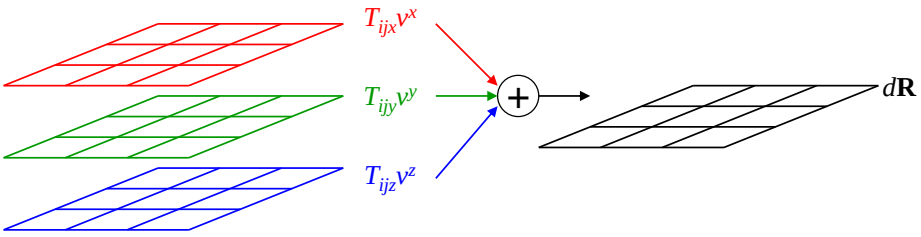
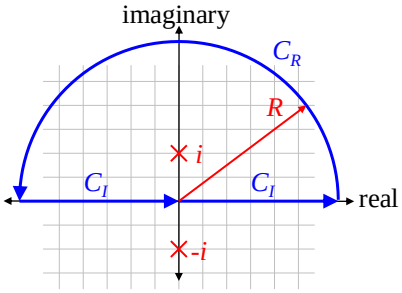
Funky Mathematical Physics Concepts

The Anti-Textbook*

A Work In Progress. See elmichelsen.physics.ucsd.edu/ for the latest versions of the Funky Series.
Please send me comments.

Eric L. Michelsen





"I study mathematics to learn how to think.
I study physics to have something to think about."

"Perhaps the greatest irony of all is not that the square root of two is irrational,
but that Pythagoras himself was irrational."

* Physical, conceptual, geometric, and pictorial physics that didn't fit in your textbook.

Please cite as: Michelsen, Eric L., *Funky Mathematical Physics Concepts*, elmichelsen.physics.ucsd.edu/, 4/3/2024.

2006 values from NIST. For more physical constants, see http://physics.nist.gov/cuu/Constants/ .

Speed of light in vacuum	c = 299 792 458 m s^{-1} (exact)
Boltzmann constant	k = 1.380 6504(24) x 10^{-23} J K^{-1}
Stefan-Boltzmann constant Relative standard uncertainty	σ = 5.670 400(40) x 10^{-8} W m^{-2} K^{-4} ±7.0 x 10^{-6}
Avogadro constant Relative standard uncertainty	N_A, L = 6.022 141 79(30) x 10^{23} mol^{-1} ±5.0 x 10^{-8}
Molar gas constant	R = 8.314 472(15) J mol^{-1} K^{-1}
Electron mass	m_e = 9.109 382 15(45) x 10^{-31} kg
Proton mass	m_p = 1.672 621 637(83) x 10^{-27} kg
Proton/electron mass ratio	m_p/m_e = 1836.152 672 47(80)
Elementary charge	e = 1.602 176 487(40) x 10^{-19} C
Electron g-factor	g_e = –2.002 319 304 3622(15)
Proton g-factor	g_p = 5.585 694 713(46)
Neutron g-factor	g_N = –3.826 085 45(90)
Muon mass	m_μ = 1.883 531 30(11) x 10^{-28} kg
Inverse fine structure constant	α^{-1} = 137.035 999 679(94)
Planck constant	h = 6.626 068 96(33) x 10^{-34} J s
Planck constant over 2π	$\hbar$ = 1.054 571 628(53) x 10^{-34} J s
Bohr radius	a_0 = 0.529 177 208 59(36) x 10^{-10} m
Bohr magneton	μ_B = 927.400 915(23) x 10^{-26} J T^{-1}

Reviews

"... most excellent tensor paper.... I feel I have come to a deep and abiding understanding of relativistic tensors.... The best explanation of tensors seen anywhere!" -- physics graduate student

Contents

1 Introduction 10
Mathematical Physics, or Physical Mathematics? 10
Why Physicists and Mathematicians Argue 10
Why Funky? 10
How to Use This Document 10
Thank You 11
Scope 11
Notation 11

2 Random Short Topics 14
I Always Lie 14
What's Hyperbolic About Hyperbolic Sine? 14
Basic Calculus You May Not Know 16
The Product Rule 17
Integration By Pictures 17
Theoretical Importance of IBP 21
Delta Function Surprise: Coordinates Matter 21
Spherical Harmonics Are Not Harmonics 24
The Binomial Theorem for Negative and Fractional Exponents 25
When Does a Divergent Series Converge? 26
Algebra Family Tree 27
Convoluted Thinking 27
Two Dimensional Convolution: Impulsive Behavior 28
Structure Functions 29
Correlation Functions 30

3 Vectors 32
Small Changes to Vectors 32
Why (r, θ, ϕ) Are *Not* the Components of a Vector 32
Laplacian's Place 33
Vector Dot Grad Vector 41

4 Green Functions 43
The Big Idea 43
Boundary Conditions on Green Functions 48
Introduction to Boundary Conditions 48
One Dimensional Boundary Conditions 49
2D?? and 3D Green Functions 55
Green Functions Don't Separate 55
Green Units 56
Special Case: Laplacian Operator with 3D Boundary Conditions 57
Desultory Green Topics 60
Fourier Series Method for Green Functions 60
Green-Like Methods: The Born Approximation 63

5 Complex Analytic Functions 65
Residues 66
Contour Integrals 67
Evaluating Integrals 67
Choosing the Right Path: Which Contour? 70
Evaluating Infinite Sums 75
Multi-valued Functions 77
Laplace Transforms: Joseph and Pierre-Simon 78

6 Conceptual Linear Algebra 81
Matrix Multiplication 81
Determinants 82
Cramer's Rule 83
Area and Volume as a Determinant 84
The Jacobian Determinant and Change of Variables 85
Expansion by Cofactors 87
Proof That the Determinant Is Unique 89
Getting Determined 90
Advanced Matrices 91
Getting to Home Basis 91
Diagonalizing a Self-Adjoint Matrix 92
Contraction of Matrices 94
Trace of a Product of Matrices 94
Linear Algebra Briefs 95

7 Introduction to Probability, Statistics, and Data Analysis 96
Probability and Random Variables 96
Precise Statement of the Question Is Critical 97
How to Lie With Statistics 98
Choosing Wisely: An Informative Puzzle 98
Multiple Events 99
Combining Probabilities 100
To B, or To Not B? 102
Continuous Random Variables and Distributions 104
Populations 104
Population Variance 105
Population Standard Deviation 105
New Random Variables From Old Ones 106
Some Distributions Have Infinite Variance, or Infinite Average 107
Samples and Parameter Estimation 108
Why Do We Use Least Squares, and Least Chi-Squared (χ^2)? 108
Average, Variance, and Standard Deviation 109
Functions of Random Variables 113
Statistically Speaking: What Is The Significance of This? 113
Predictive Power: Another Way to Be Significant, but Not Important 116
Unbiased vs. Maximum-Likelihood Estimators 116
Correlation and Dependence 118
Independent Random Variables are Uncorrelated 119
r You Serious? 120
Statistical Analysis Algebra 121
The Average of a Sum: Easy? 121
The Average of a Product 122
Variance of a Sum 122
Covariance Revisited 122
Capabilities and Limits of the Sample Variance 123
How to Do Statistical Analysis Wrong, and How to Fix It 125
Introduction to Data Fitting (Curve Fitting) 126
Goodness of Fit 127
Guidance Counselor: Computer Code to Fit Data 131

8 Multiple Linear Regression 134
Review of Multiple Linear Regression 134
We Fit to the Predictors, *Not* the Independent Variable 134
Homoskedastic Case: All Measurements Have the Same Uncertainty 137
The Raw Sum-of-Squares Identity 138

The Geometric View of a Least-Squares Fit 139
Algebra and Geometry of the Sum-of-Squares Identity 141
The ANOVA Sum-of-Squares Identity 141
The Failure of the ANOVA Sum-of-Squares Identity 142
Subtracting DC Before Analysis: Just Say No 143
Fitting to Orthonormal Functions 144
Hypothesis Testing with the Sum of Squares Identity 144
Introduction to Analysis of Variance (ANOVA) 144
The Temperature of Liberty 145
The F-test: The Decider for Zero Mean Gaussian Noise 149
Coefficient of Determination and Correlation Coefficient 150

9 Uncertainty Weighted Data Analysis 153
Be Sure of Your Uncertainty 153
Average of Uncertainty Weighted Data 153
Variance and Standard Deviation of Uncertainty Weighted Data 155
Normalized weights 157
Numerically Convenient Weights 158
Uncertainty Weighted Straight-Line Fit 158
Transformation to Equivalent Homoskedastic Measurements 158
Linear Regression with Individual Uncertainties 160
Linear Regression With Uncertainties and the Sum-of-Squares Identity 161
Hypothesis Testing a Model in Linear Regression with Uncertainties 166

10 Practical Considerations for Data Analysis 167
Rules of Thumb 167
Signal to Noise Ratio (SNR) 167
Computing SNR From Data 168
Spectral Method of Estimating SNR 169
Fitting Models To Histograms (Binned Data) 170
Reducing the Effect of Noise 173
Data With a Hard Cutoff: When Zero Just Isn't Enough 175
Filtering and Data Processing for Equally Spaced Samples 176
Finite Impulse Response Filters (aka Rolling Filters) and Boxcars 176
Use Smooth Filters (not Boxcars) 177

11 Bayesian Methods 178
Bayes' Rule 178
Bayesian Analysis 179
Refining an Estimate 179
Bayesian vs. Frequentist 181

12 Fourier Transforms and Digital Signal Processing 182
Brief Definitions 182
Model of Digitization and Sampling 183
Complex Sequences and Complex Fourier Transform 184
Basis Functions and Orthogonality 187
Real Sequences 188
Normalization and Parseval's Theorem 190
Continuous and Discrete, Finite and Infinite 191
White Noise, Correlation, and Gaussianity 191
Why Oversampling Does Not Improve Signal-to-Noise Ratio 191
Filters TBS?? 192
What Happens to a Sine Wave Deferred? 193
Don't Pad Your Data, Even for FFTs 195

Don't Pad Your Data, Even for Finer Frequency Resolution 195
Two Dimensional Fourier Transforms 197
Note on Continuous Fourier Series and Uniform Convergence 198
Nonuniform Sampling and Arbitrary Basis Functions 198
Fourier Transforms, Periodograms, and (Deprecated) Lomb-Scargle 200
The Discrete Fourier Transform vs. the Periodogram 202
Practical Considerations 203
The (Deprecated) Lomb-Scargle Algorithm 204
Bandwidth Correction (aka Bandwidth Penalty) 205
Analytic Signals and Hilbert Transforms 208
Summary 213

13 Period Finding 214
Sinusoids, Fourier, Nyquist, and All That 215
Overview of Period Finding Algorithms 215
References 215
Phase Dispersion Minimization 215
PDM Algorithm 216
PDM On Data With Trends 220
Subharmonic Response 221
Harmonic response 221
Spectral window function 222
References 222

14 Numerical Analysis 223
Round-Off Error, And How to Reduce It 223
How To Extend Precision In Sums Without Using Higher Precision Variables 224
Numerical Integration 225
Sequences of Real Numbers 225
Root Finding 225
Simple Iteration Equation 225
Newton-Raphson Iteration 227
Pseudo-Random Numbers 230
Generating Gaussian Random Numbers 231
Generating Poisson Random Numbers 231
Generating Useful, But More Challenging, Random Numbers 232
Exact Polynomial Fits 233

15 Computer Math Internals 235
Digital Integer Arithmetic: Two's Complement 235
Hexadecimal 236
Digital Floating Point 237
How Far Can I Go? 237
How Many Digits Do I Get, 6 or 9? 237
How many digits do I need? 238
Emulate vs. Simulate 238
Programming Guidance for Floating Point 239
IEEE Floating Point 240
Precision in Binary and Decimal 247
The Big ULP 247
Round and Round 248
Underflow 249
References 250

16 Scientific Programming: Discovering Efficiency 251
Software Development Efficiency 251

Some Do's and Don't's 251
Considerations on Development Efficiency and Languages 252
Sophistication Follows Function 252
Engineering vs. Programming 253
Object Oriented Programming 253
Beautiful Irony: An Object Lesson in Orientation 254
Libraries to Modules to Classes: the Growth of Program Structure 255
Libraries: Bunches of Functions 256
Modules: Libraries + Data 256
Classes: Like Modules, Only More 257
Declassified: Other Uses of Classes 258
Classes As Things That I Don't Like: the State-Pattern 259
The Best of Times, the Worst of Times: Run-time Efficiency 261
Example Using Matrix Addition 262
Memory Consumption vs. Run Time: Cache as Cache Can 266
Cache Withdrawal: Making the Most of Reference Locality 268
Data Structures for Efficient Cache Use 269
Algorithms for Efficient Cache Use 270
Cache Optimization Summary 270
Virtual Memory and Page Locality 270
Considerations on Run-Time Efficiency and Languages 272

17 Algorithms 273
Why Algorithms? 273
The Fast Fourier Transform (FFT) 273
(Deprecated) Lomb-Scargle: The Meaning Behind the Math 275

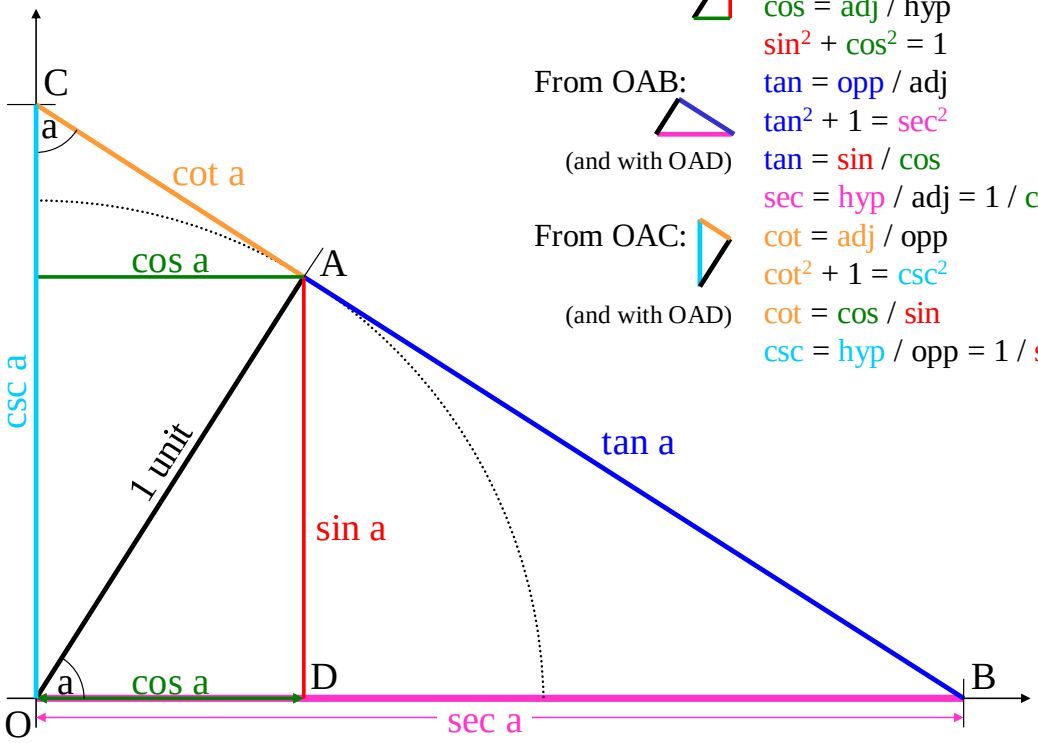
18 Tensors, Without the Tension 280
Approach 280
Two Physical Examples 280
Magnetic Susceptibility 280
Mechanical Strain 284
When Is a Matrix Not a Tensor? 286
Heading In the Right Direction 286
Some Definitions and Review 286
Vector Space Summary 287
When Vectors Collide 288
"Tensors" vs. "Symbols" 289
Notational Nightmare 289
Tensors? What Good Are They? 289
A Short, Complicated Definition 289
Building a Tensor 290
Tensors in Action 291
Tensor Fields 292
Dot Products and Cross Products as Tensors 292
The Danger of Matrices 294
Reading Tensor Component Equations 294
Adding, Subtracting, Differentiating Tensors 295
Higher Rank Tensors 295
Tensors In General 297
Change of Basis: Transformations 297
Matrix View of Basis Transformation 298
Geometric (Coordinate-Free) Dot and Cross Products 299
Non-Orthonormal Systems: Contravariance and Covariance 301
Dot Products in Oblique Coordinates 301

Covariant Components of a Vector 302
Example: Classical Mechanics with Oblique Generalized Coordinates 303
What Goes Up Can Go Down: Duality of Contravariant and Covariant Vectors 306
The *Real* Summation Convention 307
Transformation of Covariant Indexes 307
Indefinite Metrics: Relativity 308
Is a Transformation Matrix a Tensor? 308
How About the Pauli Vector? 309
Cartesian Tensors 309
The Real Reason Why the Kronecker Delta Is Symmetric 310
Tensor Appendices 310
Pythagorean Relation for 1-forms 310
Geometric Construction Of The Sum Of Two 1-Forms: 311
"Fully Anti-symmetric" Symbols Expanded 312
Metric? We Don't Need No Stinking Metric! 313
References: 315

19 Differential Geometry 316
Manifolds 316
Coordinate Bases 316
Covariant Derivatives 318
Christoffel Symbols 320
Visualization of n-Forms 321
Review of Wedge Products and Exterior Derivative 321
Wedge Products 321
Tensor Notation 322
1D 322
2D 322
3D 323

20 Math That's Fun and Interesting 325
Math Tricks That Come Up A Lot 325
The Gaussian Integral 325
Math Tricks That Are Fun and Interesting 325
Phasors 326
Miserables Coset 326
Public Key Cryptography on a Hand Calculator 327
Public Key Cryptography Overview 327
RSA Overview 328
Digital Signatures 332
Vulnerabilities 332
Secret Message 333
References 333

21 Appendices 334
References 334
Glossary 335
Formulas 340
Index 340



From OAD:
$\sin = \text{opp} / \text{hyp}$
$\cos = \text{adj} / \text{hyp}$
$\sin^2 + \cos^2 = 1$

From OAB:
$\tan = \text{opp} / \text{adj}$
$\tan^2 + 1 = \sec^2$
(and with OAD) $\tan = \sin / \cos$
$\sec = \text{hyp} / \text{adj} = 1 / \cos$

From OAC:
$\cot = \text{adj} / \text{opp}$
$\cot^2 + 1 = \csc^2$
(and with OAD) $\cot = \cos / \sin$
$\csc = \text{hyp} / \text{opp} = 1 / \sin$

1 Introduction

Mathematical Physics, or Physical Mathematics?

Is There Another Kind of Physics? Mathematical Physics is devoted to the natural emergence of mathematics from our curiosity about the universe around us. All physics is mathematical, but Mathematical Physics illustrates that math is not abstract, or capricious, but an inescapable part of the natural world. Despite its humble beginnings rooted in conceptual understanding and the practice of science, many find that Mathematical Physics holds a beauty and fascination all its own.

As with all "Funky" notes, we emphasize the physical meaning of the underlying concepts. For example, we stress a coordinate-free, geometric approach to vector operations.

Why Physicists and Mathematicians Argue

Physics goals and mathematics goals are antithetical. Physics seeks to ascribe meaning to mathematics that describe the world, to "understand" it, physically. Mathematics seeks to strip the equations of all physical meaning, and view them in purely abstract terms. These divergent goals set up a natural conflict between the two camps. Each goal has its merits: the value of physics is to make a better world; the value of mathematical abstraction, separate from any single application, is generality: the results can be used on a wide range of applications.

Why Funky?

The purpose of the "Funky" series of documents is to help develop an accurate physical, conceptual, geometric, and pictorial understanding of important physics topics. We focus on areas that don't seem to be covered well in most texts. The Funky series attempts to clarify those neglected concepts, and others that seem likely to be challenging and unexpected (funky?). The Funky documents are intended for serious students of physics; they are not "popularizations" or oversimplifications.

Physics includes math, and we're not shy about it, but we also don't hide behind it.

Without a conceptual understanding, math is gibberish.

This work is one of several aimed at graduate and advanced-undergraduate physics students. Go to our web page (in the page header) for the latest versions of the Funky Series, and for contact information. We're looking for feedback, so please let us know what you think.

How to Use This Document

This work is not a text book.

There are plenty of those, and they cover most of the topics quite well. This work is meant to be used *with* a standard text, to help emphasize those things that are most confusing for new students. When standard presentations don't make sense, come here.

You should read all of this introduction to familiarize yourself with the notation and contents. After that, this work is meant to be read in the order that most suits you. Each section stands largely alone, though the sections are ordered logically. Simpler material generally appears before more advanced topics. You may read it from beginning to end, or skip around to whatever topic is most interesting. The "Shorts" chapter is a diverse set of very short topics, meant for quick reading.

If you don't understand something, read it again once, then *keep reading*.
Don't get stuck on one thing. Often, the following discussion will clarify things.

The index is not yet developed, so go to the web page on the front cover, and text-search in this document.

Thank You

I owe a big thank you to many professors at both SDSU and UCSD, for their generosity even when I wasn't a real student: Dr. Herbert Shore, Dr. Peter Salamon, Dr. Arlette Baljon , Dr. Andrew Cooksy, Dr. George Fuller, Dr. Tom O'Neil, Dr. Terry Hwa, and others.

Scope

What This Text Covers

This text covers some of the unusual or challenging concepts in graduate mathematical physics. It is also very suitable for upper-division undergraduate level, as well. We expect that you are taking or have taken such a course, and have a good text book. *Funky Mathematical Physics Concepts* supplements those other sources.

What This Text Doesn't Cover

This text is not a mathematical physics course in itself, nor a review of such a course. We do not cover all basic mathematical concepts; only those that are very important, unusual, or especially challenging (funky?).

What You Already Know

This text assumes you understand basic integral and differential calculus, and partial differential equations. Further, it assumes you have a mathematical physics text for the bulk of your studies, and are using *Funky Mathematical Physics Concepts* to supplement it.

Notation

Sometimes the variables are inadvertently not written in italics, but I hope the meanings are clear.

?? refers to places that need more work.

TBS To be supplied (one hopes) in the future.

Interesting points that you may skip are "asides," shown in smaller font and narrowed margins. Notes to myself may also be included as asides.

Common misconceptions are sometimes written in dark red dashed-line boxes.

Formulas: We write the integral over the entire domain as a subscript "∞", for any number of dimensions:

$$\text{1-D: } \int_\infty dx \qquad \text{3-D: } \int_\infty d^3x$$

Evaluation between limits: we use the notation $[function]_a^b$ to denote the evaluation of the function between a and b, i.e.,

$$[f(x)]_a^b \equiv f(b) - f(a). \qquad \text{For example, } \int_0^1 3x^2\,dx = [x^3]_0^1 = 1^3 - 0^3 = 1.$$

We write the probability of an event as "Pr(event)."

Column vectors: Since it takes a lot of room to write column vectors, but it is often important to distinguish between column and row vectors, I sometimes save vertical space by using the fact that a column vector is the transpose of a row vector:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = (a,b,c,d)^T$$

Random variables: We use a capital letter, e.g. X, to represent the *population* from which instances of a random variable, x (lower case), are observed. In a sense, X is a representation of the PDF of the random variable, $\text{pdf}_X(x)$.

We denote that a random variable X comes from a population PDF as: $X \in \text{pdf}_X$, e.g.: $X \in \chi^2(n)$. To denote that X is a constant times a random variable from pdf_Y, we write: $X \in k\ \text{pdf}_Y$, e.g. $X \in k\ \chi^2(n)$.

For Greek letters, pronunciations, and use, see *Quirky Quantum Concepts*. Other math symbols:

Symbol	Definition
$\forall$	for all
$\exists$	there exists
$\ni$	such that
iff	if and only if
$\propto$	proportional to. E.g., $a \propto b$ means "a is proportional to b"
$\perp$	perpendicular to
$\therefore$	therefore
$\sim$	of the order of (sometimes used imprecisely as "approximately equals")
$\equiv$	is defined as; identically equal to (i.e., equal in all cases)
$\Rightarrow$	implies
$\rightarrow$	leads to
$\otimes$	tensor product, aka outer product
$\oplus$	direct sum

In mostly older texts, German type (font: Fraktur) is used to provide still more variable names:

Latin	German Capital	German Lowercase	Notes
A	$\mathfrak{A}$	$\mathfrak{a}$	Distinguish capital from U, V
B	$\mathfrak{B}$	$\mathfrak{b}$	
C	$\mathfrak{C}$	$\mathfrak{c}$	Distinguish capital from E, G
D	$\mathfrak{D}$	$\mathfrak{d}$	Distinguish capital from O, Q
E	$\mathfrak{E}$	$\mathfrak{e}$	Distinguish capital from C, G
F	$\mathfrak{F}$	$\mathfrak{f}$	
G	$\mathfrak{G}$	$\mathfrak{g}$	Distinguish capital from C, E
H	$\mathfrak{H}$	$\mathfrak{h}$	
I	$\mathfrak{I}$	$\mathfrak{i}$	Capital almost identical to J
J	$\mathfrak{J}$	$\mathfrak{j}$	Capital almost identical to I
K	$\mathfrak{K}$	$\mathfrak{k}$	
L	$\mathfrak{L}$	$\mathfrak{l}$	
M	$\mathfrak{M}$	$\mathfrak{m}$	Distinguish capital from W

N	$\mathfrak{N}$	$\mathfrak{n}$	
O	$\mathfrak{O}$	$\mathfrak{o}$	Distinguish capital from D, Q
P	$\mathfrak{P}$	$\mathfrak{p}$	
Q	$\mathfrak{Q}$	$\mathfrak{q}$	Distinguish capital from D, O
R	$\mathfrak{R}$	$\mathfrak{r}$	Distinguish lowercase from x
S	$\mathfrak{S}$	$\mathfrak{s}$	Distinguish capital from C, G, E
T	$\mathfrak{T}$	$\mathfrak{t}$	Distinguish capital from I
U	$\mathfrak{U}$	$\mathfrak{u}$	Distinguish capital from A, V
V	$\mathfrak{V}$	$\mathfrak{v}$	Distinguish capital from A, U
W	$\mathfrak{W}$	$\mathfrak{w}$	Distinguish capital from M
X	$\mathfrak{X}$	$\mathfrak{x}$	Distinguish lowercase from r
Y	$\mathfrak{Y}$	$\mathfrak{y}$	
Z	$\mathfrak{Z}$	$\mathfrak{z}$	

2 Random Short Topics

I Always Lie

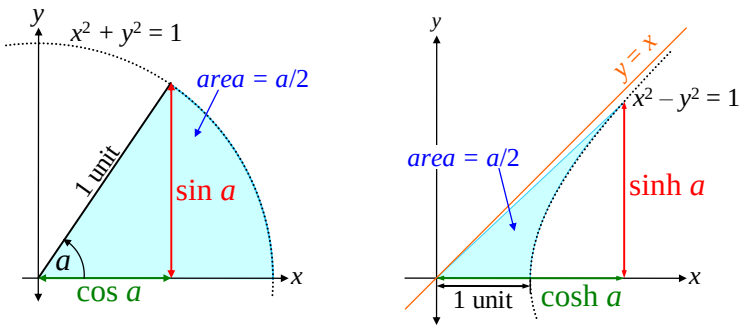
Logic, and logical deduction, are essential elements of all science. Too many of us acquire our logical reasoning abilities only through osmosis, without any concrete foundation. Unfortunately, two of the most commonly given examples of logical reasoning are both wrong. I found one in a book about Kurt Gödel (!), the famous logician.

Fallacy #1: Consider the statement, "I always lie." Wrong claim: this is a contradiction, and cannot be either true or false. Right answer: this is simply false. The negation of "I always lie" is *not* "I always tell the truth;" it is "I don't always lie," equivalent to "I at least sometimes tell the truth." Since "I always lie" cannot be true, it must be false, and it must be one of my (exceedingly rare) lies.

Fallacy #2: Consider the statement, "The barber shaves everyone who doesn't shave himself. Who shaves the barber?" Wrong answer: it's a contradiction, and has no solution. Right answer: the barber shaves himself. The original statement is about people who *don't* shave themselves; it says nothing about people who *do* shave themselves. If A then B; but if *not* A, then we know nothing about B. The barber *does* shave everyone who does not shave himself, and he also shaves one person who *does* shave himself: himself. To be a contradiction, the claim would need to be something like, "The barber shaves all and *only* those who don't shave themselves."

Logic matters.

What's Hyperbolic About Hyperbolic Sine?



From where do the hyperbolic trigonometric functions get their names? By analogy with the circular functions. We usually think of the argument of circular functions as an angle, a. But in a unit circle, the area covered by the angle a is a / 2 (above left):

$$area = \frac{a}{2\pi}\pi r^2 = \frac{a}{2} \qquad (r = 1)\,.$$

Instead of the unit circle, $x^2 + y^2 = 1$, we can consider the area bounded by the x-axis, the ray from the origin, and the unit hyperbola, $x^2 - y^2 = 1$ (above right). Then the x and y coordinates on the curve are called the **hyperbolic cosine** and **hyperbolic sine**, respectively. Notice that the hyperbola equation implies the well-known hyperbolic identity:

$$x = \cosh a, \qquad y = \sinh a, \qquad x^2 - y^2 = 1 \qquad \Rightarrow \qquad \cosh^2 - \sinh^2 = 1\,.$$

Proving that the area bounded by the x-axis, ray, and hyperbola satisfies the standard definition of the hyperbolic functions requires evaluating an elementary, but tedious, integral: (?? is the following right?)

$$area = \frac{a}{2} = \frac{1}{2}xy - \int_1^x y\,dx \qquad \text{Use:} \qquad y = \sqrt{x^2-1}$$

$$a = x\sqrt{x^2-1} - 2\int_1^x \sqrt{x^2-1}\,dx$$

For the integral, let $x = \sec\theta, \quad dx = \tan\theta\sec\theta\,d\theta \quad \Rightarrow \quad y = \sqrt{\sec^2\theta - 1} = \tan\theta$

$$\int_1^x \sqrt{x^2-1}\,dx = \int_1^x \sqrt{\sec^2\theta - 1}\,\tan\theta\sec\theta\,d\theta = \int_1^x \tan^2\theta\sec\theta\,d\theta = \int_1^x \frac{\sin^2\theta}{\cos^3\theta}\,d\theta$$

We try integrating by parts (but fail):

$$U = \tan\theta \qquad dV = \sec\theta\tan\theta\,d\theta \qquad \Rightarrow \qquad dU = \sec^2\theta\,d\theta, \quad V = \sec\theta$$

$$\int_1^x \tan^2\theta\sec\theta\,d\theta = UV - \int V\,dU = \sec\theta\tan\theta\Big|_1^x - \int_1^x \sec^3\theta\,d\theta$$

This is too hard, so we try reverting to fundamental functions sin( ) and cos( ):

$$U = \sin\theta \qquad dV = \cos^{-3}\theta\sin\theta\,d\theta \qquad \Rightarrow \qquad dU = \cos\theta\,d\theta, \quad V = \frac{1}{2}\cos^{-2}\theta$$

$$2\int_1^x \frac{\sin^2\theta}{\cos^3\theta}\,d\theta = 2UV - 2\int V\,dU = \frac{\sin\theta}{\cos^2\theta}\Big|_1^x - \int_1^x \cos^{-2}\theta\cos\theta\,d\theta \qquad \text{Use:} \quad \frac{\sin\theta}{\cos^2\theta}\Big|_1^x = \sec\theta\tan\theta = xy$$

$$= xy - \int_1^x \sec\theta\,d\theta = xy - \left(\ln\left|\sec\theta + \tan\theta\right|\right)\Big|_1^x = xy - \left(\ln\left|x + \sqrt{x^2-1}\right|\right)\Big|_1^x$$

$$= xy - \ln\left|x + \sqrt{x^2-1}\right| - \cancel{\ln 1}$$

$$a = xy - xy + \ln\left|x + \sqrt{x^2-1}\right| = \ln\left|x + \sqrt{x^2-1}\right|$$

$$e^a = x + \sqrt{x^2-1}$$

Solve for x in terms of a, by squaring both sides:

$$e^{2a} = x^2 + 2x\sqrt{x^2-1} + x^2 - 1 = 2x\underbrace{\left(x + \sqrt{x^2-1}\right)}_{e^a} - 1 = 2xe^a - 1$$

$$e^{2a} + 1 = 2xe^a$$

$$e^a + e^{-a} = 2x \qquad \Rightarrow \qquad x \equiv \cosh a = \frac{\left(e^a + e^{-a}\right)}{2}$$

The definition for sinh follows immediately from:

$$\cosh^2 - \sinh^2 = x^2 - y^2 = 1 \Rightarrow \qquad y = \sqrt{x^2-1}$$

$$\sinh a \equiv y = \sqrt{\left(\frac{e^a + e^{-a}}{2}\right)^2 - 1} = \sqrt{\frac{e^{2a} + 2 + e^{-2a}}{4} - 1} = \sqrt{\frac{e^{2a} - 2 + e^{-2a}}{4}} = \sqrt{\frac{\left(e^a - e^{-a}\right)^2}{4}} = \frac{e^a - e^{-a}}{2}$$

Basic Calculus You May Not Know

Amazingly, many calculus courses never provide a precise definition of a "limit," despite the fact that both of the fundamental concepts of calculus, derivatives and integrals, are defined as limits! So here we go:

Basic calculus relies on 4 major concepts:

1. Functions
2. Limits
3. Derivatives
4. Integrals

1. Functions: Briefly, (in real analysis) a **function** takes one or more real values as inputs, and produces one or more real values as outputs. The inputs to a function are called the **arguments**. The simplest case is a real-valued function of a real-valued argument e.g., $f(x) = \sin x$. Mathematicians would write ($f : R^1 \to R^1$), read "f is a map (or function) from the real numbers to the real numbers." A function which produces more than one output may be considered a vector-valued function.

2. Limits: Definition of "limit" (for a real-valued function of a single argument, $f : R^1 \to R^1$):

L is the **limit** of $f(x)$ as x approaches a, iff for every $\varepsilon > 0$, there exists a δ (> 0) such that $|f(x) - L| < \varepsilon$ whenever $0 < |x - a| < \delta$. In symbols:

$$L = \lim_{x \to a} f(x) \quad \text{iff } \forall \varepsilon > 0, \exists\, \delta \text{ such that } \quad |f(x) - L| < \varepsilon \quad \text{whenever } \; 0 < |x - a| < \delta \; .$$

This says that the value of the function *at a* doesn't matter; in fact, most often the function is not defined at a. However, the behavior of the function *near a* is important. If you can make the function arbitrarily close to some number, L, by restricting the function's argument to a small neighborhood around a, then L is the limit of f as x approaches a.

Surprisingly, this definition also applies to complex functions of complex variables, where the absolute value is the usual complex magnitude.

Example: Show that $\lim_{x \to 1} \dfrac{2x^2 - 2}{x - 1} = 4$.

Solution: We prove the existence of δ given any ε by computing the necessary δ from ε. Note that for $x \neq 1$, $\dfrac{2x^2 - 2}{x - 1} = 2(x + 1)$. The definition of a limit requires that

$$\left| \frac{2x^2 - 2}{x - 1} - 4 \right| < \varepsilon \quad \text{whenever} \quad 0 < |x - 1| < \delta \; .$$

We solve for x in terms of ε, which will then define δ in terms of ε. Since we don't care what the function is at $x = 1$, we can use the simplified form, $2(x + 1)$. When $x = 1$, this is 4, so we suspect the limit = 4. Proof:

$$|2(x+1) - 4| < \varepsilon \quad \Rightarrow \qquad 2|(x+1) - 2| < \varepsilon \quad \Rightarrow \qquad |x - 1| < \frac{\varepsilon}{2} \qquad \textit{or} \qquad 1 - \frac{\varepsilon}{2} < x < 1 + \frac{\varepsilon}{2} \; .$$

So by setting $\delta = \varepsilon/2$, we construct the required δ for any given ε. Hence, for every ε, there exists a δ satisfying the definition of a limit.

3. Derivatives: Only now that we have defined a limit, can we define a **derivative**:

$$f'(x) \equiv \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \; .$$

4. Integrals: A simplified definition of an **integral** is an infinite sum of areas under a function divided into equal subintervals (Figure 2.1, left):

$$\int_a^b f(x)\,dx \equiv \lim_{N\to\infty} \underbrace{\frac{b-a}{N}}_{\Delta x} \sum_{i=1}^{N} f\left((b-a)\frac{i}{N}\right) \qquad \text{(simplified definition)}.$$

For practical physics, this definition is fine. For mathematical preciseness, the actual definition of an integral is the limit over *any possible set* of subintervals [ref??], so long as the maximum of the subinterval size goes to zero. This maximum size is called "the norm of the subdivision," written as $\|\Delta x_i\|$:

$$\int_a^b f(x)\,dx \equiv \lim_{\|\Delta x_i\|\to 0} \sum_{i=1}^{N} f(x_i)\,\Delta x_i \qquad \text{(precise definition)}.$$

Figure 2.1 (Left) Simplified definition of an integral as the limit of a sum of equally spaced samples. (Right) Precise definition requires convergence for arbitrary, but small, subdivisions.

Why do mathematicians require this more precise definition? It's to avoid bizarre functions, such as: $f(x)$ is 1 if x is rational, and zero if irrational. This means $f(x)$ toggles wildly between 1 and 0 an infinite number of times over any interval. However, with the simplified definition of an integral, the following are both well defined:

$$\int_0^{3.14} f(x)\,dx = 3.14, \qquad \text{and} \qquad \int_0^{\pi} f(x)\,dx = 0 \qquad \text{(with simplified definition of integral)}.$$

In contrast, with the mathematically precise definition of an integral, both integrals are undefined. (There are other types of integrals defined, but they are beyond our scope.)

The Product Rule

Given functions $U(x)$ and $V(x)$, the product rule (aka the **Leibniz rule**) says that for differentials,

$$d(UV) = U\,dV + V\,dU\,. \tag{2.1}$$

When U and V are functions of x, we have:

$$d[U(x)V(x)] = U(x)V'(x)dx + V(x)U'(x)dx\,.$$

This leads to integration by parts, which is mostly known as an integration tool, but it is also an important theoretical (analytic) tool, and the essence of Legendre transformations.

Integration By Pictures

We assume you are familiar with integration by parts (IBP) as a tool for performing indefinite integrals. We start with a brief overview, and then discuss a specific example in detail. IBP takes a non-trivial integral into an expression with a *different* integral, which may be easier to perform analytically:

$$\int f(x)\,dx = \int U\,dV = UV - \int V\,dU \quad \textit{where} \quad U \equiv U(x), V \equiv V(x) \tag{2.2}$$

are parametric functions of x. The above comes directly from the product rule (2.1): $U\,dV = d\left(UV\right) - V\,dU$, and integrate both sides. Inserting limits of integration makes for a simple illustration of the formula's meaning (Figure 2.2a), but a slightly tedious equation:

$$\int_a^b f(x)\,dx = \int_{V(a)}^{V(b)} U\,dV = \left[UV\right]_{x=a}^{b} - \int_{U(a)}^{U(b)} V\,dU \equiv \underbrace{U(b)V(b) - U(a)V(a)}_{\text{big rectangle} - \text{small rectangle}} - \int_{U(a)}^{U(b)} V\,dU$$

$$\textit{where} \quad U \equiv U(x), V \equiv V(x).$$

The figure plots U vs. V, where we've chosen U and V to be increasing parametric functions of x. In practice, the RHS of (2.2) is usually written in terms of x as:

$$\int_a^b U(x)\underbrace{V'(x)\,dx}_{dV} = \left[U(x)V(x)\right]_{x=a}^{b} - \int_a^b V(x)\underbrace{U'(x)\,dx}_{dU}. \tag{2.3}$$

Note that x is the original integration variable (not U or V), so all the limits of integration are the original $x = a$ to $x = b$.

In practice, our job is to integrate $f(x)\,dx$ by *finding* functions $U(x)$ and $V(x)$ such that the resulting integral on the RHS of (2.3) is simpler than the original $f(x)\,dx$.

As a specific example, consider:

$$\int \underbrace{x \sin x}_{f(x)}\,dx\,.$$

Figure 2.2b illustrates the definite integral $\int_1^{2.7} f(x)\,dx$ to scale, with uniform representative intervals dx.

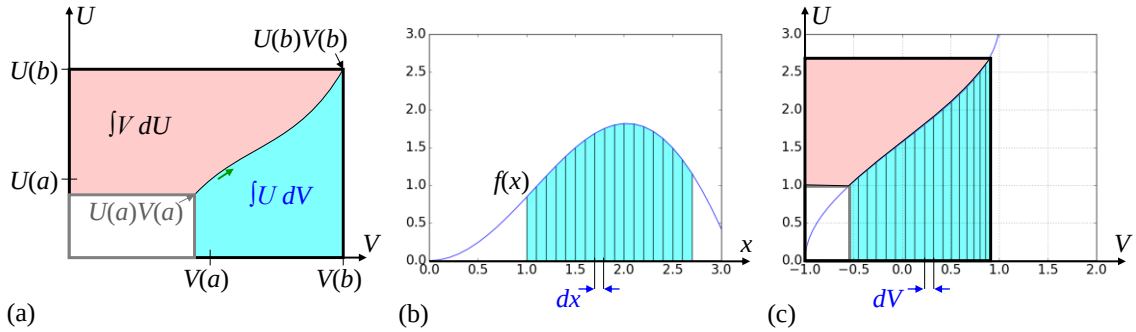


Figure 2.2 (a) Schematic identification of significant features of IBP. (b) To scale: the original integral can be reconsidered as (c) an integral of $U\,dV$; the areas are equal. U and V are parametric functions of x; dV is a function of x and dx. As shown, when the dx are uniform, the dV are not.

This integral is not immediate, so we can *try* integration by parts, though there is no guarantee that it will work. In this example, there are three ways of choosing $U(x)$ and $V(x)$:

$$U(x) = x\sin x,\ dV = dx \quad \Rightarrow \quad dU = \left(x\cos x + \sin x\right)dx,\ \ V(x) = x$$

$$U(x) = x,\ dV = \sin x\,dx \quad \Rightarrow \quad dU = dx,\ \ V(x) = -\cos x$$

$$U(x) = \sin x,\ dV = x\,dx \quad \Rightarrow \quad dU = \cos x\,dx,\ \ V(x) = x^2/2$$

More complicated integrals will have more choices for $U(x)$ and $V(x)$. It is hard to know ahead of time which choice (or choices) will succeed. However, looking at the RHS of (2.3), we see that it multiplies V and the derivative of U. Looking at our 3 choices above, on the RHS of the arrows, we find the two factors $V\,dU$ that we would be faced with integrating:

- the first choice has an ugly dU, and $V\,dU$ cannot be easily integrated;

- the second choice has $dU = dx$, which literally could not be simpler, and $V\,dU$ integrates easily;
- the last choice has $dU = \cos x\,dx$, which isn't bad, but $V\,dU$ cannot be easily integrated.

Thus our best guess is the second choice (often, the simplest dU is a good choice). Figure 2.2c illustrates $\int U\,dV$ to scale; U and V are parametric functions of x; dV is a function of x and dx. Then:

$$\int x \sin x\,dx = \underbrace{-x\cos x}_{UV} - \int \underbrace{-\cos x}_{V}\,\underbrace{dx}_{dU} = -x\cos x + \sin x\,.$$

We check by differentiating the RHS above, which yields the original integrand.

Note that when the dx in Figure 2.2b are uniform, the dV in Figure 2.2c are not. However, all the dV go to zero when the dx do, so the integral of $U\,dV$ is still valid.

The term $\left[U(x)V(x)\right]_a^b$ is called the "boundary term," or sometimes the "surface term."

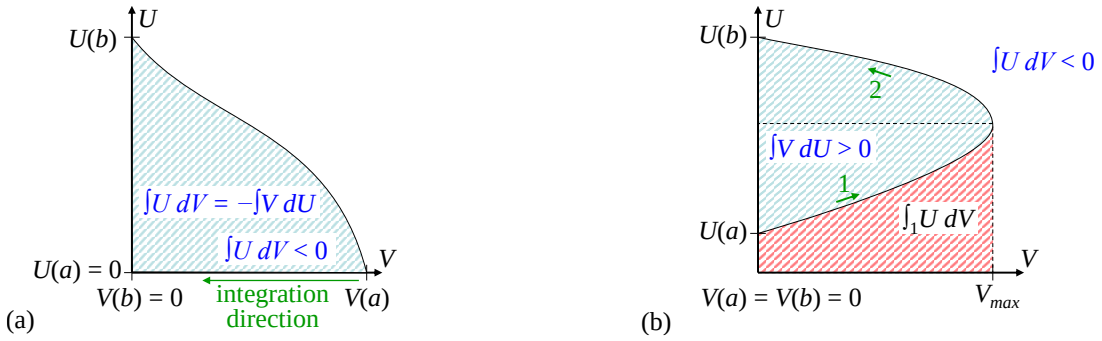


Figure 2.3 Two more cases of integration by parts: (a) $V(x)$ decreasing to 0. (b) $V(x)$ progressing from zero, to finite, and back to zero.

More advanced cases of Integration By Parts: Figure 2.3a illustrates another common case: one in which the boundary term UV is zero. In this example, $UV = 0$ at $x = a$ because $U(a) = 0$, and at $x = b$ because $V(b) = 0$. This means $V(x)$ *decreases* as x increases. Viewed as $\int U\,dV$, all the $dV < 0$. The shaded "area" is therefore negative. Viewed (sideways) as $\int V\,dU$, all the $dU > 0$ and the shaded area is positive. Thus:

$$\int f(x)\,dx = \int U\,dV = -\int V\,dU \qquad \textit{when} \qquad \left[UV\right]_a^b = 0\,,$$

in agreement with (2.3).

Figure 2.3b shows the case where $UV = 0$ at $x = a$ and b, because one of $U(x)$ or $V(x)$ starts *and* ends at 0. For illustration, we chose $V(a) = V(b) = 0$. Then the boundary term is zero, and we again have:

$$\left[U(x)V(x)\right]_{x=a}^b = 0 \qquad \Rightarrow \qquad \int_{x=a}^b U\,dV = -\int_{x=a}^b V\,dU\,.$$

For $V(x)$ to start and end at zero, $V(x)$ must grow with x to some maximum, V_{max}, and then decrease back to 0. For simplicity, we assume $U(x)$ is always increasing. The $V\,dU$ integral is the blue striped area to the left of the curve, and is > 0. The $U\,dV$ integral is the area under the curves. We break the $U\,dV$ integral into two parts: path 1, leading up to V_{max}, and path 2, going back down from V_{max} to zero. The integral from 0 to V_{max} (path 1) is the red striped area; the integral from V_{max} back down to 0 (path 2) is the negative of the entire (blue + red) striped area. Then the blue shaded region is the difference (< 0):

(1) the (red) area below path 1 (where dV is positive, because $V(x)$ is increasing), minus

(2) the (blue + red) area below path 2, where dV is negative because $V(x)$ is decreasing. Thus $\int U\,dV < 0$:

$$\underbrace{\int U\,dV}_{path1+\,path2} = \underbrace{\int_{V=0}^{V_{max}} U\,dV}_{path\,1} + \underbrace{\int_{V=V_{max}}^{0} U\,dV}_{path\,2} = \underbrace{\int_{V=0}^{V_{max}} U\,dV}_{path\,1} - \underbrace{\int_{V=0}^{V_{max}} U\,dV}_{path\,2}$$

$$= -\int_{x=a}^{b} V\,dU\ .$$

Theoretical Importance of IBP

Besides being an integration tool, an important theoretical consequence of IBP is that the variable of integration is changed, from *dV* to *dU*. Many times, one differential is unknown, but the other is known:

> Given an integral, integration by parts allows you to exchange a differential that cannot be directly evaluated, even in principle, in favor of one that can.

The classic example of this is deriving the Euler-Lagrange equations of motion from the principle of stationary action. The action of a dynamic system is defined by:

$$S \equiv \int L(q(t), \dot{q}(t))\, dt\,,$$

where the lagrangian is a given function of the trajectory $q(t)$. **Stationary action** means that the action does not change (to first order) for small changes in the trajectory. I.e., given a small variation in the trajectory, $\delta q(t)$:

$$\delta S = 0 = \int L(q + \delta q, \dot{q} + \delta\dot{q})\, dt - S = \int \left[\frac{\partial L}{\partial q}\delta q + \frac{\partial L}{\partial \dot{q}}\delta\dot{q} \right] dt\,.$$

The quantity in brackets involves both $\delta q(t)$ and its time derivative, $\delta\dot{q}(t)$. We are free to vary $\delta q(t)$ arbitrarily, but that fully determines $\delta\dot{q}(t)$. We cannot vary both δq and $\delta\dot{q}$ separately. We also know that $\delta q(t) = 0$ at its endpoints, but $\delta\dot{q}(t)$ is unconstrained at its endpoints. Therefore, it would be simpler if the quantity in brackets were written entirely in terms of $\delta q(t)$, and not in terms of $\delta\dot{q}$. This is easy:

$$\text{Use } \delta\dot{q} = \frac{d}{dt}\delta q: \qquad \delta S = 0 = \int \left[\frac{\partial L}{\partial q}\delta q + \frac{\partial L}{\partial \dot{q}}\frac{d}{dt}\delta q \right] dt\,.$$

Now in the second term, IBP allows us to eliminate the time derivative of $\delta q(t)$ (which is unconstrained) in favor of the time derivative of $\partial L / \partial\dot{q}$ (which we can easily find, since $L(q, \dot{q})$ is given). Therefore, this is a good trade. Integrating the 2nd term in brackets by parts gives:

$$\text{Let } U = \frac{\partial L}{\partial \dot{q}}, \qquad dU = \left(\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} \right) dt. \qquad dV = \frac{d}{dt}\delta q\, dt, \quad V = \delta q$$

$$\int \underbrace{\frac{\partial L}{\partial \dot{q}}}_{U} \underbrace{\frac{d}{dt}}_{V'} \delta q\, dt = UV - \int V\, dU = \cancel{\left[\frac{\partial L}{\partial \dot{q}}\delta q(t) \right]_{t=0}^{t=t}} - \int \underbrace{\delta q}_{V} \underbrace{\left(\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} \right)}_{U'} dt\,.$$

The boundary term is zero because $\delta q(t)$ is zero at both endpoints. The variation in action δS is now:

$$\delta S = \int \left[\frac{\partial L}{\partial q} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} \right] \delta q\, dt = 0 \qquad \forall\, \delta q(t)\,.$$

The only way $\delta S = 0$ can be satisfied for *any* $\delta q(t)$ is if the quantity in brackets is identically 0. Thus IBP has led us to an important *theoretical* conclusion: the Euler-Lagrange equation of motion.

This fundamental result has nothing to do with evaluating a specific difficult integral. IBP: it's not just for hard integrals any more.

Delta Function Surprise: Coordinates Matter

Rarely, one needs to consider the 3D δ-function in coordinates other than rectangular. The coordinate-free 3D δ-function is written $\delta^3(\mathbf{r} - \mathbf{r}')$. For example, in 3D Green functions, whose definition depends on a δ^3-function, it may be convenient to use cylindrical or spherical coordinates. In these cases, there are some

unexpected consequences [Wyl p280]. This section assumes you understand the basic principle of a 1D and 3D δ-function. (See the introduction to the delta function in *Quirky Quantum Concepts*.)

Recall the defining property of $\delta^3(\mathbf{r} - \mathbf{r}')$:

$$\int_\infty d^3\mathbf{r}\, \delta^3(\mathbf{r}-\mathbf{r}') = 1 \qquad \forall\, \mathbf{r}' \quad (\forall \equiv \text{" for all"}) \qquad \Rightarrow \qquad \int_\infty d^3\mathbf{r}\, \delta^3(\mathbf{r}-\mathbf{r}')\, f(\mathbf{r}) = f(\mathbf{r}')\,.$$

The above definition is "coordinate free," i.e. it makes no reference to any choice of coordinates, and is true in *every* coordinate system. As with Green functions, it is often helpful to think of the δ-function as a function of $\mathbf{r}$, which is zero everywhere except for an impulse located at $\mathbf{r}'$. As we will see, this means that it is properly a function of $\mathbf{r}$ and $\mathbf{r}'$ separately, and should be written as $\delta^3(\mathbf{r}, \mathbf{r}')$ (like Green functions are).

Rectangular coordinates: In rectangular coordinates, however, we now show that we *can* simply break up $\delta^3(x, y, z)$ into 3 components. By writing $(\mathbf{r} - \mathbf{r}')$ in rectangular coordinates, and using the defining integral above, we get:

$$\mathbf{r}-\mathbf{r}' \equiv (x-x', y-y', z-z') \qquad \Rightarrow \qquad \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy \int_{-\infty}^{\infty} dz\, \delta^3(x-x', y-y', z-z') = 1$$

$$\Rightarrow \qquad \delta^3(x-x', y-y', z-z') = \delta(x-x')\delta(y-y')\delta(z-z')\,.$$

In rectangular coordinates, the above shows that we *do* have translation invariance, so we can simply write:

$$\delta^3(x, y, z) = \delta(x)\delta(y)\delta(z)\,.$$

In other coordinates, we do *not* have translation invariance. Recall the 3D infinitesimal volume element in 4 different systems: coordinate-free, rectangular, cylindrical, and spherical coordinates:

$$d^3\mathbf{r} = dx\, dy\, dz = r\, dr\, d\phi\, dz = r^2 \sin\theta\, dr\, d\theta\, d\phi\,.$$

The presence of r and θ imply that when writing the 3D δ-function in non-rectangular coordinates, we must include a pre-factor to maintain the defining integral = 1. We now show this explicitly.

Cylindrical coordinates: In cylindrical coordinates, for $r > 0$, we have (using the imprecise notation of [Wyl p280]):

$$\mathbf{r}-\mathbf{r}' = (r-r', \phi-\phi', z-z') \qquad \Rightarrow$$

$$\int_0^\infty dr \int_0^{2\pi} d\phi \int_{-\infty}^{\infty} dz\, r\, \delta^3(r-r', \phi-\phi', z-z') = 1$$

$$\Rightarrow \qquad \delta^3(r-r', \phi-\phi', z-z') = \frac{1}{r'}\delta(r-r')\delta(\phi-\phi')\delta(z-z'), r' > 0$$

Note the $1/r'$ pre-factor on the RHS. This may seem unexpected, because the pre-factor depends on the location of $\delta^3(\,)$ in space (hence, no radial translation invariance). The rectangular coordinate version of $\delta^3(\,)$ has no such pre-factor. Properly speaking, $\delta^3(\,)$ isn't a function of $r - r'$; it is a function of r and r' separately.

In non-rectangular coordinates, $\delta^3(\,)$ does not have translation invariance, and includes a pre-factor which depends on the position of $\delta^3(\,)$ in space, i.e. depends on $\mathbf{r}'$.

At $r' = 0$, the pre-factor blows up, so we need a different pre-factor. We'd like the defining integral to be 1, regardless of ϕ, since all values of ϕ are equivalent at the origin. This means we must drop the $\delta(\phi - \phi')$, and replace the pre-factor to cancel the constant we get when we integrate out ϕ:

$$\int_0^\infty dr \int_0^{2\pi} d\phi \int_{-\infty}^{\infty} dz\; r\; \delta^3(r-r', \phi-\phi', z-z') = 1, \qquad r' = 0$$

$$\Rightarrow \qquad \delta^3(r-r', \phi-\phi', z-z') = \frac{1}{2\pi r}\delta(r)\delta(z-z'), \quad r' = 0,$$

assuming that $\int_0^\infty dr\; \delta(r) = 1.$

This last assumption is somewhat unusual, because the δ-function is usually thought of as symmetric about 0, where the above radial integral would only be ½. The assumption implies a "right-sided" δ-function, whose entire non-zero part is located at 0^+. Furthermore, notice the factor of $1/r$ in $\delta(r-0, z-z')$. This factor blows up at $r = 0$, and has no effect when $r \neq 0$. Nonetheless, it is needed because the volume element $r\, dr\, d\phi\, dz$ goes to zero as $r \to 0$, and the $1/r$ in $\delta(r-0, z-z')$ compensates for that.

Spherical coordinates: In spherical coordinates, we have similar considerations. First, away from the origin, $r' > 0$:

$$\int_0^\infty dr \int_0^{\pi} d\theta \int_0^{2\pi} d\phi\, r^2 \sin\theta\, \delta^3(r-r', \theta-\theta', \phi-\phi') = 1 \;\Rightarrow$$

$$\delta^3(r-r', \theta-\theta', \phi-\phi') = \frac{1}{r'^2 \sin\theta'}\delta(r-r')\delta(\theta-\theta')\delta(\phi-\phi'), \quad r' > 0. \quad \text{[Wyl 8.9.2 p280]}$$

Again, the pre-factor depends on the position in space, and properly speaking, $\delta^3(\,)$ is a function of r, r', θ, and θ' separately, not simply a function of $r - r'$ and $\theta - \theta'$. At the origin, we'd like the defining integral to be 1, regardless of ϕ or θ. So we drop the $\delta(\phi - \phi')\, \delta(\theta - \theta')$, and replace the pre-factor to cancel the constant we get when we integrate out ϕ and θ:

$$\int_0^\infty dr \int_0^{\pi} d\theta \int_0^{2\pi} d\phi\, r^2 \sin\theta\, \delta^3(r-0, \theta-\theta', \phi-\phi') = 1, \qquad r' = 0$$

$$\Rightarrow \qquad \delta^3(r-0, \theta-\theta', \phi-\phi') = \frac{1}{4\pi r^2}\delta(r), \qquad r' = 0,$$

assuming that $\int_0^\infty dr\; \delta(r) = 1.$

Again, this definition uses the modified $\delta(r)$, whose entire non-zero part is located at 0^+. And similar to the cylindrical case, this includes the $1/r^2$ factor to preserve the integral at $r = 0$.

2D angular coordinates: For 2D angular coordinates θ and ϕ, we have:

$$\int_0^{\pi} d\theta \int_0^{2\pi} d\phi \sin\theta\, \delta^2(\theta-\theta', \phi-\phi') = 1, \qquad \theta' > 0$$

$$\Rightarrow \qquad \delta^2(\theta-\theta', \phi-\phi') = \frac{1}{\sin\theta'}\delta(\theta-\theta')\delta(\phi-\phi'), \quad \theta' > 0.$$

Once again, we have a special case when $\theta' = 0$: we must have the defining integral be 1 for any value of ϕ. Hence, we again compensate for the 2π from the ϕ integral:

$$\int_0^{\pi} d\theta \int_0^{2\pi} d\phi \sin\theta\, \delta^2(\theta-\theta', \phi-\phi') = 1, \qquad \theta' = 0$$

$$\Rightarrow \qquad \delta^2(\theta-0, \phi-\phi') = \frac{1}{2\pi \sin\theta}\delta(\theta), \qquad \theta' = 0.$$

Similar to the cylindrical and spherical cases, this includes a $1/(\sin\theta)$ factor to preserve the integral at $\theta = 0$.

Spherical Harmonics Are Not Harmonics

See *Funky Electromagnetic Concepts* for a full discussion of harmonics, Laplace's equation, and its solutions in 1, 2, and 3 dimensions. Here is a brief overview.

Spherical harmonics are the angular parts of solid harmonics, but we will show that they are not truly "harmonics." A **harmonic** is a function which satisfies Laplace's equation:

$\nabla^2\Phi(\mathbf{r}) = 0$, with **r** typically in 2 or 3 dimensions.

Solid harmonics are 3D harmonics: they solve Laplace's equation in 3 dimensions. For example, one form of solid harmonics separates into a product of 3 functions in spherical coordinates:

$$\Phi(r,\theta,\phi) = R(r)P(\theta)Q(\phi) = \left(A_l r^l + B_l r^{-(l+1)}\right) P_l m(\cos\theta)\left(C_l \sin m\phi + D_l \cos m\phi\right)$$

where $R(r) = A_l r^l + B_l r^{-(l+1)}$ is the radial part,

$P(\theta) = P_{lm}(\cos\theta)$ is the polar angle part, the associated Legendre functions,

$Q(\phi) = \left(C_l \sin m\phi + D_l \cos m\phi\right)$ is the azimuthal part .

The spherical harmonics are just the angular (θ, ϕ) parts of these solid harmonics. But notice that the angular part alone does not satisfy the 2D Laplace equation (i.e., on a sphere of fixed radius):

$$\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2}{\partial\phi^2}, \qquad \text{but for fixed } r:$$

$$= \frac{1}{r^2}\left[\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\phi^2}\right].$$

However, direct substitution of spherical harmonics into the above Laplace operator shows that the result is *not* 0 (we let $r = 1$). We proceed in small steps:

$$Q(\phi) = C\sin m\phi + D\cos m\phi \qquad \Rightarrow \qquad \frac{\partial^2}{\partial\phi^2}Q(\phi) = -m^2 Q(\phi).$$

For integer m, the **associated Legendre functions**, $P_{lm}(\cos\theta)$, satisfy, for given l and m:

$$\frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right)P_{lm}(\cos\theta) = \left(-\frac{l(l+1)}{r^2} + m^2\right)P_{lm}(\cos\theta).$$

Combining these 2 results ($r = 1$):

$$\nabla^2\left(P(\theta)Q(\phi)\right) = \left[\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\phi^2}\right]\left(P(\theta)Q(\phi)\right)$$

$$= \left(-l(l+1) + m^2\right)P_{lm}(\cos\theta)Q(\theta) - m^2 P_{lm}(\cos\theta)Q(\phi)$$

$$= -l(l+1)P_{lm}(\cos\theta)Q(\phi)$$

Hence, the spherical harmonics are *not* solutions of Laplace's equation,
i.e. they are *not* "harmonics."

The Binomial Theorem for Negative and Fractional Exponents

You may be familiar with the **binomial theorem** for positive integer exponents, but it is very useful to know that the binomial theorem also works for negative and fractional exponents. We can use this fact to easily find series expansions for things like $\frac{1}{1-x}$ and $\sqrt{1+x} = (1+x)^{1/2}$.

First, let's review the simple case of positive integer exponents:

$$(a+b)^n = a^n b^0 + \frac{n}{1}a^{n-1}b^1 + \frac{n(n-1)}{1\cdot 2}a^{n-2}b^2 + \frac{n(n-1)(n-2)}{1\cdot 2\cdot 3}a^{n-3}b^3 + \ldots \frac{n!}{n!}a^0 b^n .$$

[For completeness, we note that we can write the general form of the m^{th} term:

$$m^{th}\text{ term} = \frac{n!}{(n-m)!m!}a^{n-m}b^m, \qquad n \text{ integer} > 0; \quad m \text{ integer}, 0 \le m \le n .]$$

But we're much more interested in the iterative procedure (recursion relation) for finding the $(m+1)^{th}$ term from the m^{th} term, because we use that to generate a power series expansion. The process is this:

1. The first term ($m = 0$) is always $a^n b^0 = a^n$, with an implicit coefficient $C_0 = 1$.
2. To find C_{m+1}, multiply C_m by the power of a in the m^{th} term, $(n - m)$,
3. divide it by $(m + 1)$, [the number of the new term we're finding]: $C_{m+1} = \frac{(n-m)}{m+1}C_m$
4. lower the power of a by 1 (*to* $n - m$), and
5. raise the power of b by 1 to $(m + 1)$.

This procedure is valid for all n, even negative and fractional n. A simple way to remember this is:

> For any real n, we generate the $(m+1)^{th}$ term from the m^{th} term
> by differentiating with respect to a, and integrating with respect to b.

The general expansion, for *any* n, is then:

$$m^{th}\ term = \frac{n(n-1)(n-2)\ldots(n-m+1)}{m!}a^{n-m}b^m, \qquad n \text{ real}; \quad m \text{ integer} \ge 0$$

Notice that for integer $n > 0$, there are $n+1$ terms. For fractional or negative n, we get an infinite series.

Example 1: Find the Taylor series expansion of $\frac{1}{1-x}$. Since the Taylor series is unique, any method we use to find a power series expansion will give us the Taylor series. So we can use the binomial theorem, and apply the rules above, with $a = 1$, $b = (-x)$:

$$\frac{1}{1-x} = (1+(-x))^{-1} = 1^{-1} + \frac{(-1)}{1}1^{-2}(-x)^1 + \frac{(-1)(-2)}{1\cdot 2}1^{-3}(-x)^2 + \frac{(-1)(-2)(-3)}{1\cdot 2\cdot 3}1^{-4}(-x)^3 + \ldots$$

$$= 1 + x + x^2 + \ldots + x^m + \ldots$$

Notice that all the fractions, all the powers of 1, and all the minus signs cancel.

Example 2: Find the Taylor series expansion of $\sqrt{1+x} = (1+x)^{1/2}$. The first term is $a^{1/2} = 1^{1/2}$:

$$(1+x)^{1/2} = 1^{1/2} + \frac{1}{2}\frac{1}{(1)}1^{-1/2}x^1 + \frac{1}{2}\left(-\frac{1}{2}\right)\frac{1}{(1\cdot 2)}1^{-3/2}x^2 + \frac{1}{2}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\frac{1}{(1\cdot 2\cdot 3)}1^{-5/2}x^3 + \ldots$$

$$= 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{3}{48}x^3 - \ldots + (-1)^{m+1}\frac{(2m-3)!!}{2^m m!}x^m$$

where $p!! \equiv p(p-2)(p-4)\ldots(2 \text{ or } 1)$

When Does a Divergent Series Converge?

Sometimes, a divergent series "converges." Consider the infinite series:

$$1 + x + x^2 + \ldots + x^n + \ldots .$$

When is it convergent? Apparently, when $|x| < 1$. What is the value of the series when $x = 2$? "Undefined!" you say. But there is a very important sense in which the series *does* converge for $x = 2$, and it's value is –1! How so?

Recall the Taylor expansion around $x = 0$ (you can use the binomial theorem, see earlier section):

$$\frac{1}{1-x} = (1-x)^{-1} = 1 + x + x^2 + \ldots + x^n + \ldots .$$

This is exactly the original infinite series above. So the series sums to $1/(1-x)$. This expression is defined for all $x \neq 1$. And its value for $x = 2$ is –1.

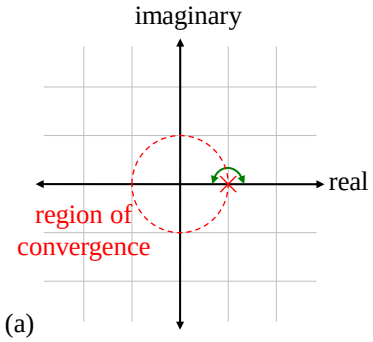


Figure 2.4 Domain of $1/(1-x)$ in the complex plane. The function is analytically continued around the pole at $x = 1$, which defines meaningful values of the function even when x is outside the region of convergence.

Why is this important? There are cases in physics when we use perturbation theory to find an expansion of a number (or function, as in QFT) in an infinite series. Sometimes, the series appears to diverge. But by finding the analytic expression corresponding to the series, we can evaluate that analytic expression at values of x that make the series diverge. In many cases, the analytic expression provides an important and meaningful answer to a perturbation problem even *outside* the original region of convergence. This happens in quantum mechanics, and quantum field theory (e.g., [M&S 2010 p291t]).

This is an example of *analytic continuation* in complex analysis. Figure 2.4 illustrates the domain of our function $1/(1-x)$ in the *complex* plane. A Taylor series is a special case of a Laurent series, and anywhere a function has a Laurent expansion it is **analytic**. If we know the Laurent series (or if we know the values of an analytic function and all its derivatives at *any one* point), then we know the function everywhere, even for complex values of x. Here, the original series is analytic around $x = 0$, with a radius of convergence of 1. However, the process of extending a function that is defined in some region to be defined in a larger (complex) region, is called **analytic continuation** (see Complex Analysis, discussed elsewhere in this document). This gives our function meaningful values for all $x \neq 1$, such as $x = 2$. Thus analytic continuation through the complex plane allows us to "hop over" the pole on the real axis, and define the function for real $x > 1$ (and for $x < -1$).

TBS: show that the sum of the integers 1 + 2 + 3 + ... = –1/12. ??

Algebra Family Tree

Doodad	Properties	Examples
group	Finite or infinite set of elements and operator (·), with closure, associativity, identity element and inverses. Possibly commutative: $a \cdot b = c$ w/ a, b, c group elements	rotations of a square by $n \times 90^o$ continuous rotations of an object
ring	Set of elements and 2 binary operators (+ and *), with: • commutative *group* under + • left and right distributivity: $a(b + c) = ab + ac$, $(a + b)c = ac + bc$ • usually multiplicative associativity	integers mod m polynomials $p(x)$ mod $m(x)$
integral domain, or **domain**	A *ring*, with: • commutative multiplication • multiplicative identity (but no inverses) • no zero divisors ($\Rightarrow$ cancellation is valid): $ab = 0$ only if $a = 0$ or $b = 0$	integers polynomials, even abstract polynomials, with abstract variable x, and coefficients from a "field"
field	"rings with multiplicative inverses (& identity)" • commutative *group* under addition • commutative *group* (excluding 0) under multiplication. • distributivity, multiplicative inverses Allows solving simultaneous linear equations. Field can be finite or infinite	integers with arithmetic modulo 3 (or any prime) real numbers complex numbers
vector space	• *field* of scalars • *group* of vectors under +. Allows solving simultaneous vector equations for unknown scalars or vectors. Finite or infinite dimensional.	physical vectors real or complex functions of space: $f(x, y, z)$ kets (and bras)
Hilbert space	*vector space* over field of complex numbers with: • a conjugate-bilinear inner product, $\langle av\|bw\rangle = (a^*)b\langle v\|w\rangle$, $\langle v\|w\rangle = \langle w\|v\rangle^*$ a, b scalars, and v, w vectors • Mathematicians require it to be infinite dimensional; physicists don't.	real or complex functions of space: $f(x, y, z)$ quantum mechanical wave functions

Convoluted Thinking

Convolution arises in many physics, engineering, statistics, and other mathematical areas. As examples, we here consider functions of time, but the concept of convolution may apply to functions of space, or anything else. Given two functions, $f(t)$ and $g(t)$, the convolution of $f(t)$ and $g(t)$ is a function of a time-displacement, Δt, defined by (Figure 2.5):

$$(f * g)(\Delta t) \equiv \int_{-\infty}^{\infty} d\tau \; f(\tau) g(\Delta t - \tau) \quad \textit{where} \quad \text{the integral covers some domain of interest .}$$

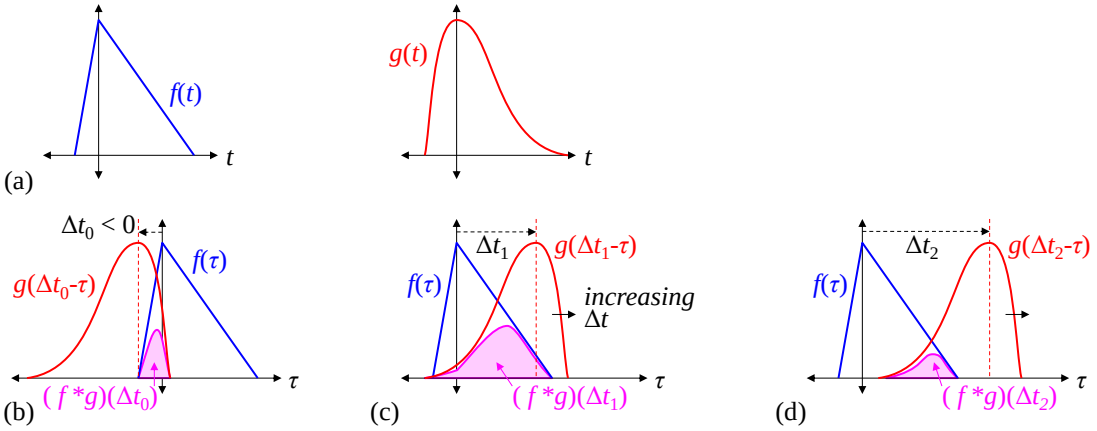


Figure 2.5 (a) Two functions, $f(t)$ and $g(t)$. (b) $(f * g)(\Delta t_0)$, $\Delta t_0 < 0$. (c) $(f * g)(\Delta t_1)$, $\Delta t_1 > 0$. (d) $f * g)(\Delta t_2)$, $\Delta t_2 > \Delta t_1$. The convolution is the magenta shaded area.

When $\Delta t < 0$, the two functions are "backing into each other" (above left). When $\Delta t > 0$, the two functions are "backing away from each other" (above middle and right).

As noted at the beginning, convolution is useful with a variety of independent variables besides time. E.g., for functions of space, $f(x)$ and $g(x)$, $f * g(\Delta x)$ is a function of spatial displacement, Δx.

Notice that convolution is

(1) commutative: $f * g = g * f$

(2) *linear* in each of the two functions:

$$f * kg = k(f * g) = (kf) * g, \quad \text{and}$$
$$f * (g + h) = f * g + f * h.$$

The verb "to convolve" means "to form the convolution of." We convolve f and g to form the convolution $f * g$. Some references use "$\otimes$" for convolution: $f \otimes g$.

Two Dimensional Convolution: Impulsive Behavior

A translation invariant linear system (TILS) is completely described by its impulse response. For example, for small angles, equivalent to narrow fields of view, an optical imaging system is approximately a TILS. In optics, the impulse response is called the Point Spread Function, or PSF. To illustrate the use of convolution in a TILS, consider an optical imager (Figure 2.6).

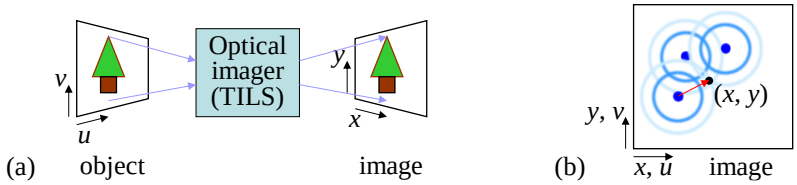


Figure 2.6 (a) Optical imager is a TILS. (b) Example image of 3 point sources, with a representative image point. Each source is spread out by the imager according to the PSF. The red arrow is the vector (**x** – **u**).

The imager has finite resolution, so a point object is spread over a region in the image. For a point object at the origin with intensity O, the image has intensity distributed over space according to:

$$I(x, y) = O \cdot PSF(x, y).$$

x and y are position coordinates, such as meters or microradians. We define the object coordinates (u, v) to be those of their image points (so we can ignore magnification). Then translation invariance says that for a point object at (u, v):

$$I(x,y) = O \cdot PSF(x-u, y-v) .$$

For incoherent sources, intensities add, so multiple point sources produce an image intensity that is the sum of the individual images (Figure 2.6b). Therefore, the PSF is real, and represents the *intensity* response function (rather than field amplitude). At each point on the image (x, y):

$$I(x,y) = O_1 PSF(x-u_1, y-v_1) + O_2 PSF(x-u_2, y-v_2) + O_3 PSF(x-u_3, y-v_3)$$

$$= \sum_{i=1}^{3} PSF(x-u_i, y-v_i)$$

For a continuous object, each infinitesimal region of size (du, dv) around each point (u, v) is essentially a point source. The image is the infinite sum of images of all these "point" sources. Then the sum above becomes a continuous integral:

$$I(x,y) = \iint_{\text{object}} du\, dv\, O(u,v)\, PSF(x-u, y-v) \equiv O \otimes PSF . \tag{2.4}$$

This is the definition of a 2D convolution. Some references use "*" for convolution: $O*PSF$.

In general, for a TILS:

> A convolution is an infinite sum of responses to a continuous input.
> Translation invariant linear systems are fully described by their impulse response (aka PSF). The output of such a system is the convolution of the input with the PSF.

All of the above is true for arbitrary PSF, symmetric or not. Some systems exhibit symmetry, e.g. many optical systems are axially symmetric. In such a symmetric case, the arguments to the PSF may be negated, though we find such expressions misleading.

For coherent systems, the PSF is generally complex, and it denotes the magnitude and phase of the light at the image relative to the object. Such a PSF represents the field *amplitude* response function (rather than intensity).

In vector notation, the convolution (2.4) can be written:

$$I(\mathbf{x}) = \iint_{\text{object}} d^2u\, O(\mathbf{u})\, PSF(\mathbf{x}-\mathbf{u}) \equiv O \otimes PSF .$$

Structure Functions

The term "correlation" has two distinct meanings, both of which are used in astronomy: (1) correlation between random variables, and (2) correlation between functions (of space or of time). In both meanings, correlations are used to compare two things. For example, we might compare light, as a function of time, at point A in space with that at point B, i.e. $I_A(t)$ compared to $I_B(t)$. If these intensities vary randomly in time, we might ask, how are the two related?

Correlations between random variables: The **correlation** of two random variables (RVs) describes to what extent the two RVs are *linearly* related to each other. The correlation is quantified with a **correlation coefficient** ρ, where $\rho = 1$ means the two RVs are actually identical. ρ is proportional to the covariance of the RVs. Two uncorrelated RVs have *no* linear relationship (though they may be related in other ways), and $\rho = 0$. (See *Funky Mathematical Physics Concepts* for details.)

In many systems, there are an infinite number of RVs, one at each point in space. For example, above a telescope, at each atmospheric space point $\mathbf{x}$, there may be a randomly-varying temperature $T(t, \mathbf{x})$, index of refraction $N(t, \mathbf{x})$, or optical phase $\phi(t, \mathbf{x})$. The variations are over *time*. It is common that there are correlations between the RVs at different points in space. For two very nearby points, ρ is near 1: the two

RVs are almost identical. For far separations, ρ is near 0, because the two RVs are essentially unrelated. In general, at two points $\mathbf{x}_1$ and $\mathbf{x}_2$, and near some time t_0, using optical phase as an example, the *two-point structure function* is [Quirr eq. 1]:

$$D_\phi(\mathbf{x}_1, \mathbf{x}_2) \equiv \left\langle \left|\phi(t, \mathbf{x}_1) - \phi(t, \mathbf{x}_2)\right|^2 \right\rangle_{T \sim \text{seconds}} \equiv \int_{t_0}^{t_0+T} dt \left|\phi(t, \mathbf{x}_1) - \phi(t, \mathbf{x}_2)\right|^2 .$$

The averaging duration is of the order of the exposure time, typically some seconds [Fried 1966 Sec III-IV]. The weather typically changes much slower, of order at least minutes. For translation invariant, isotropic systems, the above depends only on the spatial distance $r \equiv |\mathbf{x}_1 - \mathbf{x}_2|$. This defines a *structure function* of a single variable, the distance r:

$$D_\phi(r) \equiv \left\langle \left|\phi(\mathbf{x}_1) - \phi(\mathbf{x}_1 + \mathbf{r})\right|^2 \right\rangle_{T \sim \text{seconds}} .$$

Since the system is translation invariant, D_ϕ can be evaluated at any choice of $\mathbf{x}_1$. Because the system is isotropic, D_ϕ can be evaluated at any $\mathbf{r}$ such that $|\mathbf{r}| = r$.

A structure function $D(r)$ gives the correlation (linear relationship) for a time-varying physical quantity between two space points separated by a distance r.

Correlation Functions

The correlation between two functions is a measure of how linearly related they are. The functions are often functions of time, or functions of space. A measure of their linear relationship is given by the integral of their product:

$$C_{fg} \equiv \int_{-\infty}^{\infty} dt\ f(t)g(t) \qquad \text{or}$$

$$C_{fg} \equiv \int_{-\infty}^{\infty} dx\ f(x)g(x) \qquad \text{or} \qquad C_{fg} \equiv \iiint_{\text{volume}} d^3\mathbf{x}\ f(\mathbf{x})g(\mathbf{x}) .$$

It is often useful to compare the two function with some offset in one of them. Then the correlation is a function of this offset:

$$C_{fg}(\tau) \equiv \int_{-\infty}^{\infty} dt\ f(t+\tau)g(t) \qquad \text{or} \qquad C_{fg}(\mathbf{r}) \equiv \iiint_{\text{volume}} d^3x\ f(\mathbf{x}+\mathbf{r})g(\mathbf{x}) .$$

For two unrelated zero-mean functions, the correlation function is zero.

It is frequently useful to compute the correlation of a function with an offset version of itself, called the **autocorrelation function**. For example, at a fixed instant in time, consider the temperature *variations* throughout the 3D atmosphere, $T(\mathbf{x})$. Then:

$$C_{TT}(\mathbf{r}) \equiv \iiint_{\text{volume}} d^3x\, T(\mathbf{x}+\mathbf{r})T(\mathbf{x}) .$$

We expect that nearby temperatures are similar, and that distant temperatures are unrelated. Since $T(\mathbf{x})$ is zero-mean, we expect the autocorrelation function to be large for small offset, and small for large offset. The distance at which the autocorrelation becomes small is a measure of the size of atmospheric volumes with similar temperature. A 2D or higher autocorrelation function is not necessarily isotropic. For example, the temperature may vary differently in the vertical direction than in horizontal ones.

References

[Fried 1966] D. L. Fried, "Optical Resolution Through a Randomly Inhomogeneous Medium for Very Long and Very Short Exposures," Journal of the Optical Society of America, Volume 56, Number 10 October 1966, p1372.

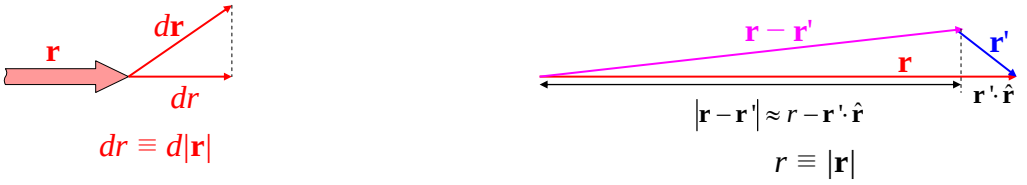
[Quirr] Andreas Quirrenbach, "The Effects of Atmospheric Turbulence on Astronomical Observations," unpublished,

https://pdfs.semanticscholar.org/d6bc/66a77fdabd708e06c6c61fc96f6647101920.pdf, retrieved 2020-01-25.

3 Vectors

Small Changes to Vectors

Projection of a Small Change to a Vector Onto the Vector



(Left) A small change to a vector, and its projection onto the vector. (Right) Approximate magnitude of the difference between a big and small vector.

It is sometimes useful (in orbital mechanics, for example) to relate the change in a vector to the change in the vector's magnitude. The diagram above (left) leads to a somewhat unexpected result:

$d\mathbf{r}\cdot\hat{\mathbf{r}} = dr$ *or* (multiplying both sides by r and using $\mathbf{r} = r\hat{\mathbf{r}}$)

$\mathbf{r}\cdot d\mathbf{r} = r\,dr$

And since this is true for any small change, it is also true for any rate of change (just divide by dt):

$\mathbf{r}\cdot\dot{\mathbf{r}} = r\,\dot{r}$

Vector Difference Approximation

It is sometimes useful to approximate the magnitude of a large vector minus a small one. (In electromagnetics, for example, this is used to compute the far-field from a small charge or current distribution.) The diagram above (right) shows that:

$|\mathbf{r} - \mathbf{r}'| \approx |\mathbf{r}| - \mathbf{r}'\cdot\hat{\mathbf{r}}, \qquad |\mathbf{r}| >> |\mathbf{r}'|$

Why (r, θ, ϕ) Are *Not* the Components of a Vector

(r, θ, ϕ) are *parameters* of a vector, but not *components*. That is, the parameters (r, θ, ϕ) uniquely define the vector, but they are not components, because you can't add them. This is important in much physics, e.g. involving magnetic dipoles (ref Jac problem on mag dipole field). **Components** of a vector are *defined* as coefficients of basis vectors. For example, the components $\mathbf{v}$ = (x, y, z) can multiply the basis vectors to construct $\mathbf{v}$:

$$\mathbf{v} = x\hat{\mathbf{x}} + y\hat{\mathbf{y}} + z\hat{\mathbf{z}}$$

There is no similar equation we can write to construct $\mathbf{v}$ from it's spherical components (r, θ, ϕ). Position vectors are displacements from the origin, and there are no $\hat{\mathbf{r}}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\varphi}}$ defined at the origin.

Put another way, you can always add the components of two vectors to get the vector sum:

Let $\mathbf{w} = (a, b, c)$ rectangular components. *Then* $\mathbf{v} + \mathbf{w} = (a + x)\hat{\mathbf{x}} + (b + y)\hat{\mathbf{y}} + (c + z)\hat{\mathbf{z}}$

We can't do this in spherical coordinates:

Let $\mathbf{w} = (r_w, \theta_w, \phi_w)$ spherical components. *Then* $\mathbf{v} + \mathbf{w} \neq (r_v + r_w, \theta_v + \theta_w, \phi_v + \phi_w)$

However, at a point off the origin, the basis vectors $\hat{\mathbf{r}}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\varphi}}$ are well defined, and can be used as a basis for general vectors. [In differential geometry, vectors referenced to a point in space are called **tangent**

vectors, because they are "tangent" to the space, in a higher dimensional sense. See Differential Geometry elsewhere in this document.]

Laplacian's Place

What is the physical meaning of the Laplacian operator? And how can I remember the Laplacian operator in any coordinates? These questions are related because understanding the physical meaning allows you to quickly derive in your head the Laplacian operator in any of the common coordinates.

Let's take a step-by-step look at the action of the Laplacian, first in 1D, then on a 3D differential volume element, with physical examples at each step. After rectangular, we go to spherical coordinates, because they illustrate all the principles involved. Finally, we apply the concepts to cylindrical coordinates, as well. We follow this outline:

1. Overview of the Laplacian operator
2. 1D examples of heat flow
3. 3D heat flow in rectangular coordinates
4. Examples of physical scalar fields [temperature, pressure, electric potential (2 ways)]
5. 3D differential volume elements in other coordinates
6. Description of the physical meaning of Laplacian operator terms, such as

$$\nabla T, \qquad \frac{\partial T}{\partial r}, \qquad r^2 \frac{\partial T}{\partial r}, \qquad \frac{\partial}{\partial r}\left(r^2 \frac{\partial T}{\partial r} \right), \qquad r^2 \frac{\partial}{\partial r}\left(r^2 \frac{\partial T}{\partial r} \right).$$

Overview of Laplacian operator: Let the Laplacian act on a scalar field $T(\mathbf{r})$, a physical function of space, e.g. temperature. Usually, the Laplacian represents the net outflow per unit volume of some physical quantity: something/volume, e.g., something/m^3. The Laplacian operator itself involves spatial second-derivatives, and so carries units of inverse area, say m^{-2}.

1D Example: Heat Flow: Consider a temperature gradient along a line. It could be a perpendicular wire through the wall of a refrigerator (Figure 3.1a). It is a 1D system, i.e. only the gradient *along* the wire matters.

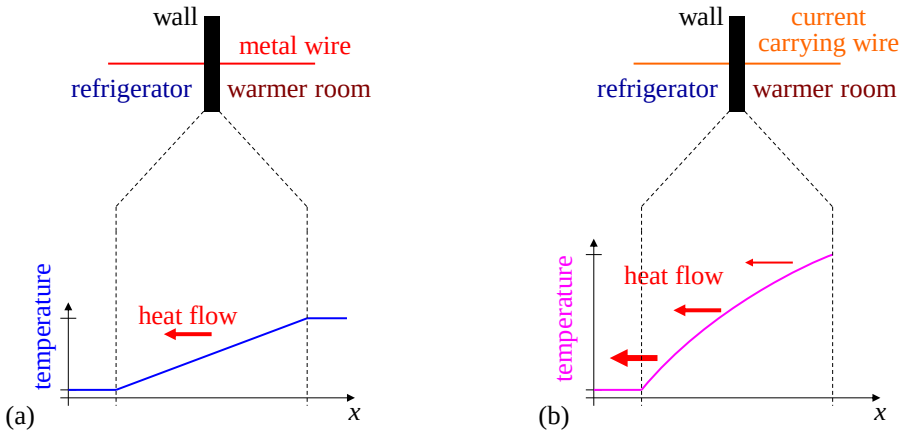


Figure 3.1 Heat condition (a) in a passive wire, and (b) in a heat-generating wire.

Let the left and right sides of the wire be in thermal equilibrium with the refrigerator and room, at 2 C and 27 C, respectively. The wire is passive, and can neither generate nor dissipate heat; it can only conduct it. Let the 1D thermal conductivity be k = 100 mW-cm/C. Consider the part of the wire inside the insulated wall, 4 cm thick. How much heat (power, J/s or W) flows through the wire?

$$P = k\frac{dT}{dx} = (100 \text{ mW-cm/C})\frac{25\,C}{4 \text{ cm}} = 625 \text{ mW}\,.$$

There is no heat generated or dissipated in the wire, so the heat that flows into the right side of any segment of the wire (differential or finite) must later flow out the left side. Thus, the heat flow must be constant along the wire. Since heat flow is proportional to dT/dx, dT/dx must be constant, and the temperature profile is linear. In other words, (1) since no heat is created or lost in the wire, heat-in = heat-out; (2) but heat flow ~ dT/dx; so (3) the change in the temperature *gradient* is zero:

$$\frac{d}{dx}\left(\frac{dT}{dx}\right) = 0 = \frac{d^2T}{dx^2}.$$

(At the edges of the wall, the 1D approximation breaks down, and the inevitable nonlinearity of the temperature profile in the x direction is offset by heat flow out the sides of the wire.)

Now consider a current carrying wire which generates heat all along its length from its resistance (Figure 3.1b). The heat that flows into the wire from the room is added to the heat generated in the wire, and the sum of the two flows into the refrigerator. The heat generated in a length dx of wire is

$$P_{gen} = I^2 \rho\, dx \qquad \textit{where} \quad \rho \equiv \text{resistance per unit length, and } I^2\rho = const.$$

In steady state, the net outflow of heat from a segment of wire must equal the heat generated in that segment. In an infinitesimal segment of length dx, we have heat-out = heat-in + heat-generated:

$$P_{out} = P_{in} + P_{gen} \quad \Rightarrow \quad \left.\frac{dT}{dx}\right|_a = \left.\frac{dT}{dx}\right|_{a+dx} + I^2\rho\, dx$$

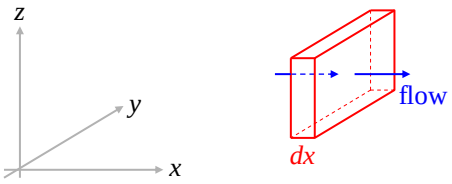
$$\left.\frac{dT}{dx}\right|_{a+dx} - \left.\frac{dT}{dx}\right|_a = -I^2\rho\, dx$$

$$\frac{d}{dx}\left(\frac{dT}{dx}\right)dx = -I^2\rho\, dx \quad \Rightarrow \quad \frac{d^2T}{dx^2} = -I^2\rho$$

The negative sign means that when the temperature gradient is positive (increasing to the right), the heat flow is negative (to the left), i.e. the heat flow is opposite the gradient. Many physical systems have a similar negative sign. Thus the 2nd derivative of the temperature is the negative of heat outflow (net inflow) from a segment, per unit length of the segment. Longer segments have more net outflow (generate more heat).

3D Rectangular Volume Element

Now consider a 3D bulk resistive material, carrying some current. The current generates heat in each volume element of material. Consider the heat flow in the x direction, with this volume element:



The temperature gradient normal to the y-z face drives a heat flow per unit area, in W/m^2. For a net flow to the right, the temperature gradient must be increasing in magnitude (becoming more negative) as we move to the right. The change in gradient is proportional to dx, and the heat outflow flow is proportional to the area, and the change in gradient:

$$P_{out} - P_{in} = -k\frac{d}{dx}\left(\frac{dT}{dx}\right)dx\, dy\, dz \quad \Rightarrow \quad \frac{P_{out} - P_{in}}{dx\, dy\, dz} = -k\frac{d^2T}{dx^2}.$$

Thus the net heat outflow per unit volume, due to the x contribution, goes like the 2nd derivative of T. Clearly, a similar argument applies to the y and z directions, each also contributing net heat outflow per unit

volume. Therefore, the *total* heat outflow per unit volume from all 3 directions is simply the sum of the heat flows in each direction:

$$\frac{P_{out} - P_{in}}{dx\,dy\,dz} = -k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right).$$

We see that in all cases, the

net outflow of flux per unit volume = change in (flux per unit area), per unit distance

We will use this fact to derive the Laplacian operator in spherical and cylindrical coordinates.

General Laplacian

We now generalize. For the Laplacian to mean anything, it must act on a scalar field whose gradient drives a flow of some physical thing.

Example 1: My favorite is $T(\mathbf{r})$ = temperature. Then $\nabla T(\mathbf{r})$ drives heat (energy) flow, heat per unit time, per unit area:

$$\frac{heat/t}{area} \equiv \mathbf{q} = -k\nabla T(\mathbf{r}) \qquad where \quad k \equiv thermal\ conductivity$$

$$\mathbf{q} \equiv heat\ \ flow\ vector$$

Then $$\frac{\partial T}{\partial r} \sim q_r = radial\ component\ of\ \ heat\ \ flow$$

Example 2: $T(\mathbf{r})$ = pressure of an incompressible viscous fluid (e.g. honey). Then $\nabla T(\mathbf{r})$ drives fluid mass (or volume) flow, mass per unit time, per unit area:

$$\frac{mass/t}{area} \equiv \mathbf{j} = -k\nabla T(\mathbf{r}) \qquad where \quad k \equiv some\ viscous\ \ friction\ coefficient$$

$$\mathbf{j} \equiv mass\ \ flow\ density\ vector$$

Then $$\frac{\partial T}{\partial r} \sim j_r = radial\ component\ of\ \ mass\ \ flow$$

Example 3: $T(\mathbf{r})$ = electric potential in a resistive material. Then $\nabla T(\mathbf{r})$ drives charge flow, charge per unit time, per unit area:

$$\frac{\text{charge}/t}{area} \equiv \mathbf{j} = -\sigma\nabla T(\mathbf{r}) \quad where \quad \sigma \equiv electrical\ conductivity$$

$$\mathbf{j} \equiv current\ density\ vector$$

Then $$\frac{\partial T}{\partial r} \sim j_r = radial\ component\ of\ \ current\ density\,.$$

Example 4: Here we abstract a little more, to add meaning to the common equations of electromagnetics. Let $T(\mathbf{r})$ = electric potential in a vacuum. Then $\nabla T(\mathbf{r})$ measures the energy per unit distance, per unit area, required to push a fixed charge density ρ through a surface, by a distance of dn, normal to the surface:

$$\frac{\text{energy/distance}}{area} \equiv \rho\nabla T(\mathbf{r})\ \ where \quad \rho \equiv \text{electric charge volume density}\,.$$

Then $\partial T/\partial r \sim$ net energy per unit radius, per unit area, to push charges of density ρ out the same distance through both surfaces.

In the first 3 examples, we use the word "flow" to mean the flow in time of some physical quantity, per unit area. In the last example, the "flow" is energy expenditure per unit distance, per unit area. The requirement of "per unit area" is essential, as we soon show.

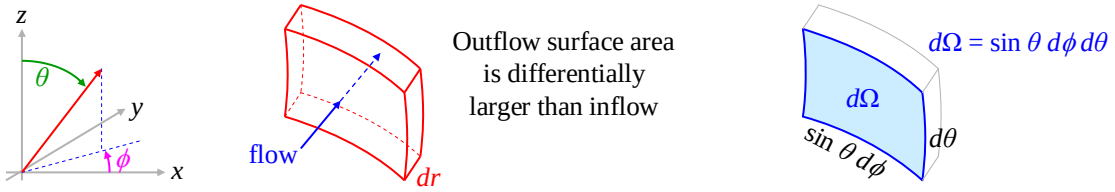
Laplacian In Spherical Coordinates

To understand the Laplacian operator terms in other coordinates, we need to take into account two effects:

1. The outflow surface area may be different than the inflow surface area
2. The derivatives with respect to angles (θ *or* ϕ) need to be converted to rate-of-change *per unit distance*.

We'll see how these two effects come into play as we develop the spherical terms of the Laplacian operator. The cylindrical terms are simplifications of the spherical terms.

Spherical radial contribution: We first consider the radial contribution to the spherical Laplacian operator, from this volume element:



The differential volume element has thickness *dr*, which can be made arbitrarily small compared to the lengths of the sides. The inner surface of the element has area $r^2\,d\Omega$. The outer surface has infinitesimally more area. Thus the radial contribution includes both the "surface-area" effect, but not the "converting-derivatives" effect.

The increased area of the outflow surface means that for the same flux-density (flow) on inner and outer surfaces, there would be a net outflow of flux, since flux = (flux-density)(area). Therefore, we must take the derivative of the flux itself, not the flux density, and then convert the result back to per-unit-volume. We do this in 3 steps:

$$\text{flux} = (\text{area})(\text{flux-density}) \sim \left(r^2 d\Omega\right)\left(\frac{\partial}{\partial r}\right)$$

$$\frac{d(\text{flux})}{dr} = \frac{\partial}{\partial r}\left(r^2 d\Omega\right)\left(\frac{\partial}{\partial r}\right)$$

$$\frac{\text{outflow}}{\text{volume}} = \frac{d(\text{flux})}{(\text{area})\,dr} = \frac{1}{r^2 d\Omega}\frac{\partial}{\partial r}\left(r^2 d\Omega\right)\left(\frac{\partial}{\partial r}\right) = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\right)\left(\frac{\partial}{\partial r}\right)$$

The constant $d\Omega$ factor from the area cancels when converting to flux, and back to flux-density. In other words, we can think of the fluxes as per-steradian.

We summarize the stages of the spherical radial Laplacian operator as follows:

$$\nabla^2_r\, T(\mathbf{r}) = \frac{1}{r^2}\frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r} T(\mathbf{r})$$

$$\frac{\partial}{\partial r} T = \text{radial flux per unit area}$$

$$r^2 \frac{\partial}{\partial r} T = \text{radial flux, per unit solid-angle} = \frac{(area)(\,flow\ per\ unit\ area\,)}{d\Omega}$$

$$\frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r} T = \text{change in radial flux per unit length, per unit solid-angle;} \qquad \text{positive is increasing flux}$$

$$\frac{1}{r^2}\frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r} T = \text{change in radial flux per unit length, per unit area}$$

$$= \text{net outflow of flux per unit volume}$$

$$\underbrace{\frac{1}{r^2}\ \underbrace{\frac{\partial}{\partial r} r^2\ \underbrace{\underbrace{\frac{\partial}{\partial r} T}_{\substack{\text{radial flow}\\ \text{per unit area}}}}_{\substack{\text{radial flux}\\ \text{per steradian}}}}_{\substack{\text{change in radial flux per}\\ \text{unit length per steradian}}}}_{\substack{\text{change in radial flux per}\\ \text{unit length, per unit area}}}$$

Following the steps in the example of heat flow, let $T(\mathbf{r})$ = temperature. Then

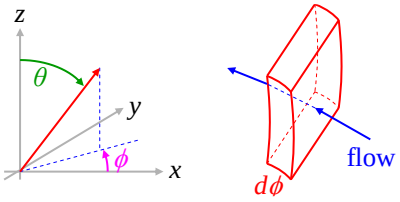
$$\frac{\partial}{\partial r} T = \text{radial heat flow per unit area, W/m}^2$$

$$r^2 \frac{\partial}{\partial r} T = \text{radial heat flux, W/solid-angle} = \frac{\text{Watts}}{\text{steradian}}$$

$$\frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r} T = \text{change in radial heat flux per unit length, per unit solid-angle}$$

$$\frac{1}{r^2}\frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r} T = \text{net outflow of heat flux per unit volume}$$

Spherical azimuthal contribution: The spherical ϕ contribution to the Laplacian has no area-change, but does require converting derivatives. Consider the volume element:



Outflow surface area is identical to inflow

The inflow and outflow surface areas are the same, and therefore area-change contributes nothing to the derivatives.

However, we must convert the derivatives with respect to ϕ into rates-of-change with respect to distance, because physically, the flow is driven by a derivative with respect to distance. In the spherical ϕ case, the effective radius for the arc-length along the flow is $r \sin\theta$, because we must project the position vector into the plane of rotation. Thus, $(\partial/\partial\phi)$ is the rate-of-change per $(r \sin\theta)$ meters. Therefore,

$$\text{rate-of-change-per-meter} = \frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}$$

Performing the two derivative conversions, we get

$$\nabla^2_\phi T(\mathbf{r}) = \frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}T(\mathbf{r})$$

$$\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}T = \text{azimuthal flux per unit area}$$

$$\frac{\partial}{\partial\phi}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}T = \text{change in (azimuthal flux per unit area) per radian}$$

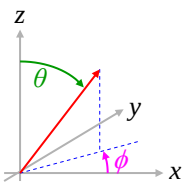
$$\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}T = \text{change in (azimuthal flux per unit area) per unit distance}$$

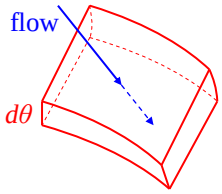
$$= \text{net azimuthal outflow of flux per unit volume}$$

$$\underbrace{\frac{1}{r\sin\theta}\underbrace{\frac{\partial}{\partial\phi}\underbrace{\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}T}_{\substack{\text{azimuthal flux}\\ \text{per unit area}}}}_{\substack{\text{change in (azimuthal flux}\\ \text{per unit area) per radian}}}}_{\substack{\text{change in (azimuthal flux per}\\ \text{unit area) per unit distance}}} = \frac{1}{r^2\sin^2\theta}\frac{\partial^2}{\partial\phi^2}T$$

Notice that the $r^2 \sin^2\theta$ in the denominator is not a physical area; it comes from two derivative conversions.

Spherical polar angle contribution:





Outflow surface area is differentially larger than inflow

The volume element is like a wedge of an orange: it gets wider (in the northern hemisphere) as θ increases. Therefore the outflow area is differentially larger than the inflow area (in the northern hemisphere). In particular, $area = (r\sin\theta)\,dr$, but we only need to keep the θ dependence, because the factors of r cancel, just like dΩ did in the spherical radial contribution. So we have

$$area \propto \sin\theta\,.$$

In addition, we must convert the $\partial/\partial\theta$ to a rate-of-change with distance. Thus the spherical polar angle contribution has *both* area-change and derivative-conversion.

Following the steps of converting to flux, taking the derivative, then converting back to flux-density, we get

$$\nabla^2{}_\theta\, T(\mathbf{r}) = \frac{1}{\sin\theta}\frac{1}{r}\frac{\partial}{\partial\theta}\sin\theta\frac{1}{r}\frac{\partial}{\partial\theta}T(\mathbf{r})$$

$$\frac{1}{r}\frac{\partial}{\partial\theta}T = \hat\theta\text{-flux per unit area}$$

$$\sin\theta\frac{1}{r}\frac{\partial}{\partial\theta}T = \hat\theta\text{-flux, per unit radius} = \frac{(area)(\,flux\ per\ unit\ area)}{dr}$$

$$\frac{\partial}{\partial\theta}\sin\theta\frac{1}{r}\frac{\partial}{\partial\theta}T = \text{change in } \left(\hat\theta\text{-flux per unit radius}\right), \text{ per radian}$$

$$\frac{1}{r}\frac{\partial}{\partial\theta}\sin\theta\frac{1}{r}\frac{\partial}{\partial\theta}T = \text{change in } \left(\hat\theta\text{-flux per unit radius}\right), \text{ per unit distance}$$

$$\frac{1}{\sin\theta}\frac{1}{r}\frac{\partial}{\partial\theta}\sin\theta\frac{1}{r}\frac{\partial}{\partial\theta}T = \text{change in } (\hat\theta\text{-flux per unit area}), \text{ per unit distance}$$

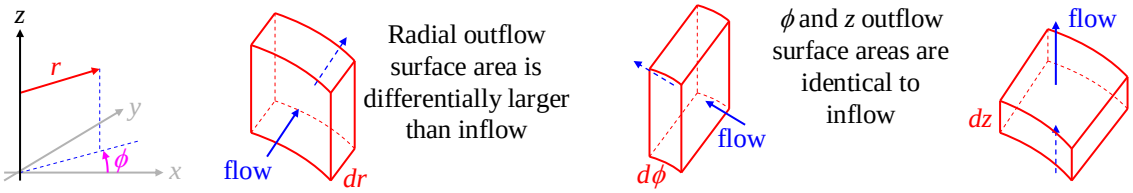
$$= \text{net outflow of flux per unit volume}$$

$$\underbrace{\frac{1}{\sin\theta}\underbrace{\frac{1}{r}\underbrace{\frac{\partial}{\partial\theta}\sin\theta\underbrace{\frac{1}{r}\underbrace{\frac{\partial}{\partial\theta}T}_{\hat\theta\text{-flux per unit area}}}_{\hat\theta\text{-flux, per unit radius}}}_{\text{change in } (\hat\theta\text{-flux per unit radius}), \text{ per radian}}}_{\text{change in } (\hat\theta\text{-flux per unit radius}), \text{ per unit distance}}}_{\text{change in } (\hat\theta\text{-flux per unit area}), \text{ per unit distance}} = \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\sin\theta\frac{\partial}{\partial\theta}T$$

Notice that the r^2 in the denominator is not a physical area; it comes from two derivative conversions.

Cylindrical Coordinates

The cylindrical terms are simplifications of the spherical terms.



Cylindrical radial contribution: The picture of the cylindrical radial contribution is essentially the same as the spherical, but the "height" of the slab is exactly constant. We still face the issues of varying inflow and outflow surface areas, and converting derivatives to rate of change per unit distance. The change in area is due only to the arc length $r\,d\phi$, with the z (height) fixed. Thus we write the radial result directly:

$$\nabla^2_r\, T(\mathbf{r}) = \frac{1}{r}\frac{\partial}{\partial r} r \frac{\partial}{\partial r} T(\mathbf{r}) \qquad \text{(Cylindrical Coordinates)}$$

$$\frac{\partial}{\partial r} T = \text{radial flow per unit area}$$

$$r\frac{\partial}{\partial r} T = \text{radial flux per unit angle} = \frac{(\text{flow per unit area})(\text{area})}{d\phi\, dz}$$

$$\frac{\partial}{\partial r} r \frac{\partial}{\partial r} T = \text{change in (radial flux per unit angle), per unit radius}$$

$$\frac{1}{r}\frac{\partial}{\partial r} r \frac{\partial}{\partial r} T = \text{change in (radial flux per unit area), per unit radius}$$

$$= \text{net outflow of flux per unit volume}$$

$$\underbrace{\frac{1}{r}\ \underbrace{\frac{\partial}{\partial r} r\ \underbrace{\underbrace{\frac{\partial}{\partial r} T}_{\substack{\text{radial flow}\\ \text{per unit area}}}}_{\substack{\text{radial flux}\\ \text{per radian}}}}_{\substack{\text{change in radial flux per}\\ \text{unit length per radian}}}}_{\substack{\text{change in (radial flux per}\\ \text{unit area), per unit radius}}}$$

Cylindrical azimuthal contribution: Like the spherical case, the inflow and outflow surfaces have identical areas. Therefore, the ϕ contribution is similar to the spherical case, except there is no sin θ factor; r contributes directly to the arc-length and rate-of-change per unit distance:

$$\nabla^2_\phi\, T(\mathbf{r}) = \frac{1}{r}\frac{\partial}{\partial \phi}\frac{1}{r}\frac{\partial}{\partial \phi} T(\mathbf{r})$$

$$\frac{1}{r}\frac{\partial}{\partial \phi} T = \text{azimuthal flux per unit area}$$

$$\frac{\partial}{\partial \phi}\frac{1}{r}\frac{\partial}{\partial \phi} T = \text{change in } \left(\text{azimuthal flux per unit area}\right) \text{ per radian}$$

$$\frac{1}{r}\frac{\partial}{\partial \phi}\frac{1}{r}\frac{\partial}{\partial \phi} T = \text{change in (azimuthal flux per unit area) per unit distance}$$

$$= \text{net azimuthal outflow of flux per unit volume}$$

$$\underbrace{\frac{1}{r}\frac{\partial}{\partial \phi}\ \underbrace{\underbrace{\frac{1}{r}\frac{\partial}{\partial \phi} T}_{\substack{\textit{azimuthal flow}\\ \textit{per unit area}}}}_{\substack{\textit{change in azimuthal}\\ \textit{flow per radian}}}}_{\substack{\text{change in (azimuthal flux per}\\ \text{unit area) per unit distance}}} = \frac{1}{r^2}\frac{\partial^2}{\partial \phi^2} T$$

Cylindrical z contribution: This is identical to the rectangular case: the inflow and outflow areas are the same, and the derivative is already per unit distance, ergo: (add cylindrical volume element picture??)

$$\nabla^2{}_z T(\mathbf{r}) = \frac{\partial}{\partial z}\frac{\partial}{\partial z}T(\mathbf{r})$$

$$\frac{\partial}{\partial z}T = \text{vertical flux per unit area}$$

$$\frac{\partial}{\partial z}\frac{\partial}{\partial z}T = \text{change in (vertical flux per unit area) per unit distance}$$

$$= \text{net outflow of flux per unit volume}$$

$$\underbrace{\frac{\partial}{\partial z}\underbrace{\frac{\partial}{\partial z}T}_{\text{vertical flux per unit area}}}_{\text{change in (vertical flux per unit area) per unit distance}} = \frac{\partial^2}{\partial z^2}T$$

Laplacian of a Vector Field

It gets worse: there's a vector form of ∇^2. If $\mathbf{E}(x, y, z)$ is a vector field, then in rectangular coordinates:

$$\nabla^2\mathbf{E} \equiv \nabla\bullet\nabla\mathbf{E} = \nabla^2 E_x\mathbf{i} + \nabla^2 E_y\mathbf{j} + \nabla^2 E_z\mathbf{k}\ .$$

This arises in E&M propagation, and not much in QM. However, the above equality is only true in rectangular coordinates [I have a ref for this, but lost it??]. This is the divergence of the gradient of a vector field, which is a vector. In oblique or non-normal coordinates, the gradient and divergence must be covariant, and include the Christoffel symbols.

Vector Dot Grad Vector

In electromagnetic propagation, and elsewhere, one encounters the "dot product" of a vector field with the gradient operator, acting on a vector field. What is this $\mathbf{v}\cdot\nabla$ operator? Here, $\mathbf{v}(\mathbf{r})$ is a given vector field. The simple view is that $\mathbf{v}(\mathbf{r})\cdot\nabla$ is just a notational shorthand for

$$\mathbf{v}(\mathbf{r})\cdot\nabla \equiv \left(v_x\frac{\partial}{\partial x} + v_y\frac{\partial}{\partial y} + v_z\frac{\partial}{\partial z}\right),$$

$$\textit{because}\ \ \mathbf{v}(\mathbf{r})\cdot\nabla = \left(v_x\hat{\mathbf{x}} + v_y\hat{\mathbf{y}} + v_z\hat{\mathbf{z}}\right)\cdot\left(\frac{\partial}{\partial x}\hat{\mathbf{x}} + \frac{\partial}{\partial y}\hat{\mathbf{y}} + \frac{\partial}{\partial z}\hat{\mathbf{z}}\right) = \left(v_x\frac{\partial}{\partial x} + v_y\frac{\partial}{\partial y} + v_z\frac{\partial}{\partial z}\right)$$

by the usual rules for a dot product in rectangular coordinates.

There is a deeper meaning, though, which is an important bridge to the topics of tensors and differential geometry.

> We can view the $\mathbf{v}\cdot\nabla$ operator as simply the dot product of the vector field $\mathbf{v}(r)$ with the gradient of a vector field.

You may think of the gradient operator as acting on a *scalar* field, to produce a vector field. But the gradient operator can also act on a vector field, to produce a tensor field. Here's how it works: You are probably familiar with derivatives of a vector field:

Let $\mathbf{A}(x, y, z)$ be a vector field. Then $\frac{\partial \mathbf{A}}{\partial x} = \left(\frac{\partial A_x}{\partial x} \hat{\mathbf{x}} + \frac{\partial A_y}{\partial x} \hat{\mathbf{y}} + \frac{\partial A_z}{\partial x} \hat{\mathbf{z}} \right)$ is a vector field.

Writing spatial vectors as column vectors, $A = \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix}$, *and* $\frac{\partial \mathbf{A}}{\partial x} = \begin{pmatrix} \frac{\partial A_x}{\partial x} \\ \frac{\partial A_y}{\partial x} \\ \frac{\partial A_z}{\partial x} \end{pmatrix}$

Similarly, $\frac{\partial \mathbf{A}}{\partial y}$ *and* $\frac{\partial \mathbf{A}}{\partial z}$ are also vector fields.

By the rule for total derivatives, for a small displacement (dx, dy, dz),

$$d\mathbf{A} \equiv \begin{pmatrix} dA_x \\ dA_y \\ dA_z \end{pmatrix} = \frac{\partial \mathbf{A}}{\partial x} dx + \frac{\partial \mathbf{A}}{\partial y} dy + \frac{\partial \mathbf{A}}{\partial z} dz = \begin{pmatrix} \frac{\partial A_x}{\partial x} & \frac{\partial A_x}{\partial y} & \frac{\partial A_x}{\partial z} \\ \frac{\partial A_y}{\partial x} & \frac{\partial A_y}{\partial y} & \frac{\partial A_y}{\partial z} \\ \frac{\partial A_z}{\partial x} & \frac{\partial A_z}{\partial y} & \frac{\partial A_z}{\partial z} \end{pmatrix} \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} \frac{\partial A_x}{\partial x} \\ \frac{\partial A_y}{\partial x} \\ \frac{\partial A_z}{\partial x} \end{pmatrix} dx + \begin{pmatrix} \frac{\partial A_x}{\partial y} \\ \frac{\partial A_y}{\partial y} \\ \frac{\partial A_z}{\partial y} \end{pmatrix} dy + \begin{pmatrix} \frac{\partial A_x}{\partial z} \\ \frac{\partial A_y}{\partial z} \\ \frac{\partial A_z}{\partial z} \end{pmatrix} dz \, .$$

This says that the vector $d\mathbf{A}$ is a linear combination of 3 column vectors $\partial\mathbf{A}/\partial x$, $\partial\mathbf{A}/\partial$y, and $\partial\mathbf{A}/\partial$z, weighted respectively by the displacements dx, dy, and dz. The 3 x 3 matrix above is the gradient of the vector field $\mathbf{A}(\mathbf{r})$. It is the natural extension of the gradient (of a scalar field) to a vector field. It is a rank-2 tensor, which means that given a vector (dx, dy, dz), it produces a vector ($d\mathbf{A}$) which is a linear combination of 3 (column) vectors ($\nabla\mathbf{A}$), each weighted by the components of the given vector (dx, dy, dz).

Note that $\nabla\mathbf{A}$ and $\nabla\cdot\mathbf{A}$ are very different: the former is a rank-2 tensor field, the latter is a scalar field.

This concept extends further to derivatives of rank-2 tensors, which are rank-3 tensors: 3 x 3 x 3 cubes of numbers, producing a linear combination of 3 x 3 arrays, weighted by the components of a given vector (dx, dy, dz). And so on.

Note that in other coordinates (e.g., cylindrical or spherical), $\nabla\mathbf{A}$ is *not* given by the derivative of its components with respect to the 3 coordinates. The components interact, because the basis vectors also change through space. That leads to the subject of differential geometry, discussed elsewhere in this document.

4 Green Functions

We follow [Jac p??] and [Bra] in using the term "Green function," rather than "Green's function." Though we agree with Jackson's logic, we do it mostly because it's easier to say and type.

Green functions are a big topic, with lots of subtopics. Many references describe only a subset, but use words that imply they are covering *all* of Green functions. If you are looking for a specific application of Green functions, such as electrostatics, you may want to skip right to that section, but the "big idea" applies to all Green functions.

Though Green functions are used to solve linear operator equations (such as differential equations), the concepts involved apply to other applications, such as the Born approximation, impulse response analysis, and quantum propagators.

The Big Idea

Green functions are a method of solving linear operator equations (such as inhomogeneous linear differential equations) of the form:

$$\mathcal{L}\{f(x)\} = \underbrace{s(x)}_{\text{source}} \qquad \textit{where} \quad \mathcal{L}\{\ \} \text{ is a linear operator .} \tag{4.1}$$

$s(x)$ is called the "source" function. We use Green functions when other methods are hard, or to make a useful approximation (the Born approximation). The big idea is to break up the source $s(x)$ into infinitesimal pieces (δ-functions), solve each piece separately, and add up the solutions. Since the $\mathcal{L}$ is linear, the sum of solutions is also a solution, and is the solution to the original problem.

Sometimes, the Green function itself can be given physical meaning, as in E&M where it is essentially Huygen's Principle, but with accurate phase information, or in Quantum Field Theory where it is the propagator of a quantized field. Green functions can generate particular (i.e. inhomogeneous) solutions, and solutions matching boundary conditions. They don't generate homogeneous solutions (i.e., where the right hand side is zero). We explore Green functions through the following steps:

1. Extremely brief review of the δ-function.
2. The tired, but inevitable, electromagnetic example.
3. Linear differential equations of one variable (1-dimensional), with sources.
4. Delta function expansions.
5. Green functions of two variables (but 1 dimension).
6. When you can collapse a Green function to one variable ("portable Green functions": translational invariance)
7. Dealing with boundary conditions: at least 5 (6??) kinds of BC
8. Green-like methods: the Born approximation

You will find *no* references to "Green's Theorem" or "self-adjoint" until we get to non-homogeneous boundary conditions, because until then, those topics are unnecessary and confusing. We will see that:

The biggest hurdle in understanding Green functions is the boundary conditions.

Some references derive Green functions from Green's Theorem, which derives from Gauss' Law. That is only a special case. In general, Green functions do *not* rely on Green's Theorem.

We return to this point later, after discussing general boundary conditions.

Dirac Delta Function

Recall that the Dirac δ-function is an "impulse," an infinitely narrow, tall spike function, *defined* as:

$$\delta(x) = 0, \quad for\ x \neq 0, \qquad \text{and} \qquad \int_{-a}^{a} \delta(x)\, dx = 1, \quad \forall a > 0 \text{ (the area under the } \delta\text{-function is 1)}.$$

(This also implies $\delta(0) \to \infty$, but we don't focus on that here.) The linearity of integration implies the delta function can be offset, and weighted, so that:

$$\int_{b-a}^{b+a} w\delta(x-b)\, dx = w \qquad \forall a > 0\,.$$

Since the δ-function is infinitely narrow, it can "pick out" a single value from a function:

$$\int_{b-a}^{b+a} \delta(x-b)\, f(x)\, dx = f(b) \qquad \forall a > 0\,. \tag{4.2}$$

This is called the "filtering property" of the δ-function. See *Quirky Quantum Concepts* for more on the delta function. The units of $\delta(\ )$ are $[x]^{-1}$.

The Tired, But Inevitable, Electromagnetic Example

You probably have seen Poisson's equation relating the electrostatic potential at a point to the charge distribution creating the potential (in gaussian units):

$$-\nabla^2\phi(\mathbf{r}) = 4\pi\rho(\mathbf{r}) \qquad where \quad \phi \equiv \text{electrostatic potential},\ \rho \equiv \text{charge density}\,. \tag{4.3}$$

We solved this by noting three things: (1a) electrostatic potential, ϕ, obeys "superposition:" the potential due to multiple charges is the sum of the potentials of the individual charges; (1b) the potential is proportional to the source charge; and (2) *if we take the potential at infinity to be zero*, the potential due to a point charge is:

$$\phi(\mathbf{r}) = q\frac{1}{|\mathbf{r}-\mathbf{r}'|} \qquad (\text{point charge at } \mathbf{r}')\,. \tag{4.4}$$

(We say much more about boundary conditions later.) The properties (1a) and (1b) above, taken together, define a **linear** relationship:

$$\text{Given:} \quad \rho_1(\mathbf{r}') \to \phi_1(\mathbf{r}), \quad \text{and} \quad \rho_2(\mathbf{r}') \to \phi_2(\mathbf{r}),$$
$$\text{then:} \quad a\rho_1(\mathbf{r}') + \rho_2(\mathbf{r}') \to \phi_{total}(\mathbf{r}) = a\phi_1(\mathbf{r}) + \phi_2(\mathbf{r})\,.$$

To solve (4.3), we break up the source charge distribution $\rho(\mathbf{r})$ into an infinite number of little point charges. The set of points is spread out over space, each of charge $\rho(\mathbf{r})\, d^3r$. The solution for ϕ is the sum of potentials from all the point charges, and the infinite sum is an integral, so we find ϕ as:

$$\phi(\mathbf{r}) = \lim_{d^3r' \to 0} \sum_{i=1}^{\#\,points} \rho(\mathbf{r}'_i)\, d^3r' \frac{1}{|\mathbf{r}-\mathbf{r}'_i|} = \int \rho(\mathbf{r}')\, d^3r' \frac{1}{|\mathbf{r}-\mathbf{r}'|}\,.$$

Note that the charge "distribution" for a point charge is a δ-function: infinite charge density, but finite total charge. Also, $\phi(\mathbf{r})$ for a point charge at $\mathbf{r}'$ is translationally invariant: it has the same form for all $\mathbf{r}'$. We will remove this restriction later.

All of this followed from simple mathematical properties of Eq (1) that have nothing to do with electromagnetics. All we used to solve for ϕ was that the left-hand side is a linear operator on ϕ (so superposition applies), and we have a known solution when the right-hand side is a delta function at $\mathbf{r}'$:

$$\underbrace{-\nabla^2}_{\text{linear operator}}\ \underbrace{\phi(\mathbf{r})}_{\text{unknown function}} = \underbrace{4\pi\rho(\mathbf{r})}_{\text{given "source" function}} \qquad \text{and} \qquad \underbrace{-\nabla^2}_{\text{linear operator}}\ \underbrace{\frac{1}{|\mathbf{r}-\mathbf{r}'|}}_{\text{known solution}} = \underbrace{\delta(\mathbf{r}-\mathbf{r}')}_{\text{given point "source" at } \mathbf{r}'}\,.$$

Since any given ρ can be written as a sum of weighted δ-functions, the solution for that given ρ is a sum of delta-function solutions. Now we generalize this electromagnetic example to arbitrary (for now, 1D) linear operator equations by letting $\mathbf{r} \to x$, $\phi \to f$, $-\nabla^2 \to \mathcal{L}$, $\rho \to s$, and call the known δ-function solution $G(x)$:

$$\underbrace{-\nabla^2}_{\mathcal{L}} \underbrace{\phi(\mathbf{r})}_{f(x)} = \underbrace{4\pi\rho(\mathbf{r})}_{s(x)} \qquad \text{and} \qquad \underbrace{-\nabla^2}_{\mathcal{L}} \underbrace{\frac{1}{|\mathbf{r}-\mathbf{r}'|}}_{G(\mathbf{r}-\mathbf{r}')} = \underbrace{\delta(\mathbf{r}-\mathbf{r}')}_{\substack{\text{given point} \\ \text{"source" at } \mathbf{r}'}} \qquad \to$$

$$\text{Given} \quad \mathcal{L}\{f(x)\} = s(x) \quad \text{and} \quad \mathcal{L}\{G(x-x')\} = \delta(x-x'),$$

$$\text{then} \quad f(x) = \int_{-\infty}^{\infty} dx'\, s(x')\, G(x-x').$$

This assumes, as above, that our linear operator, and any boundary conditions, are translationally invariant.

A Fresh, New Signal Processing Example

If the following example doesn't make sense to you, just skip it. Signal processing and control theory folk have long used a Green function-like concept, but with different words. A time-invariant linear system (TILS) produces an output which is a linear operation on its input:

$$o(t) = \mathcal{M}\{i(t)\} \qquad \textit{where} \quad \mathcal{M}\{\ \} \text{ is a linear operation taking input to output .}$$

In this case, we aren't given $\mathcal{M}\{\}$, and we don't solve for it (also it's on the right-, rather than the left-side of the equation). However, we *are* given a measurement (or computation) of the system's **impulse response**, called $h(t)$. If you poke the system with a very short spike (i.e., if you feed an impulse into the system, $i(t) = \delta(t)$), the system responds with $h(t)$:

$$h(t) = \mathcal{M}\{\delta(t)\} \qquad \textit{where} \quad h(t) \text{ is the system's impulse response .}$$

$h(t)$ acts like a Green function, giving the system response at time t to a delta function at $t = 0$. Note that $h(t)$ is spread out over time, and usually of (theoretically) infinite duration. $h(t)$ fully characterizes the system, because we can express any input function as a series of impulses (with the delta-function expansion below), and sum up all the responses. Therefore, we find the output for any input, $i(t)$, with:

$$o(t) = \int_{-\infty}^{\infty} i(t')\, h(t-t')\, dt'.$$

Caution: many references do not distinguish between a Green function $G(x)$ and an impulse response $h(x)$. The two are similar, but they differ because:

$$\mathcal{L}\{G(x)\} = \delta(x), \qquad \text{but} \qquad h(x) = \mathcal{M}\{\delta(x)\}.$$

The δ-function is in a different place for a Green function vs. an impulse response. For example, in electromagnetics, sources (charges and currents) are the stimulus that result in fields (**E** and **B**). Maxwell's equations have linear operators acting on the *result* (fields) to give you the stimulus. A TILS does the reverse: it produces a *result* which is a linear operation on its input (stimulus).

We can see a relationship between a Green function and an impulse response by taking $\mathcal{M}^{-1}$ (if it exists) of both sides of the second equation:

$$\mathcal{M}^{-1}\{h(x)\} = \delta(x).$$

Thus the impulse response for an operator $\mathcal{M}$ is the Green function for the operator $\mathcal{M}^{-1}$. In particular, quantum field theory calls the field "propagator" a Green function, but it is more directly thought-of as an impulse response.

Linear differential equations of one variable, with sources

We wish to solve for $f(x)$, given $s(x)$:

$$\mathcal{L}\{f(x)\} = s(x), \qquad \textit{where} \quad \mathcal{L}\{\ \} \text{ is a linear operator.}$$

$s(x)$ is called the "source," or forcing function.

E.g., $$\left(\frac{d^2}{dx^2} + \omega^2\right) f(x) \equiv \frac{d^2}{dx^2} f(x) + \omega^2 f(x) = s(x).$$

We ignore boundary conditions for now (to be dealt with later). The differential equations often have 3D space as their domain. Note that we are *not* differentiating $s(x)$, which will be important when we get to the delta function expansion of $s(x)$.

Green functions solve the above equation by first solving a related equation: if we can find a function (i.e., a "Green function") such that:

$$\mathcal{L}\{G(x)\} = \delta(x), \qquad \textit{where} \quad \delta(x) \text{ is the Dirac delta function,}$$

e.g., $$\left(\frac{d^2}{dx^2} + \omega^2\right) G(x) = \delta(x),$$

then we can use that Green function to solve our original equation. This might seem weird, because $\delta(0) \to \infty$, but it just means that Green functions often have discontinuities in them or their derivatives. For example, suppose $G(x)$ is a step function:

$$\left.\begin{array}{lll} G(x) & = 0, & x < 0 \\ & = 1, & x > 0 \end{array}\right\} \qquad \text{Then} \quad \frac{d}{dx} G(x) = \delta(x).$$

Now suppose our source isn't centered at the origin, i.e., $s(x) = \delta(x-a)$. If $\mathcal{L}\{\ \}$ is translation invariant [along with any boundary conditions], then $G(\ )$ can still solve the equation by translation:

$$\mathcal{L}\{f(x)\} = s(x) = \delta(x-a), \qquad \Rightarrow \qquad f(x) = G(x-a) \quad \text{is a solution.}$$

If $s(x)$ is a weighted sum of delta functions at different places, then because $\mathcal{L}\{\ \}$ is linear, the solution is immediate: we just add up the solutions from all the δ-functions:

$$\mathcal{L}\{f(x)\} = s(x) = \sum_i w_i \delta(x - x_i) \qquad \Rightarrow \qquad f(x) = \sum_i w_i G(x - x_i).$$

Usually the source $s(x)$ is continuous. Then we can break up $s(x)$ into infinitesimally small pieces (i.e., expand it as an infinite sum of delta functions, described in a moment), and sum the solutions for the pieces. The summation goes over to an integral, and a solution is:

$$\mathcal{L}\{f(x)\} = s(x) = \sum_{i=1}^{\infty} w_i \delta(x - x_i) \qquad \begin{array}{c} x_i \to x' \\ w_i \to s(x')dx' \\ \to \end{array}$$

$$\mathcal{L}\{f(x)\} = s(x) = \int_{\text{source}} s(x')\, dx'\, \delta(x - x') \qquad \text{and} \qquad f(x) = \int_{\text{source}} dx'\, s(x') G(x - x')$$

We can show directly that $f(x)$ is a solution of the original equation by plugging it in, and noting that $\mathcal{L}\{\ \}$ acts in the x domain, and "goes through" (i.e., commutes with) any operation in x':

$$\mathcal{L}\{f(x)\} = \mathcal{L}\left\{\int dx'\, s(x')G(x-x')\right\}$$

$$= \int dx'\, s(x')\mathcal{L}\{G(x-x')\} \qquad \text{moving } \mathcal{L}\{\ \} \text{ inside the integral}$$

$$= \int dx'\, s(x')\delta(x-x') = s(x) \qquad \delta(\ ) \text{ picks out the value of } s(x). \; QED.$$

We now digress for a moment to understand the δ-function expansion.

Delta Function Expansion

As in the EM example, it is frequently quite useful to expand a given function $s(x)$ as a sum of δ-functions:

$$s(x) \approx \sum_{i=1}^{N} w_i \delta(x - x_i), \qquad \textit{where} \quad w_i \text{ are the weights of the basis delta functions.}$$

[This same expansion is used to characterize the response of linear systems to input $i(t)$.]

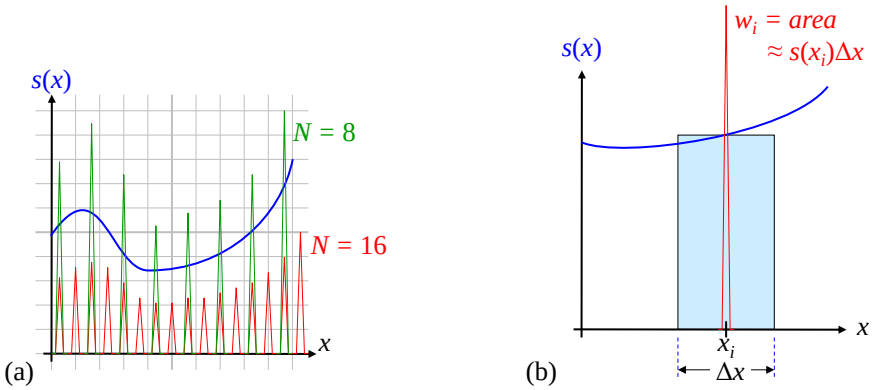


Figure 4.1 (a) Approximating a function with δ-functions. (b) The weight of each δ-function is such that its integral approximates the integral of the given function, $s(x)$, over the interval "covered" by the δ-function.

In Figure 4.1a, we approximate $s(x)$ first with $N = 8$ δ-functions (green), then with $N = 16$ δ-functions (red). As we double N, the weight of each δ-function is roughly cut in half, but there are twice as many of them. Hence, the integral of the δ-function approximation remains about the same. Of course, the approximation gets better as N increases. As usual, we let the number of δ-functions go to infinity: $N \to \infty$.

In Figure 4.1b, we show how to choose the weight of each δ-function: its weight is such that its integral approximates the integral of the given function, $s(x)$, over the interval "covered" by the δ-function. In the limit of $N \to \infty$, the approximation becomes arbitrarily good.

In what sense is the δ-function series an approximation to $s(x)$? Certainly, if we need the derivative $s'(x)$, the delta function expansion seems to be *terrible*. However, if we want the integral of $s(x)$, or any integral operator, such as an inner product or a convolution, then the delta function series is a good approximation. Examples:

$$\text{For} \quad \int s(x)\, dx, \qquad \text{or} \qquad \int f(x)s(x)\, dx, \qquad \text{or} \qquad \int f(x'-x)s(x)\, dx,$$

$$\text{then} \quad s(x) \approx \sum_{i=1}^{N} w_i \delta(x - x_i) \qquad \textit{where} \quad w_i = s(x_i)\Delta x.$$

As $N \to \infty$, we expand $s(x)$ in an infinite sum (an integral) of δ-functions:

$$s(x) = \sum_i w_i \delta(x - x_i) \qquad \begin{matrix} x_i \to x' \\ \Delta x \to dx' \\ w_i \to s(x')dx' \end{matrix} \qquad \to \qquad s(x) = \int dx'\, s(x')\delta(x - x') \,,$$

which if you think about it, follows directly from the definition of $\delta(x)$, per (4.2).

[Aside: Delta functions are a continuous set of orthonormal basis functions, much like sinusoids from quantum mechanics and Fourier transforms. They satisfy all the usual orthonormal conditions for a continuous basis, i.e. they are **orthogonal** and **normalized**:

$$\int_{-\infty}^{\infty} dx\, \delta(x-a)\delta(x-b) = \delta(a-b) \;\; .]$$

Note that in the final solution of the prior section, we integrate $s(x)$ times other stuff:

$$f(x) = \int dx'\, s(x')G(x - x') \,,$$

and integrating over $s(x)$ is what makes the δ-function expansion of $s(x)$ valid.

[Aside: It turns out that even systems that differentiate $s(x)$ can use the δ-function expansion, but we need not bother with that here.]

Boundary Conditions on Green Functions

Most problems require boundary conditions on the solution to an equation.

Introduction to Boundary Conditions

We now impose a simple boundary condition on an equation we seek to solve. Consider a 2D problem in the plane:

$$\mathcal{L}\{f(x,y)\} = s(x,y) \qquad \text{inside the boundary;}$$
$$f(\text{boundary}) = 0, \qquad \text{where the boundary is given.}$$

We define the vectors $\mathbf{r} \equiv (x, y)$ and $\mathbf{r}' = (x', y')$, and recall that:

$$\delta^2(\mathbf{r}) \equiv \delta(x)\delta(y), \qquad \text{so that} \qquad \delta^2(\mathbf{r} - \mathbf{r}') = \delta(x - x')\delta(y - y') \,.$$

The boundary condition removes the translation invariance of the problem (Figure 4.2). The delta function response of $\mathcal{L}\{G(\mathbf{r})\}$ translates, but the boundary condition does *not*. I.e., a solution of:

$$\mathcal{L}\{G(\mathbf{r})\} = \delta(\mathbf{r}), \quad \text{and} \quad G(\text{boundary}) = 0 \quad \Rightarrow \quad \mathcal{L}\{G(\mathbf{r} - \mathbf{r}')\} = \delta(\mathbf{r} - \mathbf{r}')$$
$$\text{BUT does NOT} \quad \Rightarrow \quad G(\textit{boundary} - \mathbf{r}') = 0 \,.$$

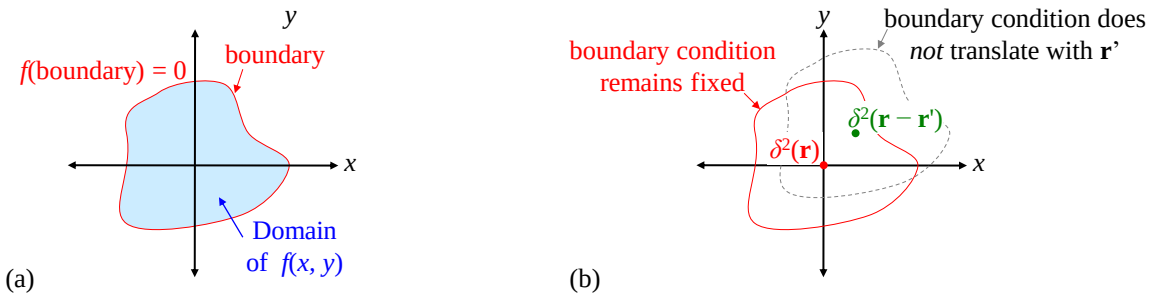


Figure 4.2 (a) The domain of interest (blue), and its boundary (red). (b) A solution meeting the BC for the source at (0, 0) does *not* translate to another point **r**' and still meet the BC.

With boundary conditions, for each source point r', we need a different Green function!

The Green function for a source point **r'**, call it $G_{\mathbf{r}'}(\mathbf{r})$, must satisfy *both*:

$$\mathcal{L}\{G_{\mathbf{r}'}(\mathbf{r})\} = \delta(\mathbf{r}-\mathbf{r}') \qquad \text{and} \qquad G_{\mathbf{r}'}(\text{boundary}) = 0\ .$$

We can think of this as a Green function of two arguments, **r** and **r'**, but really, **r** is the argument, and **r'** is a parameter. In other words, we have a *family* of Green functions, $G_{\mathbf{r}'}(\mathbf{r})$, each labeled by the location of the source point, **r'**.

Note that finding 1D Green functions is an important prerequisite for 3D Green functions, because a 3D problem sometimes separates into a 2D and a 1D problem. We give such an example in the section on 3D Laplacian operator boundary conditions.

One Dimensional Boundary Conditions

Example: Returning to a 1D example in *r*: Find the Green function for the equation:

$$\frac{d^2}{dr^2} f(r) = s(r), \quad \text{on the interval } [0,1], \text{ subject to BC: } \quad f(0) = f(1) = 0.$$

Solution: The Green function equation replaces the source $s(r)$ with $\delta(r - r')$:

$$\frac{d^2}{dr^2} G_{r'}(r) = \delta(r-r')\ .$$

Note that $G_{r'}(r)$ satisfies the *homogeneous* equation on either side of *r'*:

$$\frac{d^2}{dr^2} G_{r'}(r \neq r') = 0\ .$$

The full Green function simply matches two homogeneous solutions, one to the left of *r'*, and another to the right of *r'*, such that the discontinuity at *r'* creates the required δ-function there. First we find the homogeneous solutions $h(r)$ (*not* an impulse response):

$$\frac{d^2}{dr^2} h(r) = 0 \qquad \text{Integrate both sides:}$$

$$\frac{d}{dr} h(r) = C \qquad \textit{where} \quad \text{C is an integration constant. Integrate again:} \qquad (4.5)$$

$$h(r) = Cr + D \qquad \textit{where} \quad C, D \text{ are arbitrary constants}.$$

There are now 2 cases: (left) $r < r'$, and (right) $r > r'$. Each solution requires its own set of integration constants.

Left case: $r < r' \quad \Rightarrow \quad G_{r'}(r) = Cr + D$

Only the left boundary condition applies to $r < r'$: $\quad G_{r'}(0) = 0 \quad \Rightarrow \quad D = 0$

Right case: $r > r' \quad \Rightarrow \quad G_{r'}(r) = Er + F$

Only the right boundary condition applies to $r > r'$: $\quad G_{r'}(1) = 0 \quad \Rightarrow \quad E + F = 0,\ F = -E\ .$

So far, we have:

Left case: $G(r < r') = Cr$ Right case: $G(r > r') = Er - E\ .$

The integration constants *C* and *E* are as-yet unknown. Now we must match the two solutions at *r* = *r'*, and introduce a delta function there. The δ-function must come from the highest derivative in $\mathcal{L}\{\ \}$, in this case the 2nd derivative, because if *dG/dr* had a delta function, then the 2nd derivative d^2G/dr^2 would have the derivative of a δ-function, which cannot be canceled by any other term in $\mathcal{L}\{\ \}$. Since the derivative of a step

(discontinuity) is a δ-function, dG/dr must have a step, so that d^2G/dr^2 has a δ-function. And finally, if dG/dr has a step, then $G(r)$ has a cusp (aka "kink" or sharp point).

We can find $G(r)$ to satisfy all this by matching $G(r)$ and dG/dr of the left and right Green functions, at the point where they meet, $r = r'$:

$$\text{Left:} \quad \frac{d}{dr}G_{r'}(r<r') = C \qquad \text{Right:} \quad \frac{d}{dr}G_{r'}(r>r') = E\ .$$

There must be a unit step in the derivative across $r = r'$:

$$\left.\frac{\partial G}{\partial r}\right|_{r'_-} + 1 = \left.\frac{\partial G}{\partial r}\right|_{r'_+} \qquad \Rightarrow \qquad C+1 = E\ . \tag{4.6}$$

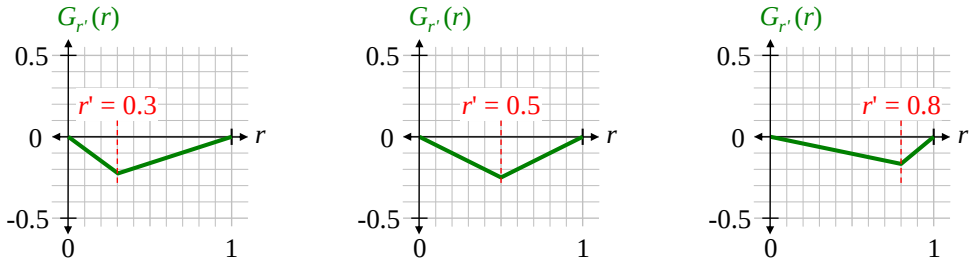
So we eliminate E in favor of C. Also, $G(r)$ must be continuous (or else dG/dr would have a δ-function), which means:

$$G_{r'}(r=r'_-) = G_{r'}(r=r'_+) \qquad \Rightarrow \qquad Cr' = (C+1)r' - C - 1, \qquad C = r' - 1\,,$$

yielding the final Green function for the given differential equation and boundary conditions:

$$G_{r'}(r<r') = (r'-1)r, \qquad G_{r'}(r>r') = r'r - r' = r'(r-1)\,.$$

Here's a plot of these Green functions for different values of r':



Normalization is important, because the δ-function in $\mathcal{L}\{G(r)\} = \delta(r)$ must have unit magnitude.

To find the solution $f(r)$, we need to integrate over r'; therefore, it is convenient to write the Green function as a true function of two variables:

$$G(r\,;r') \equiv G_{r'}(r) \ \Rightarrow \qquad \mathcal{L}\{G(r\,;r')\} = \delta(r-r'), \qquad \text{and} \qquad G(\text{boundary}\,;r') = 0\,,$$

where the ";" between r and r' emphasizes that $G(r\,;r')$ is a function of r, parameterized by r'. I.e., we can still think of $G(r;r')$ as a family of functions of r, where each family member is labeled by r', and each family member satisfies the homogeneous boundary condition.

> It is important here that the boundary condition is $G = 0$, so that any sum of Green functions still satisfies the boundary condition.

Finally, the particular solution to the original equation, which now satisfies the homogeneous boundary conditions, is:

$$f(r) = \int_0^1 dr'\, s(r')G(r\,;r') = \int_0^r dr'\, s(r') \underbrace{r'(r-1)}_{G(r;r'),\, r>r'} + \int_r^1 dr'\, s(r') \underbrace{(r'-1)r}_{G(r;r'),\, r<r'}.$$

which satisfies $f(\text{boundary}) = 0$

Summary: To solve $\mathcal{L}\{G_{r'}(r)\} = \delta(r-r')$ in one dimension:

- We break $G(r)$ into left- and right- sides of r'. Each side satisfies the homogeneous equation, $\mathcal{L}\{G_{r'}(r)\} = 0$, with arbitrary integration constants.
- We establish a first matching condition on $G(r)$, which is usually that it must be continuous at r'.
- We establish another matching condition to achieve the δ-function at r'. This establishes a set of simultaneous equations for the integration constants in the homogeneous solutions.
- We solve for the constants, yielding the left-of-r' and right-of-r' pieces of the complete Green function, $G(r; r')$.

Aside: It is amusing to notice that we use solutions to the *homogeneous* equation to construct the Green function. We then use the Green function to construct the *particular* solution to the given (inhomogeneous) equation. So we are ultimately constructing a particular solution from a homogeneous solution. That's not like anything we learned in undergraduate differential equations.

When Can You Collapse a Green Function to One Variable?

"Portable" Green Functions: When we first introduced the Green function, we ignored boundary conditions, and our Green function was a function of one variable, r. If our source wasn't at the origin, we just shifted our Green function, and it was a function of just $(r - r')$. Then we saw that with (certain) boundary conditions, shifting doesn't work, and the Green function is a function of two variables, r and r'. In general, then, under what conditions can we write a Green function in the simpler form, as a function of just $(r - r')$?

When both the linear operator and the boundary conditions are translation-invariant, the Green function is also translation-invariant.

We can say it's "portable."

This is fairly common: differential operators are translation-invariant (i.e., they do not explicitly depend on position), and BCs at infinity are translation-invariant. For example, in E&M it is common to have equations such as:

$$-\nabla^2\phi(\mathbf{r}) = \rho(\mathbf{r}), \quad \text{with boundary condition} \quad \phi(\infty) = 0 .$$

Because both the operator $-\nabla^2$ and the boundary conditions are translation invariant, we don't need to introduce r' explicitly as a parameter in $G(r)$. As we did in (4.4) when introducing Green functions, we can take the origin as the location of the delta function to find $G(r)$, and use translation invariance to "move around" the delta function:

$$G(r\,;r') \equiv G_{r'}(r) = G(r - r') \qquad \text{and} \qquad \mathcal{L}\{G(r - r')\} = \delta(r - r')$$
$$\text{with BC:} \qquad G(\infty) = 0$$

Non-homogeneous Boundary Conditions

So far, we've dealt with homogeneous boundary conditions by requiring $G_{r'}(r) \equiv G(r\,;r')$ to be zero on the boundary (which may be at infinity). But there are different kinds of boundary conditions, and different ways of dealing with each kind.

[Note that in general, constraint conditions don't have to be specified at the *boundary* of anything. They are really just "constraints" or "conditions." For example, one constraint is often that the solution be a "normalized" function, which is not a statement about any boundaries. But in most physical problems, at least one condition *does* occur at a boundary, so we defer to common usage, and limit ourselves here to boundary conditions.]

Boundary Conditions Specifying Only Values of the Solution

In one common case, we are given a general (inhomogeneous) boundary condition, $m(r)$ along the boundary of the region of interest. Our problem is now to find the complete solution $c(r)$ such that

$$\mathcal{L}\{c(r)\} = s(r), \quad \text{and} \quad c(\text{boundary}) = m(\text{boundary}) .$$

One approach to find $c(r)$ is from elementary differential equations: we find a particular solution $f(x)$ to the given equation, that doesn't necessarily meet the boundary conditions. Then we add a linear combination of *homogeneous* solutions to achieve the boundary conditions, while preserving the solution of the non-homogeneous equation. There are 3 steps:

(1) First solve for $f(r)$, as above, such that:

$$\mathcal{L}\{f(r)\} = s(r), \quad \text{and} \quad f(\text{boundary}) = 0,$$

using a Green function satisfying:

$$\mathcal{L}\{G(r;r')\} = \delta(r-r') \quad \text{and} \quad G(\text{boundary};r') = 0 .$$

(2) Find homogeneous solutions $h_i(r)$ which are non-zero on the boundary, using ordinary methods (see any differential equations text):

$$\mathcal{L}\{h_i(r)\} = 0, \quad \text{and} \quad h_i(\text{boundary}) \neq 0 .$$

Recall that in finding the Green function, we already had to find homogeneous solutions, since every Green function *is* a homogeneous solution everywhere except at the δ-function position, r'.

(3) Finally, we add a linear combination of homogeneous solutions to the particular solution to yield a complete solution which satisfies both the differential equation and the boundary conditions. Thus we find coefficients A_j such that:

$$A_1h_1(r) + A_2h_2(r) + ... = m(r), \quad \text{and} \quad \mathcal{L}\{A_1h_1(r) + A_2h_2(r) + ...\} = 0 \quad \text{by superposition} .$$

Then our solution is $c(r)$:

$$c(r) = f(r) + A_1h_1(r) + A_2h_2(r) + ..., \quad \text{because,}$$

$$\mathcal{L}\{c(r)\} = \mathcal{L}\{f(r) + A_1h_1(r) + A_2h_2(r) + ...\}$$

$$= \mathcal{L}\{f(r)\} = s(r) \quad \text{and} \quad c(\text{boundary}) = m(\text{boundary})$$

Continuing Example: In our 1D example above, we have:

$$\mathcal{L}\{\ \} = \frac{\partial^2}{\partial r^2} \quad \text{and} \quad G_{r'}(r < r') = (r'-1)r, \quad G_{r'}(r > r') = r'(r-1),$$

$$\text{satisfying BC:} \quad G_{r'}(0) = G_{r'}(1) = 0 \quad \Rightarrow \quad f(0) = f(1) = 0, \ \forall s(r) .$$

We now add boundary conditions to the original problem: $c(0) = 2$, and $c(1) = 3$, in addition to the original problem. Our linearly independent homogeneous solutions are, from (4.5):

$$h_1(r) = A_1r \qquad h_0(r) = A_0 \quad \text{(a constant)} .$$

To satisfy the BC, we need

$$h_1(0) + h_0(0) = 2 \ \Rightarrow \quad A_0 = 2$$

$$h_1(1) + h_0(1) = 3 \ \Rightarrow \quad A_1 = 1$$

Thus our complete solution, satisfying the given BCs, is:

$$c(r) = \left[\int_0^1 dr'\, s(r')G(r;r')\right] + r + 2 . \tag{4.7}$$

Boundary Conditions Specifying a Value and a Derivative

Another common kind of boundary conditions specifies a value and a derivative for our complete solution. For example, in 1D:

$$c(0) = 1 \qquad \text{and} \qquad c'(0) = 5\,.$$

Recall that our previous Green function (4.7) is not required to have any particular derivative at zero. When we find a particular solution, $f(x)$, we have no idea what it's derivative at zero, $f'(0)$, will be. And in particular, different source functions, $s(r)$, will produce different $f(r)$, with different values of $f'(0)$. This is bad for our new BCs. In the previous case of BC, $f(r)$ was zero at the boundaries for *any* $s(r)$. What we need with our new BC is $f(0) = 0$ and $f'(0) = 0$ for any $s(r)$. We can easily achieve this by using a *different Green function*! We subjected our first Green function to the boundary conditions $G(0; r') = 0$ and $G(1; r') = 0$ specifically to give the same BC to $f(r)$, so we could add our homogeneous solutions *independently* of $s(r)$. Therefore, in our example $\mathcal{L}\{\ \} = \dfrac{d^2}{dr^2}$, we now choose our Green function BC to be:

$$G(0; r') = 0 \qquad \text{and} \qquad \frac{d}{dr}G(0; r') = 0, \quad \text{with} \quad \mathcal{L}\{G(r; r')\} \equiv \frac{d^2}{dr^2}G(r; r') = \delta(r - r')\,.$$

We can see by inspection that this leads to a new Green function (Figure 4.3):

$$G(r; r') = 0 \qquad r < r', \quad \text{and} \quad G(r; r') = r - r' \qquad r > r'\,.$$

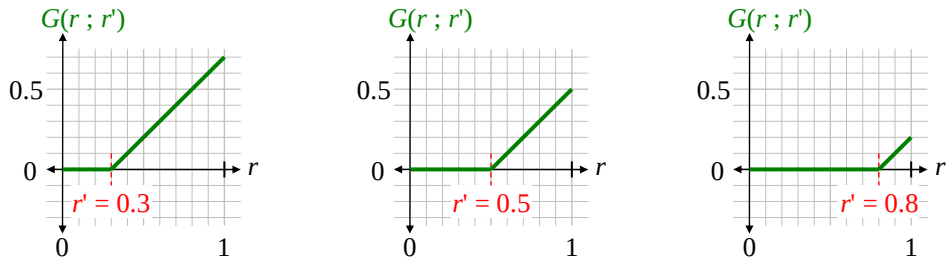


Figure 4.3 Green functions for 3 different values of r'.

The 2nd derivative of $G(r; r')$ is everywhere 0, and the first derivative changes from 0 to 1 at r'. Therefore, our new particular solution $f(r)$ also satisfies:

$$f(r) = \int_0^1 dr'\ s(r')G(r; r') \qquad \text{and} \qquad f(0) = 0, \quad f'(0) = 0, \quad \forall s(r)\,.$$

We complete the solution using our homogeneous solutions to meet the BC:

$$h_1(r) = A_1 r \qquad h_0(r) = A_0 \quad \text{(a constant)}$$

$$h_1(0) + h_0(0) = 1 \quad \Rightarrow \qquad A_0 = 1$$

$$h_1'(0) + h_0'(0) = 5 \Rightarrow \qquad A_1 = 5. \qquad \text{Then:}$$

$$c(r) = \left[\int_0^1 dr'\, s(r')G(r; r')\right] + 5r + 1$$

In general, the Green function depends not only on the particular operator, but also on the *kind* of boundary conditions specified.

The Green function does *not* depend on the *values* of the given BCs.

Boundary Conditions Specifying Ratios of Derivatives and Values

Another kind of boundary conditions specifies a ratio of the solution to its derivative, or equivalently, specifies a linear combination of the solution and its derivative be zero. This is equivalent to a homogeneous boundary condition:

$$\frac{c'(0)}{c(0)} = \alpha \qquad \text{or equivalently (if } c(0) \neq 0) \qquad c'(0) - \alpha c(0) = 0\,.$$

This BC arises, for example, in some quantum mechanics problems where the normalization of the wave-function is not yet known; the ratio cancels any normalization factor, so the solution can proceed without knowing the ultimate normalization. Note that this is only a single BC. If our differential operator is 2nd order, there is one more degree of freedom that can be used to achieve some other condition, such as normalization. (This BC is sometimes given as $\beta c'(0) - \alpha c(0) = 0$, but this simply multiplies both sides by a constant, and fundamentally changes nothing.)

Importantly, this condition is homogeneous: a linear combination of functions which satisfy the BC also satisfies the BC. This is most easily seen from the form given above, right:

If $d'(0) - \alpha d(0) = 0,$ and $e'(0) - \alpha e(0) = 0,$

then $c(r) = Ad(r) + Be(r)$ satisfies $c'(0) - \alpha c(0) = 0$

because $c'(0) - \alpha c(0) = A\big(d'(0) - \alpha d(0)\big) + B\big(e'(0) - \alpha e(0)\big)$

Therefore, if we choose a Green function which satisfies the given homogeneous BC, our particular solution $f(r)$ will also satisfy the BC. There is no need to add any homogeneous solutions.

Continuing Example: In our 1D example above, with $\mathcal{L} = d^2/dr^2$, we now specify the BC:

$$\frac{c'(0)}{c(0)} = 2 \qquad \text{or} \qquad c'(0) - 2c(0) = 0\,.$$

Green functions for this operator are always two connected line segments (because their 2nd derivatives are zero), so we have:

$r < r'$: $G(r;r') = Cr + D,$ $D \neq 0$ so that $c(0) \neq 0;$

$r > r'$: $G(r;r') = Er + F$

BC at 0: $C - 2D = 0$

With this BC, we have an unused degree of freedom, so we choose $D = 1$, implying $C = 2$. We must find E and F so that $G(r; r')$ is continuous, and $G'(r; r')$ has a unit step at r'. The latter condition requires that $E = C + 1 = 3$, and then continuity requires:

$Cr' + D = Er' + F \;\Rightarrow\; 2r' + 1 = 3r' + F, \quad F = -r' + 1.$ So:

$r < r'$: $G(r;r') = 2r + 1$ and $r > r'$: $G(r;r') = 3r - r' + 1$

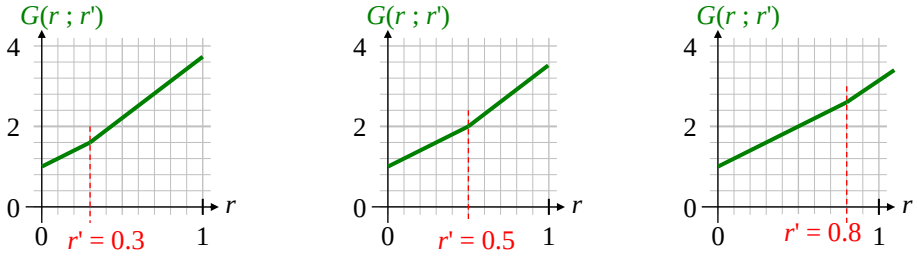


Figure 4.4 1D Green functions; the slope changes of +1 occur at r' (dotted red lines), but are subtle on this scale.

Then our complete solution is just:

$$c(r) = f(r) = \int_0^1 dr'\, s(r')G(r;r') .$$

Boundary Conditions Specifying Only Derivatives (Neumann BC)

Another common kind of BC specifies derivatives at points of the solution. For example, we might have:

$$c'(0) = 0 \qquad \text{and} \qquad c'(1) = 1 .$$

Then, analogous to the BC specifying two values for $c(\,)$, we find a Green function which has *zeros* for its derivatives at 0 and 1:

$$\frac{d}{dr}G(r=0;r') = 0 \qquad \text{and} \qquad \frac{d}{dr}G(r=1;r') = 0 .$$

Then the sum (or integral) of any number of such Green functions also satisfies the *zero* BCs:

$$f(r) = \int_0^1 dr'\, s(r')G(r;r') \qquad \text{satisfies} \qquad f'(0) = 0 \quad \text{and} \quad f'(1) = 0 .$$

We can now form the complete solution, by adding *homogeneous* solutions that satisfy the *given* BC:

$$c(r) = f(r) + A_1h_1'(r) + A_2h_2'(r) \qquad \textit{where} \quad A_1h_1'(0) + A_2h_2'(0) = 0$$
$$\textit{and} \quad A_1h_1'(1) + A_2h_2'(1) = 1$$

Example: We cannot use our previous example where $\mathcal{L}\{\ \} = d^2/dr^2$, because there is no solution to:

$$\frac{d^2}{dr^2}G(r;r') = \delta(r-r') \qquad \text{with} \qquad \frac{d}{dr}G(r=0;r') = \frac{d}{dr}G(r=1;r') = 0 .$$

This is because the homogenous solutions are straight line segments; therefore, any solution with a zero derivative at any point must be a flat line. So we must choose another operator as our example: TBS.

2D?? and 3D Green Functions

Green Functions Don't Separate

In previous sections, we described 1D Green functions, which satisfy:

$$\mathcal{L}\{G(x;x')\} = \delta(x-x') .$$

(We must change notation slightly from earlier, since in higher dimensions, "r" now has the conventional meaning: distance from the origin.) A 3D Green function satisfies:

$$\mathcal{L}\{G(\mathbf{r};\mathbf{r}')\} = \delta^3(\mathbf{r}-\mathbf{r}') \qquad (\text{coordinate free}) .$$

Note that δ^3 is a (coordinate-free) spherically symmetric function, with no preferred direction. We can *choose* to write it as a product of three coordinate functions. For example:

$$\mathcal{L}\{G(x,y,z;x',y',z')\} = \delta(x-x')\delta(y-y')\delta(z-z') \qquad (\text{rectangular coordinates}) .$$

To generalize Green functions to 3D in rectangular coordinates, you might guess that we could multiply three separate 1D Green functions together. For example, if $\mathcal{L}$ separates into x, y, and z parts, does the following hold?

Let $\mathcal{L}_x\{X(x;x')\} = \delta(x-x')$, and similar for $\mathcal{L}_y\{Y(y;y')\}$ and $\mathcal{L}_z\{Z(z;z')\}$.

Does $G(x,y,z;x',y',z') \stackrel{?}{=} X(x;x')Y(y;y')Z(z;z')$? I.e.,

$$\left(\mathcal{L}_x + \mathcal{L}_y + \mathcal{L}_z\right)\{X(x;x')Y(y;y')Z(z;z')\} \stackrel{?}{=} \delta^3(\mathbf{r}-\mathbf{r}')?$$

We now show that does *not* work. As a concrete counter-example, consider the Laplacian operator, ∇^2. In 1D, it is simply $\partial^2/\partial x^2$. Applying our guess to 3D, we would have:

$$\frac{\partial^2}{\partial x^2}X(x;x') = \delta(x-x'), \quad \text{and similar for} \quad Y(y;y') \text{ and } Z(z;z').$$

$$\nabla^2(XYZ) = \left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)(XYZ) = \delta(x-x')YZ + \delta(y-y')XZ + \delta(z-z')XY$$

$$\neq \delta(x-x')\delta(y-y')\delta(z-z').$$

Green functions do *not* separate the way solutions to Laplace's equation do.

Let us explore some properties of an actual 3D Green function. A well-known 3D Green function for the Laplacian, with BC of zero at infinity, is:

$$G(\mathbf{r};\mathbf{r}') = -\frac{1}{4\pi|\mathbf{r}-\mathbf{r}'|}.$$

For simplicity, we fix $\mathbf{r}' = \mathbf{0}_\mathbf{v}$ and drop the prefactor. For insight, we write it in rectangular coordinates:

$$G(\mathbf{r};\mathbf{0}_\mathbf{v}) \propto \frac{1}{r} = \frac{1}{\sqrt{x^2+y^2+z^2}}.$$

This is spherically symmetric, as required by the spherical symmetry of ∇^2 and the BCs, but has no other obvious structure. It does not seem to factor into $X(x)Y(y)Z(z)$. Nonetheless, we have:

$$\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\left(\frac{1}{\sqrt{x^2+y^2+z^2}}\right) = \frac{-1}{4\pi}\delta^3(r).$$

By symmetry, the three directions each contribute the same amount to the sum, which is 1/3 of the total, so:

$$\frac{\partial^2}{\partial x^2}(\bullet) = \frac{\partial^2}{\partial y^2}(\bullet) = \frac{\partial^2}{\partial z^2}(\bullet) = \frac{-1}{12\pi}\delta^3(r).$$

This means the 2nd derivative in a *single* direction is immediately a 3rd-order delta function; this $\delta^3(\,)$ does *not* result from the product of one $\delta(\,)$ in each direction.

3D Green functions are hard to understand. We give some examples in the following sections.

Green Units

Coordinates have units, operators have units, Green functions have units, and delta functions have units. As always, we can use dimensional analysis to sanity-check results, which we do later. As a 1D example:

$$[x] = L \text{ (length)}, \quad \left[\frac{\partial^2}{\partial x^2}\right] = L^{-2}, \qquad [\delta(\,)] = L^{-1}:$$

$$\frac{\partial^2}{\partial x^2} G = \delta \qquad \Rightarrow \qquad L^{-2}[G] = L^{-1}, \quad \text{and} \quad [G] = L\,.$$

If x is in meters, then so is G.

A 3D units example:

$$[x] = L \text{ (length)}, \quad \left[\nabla^2\right] = L^{-2}, \qquad [\delta(\,)] = L^{-1}:$$

$$\nabla^2 G = \delta^3 \qquad \Rightarrow \qquad L^{-2}[G] = L^{-3}, \quad \text{and} \quad [G] = L^{-1}\,.$$

If the coordinates are in meters, then G is in inverse meters.

Special Case: Laplacian Operator with 3D Boundary Conditions

In electrostatics, one often uses Green functions with the Laplacian operator, $\mathcal{L} = \nabla^2$, and boundary conditions, to find the electrostatic potential $\Phi(\mathbf{r})$. The Laplacian operator allows a "trick" (see glossary) for common boundary conditions, that gives a solution in terms of integrals. This section assumes you are thoroughly familiar with solving Laplace's equation by separation of variables into eigenfunctions (see *Funky Electromagnetics Concepts*). Beware that some references define Green functions only for this electrostatic special case, and so present an overly narrow view of them.

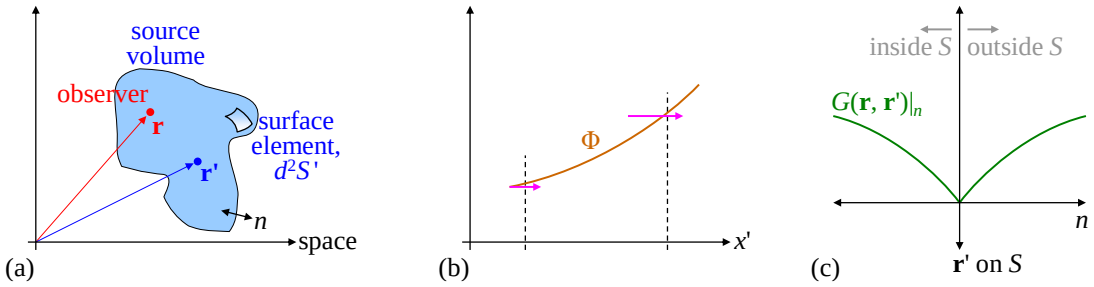


Figure 4.5 (a) A 3D distribution of charges, admired from within. (b) A 1D potential; the flux is proportional to $\partial\Phi/\partial x'$. (c) For Dirichlet BCs, form of G along the normal coordinate n for $\mathbf{r}'$ on the boundary surface S.

Consider a distribution of source charges, as in Figure 4.5a. We continue with the definition of G from earlier sections, and gaussian units:

$$\nabla^2 G(\mathbf{r};\mathbf{r}') = \delta^3(\mathbf{r}-\mathbf{r}'), \qquad \text{and} \qquad \nabla^2\Phi(\mathbf{r}) = \underbrace{-4\pi\rho(\mathbf{r})}_{s(\mathbf{r})}\,.$$

Some references include a factor of (-1) or (-4π) on the δ-function in the definition of G. That breaks the generality of the Green method, and simply moves the factor from $s(\mathbf{r})$ into the Green function itself, but the resulting integral (4.8) is identical, as it must be: $\Phi(\mathbf{r})$ is uniquely determined by $\rho(\mathbf{r}')$ and the BCs. Our convention for G is used in many references, and we believe is objectively simpler in both theory and practice.

The Laplacian boundary condition trick starts with Green's *theorem*, which relates a certain kind of volume integral to a surface integral. We give some insight to Green's theorem in the next section, but the result is: for any functions defined *inside* a volume, $\Phi(\mathbf{r}')$ and $\psi(\mathbf{r}')$:

$$\iiint_{Vol} \Phi(\mathbf{r}')\nabla'^2\psi(\mathbf{r}') - \psi(\mathbf{r}')\nabla'^2\Phi(\mathbf{r}')\, d^3r' = \oiint_{\partial Vol}\left(\Phi(\mathbf{r}')\frac{\partial\psi}{\partial n'} - \psi(\mathbf{r}')\frac{\partial\Phi}{\partial n'}\right)d^2S'$$

$$\textit{where} \quad n' \equiv \text{normal coordinate, so, e.g.,} \quad \frac{\partial\psi}{\partial n'} = \nabla'\psi\bullet\hat{\mathbf{n}}'$$

Note that the primes denote *source* coordinates. In electrostatics, we let $\Phi(\mathbf{r}')$ be the electrostatic potential *inside the volume*, and $\psi(\mathbf{r}') \rightarrow G(\mathbf{r}, \mathbf{r}')$, taking $\mathbf{r}$ as fixed. The operator ∇' tells us how a function changes as we move around the source coordinate $\mathbf{r}'$, with $\mathbf{r}$ held fixed. Then Φ is explicitly given by (gaussian units):

$$\underbrace{\Phi(\mathbf{r})}_{\substack{\mathbf{r}\text{ inside}\\ \text{volume}}} = \iiint_{Vol} G(\mathbf{r};\mathbf{r}')\underbrace{\left(-4\pi\rho(\mathbf{r}')\right)}_{s(\mathbf{r})}d^3r' + \oiint_{\partial Vol}\left(\Phi(\mathbf{r}')\frac{\partial G}{\partial n'} - G(\mathbf{r},\mathbf{r}')\frac{\partial\Phi}{\partial n'}\right)d^2S' \tag{4.8}$$

$$\textit{where} \quad \partial Vol \equiv \text{boundary of the volume}.$$

If $\mathbf{r}$ is outside the volume, it violates the terms of Green's Theorem, the volume integral is zero, and the result is meaningless. At this point, we have not given any BCs for G, so as with all Green functions, there are many G that satisfy the defining equation $\nabla^2 G = \delta^3(\mathbf{r} - \mathbf{r}')$. We must find BCs for G to make it unique.

Dirichlet BCs: There are 2 terms in the surface integral of (4.8). For Dirichlet BCs, Φ(boundary) is given. Therefore, we make G unique by choosing G(boundary; $\mathbf{r}'$) = 0, so the second surface term vanishes. Figure 4.5c illustrates $G(n, \mathbf{r}')$ along n, the normal coordinate to the boundary surface. This BC for G guarantees that $\Phi(\mathbf{r})$ from (4.8) meets the given Φ(boundary).

Neumann BCs: $d\Phi/dn' = E_{n'}$ is given everywhere on the boundary. This is equivalent to specifying $E_\perp$ or the surface charge density σ everywhere on the boundary, because:

$$\frac{d\Phi}{dn'} = -E_\perp = -4\pi\sigma \qquad \text{(gaussian units)}.$$

You might think we choose $dG/dn' = 0$ everywhere on the boundary, so the first term in the surface integral would vanish. This turns out to be a contradiction, so it fails to give a solution ([Jac 1999 p39] or [Bra p174], but note they use different δ-function conventions from each other, and from us). The contradiction appears from Gauss' Law applied to the definition of the Green function, for $\mathbf{r}$ inside the volume:

$$\nabla^2 G(\mathbf{r};\mathbf{r}') \equiv \nabla\bullet\nabla G = \delta^3(\mathbf{r} - \mathbf{r}') \qquad \Rightarrow \qquad \oiint_{\partial Vol}\nabla G\bullet\hat{\mathbf{n}}'\, d^2S' = \iiint_{Vol}\delta^3(\mathbf{r} - \mathbf{r}')\, d^3r \quad \text{or}$$

$$\oiint_{\partial Vol}\frac{dG}{dn'}d^2S' = 1.$$

So dG/dn' cannot be 0 everywhere. The simplest requirement to state (not necessarily to solve) is dG/dn' = constant = $1/S$, where $S \equiv$ surface area, which satisfies the above surface integral. However, if the system is infinitely large, as is commonly approximated, this reduce to the simple $dG/dn' = 0$.

The final solution then comes from the fact that $\Phi(\mathbf{r}')$ is defined by Neumann BCs only up to an additive constant. Therefore, there exists some $\Phi(\mathbf{r}')$ such that the first term in the surface integral of (4.8) is zero. Then *that* Φ satisfies:

$$\underbrace{\Phi(\mathbf{r})}_{\substack{\mathbf{r}\text{ inside}\\ \text{volume}}} = \iiint_{Vol} G(\mathbf{r};\mathbf{r}')\underbrace{\left(-4\pi\rho(\mathbf{r}')\right)}_{s(\mathbf{r})}d^3r' - \oiint_{\partial Vol} G(\mathbf{r},\mathbf{r}')\frac{\partial\Phi}{\partial n'}d^2S' \ .$$

This gives the solution $\Phi(\mathbf{r})$ inside the volume as an integral of the given Neumann BCs.

The BCs we choose for the Green function depends only on the *type* of BCs for Φ (Dirichlet or Neumann), but not on the boundary values themselves.

Derivation of Green's Theorem

This section is optional. Green's theorem relates a certain kind of volume integral to a surface integral. We start with a one-dimensional section of 3D space, which may be easier to think about (Figure 4.5). Consider any two functions $\Phi(x')$ and $\psi(x')$; we use primes to indicate coordinates of *source* charges. From simple integration by parts, we have:

$$\int_a^b \Phi \frac{d^2}{dx'^2}\psi \, dx' = \Phi \frac{d}{dx}\psi\Big|_a^b - \int_a^b \left(\frac{d}{dx'}\Phi\right)\left(\frac{d}{dx'}\psi\right) dx' .$$

We could just as well swap the roles of Φ and ψ, and have:

$$\int_a^b \psi \frac{d^2}{dx'^2}\Phi \, dx' = \psi \frac{d}{dx'}\Phi\Big|_a^b - \int_a^b \left(\frac{d}{dx'}\Phi\right)\left(\frac{d}{dx'}\psi\right) dx' .$$

Subtracting the latter from the former cancels the integral on the RHS:

$$\int_a^b \left(\Phi \frac{d^2}{dx'^2}\psi - \psi \frac{d^2}{dx'^2}\Phi\right) dx' = \left(\Phi \frac{d}{dx'}\psi - \psi \frac{d}{dx'}\Phi\right)\Big|_a^b \qquad \text{or}$$

$$\int_a^b \Phi \frac{d^2}{dx'^2}\psi \, dx' = \int_a^b \psi \underbrace{\frac{d^2}{dx'^2}\Phi}_{\substack{\text{charge}\\ \text{density}}} dx' + \left(\Phi \frac{d}{dx'}\psi - \psi \frac{d}{dx'}\Phi\right)\Big|_a^b \tag{4.9}$$

We recognize the charge density in the first integral on the right. We can isolate Φ on the left, at a specific point x (not x') in the volume, by choosing ψ such that $d^2\psi/dx'^2 = \delta^3(x - x')$; in other words, by choosing $\psi(x')$ to be a Green function:

$$\psi(x') \to G(x; x') \text{ such that:} \qquad \frac{\partial^2 G}{\partial x'^2} = \delta(x - x') .$$

For purposes of Green's Theorem, 'x' is a constant; x' is the variable. Green's Theorem holds for *any* functions $\Phi(x')$ and $\psi(x')$, so it holds for this choice of ψ. Then the LHS of (4.9) becomes:

$$\int_a^b \Phi \frac{d^2}{dx'^2}\psi \, dx' = \int_a^b \Phi(x)\, \delta(x - x')\, dx' = \Phi(x) . \tag{4.10}$$

So (4.9) becomes an explicit integral for $\Phi(x)$, x inside the volume:

$$\Phi(x) = \int_a^b G(x; x')\left(-4\pi\rho\right) dx' + \left(\Phi(x')\frac{dG}{dx'} - G(x; x')\frac{d\Phi}{dx'}\right)\Big|_a^b . \tag{4.11}$$

To generalize this to 3D, we throw in the requisite vector identities, and upgrade each term in our development by two additional dimensions. Start by deriving a kind-of 3D version of integration by parts:

$$\iiint_{Vol} \underbrace{\nabla' \bullet \mathbf{A}(\mathbf{r}')}_{\text{scalar}} d^3r' = \oiint_{\partial Vol} \mathbf{A}(\mathbf{r}') \bullet \hat{\mathbf{n}}' \, d^2S' \quad \textit{where} \quad \hat{\mathbf{n}}' \equiv \text{unit vector pointing outward} .$$

$$\text{Let} \quad \mathbf{A}(\mathbf{r}') = \Phi(\mathbf{r}')\nabla'\psi(\mathbf{r}') \qquad \Rightarrow$$

$$\iiint_{Vol} \nabla' \bullet \left(\Phi \nabla' \psi\right) d^3r' = \oiint_{\partial Vol} \Phi(\mathbf{r}') \frac{\partial \psi}{\partial n'} d^2S'$$

Use a vector identity for divergence:

$$\nabla'\bullet\left(\Phi\,\nabla'\psi\right)=\nabla'\Phi\bullet\nabla'\psi+\Phi\,\nabla'^2\psi\ \Rightarrow$$

$$\iiint_{Vol}\left(\nabla'\Phi\bullet\nabla'\psi+\Phi\,\nabla'^2\psi\right)d^3r'=\oiint_{\partial Vol}\Phi(\mathbf{r}')\frac{\partial\psi}{\partial n'}\,d^2S'$$

As in our 1D warmup, we can swap the roles of Φ and ψ, and subtract the result from the above. The first term on the LHS cancels, leaving:

$$\iiint_{Vol}\left[\Phi(\mathbf{r}')\nabla'^2\psi(\mathbf{r}')-\psi(\mathbf{r}')\nabla'^2\Phi(\mathbf{r}')\right]d^3r'=\oiint_{\partial Vol}\left(\Phi(\mathbf{r}')\frac{\partial\psi}{\partial n'}-\psi(\mathbf{r}')\frac{\partial\Phi}{\partial n'}\right)d^2S'\ .$$

Now choose $\psi(\mathbf{r}') \to G(\mathbf{r};\mathbf{r}')$, where $\mathbf{r}$ is a constant inside the volume, and:

$$\nabla'^2 G(\mathbf{r};\mathbf{r}')=\delta^3(\mathbf{r}-\mathbf{r}')\quad\Rightarrow$$

$$\underbrace{\Phi(\mathbf{r})}_{\substack{\mathbf{r}\text{ inside}\\ \text{volume}}}=\iiint_{Vol}G(\mathbf{r};\mathbf{r}')\underbrace{\left(-4\pi\rho(\mathbf{r}')\right)}_{s(\mathbf{r})}d^3r'+\oiint_{\partial Vol}\left(\Phi(\mathbf{r}')\frac{\partial G}{\partial n'}-G(\mathbf{r},\mathbf{r}')\frac{\partial\Phi}{\partial n'}\right)d^2S'\ .$$

The particular G we use depends on the BCs given in the original problem for $\Phi(\mathbf{r})$, as shown in the previous section.

Desultory Green Topics

Fourier Series Method for Green Functions

In some cases, we cannot find the Green function in closed form, but we can find a Fourier series for it. This section assumes you are familiar with Fourier Series, and Green functions without Fourier Series. The example below constructs a Green function from a 2D Fourier Series for the *x*-*y* parts, and for each Fourier component, uses a variant of 1D left-right construction (introduced in an earlier section) for the *z* part of that component.

To illustrate the Fourier method for Green functions, we expound on the question [Jac Q2.23 and p128-9]. There are many solutions for Q2.23 (which has no source charge) posted on the internet; most use separation of variables and eigenfunctions. (We describe such a method generally in *Funky Electromagnetic Concepts*.) We here derive one form of the Green function for such a problem [Jac 3.168 p129m]. In principle, this solves for the potential $\Phi(\mathbf{r})$ for arbitrary charge density by using (4.8).

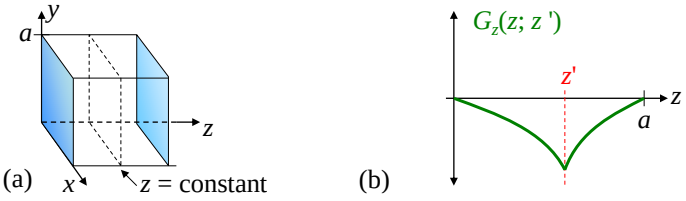


Figure 4.6 (a) A cube with specified boundary potentials. (b) Green function for the *z*-direction, requiring sinh functions.

The system is a cube of side a, with one corner at the origin, extending to (a, a, a) (Figure 4.6). The cube has arbitrary charge density $\rho(\mathbf{r})$ inside. The two faces of $z = 0$ and $z = a$ are at fixed potential $\Phi = V$, and the other 4 faces are at $\Phi = 0$. Find the potential inside the cube. As with many such problems, it is slightly ill-posed: the potential along the x and y axes, and 6 other similar edges, are specified as both 0 and V. We can ignore this by saying that the faces with $\Phi = V$ are separated by a tiny distance from the rest of the cube, so the edges don't quite touch.

The geometry favors rectangular coordinates. The BCs on Φ are Dirichlet (Φ is given everywhere on the surface of the cube), so the BCs on G are all zero. This means the three coordinate directions are all equivalent for G, and we could find G as a 3D Fourier series [Jac 3.167 p129]. However, the original problem is given with z chosen as having different BCs than x and y, so we choose to treat z differently than x and y.

We will Fourier expand the x-y surfaces (2D), but write the *z*-dependence of each Fourier component (of *G*) directly. This is desirable, because lower dimensional series usually converge faster than higher dimension.

2D Fourier Series: Recall that a well-behaved 2D function of a rectangular region of space $x \in [0,a]$, $y \in [0,b]$ can be written as a series of sinusoids:

$$f(x,y) = \sum_{l,m=1}^{\infty} A_{lm} \underbrace{\sin\left(\frac{l\pi}{a}x\right)\sin\left(\frac{m\pi}{b}y\right)}_{\text{basis function}} + \text{other cos( ) terms we won't need here .}$$

We justify the lack of cos( ) shortly. Given the function $f(x, y)$, we can find the coefficients A_{lm} of its series expansion from orthogonality of the Fourier basis functions:

$$A_{lm} = \frac{4}{ab}\int_0^b dy \int_0^a dx\, f(x,y)\sin\left(\frac{l\pi}{a}x\right)\sin\left(\frac{m\pi}{b}x\right). \tag{4.12}$$

The leading coefficient above is the inverse of the normalization of the basis function:

$$\int_0^b dy \int_0^a dx \left[\sin\left(\frac{\pi}{a}lx\right)\sin\left(\frac{\pi}{b}my\right)\right]^2 = \frac{ab}{4}.$$

3D Green function: For the potential of the cube, $\Phi(x, y, z)$, we seek the Green function, which looks like this in rectangular coordinates:

$$\nabla^2 G(x,y,z;x',y',z') = \delta^3(\mathbf{r}-\mathbf{r}') = \delta(x-x')\delta(y-y')\delta(z-z'). \tag{4.13}$$

In our parlance, we say $G( )$ is the piece of Φ at (x, y, z) due to the piece of source at (x', y', z'). As described in a previous section, G (as a whole) does *not* separate into $X(x)Y(y)Z(z)$. However, each Fourier component of G is a solution to Laplace's equation everywhere except at **r'**, so each *component can* be separated into $X(x)Y(y)Z(z)$, while still including a discontinuity. In such a separation for the ∇^2 operator in rectangular coordinates, at least one function is sin/cos, and at least one is sinh/cosh. Because we chose to Fourier expand x-y, they must be sin/cos, and therefore $Z(z)$ must be sinh/cosh. Thus G can be written:

$$G(x,y,z;x'y'z') = \sum_{l,m=1}^{\infty} A_{lm}(x',y')\sin\left(\frac{l\pi}{a}x\right)\sin\left(\frac{m\pi}{a}y\right)\underbrace{Z_{lm}(z;z')}_{\sinh/\cosh}. \tag{4.14}$$

Note that each pair of values (x', y') has its *own* distinct Fourier series. We call the z part of each component of the Green function $Z_{lm}(z; z')$. Note that each *lm* component has a *different* Z_{lm}, which is why there is no global $Z(z)$ that can be separated from the rest of G. The units of the coordinates are $[x] = [y] = [z] =$ distance, and $[A_{lm}Z_{lm}]$ are $[x]^{-1}$.

As noted earlier, the BCs for Φ given in the problem define the BCs for $G( )$, which then makes $G( )$ unique. We must impose $G( ) = 0$ everywhere on the boundary (all 6 faces):

$$G(\text{boundary}; x', y', z') = 0, \qquad \forall x', y', z'.$$

Each Fourier component satisfies Laplace's equation everywhere except at (x', y', z'), and is zero on the boundaries. The BC on G demands a square slice of z = constant has $G = 0$ around its perimeter. This can be satisfied with X and Y = sin( ), but not cos( ). Thus:

$$X_{lm}(x) = \sin\left(\frac{l\pi}{a}x\right), \qquad Y_{lm}(y) = \sin\left(\frac{m\pi}{a}y\right), \qquad l,m \text{ integer}.$$

The infinite Fourier sum in X and Y compose a $\delta(x - x')\delta(y - y')$, leaving only $\delta(z - z')$ to be constructed in Z_{lm}. In rectangular coordinates, X_{lm} depends only on l, and Y_{lm} depends only on m. We retain the "*lm*" on both because other coordinate systems don't separate so cleanly. (Y_{lm} here is *not* a spherical harmonic.)

Now to find G, "all" we must do is find the Z_{lm} and A_{lm}. Z_{lm} must provide the $\delta(z - z')$ in (4.13), so we start there. The Z_{lm} must look like Figure 4.6b, because they are zero at $z = 0$ and $z = a$, and each must have a positive step in its derivative at $z = z'$. We already know that $Z_{lm}(z)$ comprises only sinh/cosh, but because G(boundary) = 0, it must be made of only sinh. From Figure 4.6b:

$$\text{For } z < z': \qquad Z_{lm}(z; z') = A\sinh(k_{lm} z)$$

$$\textit{For } z > z': \qquad Z_{lm}(z; z') = B\sinh\big(k_{lm}(a - z)\big) \qquad \textit{where} \quad k_{lm} \equiv \frac{\pi}{a}\sqrt{l^2 + m^2}\,.$$

k_{lm} is chosen to cancel the sum of the eigenvalues from $X(x)$ and $Y(y)$, as described in the section on boundary value problems. Since k_{lm} depends on the component "lm", each Z_{lm} is a different function.

It is customary to combine these two pieces of Z_{lm} into a single form:

$$Z_{lm}(z; z') = C\sinh(k_{lm} z_<)\sinh\big(k_{lm}(a - z_>)\big) \quad \textit{where} \quad z_< \equiv \min(z, z'),\ z_> \equiv \max(z, z')\,.$$

Remember that for purposes of derivatives, z' is a given constant, so in the above form, one factor is a function of z, and the other is just a constant that depends on z'. (This combined form looks clumsy, but is helpful with deeper concepts of self-adjointness which we do not pursue here.) The coefficient C could be absorbed into the Fourier coefficients A_{lm}, but we have to do the work sooner or later. Therefore, we opt to keep all the z-dependence tidily in Z_{lm}, so we find C now:

$$\left.\frac{dZ_{lm}}{dz}\right|_{z<z'} = Ck_{lm}\cosh(k_{lm} z)\sinh\big(k_{lm}(a - z')\big), \qquad \left.\frac{dZ_{lm}}{dz}\right|_{z>z'} = -Ck_{lm}\sinh(k_{lm} z')\cosh\big(k_{lm}(a - z)\big)\,.$$

The unit step in derivative at z' gives:

$$1 = \left.\frac{dZ_{lm}}{dz}\right|_{z'_+} - \left.\frac{dZ_{lm}}{dz}\right|_{z'_-} = -Ck_{lm}\Big[\sinh(k_{lm} z')\cosh\big(k_{lm}(a - z')\big) + \cosh(k_{lm} z')\sinh\big(k_{lm}(a - z')\big)\Big]$$

$$\text{Use: } \sinh(u + v) = \sinh u \cosh v + \cosh u \sinh v:$$

$$1 = -Ck_{lm}\sinh\big(k_{lm}(z' + a - z')\big) \qquad \Rightarrow \qquad C = \frac{-1}{k_{lm}\sinh(k_{lm} a)}$$

Note that C depends on the source point z', and is negative, as shown in Figure 4.6b. The complete Z_{lm} is:

$$Z_{lm}(z; z') = \frac{-1}{k_{lm}\sinh(k_{lm} a)}\sinh(k_{lm} z_<)\sinh\big(k_{lm}(a - z_>)\big)\,. \tag{4.15}$$

As expected, Z_{lm} is *not* a 1D Green function, because it is non-zero everywhere inside the boundary. However, it *does* provide the discontinuity required in G. In fact:

$$\frac{\partial^2}{\partial z^2} Z_{lm}(z \neq z'; z') = {k_{lm}}^2 Z_{lm}, \qquad \frac{\partial^2}{\partial z^2} Z_{lm}(z = z'; z') = \delta(z - z') \quad \Leftrightarrow \quad \int_{z'_-}^{z'_+} \frac{\partial^2}{\partial z^2} Z_{lm}\, dz = 1\,.$$

(We could say, in general $\frac{\partial^2}{\partial z^2} Z_{lm}(z; z') = {k_{lm}}^2 Z_{lm} + \delta(z - z')$.) $[k_{lm}] = [z]^{-1}$, so the scaling of Z_{lm} gives it units of distance, $[z]$.

For $A_{lm}(x', y')$ we have:

$$\nabla^2 G = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)\sum_{l,m=1}^{\infty} A_{lm}(x', y')\sin\left(\frac{\pi}{a} l x\right)\sin\left(\frac{\pi}{a} m y\right) Z_{lm} = \delta(x - x')\delta(y - y')\delta(z - z')$$

$$\sum_{l,m=1}^{\infty} A_{lm}\left(-\frac{l^2\pi^2}{a^2} - \frac{m^2\pi^2}{a^2} + \frac{\partial^2}{\partial z^2}\right)\sin\left(\frac{l\pi}{a} x\right)\sin\left(\frac{m\pi}{a} y\right) Z_{lm} = \delta(x - x')\delta(y - y')\delta(z - z')\,.$$

This means the A_{lm} have units of $[x]^{-2}$. To pick out a single coefficient $A_{l'm'}$, we multiply both sides by the Fourier basis function, and integrate over the x-y region, recalling the basis function normalization is $a^2/4$:

$$A_{l'm'}\left(-\frac{l^2\pi^2}{a^2}-\frac{m^2\pi^2}{a^2}+\frac{\partial^2}{\partial z^2}\right)\frac{a^2}{4}Z_{lm} = \int_0^a\int_0^a dx\,dy \sin\left(\frac{l'\pi}{a}x\right)\sin\left(\frac{m'\pi}{a}y\right)\delta(x-x')\delta(y-y')\delta(z-z')$$
$$= \sin\left(\frac{l'\pi}{a}x'\right)\sin\left(\frac{m'\pi}{a}y'\right)\delta(z-z') \ .$$

The only term that survives on the left is from $\frac{\partial^2}{\partial z^2}Z_{lm}(z=z') = \delta(z-z')$, which cancels the $\delta(\,)$ on the right:

$$A_{l'm'}(x',y')\,\cancel{\delta(z-z')} = \frac{4}{a^2}\sin\left(\frac{l'\pi}{a}x'\right)\sin\left(\frac{m'\pi}{a}y'\right)\cancel{\delta(z-z')} \ .$$

(Equivalently, we could integrate both sides with $\int_{z'_-}^{z'_+}(\,)\,dz$.) We drop the primes from l' and m', yielding:

$$A_{lm}(x',y') = \frac{4}{a^2}\sin\left(\frac{l\pi}{a}x'\right)\sin\left(\frac{m\pi}{a}y'\right).$$

The final Green function combines these A_{lm} with Z_{lm} from (4.15):

$$G(x,y,z;z'y'z') = \sum_{l,m=1}^{\infty}\frac{-4}{a^2}\sin\left(\frac{l\pi}{a}x'\right)\sin\left(\frac{m\pi}{a}y'\right)\sin\left(\frac{l\pi}{a}x\right)\sin\left(\frac{m\pi}{a}y\right)\frac{\sinh(k_{lm}z_<)\sinh\left(k_{lm}\left(a-z_>\right)\right)}{k_{lm}\sinh\left(k_{lm}a\right)} \ .$$

Using (4.8), this G gives $\Phi(\mathbf{r})$ in integral form for arbitrary $\rho(\mathbf{r})$ and Dirichlet BCs.

Green-Like Methods: The Born Approximation

In the Born approximation, and similar problems, we have our unknown function, now called $\psi(x)$, on both sides of the equation. So both our unknown function $f(x) \to \psi(x)$, and our source $s(x) \to \psi(x)$:

(1) $\mathcal{L}\{\psi(x)\} = \psi(x)$.

The theory of Green functions still works, so that:

$$\psi(x) = \int \psi(x')G(x\,;x')\,dx' \ ,$$

but this doesn't *solve* the equation, because we still have ψ on both sides of the equation. We could try rearranging Eq (1):

$\mathcal{L}\{\psi(x)\} - \psi(x) = 0$ which is the same as

$\mathcal{L}'\{\psi(x)\} = 0,$ with $\mathcal{L}'\{\psi(x)\} \equiv \mathcal{L}\{\psi(x)\} - \psi(x)$.

But recall that Green functions require a nonzero source function $s(x)$ on the right-hand side. The method of Green functions can't solve homogeneous equations, because $s(x) = 0$ yields:

$$\mathcal{L}\{\psi(x)\} = s(x) = 0 \qquad \to \qquad \psi(x) = \int s(x')G(x\,;x')\,dx' = \int 0\,dx' = 0 \ .$$

Technically, this is a solution, but it's not very useful. So Green functions don't work when $\psi(x)$ appears on both sides. However, under the right conditions, we can make a useful approximation. If we have an approximate solution,

$$\mathcal{L}\left\{\psi^{(0)}(x)\right\} \approx \psi^{(0)}(x) \ ,$$

then we can expand ψ in a perturbation series of corrections:

$$\psi(x) = \psi^{(0)}(x) + \psi^{(1)}(x) + \psi^{(2)}(x) + \ldots$$

where $\psi^{(1)}$ is 1st order perturbation, $\psi^{(2)}$ is 2nd order, ...

Now we can use $\psi^{(0)}(x)$ as the source term, and use a method like Green functions, to get a better approximation to $\psi(x)$:

$$\mathcal{L}\{\psi(x)\} = \psi(x) \qquad \Rightarrow \qquad \psi^{(0)} + \psi^{(1)}(x) = \int \psi^{(0)}(x')G(x;x')\,dx' \tag{4.16}$$

where $G(x;x')$ is the Green's function for $\mathcal{L}$, *i.e.* $\mathcal{L}\{G(x;x')\} = \delta(x-x')$.

$\psi^{(0)}(x) + \psi^{(1)}(x)$ is called the **first Born approximation** of $\psi(x)$. This process can be extended to arbitrarily high accuracy.

In QM, $\mathcal{L}$ is the perturbed hamiltonian:

$$\mathcal{L} = \hat{H}_0 + V(\mathbf{r}),$$

where $V(\mathbf{r})$ is "small" compared to $\hat{H}_0$. $\psi^{(0)}$ is an exact solution to the unperturbed Schrodinger equation, so it can be shown that the Born approximation (4.16) reduces to:

$$\psi^{(1)}(x) = \int \psi^{(0)}(x')G(x;x')\,dx'$$

$$\psi^{(2)}(x) = \int \psi^{(1)}(x')G(x;x')\,dx' \qquad \ldots \qquad \psi^{(n+1)}(x) = \int \psi^{(n)}(x')G(x;x')\,dx'$$

This process assumes that the Green function is "small" enough to produce a converging sequence. The first Born approximation is valid when $\psi^{(1)}(x) << \psi^{(0)}(x)$ everywhere, and in many other, less stringent but harder to quantify, conditions. The extension to higher order approximations is straightforward: the Born approximation is valid when $\psi^{(n)}(x) << \psi^{(0)}(x)$. See *Quirky Quantum Concepts* for detailed information.

TBS: a real QM example?

Green function as inverse operator??

5 Complex Analytic Functions

For a review of complex numbers and arithmetic, see *Quirky Quantum Concepts*.

Notation: In this chapter, z, w are *always* complex variables; x, y, r, θ are *always* real variables. Other variables are defined as used.

A complex function of a complex variable $f(z)$ is **analytic** over some domain if it has an infinite number of continuous derivatives in that domain. It turns out, if $f(z)$ is once differentiable on a domain, then it is infinitely differentiable, and therefore analytic on that domain.

A *necessary* condition for analyticity of $f(z) = u(x, y) + iv(x, y)$ near z_0 is that the Cauchy-Riemann equations hold, to wit:

$$\frac{\partial f}{\partial x} = -i\frac{\partial f}{\partial y} \quad \Rightarrow \quad \frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x} = -i\left(\frac{\partial u}{\partial y} + i\frac{\partial v}{\partial y}\right) = -i\frac{\partial u}{\partial y} + \frac{\partial v}{\partial y} \quad \Rightarrow \quad \frac{\partial u}{\partial x} = \frac{\partial v}{\partial y},\ and\ \frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y}.$$

A *sufficient* condition for analyticity of $f(z) = u(x, y) + iv(x, y)$ near z_0 is that the Cauchy-Riemann equations hold, and the first partial derivatives of f exist and are continuous in a neighborhood of z_0. Note that if the first derivative of a complex function is continuous, then all derivatives are continuous, and the function is analytic. This condition implies

$$\nabla^2 u = \nabla^2 v = 0$$

$$\nabla u \cdot \nabla v = 0 \quad \Rightarrow \quad \text{"level lines" are perpendicular}$$

$\int_{z_1}^{z_2} f(z)\,dz$ is countour independent if $f(z)$ is single-valued

Note that a function can be analytic in some regions, but not others. **Singular** points, or **singularities**, are not in the domain of analyticity of the function, but border the domain [Det def 4.5.2 p156]. E.g., $\sqrt{z}$ is singular at 0, because it is not differentiable, but it is continuous at 0. **Poles** are singularities near which the function is unbounded (infinite), but can be made finite by multiplication by $(z - z_0)^k$ for some finite k [Det p165]. This implies $f(z)$ can be written as:

$$f(z) = a_k(z - z_0)^{-k} + a_{k-1}(z - z_0)^{-k+1} + \ldots + a_{-1}(z - z_0)^{-1} + a_0 + a_1(z - z_0)^1 + \ldots.$$

The value k is called the **order** of the pole. All poles are singularities. Some singularities are like "poles" of infinite order, because the function is unbounded near the singularity, but it is not a pole because it cannot be made finite by multiplication by any $(z - z_0)^k$, for example $e^{1/z}$. Such a singularity is called an **essential singularity**.

A Laurent series expansion of a function is similar to a Taylor series expansion, but the sum runs from $-\infty$ to $+\infty$, instead of from 1 to ∞. In both cases, an expansion is about some point, z_0:

Taylor series: $f(z) = f(z_0) + \sum_{n=1}^{\infty} b_n (z - z_0)^n$ *where* $b_n = \frac{f^{(n)}(z_0)}{n!}$

Laurent series: $f(z) = \sum_{n=-\infty}^{\infty} a_n (z - z_0)^n$, *where* $a_n = \frac{1}{2\pi i}\oint_{around\ z_0} \frac{f(z)}{(z - z_0)^{k+1}}\,dz$

[Det thm 4.6.1 p163] Analytic functions have Taylor series expansions about every point in the domain. Taylor series can be thought of as special cases of Laurent series. But analytic functions also have Laurent expansions about isolated singular points, i.e. the expansion point is not even in the domain of analyticity! The Laurent series is valid in some annulus around the singularity, but not across branch cuts. Note that in general, the a_k and b_k could be complex, but in practice, they are often real.

Properties of analytic functions:

1. If it is differentiable once, it is infinitely differentiable.
2. The Taylor and Laurent expansions are unique. This means you may use any of several methods to find them for a given function.
3. If you know a function and all its derivatives at any point, then you know the function everywhere in its domain of analyticity. This follows from the fact that every analytic function has a Laurent power series expansion. It implies that the value throughout a region is completely determined by its values at a boundary.
4. An analytic function cannot have a local extremum of absolute value. (Why not??)

Residues

Mostly, we use complex contour integrals to evaluate difficult real integrals, and to sum infinite series. To evaluate contour integrals, we need to evaluate residues. Here, we introduce residues. The **residue** of a complex function at a complex point z_0 is the a_{-1} coefficient of the Laurent expansion about the point z_0. Residues of singular points are the only ones that interest us. (In fact, residues of branch points are not defined [Sea sec 13.1].)

Common ways to evaluate residues

1. The residue of a removable singularity is zero. This is because the function is bounded near the singularity, and thus a_{-1} must be zero (or else the function would blow up at z_0):

   $$\text{For } a_{-1} \neq 0, \text{ as } \quad z \to z_0, \quad a_{-1}\frac{1}{z - z_0} \to \infty \qquad \Rightarrow \qquad a_{-1} = 0\,.$$

2. The residue of a **simple pole** at z_0 (i.e., a pole of order 1) is

   $$a_{-1} = \lim_{z \to z_0} (z - z_0)\, f(z)\,.$$

3. Extending the previous method: the residue of a pole at z_0 of order k is

   $$a_{-1} = \frac{1}{(k-1)!} \lim_{z \to z_0} \frac{d^{k-1}}{dz^{k-1}} (z - z_0)^k \; f(z) \;,$$

   which follows by substitution of the Laurent series for $f(z)$, and direct differentiation. We show it here, noting that poles of order m imply that $a_k = 0$ for $k < -m$, so we get:

   $$f(z) = a_k (z - z_0)^{-k} + a_{k-1}(z - z_0)^{-k+1} + \ldots + a_{-1}(z - z_0)^{-1} + a_0 + a_1 (z - z_0)^1 + \ldots$$

   $$(z - z_0)^k \, f(z) = a_k + a_{k-1}(z - z_0)^1 + \ldots + a_{-1}(z - z_0)^{k-1} + a_0 (z - z_0)^k + a_1 (z - z_0)^{k+1} + \ldots$$

   $$\frac{d^{k-1}}{dz^{k-1}} (z - z_0)^k \; f(z) = (k-1)! a_{-1}(z - z_0)^{k-1} + \frac{k!}{1!} a_0 (z - z_0)^k + \frac{(k+1)!}{2!} a_1 (z - z_0)^{k+1} + \ldots$$

   $$\lim_{z \to z_0} \frac{d^{k-1}}{dz^{k-1}} (z - z_0)^k \; f(z) = (k-1)! a_{-1}$$

   $$\Rightarrow \qquad \boxed{a_{-1} = \frac{1}{(k-1)!} \lim_{z \to z_0} \frac{d^{k-1}}{dz^{k-1}} (z - z_0)^k \; f(z)}$$

4. If $f(z)$ can be written as $f(z) = \dfrac{P(z)}{Q(z)}$, where P is continuous at z_0, and $Q'(z_0) \neq 0$ (and is continuous at z_0), then $f(z)$ has a simple pole at z_0, and

$$\operatorname*{Res}_{z=z_0} f(z) = \frac{P(z_0)}{\left.\frac{d}{dz}Q(z)\right|_{z_0}} \equiv \frac{P(z_0)}{Q'(z_0)}. \qquad \text{Why? Near } z_0, \quad Q(z) \approx (z - z_0)Q'(z_0).$$

$$\text{Then:} \quad \operatorname*{Res}_{z=z_0} f(z) = \lim_{z\to z_0} (z - z_0) f(z) = \lim_{z\to z_0} (z - z_0) \frac{P(z_0)}{(z - z_0)Q'(z_0)} = \frac{P(z_0)}{Q'(z_0)}.$$

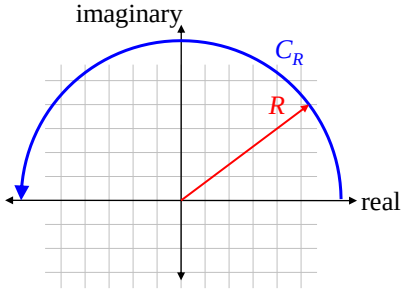
5. Find the Laurent series, and hence its coefficient of $(z - z_0)^{-1}$. This is sometimes easy if $f(z)$ is given in terms of functions with well-known power series expansions. See the sum of series example later.

We will include real-life examples of most of these as we go.

Contour Integrals

Contour integration is an invaluable tool for evaluating both real and complex-valued integrals. Contour integrals are used all over advanced physics, and we could not do physics as we know it today without them. Contour integrals are mostly useful for evaluating difficult ordinary (real-valued) integrals, and sums of series. In many cases, a function is analytic except at a set of distinct points. In this case, a contour integral may enclose, or pass near, some points of non-analyticity, i.e. **singular** points. It is these singular points that allow us to evaluate the integral.

You often let the radius of the contour integral go to ∞ for some part of the contour:



Any arc where

$$\lim_{R\to\infty} |f(z)| \to \sim \frac{1}{|z|^{1+\varepsilon}}, \qquad \varepsilon > 0$$

has an integral of 0 over the arc.

Beware that this is often stated incorrectly as "any function which goes to zero *faster* than 1/|z| has a contour integral of 0." The problem is that it has to have an *exponent* < –1; it is *not* sufficient to be simply smaller than 1/|z|. E.g. $\frac{1}{|z|+1} < \frac{1}{|z|}$, but the contour integral still diverges.

Jordan's lemma: ??.

Evaluating Integrals

Surprisingly, we can use complex contour integrals to evaluate difficult *real* integrals. The main idea is to find a contour which:

(a) includes some known (possibly complex) multiple of the desired (real) integral,

(b) includes other segments whose values are zero, and

(c) includes a known set of poles whose residues can be found.

Then you simply plug into the residue theorem:

$$\oint_C f(z)\,dz = 2\pi i \sum_{n\ residues} \text{Res}_{z_n} f(z),\ \ where \quad z_n \text{ are the finite set of isolated singularities .}$$

We can see this by considering the contour integral around the unit circle for each term in the Laurent series expanded about $z = 0$. First, consider the z^0 term (the constant term). We seek the value of $\oint_O dz$. dz is a small complex number, representable as a vector in the complex plane. Figure 5.1a shows the geometric meaning of dz. Figure 5.1b shows the geometric approximation to the desired integral.

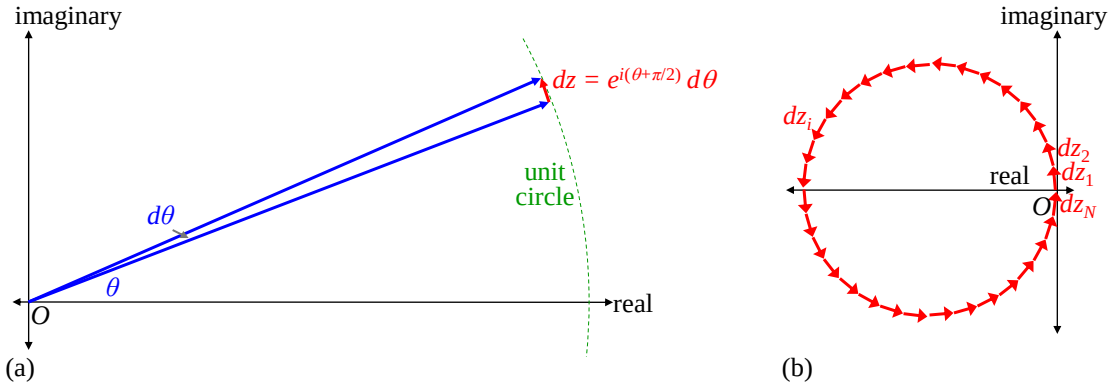


Figure 5.1 (a) Geometric description of dz. (b) Approximation of $\oint_O dz$ as a sum of 32 small complex terms (vectors).

We see that all the tiny dz elements add up to zero: the vectors add head-to-tail, and circle back to the starting point. The sum vector (displacement from start) is zero. This is true for any large number of dz, so we have:

$$\oint_O dz = 0\,.$$

Next, consider the z^{-1} term, $\oint_O \left(\frac{1}{z}\right) dz$. First, we must convert to a 1-dimensional integral, by changing the integration variable to θ:

$$\text{Let} \qquad z = e^{i\theta},\ dz = ie^{i\theta}d\theta: \qquad \oint_O \left(\frac{1}{z}\right) dz = \int_0^{2\pi} e^{-i\theta} ie^{i\theta} d\theta = \int_0^{2\pi} i\,d\theta = 2\pi i\,.$$

The change of variable maps the complex contour and z into an ordinary integral of a real variable.

Geometrically, as z goes positively (counter-clockwise) around the unit circle (Figure 5.2a), z^{-1} goes around the unit circle in the negative (clockwise) direction (Figure 5.2b). Its complex angle, $arg(1/z) = -\theta$, since $z = e^{i\theta}$. As z goes around the unit circle, dz has infinitesimal magnitude $\varepsilon = d\theta$, and argument $\theta + \pi/2$. Hence, the product of $(1/z)\,dz$ always has argument of $-\theta + \theta + \pi/2 = \pi/2$: it is always purely imaginary.

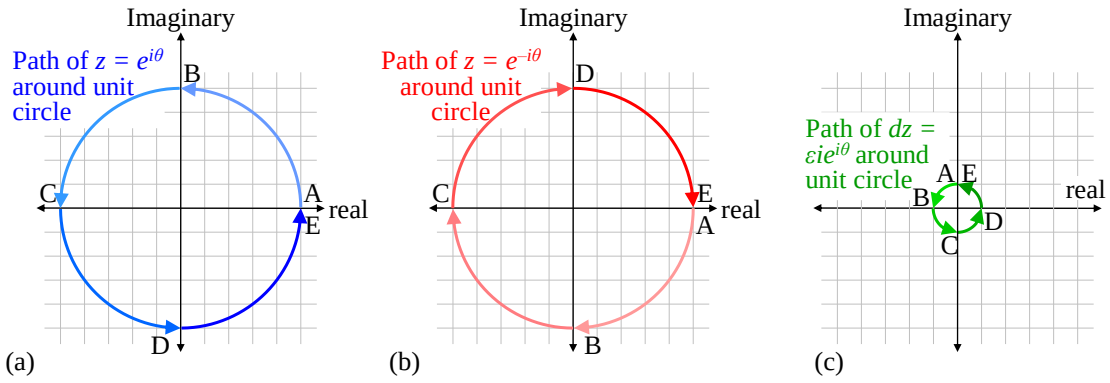


Figure 5.2 Paths of (a) z, (b) $1/z$, and (c) dz in the complex plane

The magnitude of $(1/z)\,dz = d\theta$; thus the integral around the circle is $2\pi i$. Multiplying the integrand by some constant, a_{-1} (the residue), just multiplies the integral by that constant. And any contour integral that encloses the pole $1/z$ and no other singularity has the same value. Hence, for any contour around the origin:

$$\oint_O a_{-1} z^{-1}\, dz = 2\pi i\left(a_{-1}\right) \quad \Rightarrow \qquad a_{-1} = \frac{\oint_O a_{-1} z^{-1}\, dz}{2\pi i}.$$

Now consider the other terms of the Laurent expansion of $f(z)$. We already showed that the $a_0\, z^0$ term, which on integration gives the product $a_0\, dz$, rotates uniformly about all directions, in the positive (counter-clockwise) sense, and sums to zero. Hence the a_0 term contributes nothing to the contour integral.

The $a_1 z^1\, dz$ product rotates uniformly *twice* around all directions in the positive sense, and of course, still sums to zero. Higher powers of z simply rotate more times, but always an integer number of times around the circle, and hence always sum to zero.

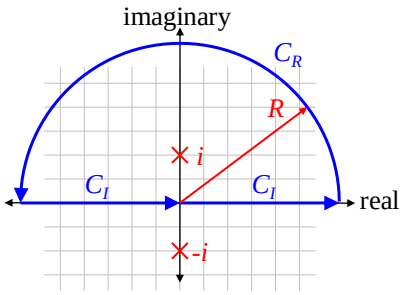
Similarly, $a_{-2} z^{-2}$, and all more negative powers, rotate uniformly about all directions, but in the *negative* (clockwise) sense. Hence, all these terms contribute nothing to the contour integral.

So in the end:

> The only term of the Laurent expansion about 0 that contributes to the contour integral is the residue term, $a_{-1}\, z^{-1}$.

The simplest contour integral: Evaluate $I = \int_0^\infty \frac{1}{x^2+1} dx$.

We know from elementary calculus (let $x = \tan u$) that $I = \pi/2$. We can see this easily from the residue theorem, using the following contour:



"C" denotes a contour, and "I" denotes the integral over that contour. We let the radius of the arc go to infinity, and we see that the closed contour integral $I_C = I + I + I_R$. But $I_R = 0$, because $f(R \to \infty) < 1/R^2$. Then $I = I_C / 2$. $f(z)$ has poles at $\pm i$. The contour encloses one pole at i. Its residue is:

$$\text{Res } f(i) = \frac{1}{\left.\frac{d}{dz}\left(z^2+1\right)\right|_{z=i}} = \frac{1}{2z\big|_{z=i}} = \frac{1}{2i}. \qquad I_C = 2\pi i \sum_n \text{Res } f(z_n) = 2\pi i \frac{1}{2i} = \pi$$

$$I = \frac{I_C}{2} = \frac{\pi}{2}.$$

Note that when evaluating a real integral with complex functions and contour integrals, the *i*'s always cancel, and you get a real result, as you must. It's a good check to make sure this happens.

Choosing the Right Path: Which Contour?

The path of integration is fraught with perils. How will I know which path to choose? There is no universal answer. Often, many paths lead to the same truth. Still, many paths lead nowhere. All we can do is use experience as our guide, and take one step in a new direction. If we end up where we started, we are grateful for what we learned, and we start anew.

We here examine several useful and general, but oft neglected, methods of contour integration. We use some sample problems to illustrate these tools. This section assumes a familiarity with contour integration, and its use in evaluating definite integrals, including the residue theorem.

Example: Evaluate $I = \int_{-\infty}^{\infty} \frac{\sin^2 x}{x^2} dx$.

The integrand is everywhere nonnegative, and somewhere positive, and the limits are in the positive direction, so *I* must be positive. We observe that the given integrand has no poles; it has only a removable singularity at $x = 0$. If we are to use contour integrals, we must somehow create a pole (or a few), to use the residue theorem. Simple poles (i.e. 1st-order) are sometimes best, because then we can also use the indented contour theorem.

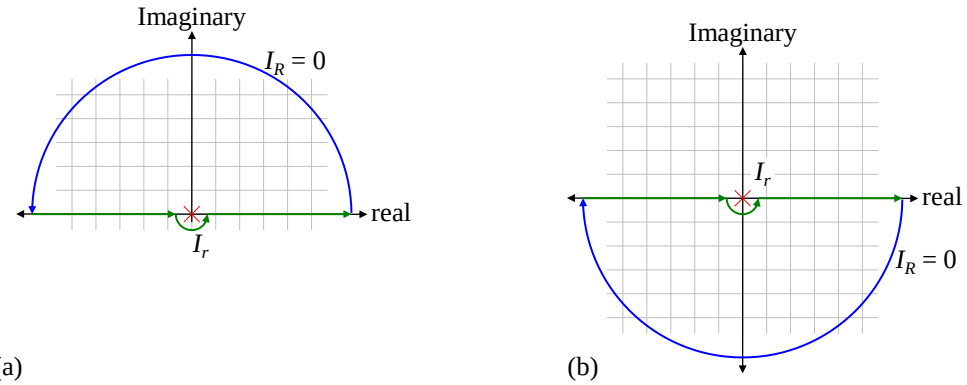


Figure 5.3 Contours for the two exponential integrals: (a) positive (counter-clockwise) exp(2*z*); (b) negative (clockwise) exp(–2*z*)

To use a contour integral (which, a priori, may or may not be a good idea), we must do two things: (1) create a pole; and (2) close the contour. The same method does both: expand the sin( ) in terms of exponentials:

$$I = \int_{-\infty}^{\infty} \frac{\sin^2 x}{x^2} dx = \int_{-\infty}^{\infty} \frac{\left(e^{iz} - e^{-iz}\right)^2}{(2i)^2 z^2} dz = -\frac{1}{4}\left[\int_{-\infty}^{\infty} \frac{e^{i2z}}{z^2} dz - \int_{-\infty}^{\infty} \frac{2}{z^2} dz + \int_{-\infty}^{\infty} \frac{e^{-i2z}}{z^2} dz\right].$$

All three integrals on the RHS have poles at $z = 0$. If we indent the contour underneath the origin, then since the function is bounded near there, the limit as $r \to 0$ leaves the original integral unchanged (Figure 5.3a). The first integral must be closed in the upper half-plane, to keep the exponential small. The second integral

can be closed in either half-plane, since it ~ $1/z^2$. The third integral must be closed in the lower half-plane, again to keep the exponential small (Figure 5.3b). Note that all three contours must use an indentation that preserves the value of the original integral. An easy way to insure this is to use the same indentation on all three.

Now the third integral encloses no poles, so is zero. The 2nd integral, by inspection of its Laurent series, has a residue of zero, so is also zero. Only the first integral contributes. By expanding the exponential in a Taylor series, and dividing by z^2, we find its residue is $2i$. Using the residue theorem, we have:

$$I = \int_{-\infty}^{\infty} \frac{\sin^2 x}{x^2}\,dx = -\frac{1}{4}\left[2\pi i\,(2i)\right] = \pi\ .$$

Example: Evaluate $I = \int_0^{\infty} \frac{\cos(ax) - \cos(bx)}{x^2}\,dx$ [B&C p?? Q1].

This innocent looking problem has a number of funky aspects:

- The integrand is two terms. Separately, each term diverges. Together, they converge.
- The integrand is even, so if we choose a contour that includes the whole real line, the contour integral includes twice the integral we seek (twice *I*).
- The integrand has no poles. How can we use any residue theorems if there are no poles? Amazingly, we can create a useful pole.
- A typical contour includes an arc at infinity, but cos(*z*) is ill-behaved for *z* far off the real-axis. How can we tame it?
- We will see that this integral leads to the indented contour theorem, which can only be applied to simple poles, i.e., first order poles (unlike the residue theorem, which applies to all poles).

Each of these funky features is important, and each arises in practical real-world integrals. Let us consider each funkiness in turn.

1. The integrand is two terms. Separately, each term diverges. Together, they converge.

Near zero, cos(*x*) ≈ 1. Therefore, the zero endpoint of either term of the integral looks like

$$\int_0^{anywhere} \frac{\cos ax}{x^2}\,dx \sim \int_0^{anywhere} \frac{1}{x^2}\,dx = -\frac{1}{x}\Big|_0^{anywhere} \to +\infty\ .$$

Thus each term, separately, diverges. However, the difference is finite. We see this by power series expanding cos(*x*):

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots \quad \Rightarrow \quad \cos(ax) - \cos(bx) = -\frac{a^2x^2}{2} + \frac{b^2x^2}{2} + O\left(x^4\right) \qquad \textit{and}$$

$$\frac{\cos(ax) - \cos(bx)}{x^2} = -\frac{a^2}{2} + \frac{b^2}{2} + O\left(x^2\right) = \frac{b^2 - a^2}{2} + O\left(x^2\right) \quad \Rightarrow$$

$$\int_0^{anywhere} \frac{\cos(ax) - \cos(bx)}{x^2}\,dx \sim \frac{b^2 - a^2}{2} \qquad \text{which is to say, is finite.}$$

2. The integrand is even, so if we choose a contour that includes the whole real line, the contour integral includes twice the integral we seek (twice *I*).

Perhaps the most common integration contour (below left) covers the real line, and an infinitely distant arc from +∞ back to –∞. When our real integral (*I* in this case) is only from 0 to ∞, the contour integral includes more than we want on the real axis. If our integrand is even, the contour integral includes twice the integral we seek (twice *I*). This may seem trivial, but the point to notice is that when integrating from –∞ to 0, *dx* is still positive (below middle).

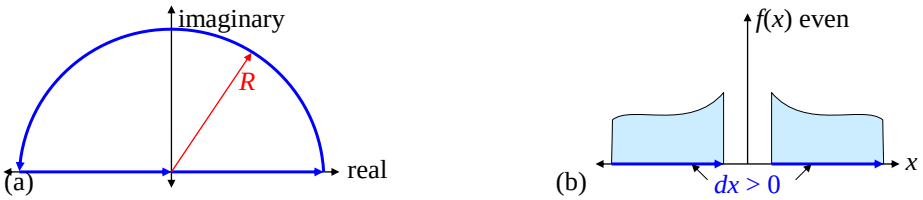


Figure 5.4 (a) A common contour. (b) An even function has integral over the real-line twice that of 0 to infinity.

Note that if the integrand is odd (below left), choosing this contour cancels out the original (real) integral from our contour integral, and the contour is of no use. Or if the integrand has no even/odd symmetry (below middle), then this contour tells us nothing about our desired integral. In these cases, a different contour may work, for example, one which only includes the positive real axis (below right).

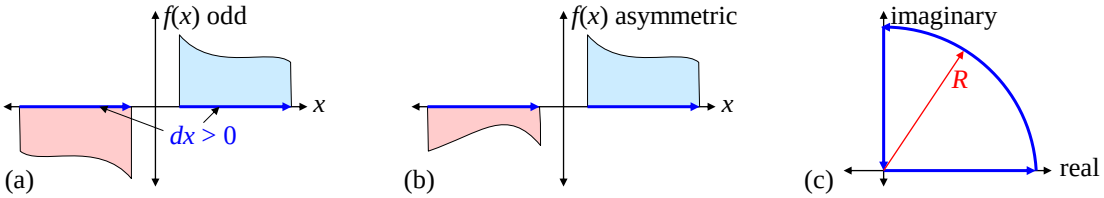


Figure 5.5 (a) An odd function has zero integral over the real line. (b) An asymmetric function has unknown integral over the real line. (c) A contour containing only the desired real integral.

3. The integrand has no poles. How can we use any residue theorems if there are no poles? Amazingly, we can create a useful pole.

This is the funkiest aspect of this problem, but illustrates a standard tool. We are given a real-valued integral with no poles. Contour integration is usually useless without a pole, and a residue, to help us evaluate the contour integral. Our integrand contains cos(*x*), and that is related to exp(*ix*). We could try replacing cosines with exponentials,

$$\cos z = \frac{\exp(iz) + \exp(-iz)}{2} \qquad \text{(does no good)}.$$

but this only rearranges the algebra; fundamentally, it buys us nothing. The trick here is to notice that we can often add a made-up imaginary term to our original integrand, perform a contour integration, and then simply take the real part of our result:

$$\textit{Given} \quad I = \int_a^b g(x)\,dx, \quad \textit{let} \qquad f(z) \equiv g(z) + ih(z). \qquad \textit{Then} \quad I = \mathrm{Re}\left\{\int_a^b f(z)\,dz\right\}.$$

For this trick to work, *ih*(*z*) must have no real-valued contribution over the contour we choose, so it doesn't mess up the integral we seek. Often, we satisfy this requirement by choosing *ih*(*z*) to be purely imaginary on the real axis, and having zero contribution elsewhere on the contour. Given an integrand containing cos(*x*), as in our example, a natural choice for *ih*(*z*) is *i* sin(*z*), because then we can write the new integrand as a simple exponential:

$$\cos(x) \rightarrow f(z) = \cos(z) + i\sin(z) = \exp(iz).$$

In our example, the corresponding substitution yields

$$I = \int_0^\infty \frac{\cos ax - \cos bx}{x^2}\,dx \;\rightarrow \qquad I = \mathrm{Re}\left\{\int_0^\infty \frac{\exp(iax) - \exp(ibx)}{x^2}\,dx\right\}.$$

Examining this substitution more closely, we find a wonderful consequence: this substitution introduced a pole! Recall that

$$\sin z = z - \frac{z^3}{3!} + \ldots \qquad \Rightarrow \qquad \frac{i \sin z}{z^2} = i\left(\frac{1}{z} - \frac{z}{3!} + \ldots\right).$$

We now have a simple pole at $z = 0$, with residue i.

> By choosing to add an imaginary term to the integrand,
> we now have a pole that we can work with to evaluate a contour integral!

It's like magic. In our example integral, our residue is:

$$\frac{i \sin az - i \sin bz}{z^2} = i\left(\frac{a-b}{z} + \ldots\right), \qquad \textit{and} \qquad \textit{residue} = i(a-b).$$

Note that if our original integrand contained $\sin(x)$ instead of $\cos(x)$, we would have made a similar substitution, but taken the *imaginary* part of the result:

$$\textit{Given} \quad I = \int_a^b \sin(x)\,dx, \quad \textit{let} \qquad f(z) \equiv \cos(z) + i\sin(z). \quad \textit{Then} \qquad I = \operatorname{Im}\left\{\int_a^b f(z)\,dz\right\}.$$

4. **A typical contour includes an arc at infinity, but cos(*z*) is ill-behaved for *z* far off the real-axis. How can we tame it?**

This is related to the previous funkiness. We're used to thinking of $\cos(x)$ as a nice, bounded, well-behaved function, but this is only true when x is real.

> When integrating cos(*z*) over a contour,
> we must remember that cos(*z*) blows up rapidly off the real axis.

In fact, $\cos(z) \sim \exp(\operatorname{Im}\{z\})$, so it blows up extremely quickly off the real axis. If we're going to evaluate a contour integral with $\cos(z)$ in it, we must cancel its divergence off the real axis. There is only one function which can exactly cancel the divergence of $\cos(z)$, and that is $\pm i \sin(z)$. The plus sign cancels the divergence *above* the real axis; the minus sign cancels it *below*. There is nothing that cancels it everywhere. We show this cancellation simply:

$$\textit{Let} \qquad z \equiv x + iy$$

$$\cos z + i \sin z = \exp(iz) = \exp\big(i(x+iy)\big) = \exp(ix)\exp(-y) \qquad \textit{and}$$

$$\big|\exp(ix)\exp(-y)\big| = \big|\exp(ix)\big| \cdot \big|\exp(-y)\big| = \exp(-y)$$

For z above the real axis, this shrinks rapidly. Recall that in the previous step, we added $i \sin(x)$ to our integrand to give us a pole to work with. We see now that we also need the *same* additional term to tame the divergence of $\cos(z)$ off the real axis. For the contour we've chosen, no other term will work.

5. **We will see that this integral leads to the indented contour theorem, which can only be applied to simple poles, i.e., first order poles (unlike the residue theorem, which applies to *all* poles).**

We're now at the final step. We have a pole at $z = 0$, but it is *right on* our contour, not inside it. If the pole were inside the contour, we would use the residue theorem to evaluate the contour integral, and from there, we'd find the integral on the real axis, cut it in half, and take the real part. That is the integral we seek.

But the pole is not inside the contour; it is *on* the contour. The indented contour theorem allows us to work with poles on the contour. We explain the theorem geometrically in the next section, but state it briefly here:

> Indented contour theorem: For a simple pole, the integral of an arc of tiny radius around the pole,
> of angle θ, equals $(i\theta)$(residue). See diagram below.

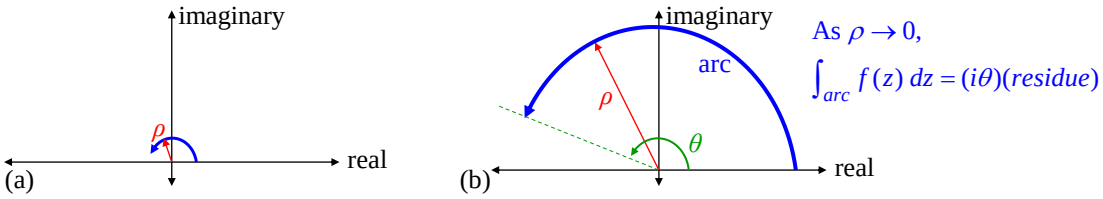


Figure 5.6 (a) A tiny arc around a simple pole. (b) A magnified view; we let $\rho \to 0$.

Note that if we encircle the pole completely, $\theta = 2\pi$, and we have the special case of the residue theorem for a simple pole:

$$\oint f(z)\,dz = 2\pi i\,(residue)\,.$$

However, the residue theorem is true for *all* poles, not just simple ones (see The Residue Theorem earlier).

Putting it all together: We now solve the original integral using all of the above methods. First, we add $i\sin(z)$ to the integrand, which is equivalent to replacing $\cos(z)$ with $\exp(iz)$:

$$I = \int_0^\infty \frac{\cos ax - \cos bx}{x^2}\,dx \quad \to \qquad I = \mathrm{Re}\left\{\int_0^\infty \frac{\exp(iax) - \exp(ibx)}{x^2}\,dx\right\}$$

$$Define \quad J \equiv \int_0^\infty \frac{\exp(iax) - \exp(ibx)}{x^2}\,dx, \qquad so \qquad I = \mathrm{Re}\{J\}$$

We choose the contour Figure 5.7a, with $R \to \infty$, and $\rho \to 0$.

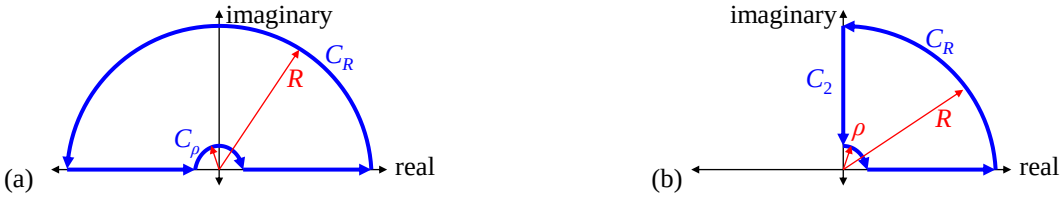


Figure 5.7 (a) A contour for the integral. (b) Alternate contour for the integral.

There are no poles enclosed, so the contour integral is zero. The contour includes twice the desired integral, so define:

$$f(z) \equiv \frac{\exp(iaz) - \exp(ibz)}{z^2}. \qquad \text{Then} \qquad \oint f(z)\,dz = \int_{C_R} f(z)\,dz + 2J + \int_{C_\rho} f(z)\,dz = 0\,. \quad (5.1)$$

For C_R, $|f(z)| < 1/R^2$, so as $R \to \infty$, the integral goes to 0. For C_ρ, the residue is $i(a - b)$, and the arc is π radians in the negative direction, so the indented contour theorem says:

$$\lim_{\rho \to 0} \int_{C_\rho} f(z)\,dz = -(\pi i)\,i\,(a-b) = \pi\,(a-b)\,.$$

Plugging into (5.1), we finally get

$$2J + \pi\,(a-b) = 0 \qquad \Rightarrow \qquad I = \mathrm{Re}\{J\} = \frac{\pi}{2}(b-a)\,.$$

In this example, the contour integral J happened to be real, so taking $I = \mathrm{Re}\{J\}$ is trivial, but in general, there's no reason why J must be real. It could well be complex, and we would need to take the real part of it.

To illustrate this and more, we evaluate the integral again, now with the alternate contour Figure 5.7b. Again, there are no poles enclosed, so the contour integral is zero. Again, the integral over $C_R = 0$. We then have:

$$\oint f(z)\,dz = \cancel{\int_{C_R} f(z)\,dz} + \int_{C_2} f(z)\,dz + J + \int_{C_\rho} f(z)\,dz = 0\,.$$

And $\lim_{\rho\to 0} \int_{C_\rho} f(z)\,dz = -(i\pi/2)i(a-b) = \frac{\pi}{2}(a-b)\,.$

The integral over C_2 is down the imaginary axis:

Let $z = x + iy = 0 + iy = iy$, then $dz = i\,dy$, and

$$\int_{C_2} f(z)\,dz = \int_{C_2} \frac{\exp(iaz) - \exp(ibz)}{z^2}\,dz = \int_{\infty}^{0} \frac{\exp(-ay) - \exp(-by)}{-y^2}\,i\,dy\,.$$

We don't know what this integral is, but *we don't care*! In fact, it is divergent, but we see that it is purely imaginary, so will contribute only to the imaginary part of J. But we seek $I = \text{Re}\{J\}$, and therefore:

$$I = \lim_{\rho\to 0} \text{Re}\{J\} \quad \text{is well-defined.}$$

Therefore we ignore the divergent imaginary contribution from C_2. We then have:

$$i(\textit{something}) + J + \frac{\pi}{2}(a-b) = 0 \quad \Rightarrow \quad I = \text{Re}\{J\} = \frac{\pi}{2}(b-a)\,,$$

as before.

Evaluating Infinite Sums

Perhaps the simplest infinite sum in the world is $S = \sum_{n=1}^{\infty} \frac{1}{n^2}$. The general method for using contour integrals is to find a countably infinite set of residues whose values are the terms of the sum, and whose contour integral can be evaluated by other means. Then:

$$I_C = 2\pi i \sum_{n=1}^{\infty} \text{Res}\, f(z_n) = 2\pi i S \quad \Rightarrow \quad S = \frac{I_C}{2\pi i}\,.$$

The hard part is finding the function $f(z)$ that has the right residues. Such a function must first have poles at all the integers, and then also have residues at those poles equal to the terms of the series.

To find such a function, consider the complex function $\pi\cot(\pi z)$. Clearly, this has poles at all real integer z, due to the $\sin(\pi z)$ function in the denominator of $\cot(z)$. Hence:

$$\text{For } z_n = n \text{ (integer)}, \quad \text{Res}\left[\pi\cot(z_n)\right] = \text{Res}\left[\pi\frac{\cos(\pi z_n)}{\sin(\pi z_n)}\right] = \pi\frac{\cos(\pi z_n)}{\pi\cos(\pi z_n)} = 1\,,$$

where in the last step we used: if $Q(z) = 0$, then $\underset{z=z_0}{\text{Res}} \frac{P(z)}{Q(z)} = \frac{P(z)}{Q'(z_0)}$, if this is defined.

Thus $\pi\cot(\pi z)$ can be used to generate lots of infinite sums, by simply multiplying it by a continuous function of z that equals the terms of the infinite series when z is integer. For example, for the sum above, $S = \sum_{n=1}^{\infty} \frac{1}{n^2}$, we simply define:

$$f(z) = \frac{1}{z^2}\pi\cot(\pi z), \quad \text{and its residues are} \quad \text{Res}\, f(z_n) = \frac{1}{n^2},\ n \neq 0\,.$$

[In general, to find $\sum_{n=1}^{\infty} s(n)$, define:

$$f(z) = s(z)\left[\pi\cot(\pi z)\right], \quad \text{and its residues are} \qquad \operatorname*{Res}_{z=n} f(z) = s(n) .$$

Depending on your contour, you may now have to deal with the residues for n ≤ 0.]

Continuing our example, we might use the contour of Figure 5.8. Then we need the residue at $n = 0$. Since cot(z) has a simple pole at zero, cot(z)/z^2 has a 3rd order pole at zero. We optimistically try tedious brute force for an m^{th} order pole with $m = 3$, only to find that it fails:

$$\operatorname*{Res}_{z=0}\frac{\pi\cot\pi z}{z^2} = \lim_{z\to 0}\left[\frac{1}{2!}\frac{d^2}{dz^2}z^3\frac{\pi\cot\pi z}{z^2}\right] = \lim_{z\to 0}\left[\frac{1}{2!}\frac{d^2}{dz^2}\pi z\cot\pi z\right]$$

$$= \frac{\pi}{2}\lim_{z\to 0}\frac{d}{dz}\left[\cot\pi z - \pi z\csc^2\pi z\right] = \frac{\pi}{2}\lim_{z\to 0}\frac{d}{dz}\left[\frac{\cos\pi z\sin\pi z - \pi z}{\sin^2\pi z}\right] = \frac{\pi}{2}\lim_{z\to 0}\frac{d}{dz}\left[\frac{\frac{1}{2}\sin 2\pi z - \pi z}{\sin^2\pi z}\right].$$

Use $d\frac{U}{V} = \frac{VdU - UdV}{V^2}$:

$$\operatorname*{Res}_{z=0}\frac{\pi\cot\pi z}{z^2} = \frac{\pi}{2}\lim_{z\to 0}\frac{\sin^2\pi z\left(\pi\cos 2\pi z - \pi\right) - \left(\frac{1}{2}\sin 2\pi z - \pi z\right)2\pi\sin\pi z\cos\pi z}{\sin^4\pi z}$$

$$= \frac{\pi}{2}\lim_{z\to 0}\frac{\sin\pi z\left(\pi\cos 2\pi z - \pi\right) - \left(\frac{1}{2}\sin 2\pi z - \pi z\right)2\pi\cos\pi z}{\sin^3\pi z}$$

Use L'hopital's rule:

$$\operatorname*{Res}_{z=0}\frac{\pi\cot\pi z}{z^2} = \frac{\pi}{2}\lim_{z\to 0}\frac{1}{3\pi\sin^2\pi z\cos\pi z}\Big[\pi\cos\pi z\left(\pi\cos 2\pi z - \pi\right) + \sin\pi z\left(-2\pi\sin 2\pi z - 1\right)$$

$$-\left(\pi\cos 2\pi z - \pi\right)2\pi\cos\pi z - \left(\frac{1}{2}\sin 2\pi z - \pi z\right)2\pi^2\sin\pi z\Big]$$

$$= \frac{\pi}{2}\lim_{z\to 0}\frac{-\pi^2\cos\pi z\left(\cos 2\pi z - 1\right) + \sin\pi z\left(-2\pi\sin 2\pi z - 1\right) - 2\pi^2\left(\frac{1}{2}\sin 2\pi z - \pi z\right)\sin\pi z}{3\pi\sin^2\pi z\cos\pi z}$$

At this point, we give up on brute force, because we see from the denominator that we'll have to use L'Hopital's rule twice more to eliminate the zero there, and the derivatives will get untenably complicated.

But in 2 lines, we can find the a_{-1} term of the Laurent series (about the origin) from the series expansions of *sin* and *cos*. The z^1 coefficient of cot(z) becomes the z^{-1} coefficient of $f(z) = \cot(z)/z^2$:

$$\cot z = \frac{\cos z}{\sin z} = \frac{1 - z^2/2 + \ldots}{z - z^3/6 + \ldots} \approx \left(\frac{1}{z}\right)\frac{1 - z^2/2}{1 - z^2/6} \approx \left(\frac{1}{z}\right)\left(1 - z^2/2\right)\left(1 + z^2/6\right) = \left(\frac{1}{z}\right)\left(1 - z^2/3\right) = \frac{1}{z} - \frac{z}{3}$$

$$\cot\pi z \approx \frac{1}{\pi z} - \frac{\pi z}{3} \qquad \Rightarrow \qquad \operatorname*{Res}_{z=0}\pi\frac{\cot\pi z}{z^2} = -\frac{\pi^2}{3} \equiv K_0 .$$

Now we take a contour integral over a large circle centered at the origin (Figure 5.8), and passing through the real axis between integers (because cot(πz) blows up every integer !).

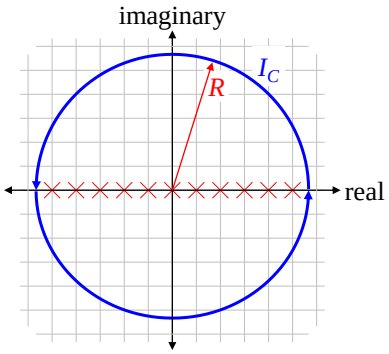


Figure 5.8 A contour for the integral leading to an infinite sum.

As $R \to \infty$, $I_C \to 0$ (why??). Hence:

$$I_C = 0 = 2\pi i\left(\sum_{n=-1}^{-\infty}\frac{1}{n^2} + K_0 + \sum_{n=1}^{\infty}\frac{1}{n^2}\right) \qquad \Rightarrow \qquad K_0 + 2\sum_{n=1}^{\infty}\frac{1}{n^2} = 0, \qquad \sum_{n=1}^{\infty}\frac{1}{n^2} = \frac{-K_0}{2} = \frac{\pi^2}{6}.$$

Multi-valued Functions

Many functions are **multi-valued** (despite the apparent oxymoron), i.e. for a single point z in the domain, the function can have multiple values. A simple example is a square-root "function": given a complex number, there are two complex square roots of it. Thus, the square root function is two-valued. Another example is arc-tangent: given any complex number, there are an infinite number of complex numbers whose tangent is the given complex number.

[picture??]

We refer now to "nice" functions, which are locally (i.e., within a small finite region) analytic, but multi-valued. If you're not careful, such "multi-valuedness" can violate the assumptions of analyticity, by introducing discontinuities in the function. Without analyticity, all our developments break down: no contour integrals, no sums of series. But, you can avoid such a breakdown, and preserve the tools we've developed, by treating multi-valued functions in a slightly special way to insure continuity, and therefore analyticity.

A **regular** function, or region, is analytic and single valued. (You can get a regular function from a multi-valued one by choosing a Riemann sheet. More below.)

A **branch point** is a point in the domain of a function $f(z)$ with this property: when you traverse a closed path around the branch point, following continuous values of $f(z)$, $f(z)$ has a different value at the end point of the path than at the beginning point, even though the beginning and end point are the same point in the domain. Example TBS: square root around the origin. Sometimes branch points are also singularities.

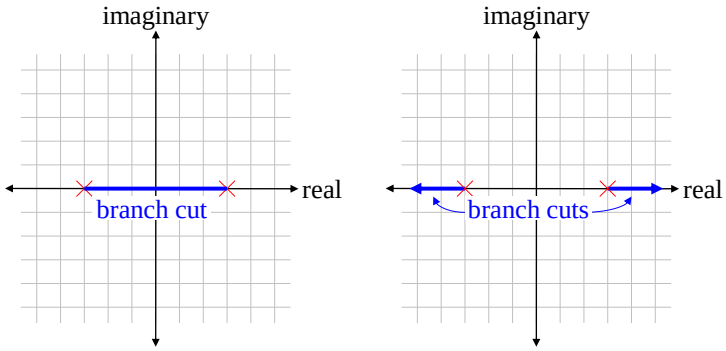
A **branch cut** is an arbitrary (possibly curved) path connecting branch points, or running from a branch point to infinity ("connecting" the branch point to infinity). If you now evaluate integrals of contours that never cross the branch cuts, you insure that the function remains continuous (and thus analytic) over the domain of the integral.

When the contour of integration is entirely in the domain of analyticity of the integrand, "ordinary" contour integration, and the residue theorem, are valid.

This solves the problem of integrating across discontinuities. Branch cuts are like fences in the domain of the function: your contour integral can't cross them. Note that you're free to choose your branch cuts wherever you like, so long as the function remains continuous when you don't cross the branch cuts. Connecting branch points is one way to insure this.

A **Riemann sheet** is the complex plane plus a choice of branch cuts, and a choice of branch. This defines a domain on which a function is regular.

A **Riemann surface** is a continuous joining of Riemann sheets, gluing the edges together. This "looks like" sheets layered on top of each other, and each sheet represents one of the multiple values a multi-valued analytic function may have. TBS: consider $\sqrt{(z-a)(z-b)}$.



Laplace Transforms: Joseph and Pierre-Simon

Laplace transforms can help solve integro-differential equations, similarly to Fourier Transforms. They arise frequently in control systems analysis, and many other scientific and mathematical areas. (In Statistical Mechanics, the partition function is the Laplace transform of the density of states; see *Statistical Mechanifesto.*) The Fourier Transform (FT) of $f(t)$ has a fairly straightforward interpretation (or "physical" description) as the sinusoidal components that sum to $f(t)$. However, the Laplace Transform (LT) is a little more subtle. The "meaning" of the LT is rarely discussed, and it is often presented as a purely mathematical device. However, the LT is closely related to the FT, and also has a conceptual interpretation.

This section assumes you are familiar with the basics of the FT, and its use in solving differential equations.

The big picture: We show below that one simple way to understand the LT is this: it is a trick to make the Fourier Transform work on a broader range of functions, such as for "infinite-energy" functions. The Laplace Transform of $f(t)$ can be thought of as a Fourier Transform of $f(t)$ times a decreasing exponential:

$$L(s) \equiv LT\{f(t)\} = \underbrace{FT\left\{f(t)e^{-\sigma t}\right\}}_{\text{except for limits of integration}} = \int_0^\infty f(t)e^{-\sigma t}e^{-i\omega t}\,dt \qquad \text{where} \quad s \equiv \sigma + i\omega .$$

Details: As a brief review, and to establish notation and conventions, we define the Fourier Transform as most engineers do: given a function $f(t)$, its FT $F(\omega)$ satisfies:

$$f(t) = \int_{-\infty}^{\infty} F(\omega)e^{+i\omega t}\,d\omega . \tag{5.2}$$

In other words, we represent $f(t)$ as an infinite sum of sinusoidal components. $F(\omega)$ is generally complex. This expression explicitly defines the **inverse Fourier Transform** ($F(\omega) \rightarrow f(t)$), and *implicitly* defines the FT ($f(t) \rightarrow F(\omega)$). It is readily shown that this definition demands that $F(\omega)$ is given explicitly by:

$$F(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(t)e^{-i\omega t}\,dt . \tag{5.3}$$

(NB: Some references use different conventions for the sign of the exponential, and the factors of 2π).

A major property of the FT is that it converts a differential equation into an algebraic one, through the integral and differential identities (dropping some subtleties):

$$FT\left\{\frac{d}{dt}f(t)\right\} = i\omega F(\omega), \qquad FT\left\{\int_{-\infty}^{t} f(t')\,dt'\right\} = \frac{1}{i\omega}F(\omega) . \tag{5.4}$$

We can see these identities quickly from (5.2), by differentiating both sides:

$$\frac{d}{dt} f(t) = \frac{d}{dt}\int_{-\infty}^{\infty} F(\omega) e^{+i\omega t}\, d\omega = \int_{-\infty}^{\infty} i\omega F(\omega) e^{+i\omega t}\, d\omega \qquad \Rightarrow \qquad FT\left\{ f'(t) \right\} = i\omega F(\omega)\,.$$

The third expression above is the definition of the inverse Fourier Transform (5.2). (Again: ignoring some complication for now.) For integration: if differentiating $f(t)$ corresponds to multiplying $F(\omega)$ by $i\omega$, then integrating $f(t)$ corresponds to dividing $F(\omega)$ by $i\omega$.

One major limitation on Fourier Transforms is that the Transform (5.2), (5.3) must exist.

This is guaranteed if $f(t)$ is absolutely integrable [ref??]:

$$\int_{-\infty}^{\infty} \left| f(t) \right| dt < \infty\,,$$

and sometimes even if not [ref??]. But in some applications, this is a crippling restriction. For example, in control theory, a crucial aspect of a system is its step-response: how does the system respond to a (Heavyside) step input:

$$\begin{aligned} f(t) &= 0, && t < 0; \\ &= 1, && t > 0\,. \end{aligned}$$

For example, you change the set-point of a temperature control system. How does the system arrive at the new set-point? The input to this system is a step-function (step change in set-point). But this function has no (simple) Fourier Transform.

A more-divergent example: the system's "ramp response" is also often important, e.g., how does a telescope track a moving star across the sky? The input (position) to the tracking system is $f(t) = vt$; this is not only non-Fourier-integrable, it isn't even bounded.

The Laplace Transform (LT) gives us a simple way to handle such cases. To develop it, we must extend Fourier-like methods to non-transformable functions. So we must first consider how the benefits of existing FTs derive. The whole benefit of converting differential equations into algebraic equations comes from these properties:

- the derivative and integral rules (5.4);
- linearity;
- the convolution rule.

But notice that these properties do *not* depend on the specific factor $i\omega$ in the exponential, nor on the limits of integration; their validity derives entirely from writing a function $f(t)$ as a superposition of exponentials. If we can write:

$$f(t) = \int_a^b F(s) e^{+st}\, ds\,, \tag{5.5}$$

for some limits a and b, and for some set of s, then the derivative and integral rules (5.4), and the two other properties, follow. Furthermore, s need not be real. If s is complex, then the integral can even be a contour integral over a specified path. Indeed, in the FT, s is replaced by $i\omega$.

Let us write a complex s in rectangular components: $s = \sigma + i\omega$. Then our Fourier Transform (5.3) becomes:

$$F(s) = \int_a^b f(t) e^{-\sigma t} e^{-i\omega t}\, ds = FT\left\{ f(t) e^{-\sigma t} \right\} \qquad \text{(except maybe the limits of integration)}\,. \tag{5.6}$$

(We will use the term "Fourier Transform" for any Fourier integral like (5.3), regardless of the limits of integration.) This is the Laplace Transform: the FT of a modified time-domain function: $f(t) \to f(t) e^{-\sigma t}$.

We can always choose σ constant and large enough to make a step function, or a ramp, or any polynomial, and most other practical functions, into a well-behaved, convergent function. One *with* a Fourier Transform.

It is most common to define the Laplace Transform with limits from 0 to ∞:

$$F(s) \equiv \int_0^\infty f(t) e^{-st}\, ds, \qquad s \text{ complex} \qquad \text{(most common definition).}$$

However, there are left-sided, two-sided, and other variations which differ only by the limits of integration.

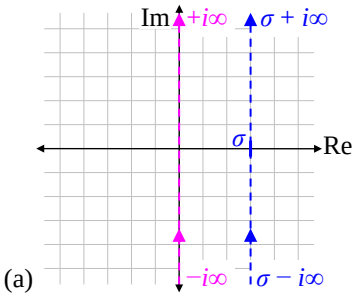
Now we see that the FT (5.3) is a special case of the LT (5.6): that where $\sigma = 0$, and $(a, b) = (-\infty, +\infty)$.

Inverse Laplace Transform: We often don't need an explicit formula for the LT^{-1}, because we get much information from $F(s)$ itself. When we *do* need an explicit inverse, it usually comes from tables and theorems. However, we can find an explicit formula for inversion by inspection of the LT (5.6). Working in reverse, we invert the LT by (1) taking the inverse FT to recover the modified time-domain function, and then (2) removing the modification by multiplying by the inverse exponential:

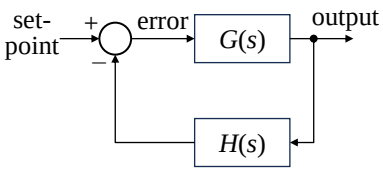
$$f(t) \equiv LT^{-1}\{F(s)\} = FT^{-1}\{F(i\omega)\} e^{+\sigma t} = \int_{-\infty}^{\infty} F(i\omega) e^{+i\omega t}\, d\omega\, e^{+\sigma t} \qquad \text{or}$$

$$f(t) = \int_C F(s) e^{+st}\, ds,$$

where in the last integral, the contour is as shown in Figure 5.9a.



(a)



(b)

Figure 5.9 (a) Integration contours for the inverse Laplace Transform (blue), and the inverse Fourier Transform (magenta). (b) Typical feedback-control system.

6 Conceptual Linear Algebra

Instead of lots of summation signs, we describe linear algebra concepts, visualizations, and ways to think about linear operations as algebraic operations. This allows fast understanding of linear algebra methods that is extremely helpful in almost all areas of physics. Tensors rely heavily on linear algebra methods, so this section is a good warm-up for tensors. Matrices and linear algebra are also critical for quantum mechanics.

Caution In this section, **vector** means a column or row of numbers. In other sections, "vector" has a more general meaning.

In this section, we use bold capitals for matrices (**A**), and bold lower-case for vectors (**a**).

Matrix Multiplication

It is often helpful to view a matrix as a horizontal concatenation of column-vectors. You can think of it as a row-vector, where each element of the row-vector is itself a column vector.

$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{array}\right] \qquad \textit{or} \qquad \mathbf{A} = \left[\begin{array}{c} \mathbf{d} \\ \hline \mathbf{e} \\ \hline \mathbf{f} \end{array}\right].$$

Equally valid, you can think of a matrix as a vertical concatenation of row-vectors, like a column-vector where each element is itself a row-vector.

Matrix multiplication is defined to be the operation of linear transformation, e.g., from one set of coordinates to another. The following properties follow from the standard definition of matrix multiplication:

Matrix times a vector: A matrix **B** times a column vector **v**, is a weighted sum of the columns of **B:**

$$\mathbf{Bv} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} \equiv v^x \begin{bmatrix} B_{11} \\ B_{21} \\ B_{31} \end{bmatrix} + v^y \begin{bmatrix} B_{12} \\ B_{22} \\ B_{32} \end{bmatrix} + v^z \begin{bmatrix} B_{13} \\ B_{23} \\ B_{33} \end{bmatrix}$$

We can visualize this by laying the vector on its side above the columns of the matrix, multiplying each matrix-column by the vector component, and summing the resulting vectors:

$$\mathbf{Bv} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} = \left[\begin{array}{c} v^x \\ \times \\ B_{11} \\ B_{21} \\ B_{31} \end{array}\middle|\begin{array}{c} v^y \\ \times \\ B_{12} \\ B_{22} \\ B_{32} \end{array}\middle|\begin{array}{c} v^z \\ \times \\ B_{13} \\ B_{23} \\ B_{33} \end{array}\right] \begin{bmatrix} \\ \\ \\ \end{bmatrix} = v^x \begin{bmatrix} B_{11} \\ B_{21} \\ B_{31} \end{bmatrix} + v^y \begin{bmatrix} B_{12} \\ B_{22} \\ B_{32} \end{bmatrix} + v^z \begin{bmatrix} B_{13} \\ B_{23} \\ B_{33} \end{bmatrix}$$

The columns of **B** are the vectors which are weighted by each of the input vector components, v^j.

Another important way of conceptualizing a matrix times a vector: the resultant vector is a column of dot products. The i^{th} element of the result is the dot product of the given vector, **v**, with the i^{th} row of **B**. Writing **B** as a column of row-vectors:

$$\mathbf{B} = \begin{bmatrix} \mathbf{r}_1 \\ \hdashline \mathbf{r}_2 \\ \hdashline \mathbf{r}_3 \end{bmatrix} \quad \rightarrow \quad \mathbf{Bv} = \begin{bmatrix} \mathbf{r}_1 \\ \hdashline \mathbf{r}_2 \\ \hdashline \mathbf{r}_3 \end{bmatrix} \begin{bmatrix} \\ \mathbf{v} \\ \\ \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \bullet \mathbf{v} \\ \mathbf{r}_2 \bullet \mathbf{v} \\ \mathbf{r}_3 \bullet \mathbf{v} \end{bmatrix}.$$

This view derives from the one above, where we lay the vector on its side above the matrix, but now consider the effect on each row separately: it is exactly that of a dot product.

In linear algebra, even if the matrices are complex, we do not conjugate the left vector in these dot products. If they need conjugation, the application must conjugate them separately from the matrix multiplication, i.e. during the construction of the matrix.

We use this dot product concept later when we consider a change of basis.

Matrix times a matrix: Multiplying a matrix **B** times another matrix **C** is *defined* as multiplying each column of **C** by the matrix **B**. Therefore, *by definition*, matrix multiplication distributes to the right across the columns:

$$\textit{Let} \qquad \mathbf{C} = \left[\begin{array}{c:c:c} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{array}\right], \quad \textit{then} \qquad \mathbf{BC} = \mathbf{B}\left[\begin{array}{c:c:c} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{array}\right] \equiv \left[\begin{array}{c:c:c} \mathbf{Bx} & \mathbf{By} & \mathbf{Bz} \end{array}\right].$$

[Matrix multiplication also distributes to the left across the rows, but we don't use that as much.]

Determinants

This section assumes you've seen matrices and determinants, but probably didn't understand the reasons why they work.

The determinant operation on a matrix produces a scalar. It is the only operation (up to a constant factor) which is (1) linear in each row and each column of the matrix; and (2) antisymmetric under exchange of any two rows or any two columns.

The above two rules, linearity and antisymmetry, allow determinants to help solve simultaneous linear equations, as we show later under "Cramer's Rule." In more detail:

1. The determinant is linear in each column-vector (and row-vector). This means that multiplying any column (or row) by a scalar multiplies the determinant by that scalar. E.g.,

$$\det\left|\begin{array}{c:c:c} k\mathbf{a} & \mathbf{b} & \mathbf{c} \end{array}\right| = k\det\left|\begin{array}{c:c:c} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{array}\right|; \quad \textit{and} \quad \det\left|\begin{array}{c:c:c} \mathbf{a}+\mathbf{d} & \mathbf{b} & \mathbf{c} \end{array}\right| = \det\left|\begin{array}{c:c:c} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{array}\right| + \det\left|\begin{array}{c:c:c} \mathbf{d} & \mathbf{b} & \mathbf{c} \end{array}\right|.$$

2. The determinant is anti-symmetric with respect to any two column-vectors (or row-vectors). This means swapping any two columns (or rows) of the matrix negates its determinant.

The above properties of determinants imply some others:

3. Expansion by minors/cofactors (see below), whose derivation proves the determinant operator is unique (up to a constant factor).

4. The determinant of a matrix with any two columns equal (or proportional) is zero. (From anti-symmetry, swap the two equal columns, the determinant must negate, but its negative now equals itself. Hence, the determinant must be zero.)

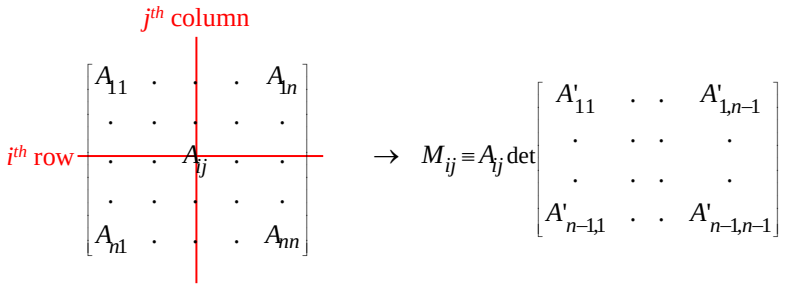
$$\det\left|\begin{array}{c:c:c} \mathbf{b} & \mathbf{b} & \mathbf{c} \end{array}\right| = -\det\left|\begin{array}{c:c:c} \mathbf{b} & \mathbf{b} & \mathbf{c} \end{array}\right| \quad \Rightarrow \quad \det\left|\begin{array}{c:c:c} \mathbf{b} & \mathbf{b} & \mathbf{c} \end{array}\right| = 0.$$

5. $\det|\mathbf{A}|\cdot\det|\mathbf{B}| = \det|\mathbf{AB}|$. This is crucially important. It also fixes the overall constant factor of the determinant, so that the determinant (with this property) is a completely unique operator.
6. Adding a multiple of any column (row) to any other column (row) does not change the determinant:

$$\det|\mathbf{a}+k\mathbf{b}\ \ \mathbf{b}\ \ \mathbf{c}| = \det|\mathbf{a}\ \ \mathbf{b}\ \ \mathbf{c}| + \det|k\mathbf{b}\ \ \mathbf{b}\ \ \mathbf{c}| = \det|\mathbf{a}\ \ \mathbf{b}\ \ \mathbf{c}| + k\det|\mathbf{b}\ \ \mathbf{b}\ \ \mathbf{c}| = \det|\mathbf{a}\ \ \mathbf{b}\ \ \mathbf{c}|.$$

7. $\det|\mathbf{A}+\mathbf{B}| \neq \det|\mathbf{A}| + \det|\mathbf{B}|$. The determinant operator is *not* distributive over matrix addition.
8. $\det|k\mathbf{A}| = k^n \det|\mathbf{A}|$.

The ij-th **minor**, M_{ij}, of an $n\times n$ matrix ($\mathbf{A} \equiv A_{ab}$) is the product A_{ij} times the determinant of the $(n-1)\times(n-1)$ matrix formed by crossing out the i-th row and j-th column:



A **cofactor** is just a minor with a plus or minus sign affixed:

$$C_{ij} = (-1)^{i+j} M_{ij} = (-1)^{i+j} A_{ij} \det\Big[\big[\mathbf{A}\big] \text{ without } i^{th} \text{ row and } j^{th} \text{ column}\Big].$$

Cramer's Rule

It's amazing how many textbooks describe Cramer's rule, and how few explain or derive it. I spent years looking for this, and finally found it in [Arf ch 3]. Cramer's rule is a turnkey method for solving simultaneous linear equations. It is horribly inefficient, and virtually worthless above 3 × 3, however, it does have important theoretical implications. Cramer's rule solves for n equations in n unknowns:

Given $\mathbf{Ax} = \mathbf{b}$, *where* $\mathbf{A}$ is a coefficient matrix,
$\mathbf{x}$ is a vector of unknowns, x_i
$\mathbf{b}$ is a vector of constants, b_i

To solve for the i^{th} unknown x_i, we replace the i^{th} column of $\mathbf{A}$ with the constant vector $\mathbf{b}$, take the determinant, and divide by the determinant of $\mathbf{A}$. Mathematically:

Let $\mathbf{A} = \big[\mathbf{a}_1 \ \big|\ \mathbf{a}_2 \ \big|\ \cdots \ \big|\ \mathbf{a}_n\big]$ *where* $\mathbf{a}_i$ is the i^{th} column of $\mathbf{A}$. We can solve for x_i as

$$x_i = \frac{\det|\mathbf{a}_1 \ \ldots \mathbf{a}_{i-1}\ \ \mathbf{b}\ \ \mathbf{a}_{i+1} \ldots\ \ \mathbf{a}_n|}{\det|\mathbf{A}|} \qquad \textit{where} \quad \mathbf{a}_i \text{ is the } i^{th} \text{ column of } \mathbf{A}$$

This seems pretty bizarre, and one has to ask, why does this work? It's quite simple, if we recall the properties of determinants. Let's solve for x_1, noting that all other unknowns can be solved analogously. Start by simply multiplying x_1 by $\det|\mathbf{A}|$:

$$x_1 \det|\mathbf{A}| = \det\left|\, x_1\mathbf{a}_1 \;\vdots\; \mathbf{a}_2 \;\vdots\; \ldots \;\vdots\; \mathbf{a}_n \,\right|$$

from linearity of det[ ]

$$= \det\left|\, x_1\mathbf{a}_1 + x_2\mathbf{a}_2 \;\vdots\; \mathbf{a}_2 \;\vdots\; \ldots \;\vdots\; \mathbf{a}_n \,\right|$$

adding a multiple of any column to another doesn't change the determinant

$$= \det\left|\, x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \ldots x_n\mathbf{a}_n \;\vdots\; \mathbf{a}_2 \;\vdots\; \ldots \;\vdots\; \mathbf{a}_n \,\right|$$

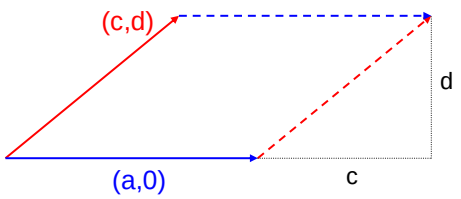
ditto $(n-2)$ more times

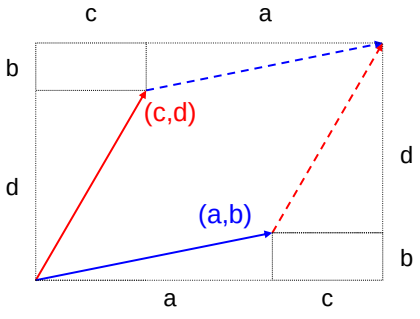
$$= \det\left|\, \mathbf{Ax} \;\vdots\; \mathbf{a}_2 \;\vdots\; \ldots \;\vdots\; \mathbf{a}_n \,\right| = \det\left|\, \mathbf{b} \;\vdots\; \mathbf{a}_2 \;\vdots\; \ldots \;\vdots\; \mathbf{a}_n \,\right|$$

rewriting the first column

$$\Rightarrow \qquad x_1 = \frac{\det\left|\, \mathbf{b} \;\vdots\; \mathbf{a}_2 \;\vdots\; \ldots \;\vdots\; \mathbf{a}_n \,\right|}{\det|\mathbf{A}|}\,.$$

Area and Volume as a Determinant





Determining areas of regions defined by vectors is crucial to geometric physics in many areas. It is the essence of the Jacobian matrix used in variable transformations of multiple integrals. What is the area of the parallelogram defined by two vectors? This is the archetypal area for generalized (oblique, non-normal) coordinates. We will proceed in a series of steps, gradually becoming more general.

First, consider that the first vector is horizontal (above left). The area is simply base × height: $A = ad$. We can obviously write this as a determinant of the matrix of column vectors, though it is as-yet contrived:

$$A = \det\begin{vmatrix} a & c \\ 0 & d \end{vmatrix} = ad - (0)c = ad \ .$$

For a general parallelogram (above right), we can take the big rectangle and subtract the smaller rectangles and triangles, by brute force:

$$A = (a+c)(b+d) - 2bc - 2\left(\frac{1}{2}\right)cd - 2\left(\frac{1}{2}\right)ab = \cancel{ab} + ad + cb + \cancel{cd} - 2bc - \cancel{cd} - \cancel{ab}$$

$$= ad - bc = \det\begin{vmatrix} a & c \\ b & d \end{vmatrix}.$$

This is simple enough in 2-D, but is incomprehensibly complicated in higher dimensions. We can achieve the same result more generally, in a way that allows for extension to higher dimensions by induction. Start again with the diagram above left, where the first vector is horizontal. We can rotate that to arrive at any arbitrary pair of vectors, thus removing the horizontal restriction:

Let **R** = the rotation matrix. Then the rotated vectors are $\mathbf{R}\begin{bmatrix} a \\ 0 \end{bmatrix}$ *and* $\mathbf{R}\begin{bmatrix} c \\ d \end{bmatrix}$

$$\det\left|\mathbf{R}\begin{bmatrix} a \\ 0 \end{bmatrix} \quad \mathbf{R}\begin{bmatrix} c \\ d \end{bmatrix}\right| = \det\left(\mathbf{R}\begin{bmatrix} a & c \\ 0 & d \end{bmatrix}\right) = \cancel{\left(\det|\mathbf{R}|\right)} \det\begin{vmatrix} a & c \\ 0 & d \end{vmatrix} = \det\begin{vmatrix} a & c \\ 0 & d \end{vmatrix}$$

The final equality is because rotation matrices are orthogonal, with det = 1. Thus the determinant of arbitrary vectors defining arbitrary parallelograms equals the determinant of the vectors spanning the parallelogram rotated to have one side horizontal, which equals the area of the parallelogram.

What about the sign? If we reverse the two vectors, the area comes out negative! That's ok, because in differential geometry, 2-D areas are signed: positive if we travel counter-clockwise from the first vector to the 2nd, and negative if we travel clockwise. The above areas are positive.

In 3-D, the **signed volume** of the parallelepiped defined by 3 vectors **a**, **b**, and **c**, is the determinant of the matrix formed by the vectors as columns (positive if **abc** form a right-handed set, negative if **abc** are a left-handed set). We show this with rotation matrices, similar to the 2-D case: First, assume that the parallelogram defined by **bc** lies in the *x*-*y* plane ($b_z = c_z = 0$). Then the volume is simply (area of the base) × height:

$$V = (\textit{area of base})(\textit{height}) = \left(\det\left|\mathbf{b} \;\vdots\; \mathbf{c}\right|\right)(a_z) = \det\begin{vmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & 0 & 0 \end{vmatrix}.$$

where the last equality is from expansion by cofactors along the bottom row. But now, as before, we can rotate such a parallelepiped in 3 dimensions to get any arbitrary parallelepiped. As before, the rotation matrix is orthogonal (det = 1), and does not change the determinant of the matrix of column vectors.

This procedure generalizes to arbitrary dimensions: the signed hyper-volume of a parallelepiped defined by *n* vectors in *n*-D space is the determinant of the matrix of column vectors. The sign is positive if the 3-D submanifold spanned by each contiguous subset of 3 vectors ($\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$, $\mathbf{v}_2\mathbf{v}_3\mathbf{v}_4$, $\mathbf{v}_3\mathbf{v}_4\mathbf{v}_5$, ...) is right-handed, and negated for each subset of 3 vectors that is left-handed.

The Jacobian Determinant and Change of Variables

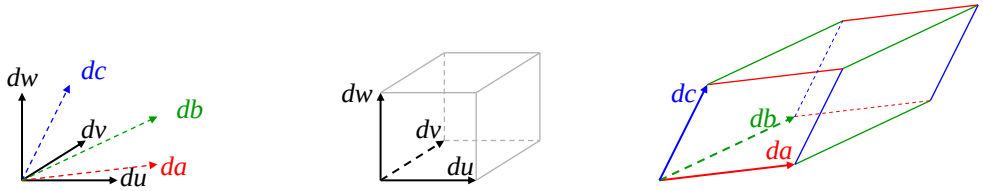
How do we change multiple variables in a multiple integral? Given

$\iiint f(a,b,c)\, da\, db\, dc$ and the change of variables to u, v, w:

$a = a(u,v,w), \quad b = b(u,v,w), \quad c = c(u,v,w).$ The simplistic

$$\iiint f(a,b,c)\, da\, db\, dc \quad \rightarrow \quad \iiint f\left[a(u,v,w), b(u,v,w), c(u,v,w)\right] du\, dv\, dw \quad (wrong!)$$

fails, because the "volume" *du dv dw* associated with each point of $f(\cdot)$ is different than the volume *da db dc* in the original integral.



Example of new-coordinate volume element (*du dv dw*), and its corresponding old-coordinate volume element (*da db dc*). The new volume element is a rectangular parallelepiped. The old-coordinate parallelepiped has sides straight to first order in the original integration variables.

In the diagram above, we see that the "volume" (*du dv dw*) is smaller than the old-coordinate "volume" (*da db dc*). Note that "volume" is a *relative* **measure** of volume in coordinate space; it has nothing to do with a "metric" on the space, and "distance" need not even be defined.

There is a concept of relative "volume" in any space, even if there is no definition of "distance." Relative volume is defined as products of coordinate differentials.

The integrand is constant (to first order in the integration variables) over the whole volume element.

Without some correction, the weighting of $f(\cdot)$ throughout the new-coordinate domain is different than the original integral, and so the integrated sum (i.e., the integral) is different. We correct this by putting in the *original-coordinate* differential volume (*da db dc*) as a function of the *new* differential coordinates, *du*, *dv*, *dw*. Of course, this function varies throughout the domain, so we can write

$$\iiint f(a,b,c)\, da\, db\, dc \quad \rightarrow \quad \iiint f\left[a(u,v,w), b(u,v,w), c(u,v,w)\right] V(u,v,w)\, du\, dv\, dw$$

$$where \quad V(u,v,w)\ takes\ \left(du\, dv\, dw\right) \rightarrow \left(da\, db\, dc\right)$$

To find $V(\cdot)$, consider how the *a*-*b*-*c* space vector $da\hat{\mathbf{a}}$ is created from the new *u*-*v*-*w* space. It has contributions from displacements in all 3 new dimensions, *u*, *v*, and *w*:

$$da\hat{\mathbf{a}} = \left(\frac{\partial a}{\partial u} du + \frac{\partial a}{\partial v} dv + \frac{\partial a}{\partial w} dw\right)\hat{\mathbf{a}}. \qquad Similarly,$$

$$db\hat{\mathbf{b}} = \left(\frac{\partial b}{\partial u} du + \frac{\partial b}{\partial v} dv + \frac{\partial b}{\partial w} dw\right)\hat{\mathbf{b}}$$

$$dc\hat{\mathbf{c}} = \left(\frac{\partial c}{\partial u} du + \frac{\partial c}{\partial v} dv + \frac{\partial c}{\partial w} dw\right)\hat{\mathbf{c}}$$

The volume defined by the 3 vectors $du\hat{\mathbf{u}}, dv\hat{\mathbf{v}}, and\ dw\hat{\mathbf{w}}$ maps to the volume spanned by the corresponding 3 vectors in the original *a*-*b*-*c* space. The *a*-*b*-*c* space volume is given by the determinant of the components of the vectors $d\mathbf{a}$, $d\mathbf{b}$, and $d\mathbf{c}$ (written as rows below, to match equations above):

$$volume = \det\begin{vmatrix} \frac{\partial a}{\partial u}du & \frac{\partial a}{\partial v}dv & \frac{\partial a}{\partial w}dw \\ \frac{\partial b}{\partial u}du & \frac{\partial b}{\partial v}dv & \frac{\partial b}{\partial w}dw \\ \frac{\partial c}{\partial u}du & \frac{\partial c}{\partial v}dv & \frac{\partial c}{\partial w}dw \end{vmatrix} = \det\begin{vmatrix} \frac{\partial a}{\partial u} & \frac{\partial a}{\partial v} & \frac{\partial a}{\partial w} \\ \frac{\partial b}{\partial u} & \frac{\partial b}{\partial v} & \frac{\partial b}{\partial w} \\ \frac{\partial c}{\partial u} & \frac{\partial c}{\partial v} & \frac{\partial c}{\partial w} \end{vmatrix}(du\ dv\ dw).$$

where the last equality follows from linearity of the determinant. Note that all the partial derivatives are functions of *u*, *v*, and *w*. Hence,

$$V(u,v,w) = \det\begin{vmatrix} \frac{\partial a}{\partial u} & \frac{\partial a}{\partial v} & \frac{\partial a}{\partial w} \\ \frac{\partial b}{\partial u} & \frac{\partial b}{\partial v} & \frac{\partial b}{\partial w} \\ \frac{\partial c}{\partial u} & \frac{\partial c}{\partial v} & \frac{\partial c}{\partial w} \end{vmatrix} \equiv J(u,v,w) \quad \left[\text{the Jacobian}\right], \qquad \textit{and}$$

$$\iiint f(a,b,c)\,da\,db\,dc \qquad \rightarrow \qquad \iiint f\left[a(u,v,w),b(u,v,w),c(u,v,w)\right]J(u,v,w)\,du\,dv\,dw$$

QED.

Expansion by Cofactors

Let us construct the determinant operator from its two defining properties: linearity, and antisymmetry. First, we'll define a linear operator, then we'll make it antisymmetric. [This section is optional, though instructive.]

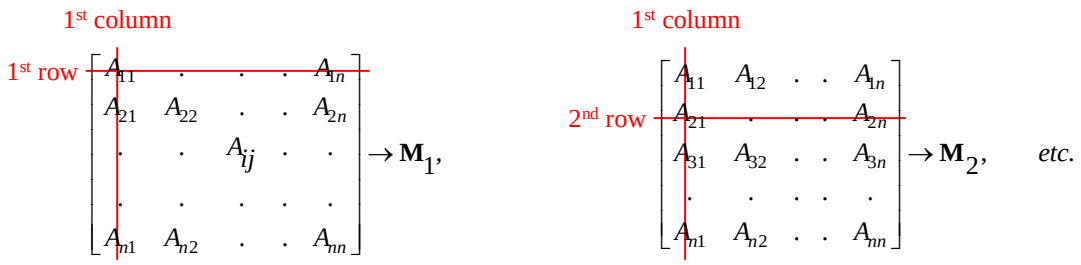
We first construct an operator which is linear in the first column. For the determinant to be linear in the first column, it must be a sum of terms each containing exactly one factor from the first column:

$$\text{Let} \qquad \mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}, \qquad \text{then} \qquad \det\mathbf{A} = A_{11}(\ldots) + A_{21}(\ldots) + \cdots + A_{n1}(\ldots).$$

To be linear in the first column, the parentheses above must have *no* factors from the first column (else they would be quadratic in some components). Now to also be linear in the 2nd column, *all* of the parentheses above must be linear in all the remaining columns. Therefore, to fill in the parentheses we need a linear operator on columns 2...*n*. But that is the same kind of operator we set out to make: a linear operator on columns 1..*n*. Recursion is clearly called for, therefore the parentheses should be filled in with more determinants:

$$\det\mathbf{A} = A_{11}(\det\mathbf{M}_1) + A_{21}(\det\mathbf{M}_2) + \cdots + A_{n1}(\det\mathbf{M}_n) \qquad \text{(so far)}.$$

We now note that the determinant is linear both in the columns, and in the rows. This means that det $\mathbf{M}_1$ must not have any factors from the first row *or* the first column of **A**. Hence, $\mathbf{M}_1$ must be the submatrix of **A** with the first row and first column stricken out.

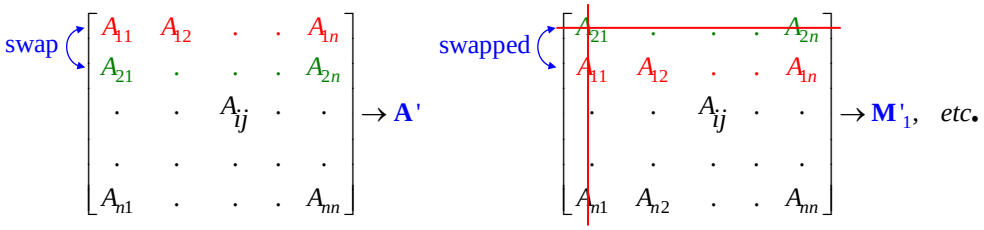


Similarly, $\mathbf{M}_2$ must be the submatrix of $\mathbf{A}$ with the 2nd row and first column stricken out. And so on, through $\mathbf{M}_n$, which must be the submatrix of $\mathbf{A}$ with the n^{th} row and first column stricken out. We now have an operator that is linear in all the rows and columns of $\mathbf{A}$.

So far, this operator is not unique. We could multiply each term in the operator by a constant, and still preserve linearity in all rows and columns:

$$\det \mathbf{A} = k_1 A_{11}\left(\det \mathbf{M}_1\right) + k_2 A_{21}\left(\det \mathbf{M}_2\right) + \cdots + k_n A_{n1}\left(\det \mathbf{M}_n\right).$$

We choose these constants to provide the 2nd property of determinants: antisymmetry. The determinant is antisymmetric on interchange of any two rows. We start by considering swapping the first two rows: Define **A'** ≡ (**A** with $A_{1*} \leftrightarrow A_{2*}$).



Recall that $\mathbf{M}_1$ strikes out the first row, and $\mathbf{M}_2$ strikes out the 2nd row, so swapping row 1 with row 2 replaces the first two terms of the determinant:

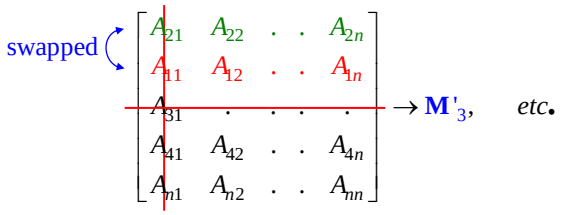
$$\det \mathbf{A} = k_1 A_{11}\left(\det \mathbf{M}_1\right) + k_2 A_{21}\left(\det \mathbf{M}_2\right) + \ldots \quad \rightarrow \quad \det \mathbf{A}' = k_1 A_{21}\left(\det \mathbf{M}'_1\right) + k_2 A_{11}\left(\det \mathbf{M}'_2\right) + \ldots$$

But $\mathbf{M'}_1 = \mathbf{M}_2$, and $\mathbf{M'}_2 = \mathbf{M}_1$. So we have:

$$\rightarrow \quad \det \mathbf{A}' = k_1 A_{21}\left(\det \mathbf{M}_2\right) + k_2 A_{11}\left(\det \mathbf{M}_1\right) + \ldots .$$

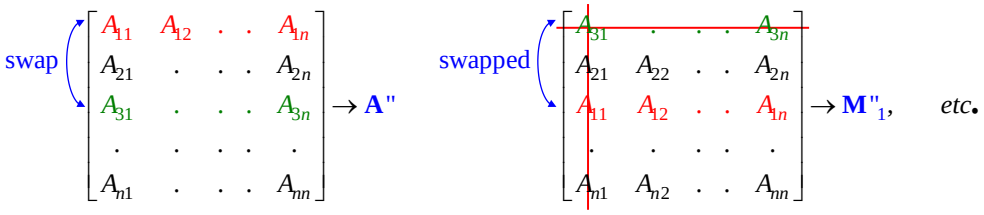
This last form is the same as det **A**, but with k_1 and k_2 swapped. To make our determinant antisymmetric, we must choose constants k_1 and k_2 such that terms 1 and 2 are antisymmetric on interchange of rows 1 and 2. This simply means that $k_1 = -k_2$. So far, the determinant is unique only up to an arbitrary factor, so we choose the simplest such constants: $k_1 = 1$, $k_2 = -1$.

For $\mathbf{M}_3$ through $\mathbf{M}_n$, swapping the first two rows of **A** swaps the first two rows of $\mathbf{M'}_3$ through $\mathbf{M'}_n$:



Since $\mathbf{M}_3$ through $\mathbf{M}_n$ appear inside determinant operators, and such operators are defined to be antisymmetric on interchange of rows, terms 3 through n also change sign on swapping the first two rows of **A**. Thus, all the terms 1 through n change sign on swapping rows 1 and 2, and det **A** = –det **A'**.

We are almost done. We have now a unique determinant operator, with $k_1 = 1$, $k_2 = -1$. We must determine k_3 through k_n. So consider swapping rows 1 and 3 of **A**, which must also negate our determinant:



Again, **M"**$_4$ through **M"**$_n$ have rows 1 & 3 swapped, and thus terms 4 through n are negated by their determinant operators. Also, **M"**$_2$ (formed by striking out row 2 of **A**) has its rows 1 & 2 swapped, and is also thus negated.

The terms remaining to be accounted for are $A_{11}\left(\det \mathbf{M}_1\right)$ and $k_3 A_{31}\left(\det \mathbf{M}_3\right)$. The new **M"**$_1$ is the same as the old $\mathbf{M}_3$, but with its first two rows swapped. Similarly, the new **M"**$_3$ is the same as the old $\mathbf{M}_1$, but with its first two rows swapped. Hence, both terms 1 and 3 are negated by their determinant operators, so we must choose $k_3 = 1$ to preserve that negation.

Finally, proceeding in this way, we can consider swapping rows 1 & 4, etc. We find that the odd numbered k's are all 1, and the even numbered k's are all –1.

We could also have started from the beginning by linearizing with column 2, and then we find that the k are opposite to those for column 1: this time for odd numbered rows, $k_{odd} = -1$, and for even numbered rows, $k_{even} = +1$. The k's simply alternate sign. This leads to the final form of cofactor expansion about any column c:

$$\det \mathbf{A} = (-1)^{1+c} A_{1c}\left(\det \mathbf{M}_1\right) + (-1)^{2+c} A_{2c}\left(\det \mathbf{M}_2\right) + \cdots + (-1)^{n+c} A_{nc}\left(\det \mathbf{M}_n\right).$$

Note that:

We can perform a cofactor expansion down any column, or across any row, to compute the determinant of a matrix.

We usually choose an expansion order which includes as many zeros as possible, to minimize the computations needed.

Proof That the Determinant Is Unique

If we compute the determinant of a matrix two ways, from two different cofactor expansions, do we get the same result? Yes. We here prove the determinant is unique by showing that in a cofactor expansion, every possible combination of elements from the rows and columns appears exactly once. This is true no matter what row or column we expand on. Thus all expansions include the same terms, but just written in a different order.

Also, this complete expansion of *all* combinations of elements is a useful property of the cofactor expansion which has many applications beyond determinants. For example, by performing a cofactor expansion without the alternating signs (in other word, an expansion in *minors*), we can fully symmetrize a set of functions (such as boson wave functions).

The proof: let's count the number of terms in a cofactor expansion of a determinant for an $n \times n$ matrix. We do this by mathematical induction. For the first level of expansion, we choose a row or column, and construct n terms, where each term includes a cofactor (a sub-determinant of an $n-1 \times n-1$ matrix). Thus, the number of terms in an $n \times n$ determinant is n times the number of terms in an $n-1 \times n-1$ determinant. Or, turned around,

$$\#terms\ in\ (n+1 \times n+1) = (n+1)(\#terms\ in\ n \times n).$$

There is one term in a 1×1 determinant, 2 terms in a 2×2, 6 terms in a 3×3, and thus $n!$ terms in an $n \times n$ determinant. Each term is unique within the expansion: by construction, no term appears twice as we work our way through the cofactor expansion.

Let's compare this to the number of terms *possible* which are linear in every row and column: we have n choices for the first factor, $n–1$ choices for the second factor, and so on down to 1 choice for the last factor. That is, there are $n!$ ways to construct terms linear in all the rows and columns. That is exactly the number of terms in the cofactor expansion, which means every cofactor expansion is a sum of *all possible* terms which are linear in the rows and columns. This proves that the determinant is unique up to a sign.

To prove the sign of the cofactor expansion is also unique, we can consider one specific term in the sum. Consider the term which is the product of the main diagonal elements. This term is always positive, since TBS ??

Getting Determined

You may have noticed that computing a determinant by cofactor expansion is computationally infeasible for $n > \sim 15$. There are $n!$ terms of n factors each, requiring $O(n \cdot n!)$ operations. For $n = 15$, this is $\sim 10^{13}$ operations, which would take about a day on a few GHz computer. For $n = 20$, it would take years.

Is there a better way? Fortunately, yes. It can be done in $O(n^3)$ operations, so one can easily compute the determinant for $n = 1000$ or more. We do this by using the fact that adding a multiple of any row to another row does not change the determinant (which follows from anti-symmetry and linearity). Performing such row operations, we can convert the matrix to **upper-right-triangular** form, i.e., all the elements of **A'** below the main diagonal are zero.

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \quad \rightarrow \quad \mathbf{A}' = \begin{bmatrix} A'_{11} & A_{12} & \cdots & A'_{1,n-1} & A'_{1n} \\ 0 & A'_{22} & \cdots & A'_{2,n-1} & A'_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & A'_{n-1,n-1} & A'_{n-1,n} \\ 0 & 0 & \cdots & 0 & A'_{nn} \end{bmatrix}.$$

By construction, $\det|\mathbf{A'}| = \det|\mathbf{A}|$. Using the method of cofactors on **A'**, we expand down the first column of **A'** and first column of every submatrix in the expansion. E.g.,

$$\mathbf{A}' = \begin{bmatrix} A'_{11} & x & x & x \\ 0 & A'_{22} & x & x \\ 0 & 0 & A'_{33} & x \\ 0 & 0 & 0 & A'_{44} \end{bmatrix}$$

Only the first term in each expansion survives, because all the others are zero. Hence, $\det|\mathbf{A'}|$ is the product of its diagonal elements:

$$\det \mathbf{A} = \det \mathbf{A}' = \prod_{i=1}^{n} A'_{ii} \qquad \textit{where} \quad A'_{ii} \text{ are the diagonal elements of } \mathbf{A}'.$$

Let's look at the row operations needed to achieve upper-right-triangular form. We multiply the first row by (A_{21} / A_{11}) and subtract it from the 2nd row. This makes the first element of the 2nd row zero (below left):

$$\mathbf{A} \to \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{1n} \\ 0 & B_{22} & B_{23} & B_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \to \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{1n} \\ 0 & B_{22} & B_{23} & B_{24} \\ 0 & B_{32} & B_{33} & B_{34} \\ 0 & B_{42} & B_{43} & B_{44} \end{bmatrix} \to \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{1n} \\ 0 & B_{22} & B_{23} & B_{24} \\ 0 & 0 & C_{33} & C_{34} \\ 0 & 0 & C_{43} & C_{44} \end{bmatrix}$$

Perform this operation for rows 3 through n, and we have made the first column below row 1 all zero (above middle). Similarly, we can zero the 2nd column below row 2 by multiplying the (new) 2nd row by (B_{32} / B_{22}) and subtracting it from the 3rd row. Perform this again on the 4th row, and we have the first two columns of the upper-right-triangular form (above right). Iterating for the first (n – 1) columns, we complete the upper-right-triangular form. The determinant is now the product of the diagonal elements.

About how many operations did that take? There are $n(n-1)/2$ row-operations needed, or $O(n^2)$. Each row-operation takes from 1 to n multiplies (average $n/2$), and 1 to n additions (average $n/2$), summing to $O(n)$ operations. Total operations is then of order

$$O(n)O\left(n^2\right) \sim O\left(n^3\right).$$

TBS: Proof that det|**AB**| = det|**A**| det|**B**|

Advanced Matrices

Getting to Home Basis

We often wish to change the basis in which we express vectors and matrix operators, e.g. in quantum mechanics. We use a transformation matrix to transform the components of the vectors from the old basis to the new basis. Note that:

> We are not transforming the vectors; we are transforming the components of the vector from one basis to another. The vector itself is unchanged.

There are two ways to visualize the transformation. In the first method, we write the decomposition of a vector into components in matrix form. We use the visualization from above that a matrix times a vector is a weighted sum of the columns of the matrix:

$$\mathbf{v} = \left[\begin{array}{c|c|c} : & : & : \\ \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z \\ : & : & : \end{array}\right] \left[\begin{array}{c} v^x \\ v^y \\ v^z \end{array}\right] = v^x\mathbf{e}_x + v^y\mathbf{e}_y + v^z\mathbf{e}_z$$

This is a vector equation which is true in any basis. In the x-y-z basis, it looks like this:

$$\mathbf{v} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} = \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} \qquad \textit{where} \quad \mathbf{e}_x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{e}_z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

If we wish to convert to the $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ basis, we simply write $\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$ in the 1-2-3 basis:

$$\mathbf{v} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} = \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} \qquad \textit{where} \quad (\text{in the 1-2-3 basis}): \quad \mathbf{e}_x = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad \mathbf{e}_y = \begin{bmatrix} d \\ e \\ f \end{bmatrix}, \quad \mathbf{e}_z = \begin{bmatrix} g \\ h \\ i \end{bmatrix}.$$

Thus:

The columns of the transformation matrix are the old basis vectors written in the new basis.
This is true even for non-ortho-normal bases.

Now let us look at the same transformation matrix, from the viewpoint of its rows. For this, we must restrict ourselves to ortho-normal bases. This is usually not much of a restriction. Recall that the component of a vector **v** in the direction of a basis vector $\mathbf{e}_i$ is given by**:**

$$v^i = \mathbf{e}_i \cdot \mathbf{v} \qquad \Rightarrow \qquad \mathbf{v} = (\mathbf{e}_x \cdot \mathbf{v})\mathbf{e}_x + (\mathbf{e}_y \cdot \mathbf{v})\mathbf{e}_y + (\mathbf{e}_z \cdot \mathbf{v})\mathbf{e}_z \,.$$

But this is a vector equation, valid in any basis. So *i* above could also be 1, 2, or 3 for the new basis:

$$v^1 = \mathbf{e}_1 \cdot \mathbf{v}, \quad v^2 = \mathbf{e}_2 \cdot \mathbf{v}, \quad v^3 = \mathbf{e}_3 \cdot \mathbf{v} \qquad\qquad \mathbf{v} = (\mathbf{e}_1 \cdot \mathbf{v})\mathbf{e}_1 + (\mathbf{e}_2 \cdot \mathbf{v})\mathbf{e}_2 + (\mathbf{e}_3 \cdot \mathbf{v})\mathbf{e}_3 \,.$$

Recall from the section above on matrix multiplication that multiplying a matrix by a vector is equivalent to making a set of dot products, one from each row, with the vector:

$$\left[\begin{array}{c} \mathbf{e}_1 \\ \hdashline \mathbf{e}_2 \\ \hdashline \mathbf{e}_3 \end{array}\right] \left[\mathbf{v}\right] = \left[\begin{array}{c} \mathbf{e}_1 \cdot \mathbf{v} \\ \mathbf{e}_2 \cdot \mathbf{v} \\ \mathbf{e}_3 \cdot \mathbf{v} \end{array}\right] = \left[\begin{array}{c} v^1 \\ v^2 \\ v^3 \end{array}\right] \qquad \textit{or} \qquad \left[\begin{array}{ccc} (\mathbf{e}_1)_x & (\mathbf{e}_1)_y & (\mathbf{e}_1)_z \\ \hdashline (\mathbf{e}_2)_x & (\mathbf{e}_2)_y & (\mathbf{e}_2)_z \\ \hdashline (\mathbf{e}_3)_x & (\mathbf{e}_3)_y & (\mathbf{e}_3)_z \end{array}\right] \left[\begin{array}{c} v^x \\ v^y \\ v^z \end{array}\right] = \left[\begin{array}{c} \mathbf{e}_1 \cdot \mathbf{v} \\ \mathbf{e}_2 \cdot \mathbf{v} \\ \mathbf{e}_3 \cdot \mathbf{v} \end{array}\right] = \left[\begin{array}{c} v^1 \\ v^2 \\ v^3 \end{array}\right].$$

Thus:

The rows of the transformation matrix are the new basis vectors written in the old basis.
This is only true for ortho-normal bases.

There is a beguiling symmetry, and nonsymmetry, in the above two boxed statements about the columns and rows of the transformation matrix.

For complex vectors, we must use the dot product defined with the conjugate of the row basis vector, i.e. the rows of the transformation matrix are the hermitian adjoints of the new basis vectors written in the old basis:

$$\left[\begin{array}{c} \mathbf{e}_1{}^\dagger \\ \hdashline \mathbf{e}_2{}^\dagger \\ \hdashline \mathbf{e}_3{}^\dagger \end{array}\right] \left[\mathbf{v}\right] = \left[\begin{array}{c} \mathbf{e}_1 \cdot \mathbf{v} \\ \mathbf{e}_2 \cdot \mathbf{v} \\ \mathbf{e}_3 \cdot \mathbf{v} \end{array}\right] = \left[\begin{array}{c} v^1 \\ v^2 \\ v^3 \end{array}\right].$$

Diagonalizing a Self-Adjoint Matrix

A special case of basis changing comes up often in quantum mechanics: we wish to change to the basis of eigenvectors of a given operator. In this basis, the basis vectors (which are also eigenvectors) always have the form of a single '1' component, and the rest 0. E.g.,

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

The matrix operator **A**, in this basis (its own eigenbasis), is diagonal, because:

$$\left.\begin{aligned}\mathbf{A}\mathbf{e}_1 &= \lambda_1\mathbf{e}_1\\ \mathbf{A}\mathbf{e}_2 &= \lambda_2\mathbf{e}_2\\ \mathbf{A}\mathbf{e}_3 &= \lambda_3\mathbf{e}_3\end{aligned}\right\} \quad\Rightarrow\quad \mathbf{A} = \begin{bmatrix}\lambda_1 & & \\ & \lambda_2 & \\ & & \lambda_3\end{bmatrix}.$$

Finding the unitary (i.e., unit magnitude) transformation from a given basis to the eigenbasis of an operator is called **diagonalizing the matrix**. We saw above that the transformation matrix from one basis to another is just the hermitian adjoint of the new basis vectors written in the old basis. We call this matrix **U**:

$$\left[\begin{array}{c}\mathbf{e}_1^{\dagger}\\ \hdashline \mathbf{e}_2^{\dagger}\\ \hdashline \mathbf{e}_3^{\dagger}\end{array}\right]\left[\mathbf{v}\right] = \begin{bmatrix}\mathbf{e}_1\cdot\mathbf{v}\\ \mathbf{e}_2\cdot\mathbf{v}\\ \mathbf{e}_3\cdot\mathbf{v}\end{bmatrix} = \begin{bmatrix}v^1\\ v^2\\ v^3\end{bmatrix} \quad\Rightarrow\quad \mathbf{U} = \left[\begin{array}{c}\mathbf{e}_1^{\dagger}\\ \hdashline \mathbf{e}_2^{\dagger}\\ \hdashline \mathbf{e}_3^{\dagger}\end{array}\right].$$

U transforms vectors, but how do we transform the operator matrix **A** itself? The simplest way to see this is to note that we can perform the operation **A** in any basis by transforming the vector back to the original basis, using **A** in the original basis, and then transforming the result to the new basis:

$$\mathbf{v}_{new} = \mathbf{U}\mathbf{v}_{old} \quad\Rightarrow\quad \mathbf{v}_{old} = \mathbf{U}^{-1}\mathbf{v}_{new}$$

$$\mathbf{A}_{new}\mathbf{v}_{new} = \mathbf{U}\left(\mathbf{A}_{old}\mathbf{v}_{old}\right) = \mathbf{U}\left(\mathbf{A}_{old}\mathbf{U}^{-1}\mathbf{v}_{new}\right) = \left(\mathbf{U}\mathbf{A}_{old}\mathbf{U}^{-1}\right)\mathbf{v}_{new} \quad\Rightarrow\quad \mathbf{A}_{new} = \left(\mathbf{U}\mathbf{A}_{old}\mathbf{U}^{-1}\right)$$

where we used the fact that matrix multiplication is associative. Thus:

The unitary transformation that diagonalizes a (complex) self-adjoint matrix is the matrix of normalized eigen-row-vectors.

We can see this another way by starting with:

$$\mathbf{A}\mathbf{U}^{-1} = \mathbf{A}\left[\begin{array}{c|c|c}\mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3\end{array}\right] = \left[\begin{array}{c|c|c}\mathbf{A}\mathbf{e}_1 & \mathbf{A}\mathbf{e}_2 & \mathbf{A}\mathbf{e}_3\end{array}\right] = \left[\begin{array}{c|c|c}\lambda_1\mathbf{e}_1 & \lambda_2\mathbf{e}_2 & \lambda_3\mathbf{e}_3\end{array}\right]$$

where $\mathbf{e}_i$ are the otho-normal eigenvectors

λ_i are the eigenvalues

Recall the eigenvectors (of self-adjoint matrices) are orthogonal, so we can now pre-multiply by the hermitian conjugate of the eigenvector matrix:

$$\mathbf{U}\mathbf{A}\mathbf{U}^{-1} = \left[\begin{array}{c}\mathbf{e}_1^{\dagger}\\ \hdashline \mathbf{e}_2^{\dagger}\\ \hdashline \mathbf{e}_3^{\dagger}\end{array}\right]\mathbf{A}\left[\begin{array}{c|c|c}\mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3\end{array}\right] = \left[\begin{array}{c}\mathbf{e}_1^{\dagger}\\ \hdashline \mathbf{e}_2^{\dagger}\\ \hdashline \mathbf{e}_3^{\dagger}\end{array}\right]\left[\begin{array}{c|c|c}\lambda_1\mathbf{e}_1 & \lambda_2\mathbf{e}_2 & \lambda_3\mathbf{e}_3\end{array}\right]$$

$$= \begin{bmatrix}\lambda_1(\mathbf{e}_1\cdot\mathbf{e}_1) & \lambda_2(\mathbf{e}_1\cancel{\cdot}\mathbf{e}_2) & \cancel{\cdots}\\ \lambda_1(\mathbf{e}_2\cancel{\cdot}\mathbf{e}_1) & \lambda_2(\mathbf{e}_2\cdot\mathbf{e}_2) & \cancel{\cdots}\\ \cancel{\cdots} & \cancel{\cdots} & \lambda_3(\mathbf{e}_3\cdot\mathbf{e}_3)\end{bmatrix} = \begin{bmatrix}\lambda_1 & 0 & 0\\ 0 & \lambda_2 & 0\\ 0 & 0 & \lambda_3\end{bmatrix}$$

where the final equality is because each element of the result is the inner product of two eigenvectors, weighted by an eigenvalue. The only non-zero inner products are between the same eigenvectors

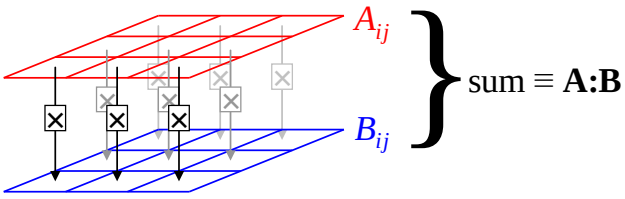
(orthogonality), so only diagonal elements are non-zero. Since the eigenvectors are normalized, their inner product is 1, leaving only the weight (i.e., the eigenvalue) as the result.

Warning Some reference write the diagonalization as $\mathbf{U}^{-1}\mathbf{AU}$, instead of the correct $\mathbf{UAU}^{-1}$. This is confusing, and inconsistent with vector transformation. Many of these very references then *change* their notation when they have to transform a vector, because nearly all references agree that vectors transform with **U**, and not $\mathbf{U}^{-1}$.

Contraction of Matrices

You don't see a dot product of matrices defined very often, but the concept comes up in physics, even if they don't call it a "dot product." We see such products in QM density matrices, and in tensor operations on vectors. We use it below in the "Trace" section for traces of products.

For two matrices of the same size, we define the **contraction of two matrices** as the sum of the products of the corresponding elements (much like the dot product of two vectors). The contraction is a scalar. Picture the contraction as overlaying one matrix on top of the other, multiplying the stacked numbers (elements), and adding all the products:



We use a colon to convey that the summation is over 2 dimensions (rows and columns) of **A** and **B** (whereas the single-dot dot product of vectors sums over the 1 dimensional list of vector components):

$$\mathbf{A:B} \equiv \sum_{i,j=1}^{n} a_{ij}b_{ij}$$

For example, for 3×3 matrices:

$$\mathbf{A:B} = \begin{matrix} a_{11}b_{11} & +a_{12}b_{12} & +a_{13}b_{13} \\ +a_{21}b_{21} & +a_{22}b_{22} & +a_{23}b_{23} \\ +a_{31}b_{31} & +a_{32}b_{32} & +a_{33}b_{33} \end{matrix} = a_{11}b_{11} + a_{12}b_{12} + a_{13}b_{13} + a_{21}b_{21} + a_{22}b_{22} + a_{23}b_{23} + a_{31}b_{31} + a_{32}b_{32} + a_{33}b_{33}$$

which is a single number.

If the matrices are complex, we do not conjugate the left matrix (such conjugation is often done in defining the dot product of complex vectors).

Trace of a Product of Matrices

The trace of a matrix is defined as the sum of the diagonal elements:

$$\mathrm{Tr}(\mathbf{A}) \equiv \sum_{j=1}^{n} a_{jj} \qquad E.g.: \quad \mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \qquad \mathrm{Tr}(\mathbf{A}) = a_{11} + a_{22} + a_{33}\,.$$

The trace of a product of matrices comes up often, e.g. in quantum field theory. We first show that Tr(**AB**) = Tr(**BA**):

Let $\mathbf{C} = \mathbf{AB}$. $\Rightarrow$ $\mathrm{Tr}(\mathbf{AB}) = c_{11} + c_{22} + ... + c_{nn}$

Define a_{r*} as the r^{th} row of $\mathbf{A}$, and b_{*c} as the c^{th} column of $\mathbf{B}$

$c_{11} = a_{1*} \cdot b_{*1}$,
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \quad or \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} (\mathbf{B}^T)_{11} & (\mathbf{B}^T)_{12} & (\mathbf{B}^T)_{13} \\ . & . & . \\ . & . & . \end{pmatrix}$$

$c_{22} = a_{2*} \cdot b_{*2}$
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \quad or \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} . & . & . \\ (\mathbf{B}^T)_{21} & (\mathbf{B}^T)_{22} & (\mathbf{B}^T)_{23} \\ . & . & . \end{pmatrix}$$

and so on.

The diagonal elements of the product **C** are the sums of the overlays of the rows of **A** on the columns of **B**. But this is the same as the overlays of the rows of **A** on the *rows* of $\mathbf{B}^T$. Then we sum the overlays, i.e., we overlay **A** onto $\mathbf{B}^T$, and sum *all* the products of *all* the overlaid elements:

$$\mathrm{Tr}(\mathbf{AB}) = \mathbf{A} : \mathbf{B}^T .$$

Now consider $\mathrm{Tr}(\mathbf{BA}) = \mathbf{B} : \mathbf{A}^T$. But visually, $\mathbf{B} : \mathbf{A}^T$ overlays the same pairs of elements as $\mathbf{A} : \mathbf{B}^T$, but in the transposed order. When we sum over all the products of the pairs, we get the same sum either way:

$$\mathrm{Tr}(\mathbf{AB}) = \mathrm{Tr}(\mathbf{BA}) \qquad because \qquad \mathbf{A} : \mathbf{B}^T = \mathbf{B} : \mathbf{A}^T .$$

This leads to the important cyclic property for the trace of the product of several matrices:

$$\mathrm{Tr}(\mathbf{AB...C}) = \mathrm{Tr}(\mathbf{CAB...}) \qquad because \qquad \mathrm{Tr}((\mathbf{AB...})\mathbf{C}) = \mathrm{Tr}(\mathbf{C}(\mathbf{AB...})) .$$

and matrix multiplication is associative. By simple induction, any cyclic rotation of the matrices leaves the trace unchanged.

Linear Algebra Briefs

The determinant equals the product of the eigenvalues:

$$\det \mathbf{A} = \coprod_{i=1}^{n} \lambda_i \qquad where \quad \lambda_i \text{ are the eigenvalues of } \mathbf{A} .$$

This is because the eigenvalues are unchanged through a similarity transformation. If we diagonalize the matrix, the main diagonal consists of the eigenvalues, and the determinant of a diagonal matrix is the product of the diagonal elements (by cofactor expansion).

7 Introduction to Probability, Statistics, and Data Analysis

I think probability and statistics are among the most conceptually difficult topics in mathematical physics. We start with a brief overview of the basics, but overall, we assume you are familiar with simple probabilities, and gaussian (aka "normal") distributions.

Probability and Random Variables

We assume you have a basic idea of probability, and since we seek here understanding over mathematical purity, we give here intuitive definitions. A **random variable**, say X, is a quantity that you can observe (or measure), multiple times (at least in principle), and is not completely predictable. Each observation (**instance**) of a random variable may give a different value. Random variables may be **discrete** (the roll of a die), or **continuous** (the angle of a game spinner after you spin it). A **uniform** random variable has all its values equally likely. Thus the roll of a (fair) die is a uniform discrete random variable. The angle of a game spinner is a uniform continuous random variable. But in general, the values of a random variable are not necessarily equally likely. For example, a **gaussian** (aka "normal") random variable is more likely to be near the mean.

Given a large sample of observations of any physical quantity X, there will be some structure to the values X assumes. For discrete random variables, each possible value will appear (close to) some fixed fraction of the time in any large sample. The fraction of a large sample that a given value appears is that value's **probability**. For a 6-sided die, the probability of rolling 1 is 1/6, i.e. Pr(1) = 1/6. Because probability is a fraction of a total, it is always *dimensionless*, and between 0 and 1 inclusive:

$$0 \le \Pr(\textit{anything}) \le 1 .$$

[Note that one can imagine systems of chance specifically constructed to *not* provide consistency between samples, at least not on realistic time scales. By definition, then, observations of such a system do *not* constitute a random variable in the sense of our definition.]

Strictly speaking, a **statistic** is a number that summarizes in some way a set of random values. Many people use the word informally, though, to mean the raw data from which we compute true statistics.

Conditional Probability

Probability, in general, is a combination of physics and knowledge:
the physics of the system in question, and what you know about its state.

Conditional probability specifically addresses probability when the state of the system is partly known. **A priori probability** generally implies less knowledge of state ("a priori" means "in the beginning" or "beforehand"). But there is no true, fundamental distinction, because *all* probabilities are in some way dependent on both physics *and* knowledge.

Suppose you have one bag with 2 white and 2 black balls. You draw 2 balls without replacement. What is the chance the 2nd ball will be white? A priori, it's obviously ½. However, suppose the first ball is known white. Now Pr(2nd ball is white) = 1/3. So we say the conditional probability that the 2nd ball will be white, given that the first ball is white, is 1/3. In symbols:

$$\Pr(2\textit{nd ball white} \mid \textit{first ball white}) = 1/3 .$$

Another example of how conditional probability of an event can be different than the a priori probability of that event: I have a bag of white and a bag of black balls. I give you a bag at random. What is the chance the 2nd ball will be white? A priori, it's ½. After seeing the 1st ball is white, now Pr(2nd ball is white) = 1. In this case,

$$\Pr(2\textit{nd ball white} \mid \textit{first ball white}) = 1 .$$

Precise Statement of the Question Is Critical

Many arguments arise about probability because the questions are imprecise: each combatant has a different interpretation of the question, but neither realizes the other is arguing a different issue. Consider this:

You deal 4 cards from a shuffled standard deck of 52 cards. I tell you 3 of them are aces. What is the probability that the 4th card is also an ace?

The question is ambiguous, and could reasonably be interpreted two ways, but the two interpretations have quite different answers. It is very important to know exactly *how* I have discovered that 3 of them are aces.

Case 1: I look at the 4 cards and say "At least 3 of these cards are aces." There are 193 ways that 4 cards can hold at least 3 aces (derived below), and only 1 of those ways has 4 aces. Therefore, the chance of the 4th card being an ace is 1/193.

Case 2: I look at only 3 of the 4 cards and say, "These 3 cards are aces." There are 49 unseen cards, all equally likely to be the 4th card. Only one of them is an ace. Therefore, the chance of the 4th card being an ace is 1/49.

It may help to show that we can calculate the 1/49 chance from the 193 hands that have at least 3 aces: Of the 192 that have exactly 3 aces, we expect that 1/4 of them = 48 will show aces as their first 3 cards (because the non-ace has probability 1/4 of being last) . Additionally, the one hand of 4 aces will always show aces as its first 3 cards. Hence, of the 193 hands with at least 3 aces, 49 show aces as their first 3 cards, of which exactly 1 will be the 4-ace hand. Hence, its conditional probability, given that the first 3 cards are aces, is 1/49.

Here is an ad-hoc derivation that there are 193 sets of 4 cards with at least 3 aces: First, how many ways (combinations) are there to have *exactly* 3 aces in 4 cards? The # ways to have a set of 3 aces = # ways to omit 1 ace = 4. Then there are 48 non-ace cards left for the 4th card. Thus there are 4•48 = 192 ways (combinations) to have exactly 3 aces in 4 cards. There is 1 combination of 4 aces in 4 cards. Thus there are 192 + 1 = 193 ways (combinations) to have at least 3 aces in 4 cards.

Let's Make a Deal

This is an example of a problem that confuses many people (including me), and how to properly analyze it. We hope this example illustrates some general methods of analysis that you can use to navigate more general confusing questions. In particular, the methods used here apply to renormalizing entangled quantum states when a measurement of one value is made.

You are in the Big Deal on the game show *Let's Make a Deal*. There are 3 doors. Hidden behind two of them are goats; behind the other is the Big Prize. You choose door #1. Monty Hall, the MC, knows what's behind each door. He opens door #2, and shows you a goat. Now he asks, do you want to stick with your door choice, or switch to door #3? Should you switch?

Answer: Without loss of generality (WLOG), we assume you choose door #1 (and of course, it doesn't matter which door you choose). You know ahead of time that no matter what, Monty will show you a goat behind one of the doors you didn't pick. For simple questions like this, you can make an exhaustive chart of all the mutually exclusive events, and their probabilities:

Bgg	shows door #2 1/6 shows door #3 1/6
gBg	shows door #3 1/3
ggB	shows door #2 1/3

After you choose, Monty shows you that door #2 is a goat. So from the population of possibilities, we strike out those that are no longer possible (i.e., where he shows door #3, and those where the big prize is #2), and renormalize the remaining probabilities:

Bgg	shows door #2 ~~1/6~~ 1/3 ~~shows door #3 1/6~~
~~gBg~~	~~shows door #3 1/3~~
ggB	shows door #2 ~~1/3~~ 2/3

Another way to think of this: Monty showing you door #2 is equivalent to saying, "The big prize is either the door you picked, or it's door #3." Since your chance of having picked right (1/3) is unaffected by Monty telling you this, Pr(big prize is #3) = 2/3. Monty uses his knowledge to always pick a door with a goat. That gives you information, which improves your ability to guess right on your second guess.

You can also see it this way: There's a 1/3 chance you picked right the first time. If you switch, you'll lose. But there's a 2/3 chance you picked wrong the first time. If you switch, you'll win. So by switching, you win twice as often as you lose, much better odds than 1/3 of winning.

Let's take a more extreme example: suppose there are 100 doors, and you pick #1. Now Monty tells you, "The big prize is either the door you picked, or it's door #57." Should you switch? Of course. The chance you guessed right is tiny, but Monty knows for sure.

How to Lie With Statistics

In 2007, on the front page of newspapers, was a story about a big study of sexual behavior in America. The *headline point* was that on average, heterosexual men have 7 partners in their lives, and women have only 4.

Innumeracy, a book about math and statistics, uses this exact same claim from a previous study of sexual behavior, and noted that one can easily prove that the *average* number of heterosexual partners of men and women must be *exactly* the same (if there are equal numbers of men and women in the population. The US has equal numbers of men and women to better than 1%).

The only explanation for the survey results is that *many people are lying*. Typically, men lie on the high side, women lie on the low side. The article goes on to quote all kinds of statistics and "facts," oblivious to the fact that these claims are based on lies. So how much can you believe anything the study subjects said?

Even more amazing to me is that the "scientists" doing the study seem equally oblivious to the mathematical impossibility of their results. (Perhaps some graduate student got a PhD out of this study, too.)

The proof: every heterosexual encounter involves a man and a woman. If the partners are new to each other, then it counts as a new partner for both the man and the woman. The average number of partners for men is the total number of new partners for all men divided by the number of men in the US. But this is equal to the total number of new partners for all women divided by the number of women in the US. QED.

[An insightful friend noted, "Maybe to some women, some guys aren't worth counting."]

Choosing Wisely: An Informative Puzzle

Here's a puzzle which illuminates the physical meaning of the $\binom{n}{k}$ binomial forms. Try it yourself before reading the answer. Really. First, recall that:

$$\binom{n}{k} \equiv n \text{ choose } k \equiv \frac{n!}{k!(n-k)!}.$$

is the number of combinations of k items taken from n distinct items; more precisely, $\binom{n}{k}$ is the number of ways of choosing k items from n distinct items, without replacement, where the order of choosing doesn't matter.

The puzzle: Show in words, without algebra, that $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$.

Some purists may complain that the demonstration below lacks rigor (not true), or that the algebraic demonstration is "shorter." However, though the algebraic proof is straightforward, it is dull and uninformative. Some may like the demonstration here because it uses the physical meaning of the mathematics to reach an iron-clad conclusion.

The solution: The LHS is the number of ways of choosing k items from $n + 1$ distinct items. Now there are two *distinct* subsets of those ways: those ways that include the $(n + 1)^{th}$ item, and those that don't. In the first subset, given the $(n + 1)^{th}$ item, we must choose $k - 1$ more items from the remaining n, and there are $\binom{n}{k-1}$ ways to do this. In the second subset, we must choose all k items from the first n, and there are $\binom{n}{k}$ ways to do this. Since this covers all the possible ways to choose k items from $n + 1$ items, it must be that $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$. QED.

Multiple Events

First we summarize the rules for computing the probability of combinations of independent events from their individual probabilities, then we justify them:

Pr(A and B) = Pr(A)·Pr(B), A and B independent

Pr(A or B) = Pr(A) + Pr(B), A and B mutually exclusive

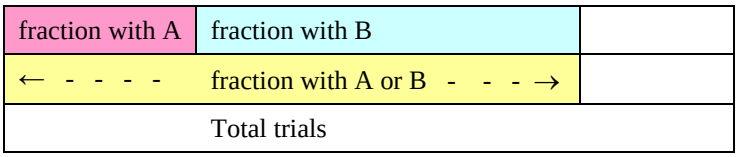
Pr(not A) = 1 – Pr(A)

Pr(A or B) = Pr(A) + Pr(B) – Pr(A)Pr(B), always .

For independent events A and B, Pr(A and B) = Pr(A)·Pr(B). This follows from the definition of probability as a fraction. If A and B are **independent** (have nothing to do with each other), then Pr(A) is the fraction of trials with event A. Then of the fraction of those with event A, the fraction that also has B is Pr(B). Therefore, the fraction of the total trials with both A and B is:

Pr(A and B) = Pr(A)·Pr(B).

For mutually exclusive events, Pr(A or B) = Pr(A) + Pr(B). This also follows from the definition of probability as a fraction. The fraction of trials with event A ≡ Pr(A); fraction with event B ≡ Pr(B). If no trial can contain both A and B, then the fraction with either is simply the sum (figure below).



Pr(not A) = 1 – Pr(A). Since Pr(A) is the fraction of trials with event A, and all trials must either have event A or not:

Pr(A) + Pr(not A) = 1.

Notice that A and (not A) are mutually exclusive events (a trial can't both have A and not have A), so their probabilities add.

By Pr(A or B) we mean Pr(A or B or both). For independent events, you might think that Pr(A or B) = Pr(A) + Pr(B), but this is not so. A simple example shows that it can't be: suppose Pr(A) = Pr(B) = 0.7. Then Pr(A) + Pr(B) = 1.4, which can't be the probability of anything. The reason for the failure of simple addition of probabilities is that doing so counts the probability of (A *and* B) twice (figure below):

fraction with A only	fraction with A and B	fraction with B only	
← - - - -	fraction with A or B - - - - - →		
	Total trials		

Note that Pr(A or B) is equivalent to Pr(A and maybe B) *or* Pr(B and maybe A). But Pr(A and maybe B) includes the probability of both A and B, as does Pr(B and maybe A), hence it is counted twice. So subtracting the probability of (A and B) makes it counted only once:

$$\Pr(A \text{ or } B) = \Pr(A) + \Pr(B) - \Pr(A)\Pr(B), \qquad A \text{ and } B \text{ independent.}$$

A more complete statement, which breaks down (A or B) into mutually exclusive events is:

$$\Pr(A \text{ or } B) = \Pr(A \text{ and not } B) + \Pr(\text{not } A \text{ and } B) + \Pr(A \text{ and } B)$$

Since the right hand side is now mutually exclusive events, their probabilities add:

$$\begin{aligned}\Pr(A \text{ or } B) &= \Pr(A)[1 - \Pr(B)] + \Pr(B)[1 - \Pr(A)] + \Pr(A)\Pr(B) \\ &= \Pr(A) + \Pr(B) - 2\Pr(A)\Pr(B) + \Pr(A)\Pr(B) \\ &= \Pr(A) + \Pr(B) - \Pr(A)\Pr(B) \ .\end{aligned}$$

TBS: Example of rolling 2 dice.

Combining Probabilities

Here is a more in-depth view of multiple events, with several examples. This section should be called "Probability Calculus," but most people associate "calculus" with something hard, and I didn't want to scare them off. In fact, **calculus** simply means "a method of calculation."

Probabilities describe binary events: an event either happens, or it doesn't.
Therefore, we can use some of the methods of Boolean algebra in probability.

Boolean algebra is the mathematics of expressions and variables that can have one of only two values: usually taken to be "true" and "false." We will use only a few simple, intuitive aspects of Boolean algebra here.

An **event** is something that can either happen, or not (it's binary!). We define the **probability** of an event as the fraction of time, out of many (possibly hypothetical) trials, that the given event happens. For example, the probability of getting a "heads" from a toss of a fair coin is 0.5, which we might write as Pr(heads) = 0.5 = 1/2. Probability is a fraction of a whole, and so lies in [0, 1].

We now consider *two* random events. Two events have one of 3 relationships: independent, mutually exclusive, or conditional (aka conditionally dependent). We will soon see that the first two are special cases of the "conditional" relationship. We now consider each relationship, in turn.

Independent: For now, we define independent events as events that have nothing to do with each other, and no effect on each other. For example, consider two events: tossing a heads, and rolling a 1 on a 6-sided die. Then Pr(heads) = 1/2, and Pr(rolling 1) = 1/6. The events are independent, since the coin cannot influence the die, and the die cannot influence the coin. We define one "trial" as two actions: a toss and a roll. Since probabilities are fractions, of all trials, ½ will have "heads", and 1/6 of those will roll a 1. Therefore, 1/12 of all trials will contain both a "heads" and a 1. We see that probabilities of independent events multiply. We write:

$$\Pr(A \text{ and } B) = \Pr(A)\Pr(B) \ . \qquad \text{(independent events).}$$

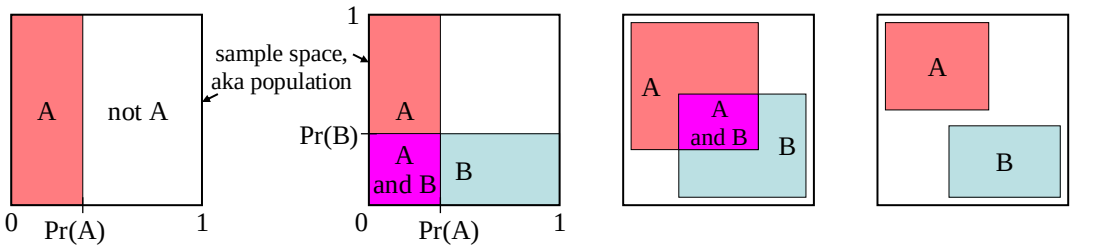
In fact, this is the precise *definition* of independence: if the probability of two events both occurring is the product of the individual probabilities, then the events are **independent**.

[Aside: This definition extends to PDFs: if the joint PDF of two random variables is the product of their individual PDFs, then the random variables are independent.]

Geometric diagrams are very helpful in understanding the probability calculus. We can picture the probabilities of *A*, *B*, and (*A* and *B*) as areas. The **sample space** or **population** is the set of all possible outcomes of trials. We draw that as a rectangle. Each point in the rectangle represents one possible outcome. Therefore, the probability of an outcome being within a region of the population is proportional to the area of the region.

Figure 7.1 (a): An event A either happens, or it doesn't. Therefore:

$$\Pr(A) + \Pr(\sim A) = 1 \qquad \text{(always)} .$$



(a) (b) independent (c) conditional (d) mutually exclusive

Figure 7.1 The (continuous) sample space is the square. Areas are proportional to probabilities. (a) An event either happens, or it doesn't. (b) Events A and B are independent. (c) A and B are dependent. (d) A and B are mutually exclusive.

Figure 7.1 (b): Pr(A) is the same whether B occurs or not, shown by the fraction of B covered by A is the same as the fraction of the sample space covered by A. Therefore, *by definition*, A and B are independent.

Figure 7.1 (c): The probability of (A or B (or both)) is the red, blue, and magenta areas. Geometrically then, we see:

$$\Pr(A \text{ or } B) = \Pr(A) + \Pr(B) - \Pr(A \text{ and } B) \qquad \text{(always)}.$$

This is always true, regardless of any dependence between A and B.

Conditionally dependent: From the diagram, when A and B are **conditionally dependent**, we see that the Pr(B) depends on whether A happens or not. Pr(B given that A occurred) is written as Pr(B | A), and read as "probability of B given A." From the ratio of the magenta area to the red, we see

$$\Pr(B \mid A) = \Pr(B \text{ and } A)/\Pr(A) . \qquad \text{(always)}.$$

Mutually exclusive: Two events are **mutually exclusive** when they cannot both happen (Figure 7.1d). Thus,

$$\Pr(A \text{ and } B) = 0, \qquad \text{and} \qquad \Pr(A \text{ or } B) = \Pr(A) + \Pr(B) \qquad \text{(mutually exclusive)} .$$

Note that Pr(A or B) follows the rule from above, which always applies.

We see that independent events are an extreme case of conditional events: independent events satisfy:

$$\Pr(B \mid A) = \Pr(B) \qquad \text{(independent)}.$$

since the occurrence of A has no effect on B. Also, mutually exclusive events satisfy:

$$\Pr(B \mid A) = 0 \qquad \text{(mutually exclusive)}.$$

Summary of Probability Calculus

Rule	Notes
Always true:	
Pr(~A) = 1 – Pr(A)	Pr(entire sample space) = 1, Figure 7.1a.
Pr(A or B) = Pr(A) + Pr(B) – Pr(A and B)	Subtract off any double-count of "A and B," Figure 7.1c.
If A & B independent:	**From Figure 7.1b**
Pr(A and B) = Pr(A)Pr(B)	Precise def'n of "independent."
Pr(B | A) = Pr(B)	special case of conditional probability.
If A & B mutually exclusive:	**From Figure 7.1d**
Pr(A and B) = 0	Def'n of "mutually exclusive."
Pr(A or B) = Pr(A) + Pr(B)	Nothing to double-count; special case of Pr(A or B) from above.
Pr(B | A) = Pr(A | B) = 0	Can't both happen.
Conditional probabilities:	**From Figure 7.1c**
Pr(B | A) = Pr(B and A) / Pr(A)	fraction of A that is also B.
Pr(B and A) = Pr(B | A)Pr(A) = Pr(A | B)Pr(B)	Bayes' Rule: Shows relationship between Pr(B | A) and Pr(A | B).
Pr(A or B) = Pr(A) + Pr(B) – Pr(A and B)	Same as "Always true" rule above.

Note that the "and" rules are often simpler than the "or" rules.

To B, or To Not B?

Sometimes its easier to compute Pr(~A) than Pr(A). Then we can find Pr(A) from Pr(A) = 1 – Pr(~A).

Example: What is the probability of rolling 4 or more with two dice?

The population has 36 possibilities. To compute the probability directly, we use:

$$\underset{\text{ways to roll 4}}{3} + \underset{\text{ways to roll 5}}{4} + 5 + 6 + 5 + 4 + \cdots + 3 + \cdots + \underset{\text{ways to roll 11}}{2} + \underset{\text{ways to roll 12}}{1} = 33 \quad \Rightarrow \quad \Pr(\geq 4) = \frac{33}{36}.$$

That's a lot of addition. It's much easier to note that:

$$\Pr(<4) = \underset{\substack{\text{ways to}\\ \text{roll 2}}}{1} + \underset{\substack{\text{ways to}\\ \text{roll 3}}}{2} = 3 \quad \Rightarrow \quad \Pr(<4) = \frac{3}{36}, \quad \text{and} \quad \Pr(\geq 4) = 1 - \Pr(<4) = \frac{33}{36}.$$

In particular, the "and" rules are often simpler than the "or" rule. Therefore, when asked for the probability of "this *or* that", it is sometimes simpler to convert to its complementary "and" statement, compute the "and" probability, and subtract it from 1 to find the "or" probability.

Example: From a standard 52-card deck, draw a single card. What is the chance it is a spade or a face-card (or both)? Note that these events are independent.

To compute directly, we use the "or" rule:

$$\Pr(spade) = 1/4, \qquad \Pr(\ facecard) = 3/13,$$

$$\Pr(spade\ or\ \ facecard) = \frac{1}{4} + \frac{3}{13} - \frac{1}{4}\cdot\frac{3}{13} = \frac{13+12-3}{52} = \frac{22}{52}$$

Because the two events are independent, it may be simpler to compute the probability of drawing neither a spade nor a face-card, and subtracting from 1:

$$\Pr(\sim spade) = 3/4, \qquad \Pr(\sim\ facecard) = 10/13,$$

$$\Pr(spade\ or\ \ facecard) = 1 - \Pr(\sim spade\ and\ \sim\ facecard) = 1 - \frac{3}{4}\cdot\frac{10}{13} = 1 - \frac{30}{52} = \frac{22}{52}$$

The benefit of converting to the simpler "and" rule increases with more "or" terms, as shown in the next example.

Example: Remove the 12 face cards from a standard 52-card deck, leaving 40 number cards (aces are 1). Draw a single card. What is the chance it is a spade (S), low (L) (4 or less), or odd (O)? Note that these 3 events are independent.

To compute directly, we can count up the number of ways the conditions can be met, and divide by the population of 40 cards. There are 10 spades, 16 low cards, and 20 odd numbers. But we can't just sum those numbers, because we would double (and triple) count many of the cards.

To compute directly, we must extend the "or" rule to 3 conditions, shown below.

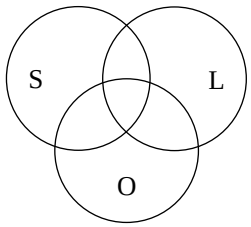


Figure 7.2 Venn diagram for Spade, Low, and Odd.

Without proof, we state that the direct computation from a 3-term "or" rule is this:

$$\Pr(S) = 1/4, \qquad \Pr(L) = 4/10, \qquad \Pr(O) = 1/2$$

$$\begin{aligned}\Pr(S\ or\ L\ or\ O) &= \Pr(S) + \Pr(L) + \Pr(O) \\ &\quad - \Pr(S)\Pr(L) - \Pr(S)\Pr(O) - \Pr(L)\Pr(O) + \Pr(S)\Pr(L)\Pr(O) \\ &= \frac{1}{4} + \frac{4}{10} + \frac{1}{2} - \left(\frac{1}{4}\cdot\frac{4}{10}\right) - \left(\frac{1}{4}\cdot\frac{1}{2}\right) - \left(\frac{4}{10}\cdot\frac{1}{2}\right) + \left(\frac{1}{4}\cdot\frac{4}{10}\cdot\frac{1}{2}\right) \\ &= \frac{10+16+20-4-5-8+2}{40} = \frac{31}{40}\end{aligned}$$

It is far easier to compute the chance that it is none of these (neither spade, nor low, nor odd):

$$\Pr(\sim S) = 3/4, \qquad \Pr(\sim L) = 6/10, \qquad \Pr(\sim O) = 1/2$$

$$\begin{aligned}\Pr(S \text{ or } L \text{ or } O) &= 1 - \Pr(\sim S \text{ and } \sim L \text{ and } \sim O) = 1 - \Pr(\sim S)\Pr(\sim L)\Pr(\sim O) \\ &= 1 - \frac{3}{4}\cdot\frac{6}{10}\cdot\frac{1}{2} = 1 - \frac{9}{40} = \frac{31}{40}.\end{aligned}$$

You may have noticed that converting "S or L or O" into "~(~S and ~L and ~O)" is an example of De Morgan's theorem from Boolean algebra.

Continuous Random Variables and Distributions

Probability is a little more complicated for continuous random variables. A continuous population is a set of random values than can take on values in a continuous interval of real numbers; for example, if I spin a board-game spinner, the little arrow can point in any direction: $0 \le \theta < 2\pi$.

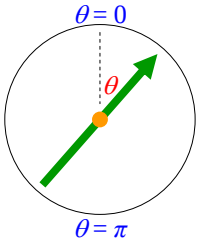


Figure 7.3 Board game spinner

Furthermore, all angles are equally likely. By inspection, we see that the probability of being in the first quadrant is ¼, i.e. $\Pr(0 \le \theta < \pi/2) = ¼$. Similarly, the probability of being in any interval $d\theta$ is:

$$\Pr\left(\theta \text{ in any interval } d\theta\right) = \frac{1}{2\pi} d\theta \,.$$

If I ask, "what is the chance that it will land at exactly $\theta = \pi$?" the probability goes to zero, because the interval $d\theta$ goes to zero. In this simple example, the probability of being in any interval $d\theta$ is the same as being in any other interval of the same size. In general, however, some systems have a probability per unit interval that varies with the value of the random variable (call it X) (I wish I had a simple, everyday example of this??). So:

$$\Pr\left(X \text{ in an infinitesimal interval } dx \text{ around } x\right) = \text{pdf}_X(x)\, dx\,, \quad \text{where}$$

$\text{pdf}_X(x) \equiv$ the **probability distribution function** of X.

$\text{pdf}_X(x)$ has units of $[X]/x$.

E.g., if X has units of meters ($[X]$ = m), and x also has units of meters ($[x]$ = m), then $\text{pdf}_X(x)$ is dimensionless. If $[X]$ = dimensionless and $[t]$ = s, then $\text{pdf}_X(t)$ has units of 1/s.

By summing mutually exclusive probabilities, the probability of X in any *finite* interval $[a, b]$ is:

$$\Pr(a \le X \le b) = \int_a^b dx\, \text{pdf}(x)\,.$$

Since any random variable X must have *some* real value, the total probability of X being between $-\infty$ and $+\infty$ must be 1:

$$\Pr\left(-\infty < X < \infty\right) = \int_{-\infty}^{\infty} dx\, \text{pdf}(x) = 1\,.$$

The probability distribution function of a random variable tells you everything there is to know about that random variable.

Populations

A **population** is a (often infinite) set of all possible values that a random variable may take on, along with their probabilities. A **sample** is a finite set of values of a random variable, where those values come from the population of all possible values. The same value may be repeated in a sample. We often use samples to estimate the characteristics of a much larger population.

A **trial** or **instance** is *one* value of a random variable.

There is enormous confusion over the binomial (and similar) distributions, because each instance of a binomial random variable comes from many **attempts** at an event, where each attempt is labeled either "success" or "failure." Superficially, an "attempt" looks like a "trial," and many sources confuse the terms. In the binomial distribution, *n* attempts go into making a *single* trial (or instance) of a binomial random variable.

Population Variance

The **variance** of a population is a measure of the "spread" of any distribution, i.e. it is some measure of how widely spread out values of a random variable are likely to be [there are other measures of spread, too]. The variance of a population or sample is among the most important parameters in statistics. Variance is always ≥ 0, and is defined as the average squared-difference between the random values and their average value:

$$\mathrm{var}(X) \equiv \left\langle \left(X - \bar{X}\right)^2 \right\rangle \qquad \textit{where} \quad \langle\ \rangle \text{ is an operator which takes the average} \qquad \bar{X} \equiv \langle X \rangle .$$

Note that:

Whenever we write an operator such as var(*X*), we can think of it as a *functional* of the PDF of *X* (recall that a functional acts on a function to produce a number).

$$\mathrm{var}(X) \equiv \mathrm{var}\left[\mathrm{pdf}_X(x)\right] \equiv \int_{-\infty}^{\infty} \left(x - \bar{X}\right)^2 \mathrm{pdf}_X(x)\, dx \equiv \left\langle \left(X - \bar{X}\right)^2 \right\rangle .$$

The units of variance are the square of the units of *X*. From the definition, we see that if I multiply a set of random numbers by a constant *k*, then I multiply their variance by k^2:

$$\mathrm{var}(kX) = k^2\, \mathrm{var}(X) \qquad \textit{where} \quad X \text{ is any set of random numbers .}$$

Any function, including variance, with the above property is **homogeneous-of-order-2** (2^{nd} order homogeneous??). We will return later to methods of estimating the variance of a population.

Population Standard Deviation

The **standard deviation** of a population is another measure of the "spread" of a distribution, defined simply as the square root of the variance. Standard deviation is always ≥ 0, and equals the root-mean-square (RMS) of the deviations from the average:

$$\mathrm{dev}(X) \equiv \sqrt{\mathrm{var}(X)} = \sqrt{\left\langle \left(X - \bar{X}\right)^2 \right\rangle} \qquad \textit{where} \quad \langle\ \rangle \text{ is an operator which takes the average .}$$

As with variance, we can think of dev(*X*) as a functional acting on $\mathrm{pdf}_X(x)$: $\mathrm{dev}[\mathrm{pdf}_X(x)]$. The units of standard deviation are the units of *X*. From the definition, we see that if I multiply a set of random numbers by a constant *k*, then I multiply their standard deviation by *k*:

$$\mathrm{dev}(kX) = k\, \mathrm{dev}(X) \qquad \textit{where} \quad X \text{ is any set of random numbers .}$$

Standard deviation and variance are useful measures, even for non-normal populations.

They have many universal properties, some of which we discuss as we go. There exist bounds on the percentage of *any* population contained with $\pm\ c\sigma$, for any number *c*. Even stronger bounds apply for all unimodal populations.

Normal (aka Gaussian) Distribution

From mathworld.wolfram.com/NormalDistribution.html : "While statisticians and mathematicians uniformly use the term 'normal distribution' for this distribution, physicists sometimes call it a gaussian distribution and, because of its curved flaring shape, social scientists refer to it as the 'bell curve.' "

A **gaussian distribution** is one of a 2-parameter family of distributions defined as a population with:

$$\text{pdf}(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \qquad \textit{where} \quad \begin{matrix} \mu \equiv \text{population average} \\ \sigma \equiv \text{population standard deviation} \end{matrix} \quad \text{[picture??].}$$

μ and σ are parameters: μ can be any real value, and $\sigma > 0$ and real. This illustrates a common feature of named distributions: they are usually a *family* of distributions, parameterized by one or more parameters. A gaussian distribution is a 2-parameter distribution: μ and σ. As noted below:

Any linear combination of gaussian random variables is another gaussian random variable.

Gaussian distributions are the *only* such distributions [ref??].

New Random Variables From Old Ones

Given two random variables X and Y, we can construct new random variables as functions of x and y (trial values of X and Y). One common such new random variable is simply the sum:

$$\text{Define} \quad Z \equiv X + Y \qquad \text{which means} \qquad \forall \text{ trials } i, \qquad z_i \equiv x_i + y_i \,.$$

We then ask, given $\text{pdf}_X(x)$ and $\text{pdf}_Y(y)$ (which is all we can know about X and Y), what is $\text{pdf}_Z(z)$? To answer this, consider a particular value x of X; we see that:

$$\text{Given} \quad x: \qquad \Pr(Z \text{ within } dz \text{ of } z) = \Pr\big(Y \text{ within } dz \text{ of } (z-x)\big).$$

But x is a value of a random variable, so the total $\Pr(Z \text{ within } dz \text{ of } z)$ is the sum (integral) over all x:

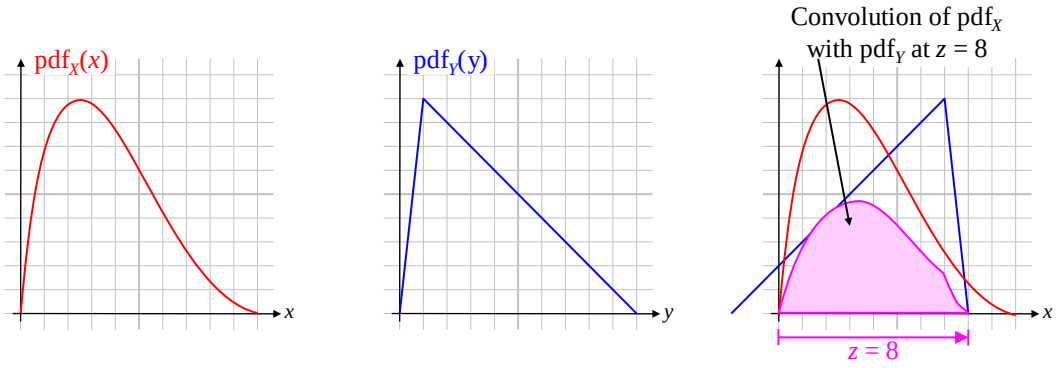
$$\Pr(Z \text{ within } dz \text{ of } z) = \int_{-\infty}^{\infty} dx\, \text{pdf}_X(x) \Pr\big(Y \text{ within } dz \text{ of } (z-x)\big), \qquad \text{but}$$

$$\Pr\big(Y \text{ within } dz \text{ of } (z-x)\big) = \text{pdf}_Y(z-x)\,dz, \qquad \text{so}$$

$$\Pr(Z \text{ within } dz \text{ of } z) = dz \int_{-\infty}^{\infty} dx\, \text{pdf}_X(x)\, \text{pdf}_Y(z-x)$$

$$\Rightarrow \qquad \text{pdf}_Z(z) = \int_{-\infty}^{\infty} dx\, \text{pdf}_X(x)\, \text{pdf}_Y(z-x).$$

This integral way of combining two functions, $\text{pdf}_X(x)$ and $\text{pdf}_Y(y)$ with a parameter z is called the **convolution** of pdf_X and pdf_Y, which is a function of a number, z.



The convolution evaluated at z is the area under the product $\text{pdf}_X(x)\text{pdf}_Y(z-x)$.

From the above, we can easily deduce the $\text{pdf}_Z(z)$ if $Z \equiv X - Y = X + (-Y)$. First, we find $\text{pdf}_{(-Y)}(y)$, and then use the convolution rule. Note that:

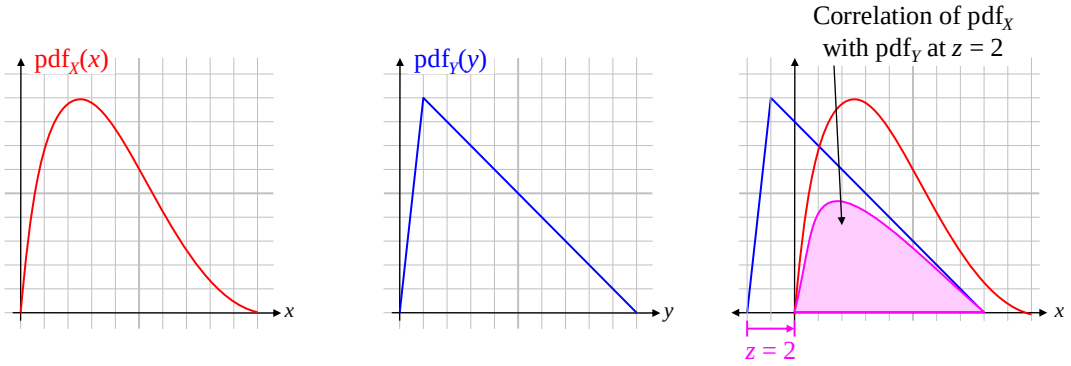
$$\text{pdf}_{(-Y)}(y) = \text{pdf}_Y(-y)$$

$$\Rightarrow \qquad \text{pdf}_Z(z) = \int_{-\infty}^{\infty} dx\, \text{pdf}_X(x)\, \text{pdf}_{(-Y)}(z-x) = \int_{-\infty}^{\infty} dx\, \text{pdf}_X(x)\, \text{pdf}_Y(x-z)$$

Since we are integrating from $-\infty$ to $+\infty$, we can shift x with no effect:

$$x \to x+z \qquad \Rightarrow \qquad \text{pdf}_Z(z) = \int_{-\infty}^{\infty} dx\, \text{pdf}_X(x+z)\, \text{pdf}_Y(x)\,,$$

which is the standard form for the **correlation function** of two functions, $\text{pdf}_X(x)$ and $\text{pdf}_Y(y)$.



The correlation function evaluated at z is the area under the product $\text{pdf}_X(x+z)\text{pdf}_Y(x)$.

The PDF of the sum of two random variables is the convolution of their PDFs.
The PDF of the difference of two random variables is the correlation function of their PDFs.

Note that the convolution of a gaussian distribution with a different gaussian is another gaussian. Therefore, the sum of a gaussian random variable with *any* other gaussian random variable is gaussian.

Some Distributions Have Infinite Variance, or Infinite Average

In principle, the only requirement on a PDF is that it be normalized:

$$\int_{-\infty}^{\infty} \text{pdf}(x)\, dx = 1\,.$$

Such a distribution has well-defined probabilities for all x. However, even given that, it is possible that the variance is infinite (or properly, undefined). For example, consider:

$$\left.\begin{array}{ll} \text{pdf}(x) = 2x^{-3} & x \ge 1 \\ = 0 & x < 1 \end{array}\right\} \quad \Rightarrow \qquad \bar{x} = \int_1^{\infty} x\, \text{pdf}(x)\, dx = 2, \qquad \text{but} \qquad \sigma^2 = \int_1^{\infty} x^2\, \text{pdf}(x)\, dx \to \infty\,.$$

The above distribution is normalized, and has finite average, but infinite deviation. The following example is even worse:

$$\left.\begin{array}{ll} \text{pdf}(x) = x^{-2} & x \ge 1 \\ = 0 & x < 1 \end{array}\right\} \quad \Rightarrow \qquad \bar{x} = \int_0^{\infty} x\, \text{pdf}(x)\, dx \to \infty, \quad \text{and} \quad \sigma^2 = \int_0^{\infty} x^2\, \text{pdf}(x)\, dx \to \infty\,.$$

This distribution is normalized, but has both infinite average and infinite deviation.

Are such distributions physically meaningful? Sometimes. The Lorentzian (aka Breit-Wigner) distribution is common in physics, or at least, a good approximation to physical phenomena. It has infinite average and deviation. It's standard and parameterized forms are:

$$L(x) = \frac{1}{\pi\left(1+x^2\right)} \qquad L(x;x_0,\gamma) = \frac{1}{\pi\gamma}\cdot\frac{1}{1+\left((x-x_0)/\gamma\right)^2}$$

where $x_0 \equiv$ location of peak, $\gamma \equiv$ half-width at half-maximum

This is approximately the energy distribution of particles created in high-energy collisions. It's CDF is:

$$\text{cdf}_{Lorentzian}(x) = \frac{1}{\pi}\arctan\left(\frac{x-x_0}{\gamma}\right)+\frac{1}{2}.$$

Samples and Parameter Estimation

Why Do We Use Least Squares, and Least Chi-Squared (χ^2)?

We frequently use "least sum-squared-residuals" (aka **least squares**) as our definition of "best." Why sum-squared-residuals? Certainly, one could use other definitions (see least-sum-magnitudes below). However, least squares residuals are most common because they have many useful properties:

- Squared residuals are reasonable: they're always positive.
- Squared residuals are continuous and differentiable functions of things like fit parameters (magnitude residual is not differentiable). Differentiable means we can analytically minimize it, and for linear fits, the resulting equations are linear.
- The sum-of-squares identity is only valid for least-squares fits; this identity allows us to cleanly separate our data variation into a "model" and "residuals."
- We can compute *many* other analytic results from least squares, which is not generally true with other residual measures.
- Variance is defined as average of squared deviation (aka "residual"), and variances of uncorrelated random values simply add.
- The central limit theorem causes gaussian distributions to appear frequently in the natural world, and one of its two natural parameters is variance: an average squared-residual.
- For gaussian residuals, least squares parameter estimates are also maximum likelihood (in the misleading statistical sense??), and minimum variance [Meyers 1996 p??].

Most other measures of residuals have fewer nice properties. Note that the residuals include both unmodeled behavior and (unmodelable) noise, so even if your *noise* is gaussian, your residuals are not.

Why Not Least-Sum-Magnitudes?

A common question is "Why not magnitude of residuals, instead of squared residuals?" Least-sum-magnitude residuals have at least two serious problems. First, they often yield clearly bad results; and second, least-sum-magnitude-residuals can be highly degenerate: there are often an infinite number of solutions that are "equally" good, and that's bad.

To illustrate, Figure 7.4a shows the least sum magnitude "average" for 3 points. Sliding the average line up or down increases the magnitude difference for points 1 and 2, and decreases the magnitude difference by the same amount for point 2. Points 1 and 2 totally dominate the result, regardless of how large point 2 is. This is intuitively undesirable for most purposes.

Figure 7.4b and (c) show the degeneracy: both lines have equal sum magnitudes, but intuitively, fit (b) is vastly better for most purposes.

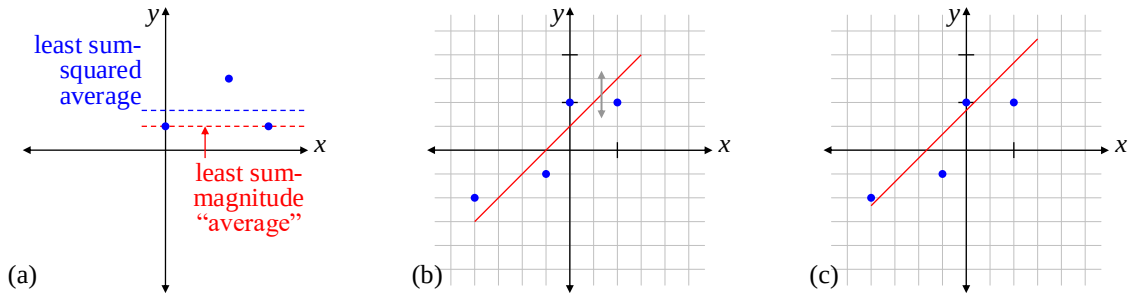


Figure 7.4 (a) least-sum-magnitude "average". (b) Example fit to least-sum-magnitude-residuals. The sum-magnitude is unchanged by moving the "fit line" straight up or down. (c) Alternative "fit" has same sum-magnitude-residuals, but is a much less-likely fit for realistic residual distributions.

Other Misfit Measures

There *are* some cases where least squares residuals does *not* work well, in particular, if you have outliers in your data. When you square the residual to an outlier, you get a really big number. This squared-residual swamps out all your (real) residuals, thus wreaking havoc with your results. The usual practice is to identify the outliers, remove them, and analyze the remaining data with least-squares. However, on rare occasions, one might work with a residual measure other than least squared residuals [Myers ??].

When working with data where each measurement has its own uncertainty, we usually replace the least squared residuals criterion with least-chi-squared. We discuss this later when considering data with individual uncertainties.

Average, Variance, and Standard Deviation

In statistics, an **efficient estimator** ≡ the *most* efficient estimator [ref??]. There is none better (i.e., none with smaller variance). You can prove mathematically that for gaussian populations, the average and variance of a sample are the most efficient estimators (least variance) of the population average and variance. It is impossible to do any better, so it's not worth looking for better ways. For gaussian populations, the most efficient estimators are **least squares** estimators, which means that over many samples, they minimize the sum-squared error from the true value. We discuss least-squares vs. maximum-likelihood estimators later.

Note, however, that given a set of measurements, some of them may not actually measure the population of interest (i.e., they may be noise). If you can identify those bad measurements from a sample, you should remove them before estimating any parameter. Usually, in real experiments, there is always some unremovable corruption of the desired signal, and this contributes to the uncertainty in the measurement.

Recall that a **sample** is a set of n random values taken from a population X. The **sample average** is defined as:

$$\bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i \,,$$

and is the least variance estimate of the average $\langle X \rangle$ of any population [ref??]. It is **unbiased** [ref??], which means the average of many sample estimates approaches the true population average:

$$\langle \bar{x} \rangle_{many\ samples} = \langle X \rangle \qquad where \quad \langle \bullet \rangle_{over\ what} \equiv \text{average, over the given parameter if not obvious .}$$

Note that the definition of unbiased is *not* that the estimator approaches the true value for large samples; it is that the *average* of the estimator approaches the true value over many samples, even small samples. See sample standard deviation, just below.

The sample variance and standard deviation are defined as:

$$s^2 \equiv \frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2 \qquad \textit{where} \quad \bar{x} \text{ is the sample average, as above: } \bar{x} \equiv \left\langle x_i \right\rangle$$

$$s \equiv \sqrt{s^2}$$

The sample variance is an efficient and unbiased estimate of var(X), which means no other estimate of var(X) is better. Note that s^2 is unbiased, but s is *biased*, because the square root of the average is not equal to the average of the square root:

$$\left\langle s \right\rangle_{\textit{many samples}} \neq \text{dev}\left(X\right) \quad \text{because} \qquad \left\langle \sqrt{s^2} \right\rangle \neq \sqrt{\left\langle s^2 \right\rangle}\,.$$

This exemplifies the importance of properly defining "bias":

$$\left\langle s \right\rangle_{\textit{many samples}} \neq \text{dev}\left(X\right) \quad \text{even though} \qquad \lim_{n\to\infty} s = \text{dev}\left(X\right).$$

Sometimes you see variance defined with $1/n$, and sometimes with $1/(n-1)$. Why? The **population variance** is *defined* as the mean-squared deviation from the population average. For a finite population (such as test scores in a given class), we find the population variance using $1/N$, where N is the number of values in the whole population:

$$\text{var}(X) \equiv \frac{1}{N}\sum_{i=1}^{N}\left(X_i - \mu\right)^2 \qquad \textit{where} \quad \begin{array}{l} N \text{ is the \# of values in the entire population} \\ X_i \text{ is the } i^{th} \text{ value of the population} \\ \mu \equiv \text{exact population average}. \end{array}$$

In contrast, the **sample variance** is the variance of a *sample* taken from a population. The population average μ is usually unknown. We can only estimate $\mu \approx \langle x \rangle$. Then to make s^2 an unbiased estimate of var(X), we must use $1/(n-1$ (as we show later)), where n is the sample size (not population size).

The sample variance is actually a special case of curve fitting, where we fit a constant, $\langle x \rangle$, to the population. This is a single parameter, and so removes 1 degree of freedom from our fit errors. Hence, the mean-squared fit error (i.e., s^2) has 1 degree of freedom less than the sample size. (Much more on curve fitting later).

For a *sample* from a population when the average μ is exactly known, we use n as the weighting for s^2 to be an unbiased estimator of var(X):

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu\right)^2, \qquad \text{which is just the above equation with } X_i \to x_i,\ N \to n.$$

In contrast, infinite populations with unknown μ can *only* have samples, and thus always use n–1. But as $n \to \infty$, it doesn't matter, so we can compute the population variance either way:

$$\text{var}(X) \equiv \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2 = \lim_{n\to\infty}\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2, \text{ because } n-1 \to n \text{ when } n \to \infty.$$

Central Limit Theorem For Continuous And Discrete Populations

The central limit theorem is important because it allows us to estimate some properties of a population given only a sample of the population, with no a priori information. Each sample has an average. If we take many samples, each will (likely) produce a different average. Hence, the average of a sample is a *new* random variable, created from the original population.

The central limit theorem says that for *any* population, as the sample size n grows, the sample average approaches a *gaussian* random variable, with average equal to the population average, and variance equal to the population variance divided by n.

Mathematically, given a random variable X, with mean μ and variance $\sigma_X^2 \equiv \text{var}(X)$:

$$\lim_{n\to\infty} \langle x_i \rangle \in gaussian\left(\mu, \frac{{\sigma_X}^2}{n}\right) \qquad where \quad \langle x_i \rangle \equiv \text{sample average}.$$

Note that the central limit theorem applies only to multiple samples from a *single* population (though there are some variations of the theorem that can be applied to multiple populations). [It is possible to construct large sums of *multiple* populations whose averages do *not* approach gaussian, e.g. in communication theory, inter-symbol interference (ISI). We will not go further into that.]

How does the Central Limit Theorem apply to a discrete population? If a population is discrete, then any sample average is also discrete. But the gaussian distribution is continuous. So how can the sample average approach a gaussian for large sample size n? Though the sample average is discrete, the density of allowed values increases with n. If you bin many averages into a histogram, the histogram approaches the gaussian curve. For very large n, the bins can be chosen narrow, so they "look" continuous.

TBS: Why binomial (discreet), Poisson (discreet), and chi-squared (continuous) distributions approach gaussian for large n (or ν).

Uncertainty of Average

A sample average $\bar{x}$ gives us an estimate of the population average μ. The sample average, when taken as a set of values of many samples, is itself a random variable. The Central Limit Theorem (CLT) says that if we know the population standard deviation σ, the sample average will have standard deviation:

$$\text{dev}(\bar{x}) = \frac{\sigma}{\sqrt{n}} \qquad \text{(proof below)}.$$

In statistics, $\text{dev}(\bar{x})$ is called the **standard error of the mean**. In experiments, $\text{dev}(\bar{x})$ is the 1-sigma **uncertainty** in our estimate of the population average μ. However, most often, we know neither μ nor σ, and must estimate *both* from our sample, using $\bar{x}$ and s. For "large" samples, we use simply $\sigma \approx s$, and then:

$$\text{dev}\left(\bar{x}\right) \approx \frac{s}{\sqrt{n}} \qquad \text{for "large" samples, i.e. } n \text{ is "large"}.$$

For small samples, we must still use s as our estimate of the population deviation, since we have nothing else. But instead of assuming that $\text{dev}(\bar{x})$ is gaussian, we use the *exact* distribution, which is a little wider, called a **t-distribution** [W&M ??]. The t-distribution is a 1-parameter family, with the parameter "degrees of freedom" $= n - 1$. $pdf_t(t)$ is complicated to write explicitly. The argument t is similar to the gaussian $z \equiv (x - \mu)/\sigma$; both are measures of dimensionless distance from the mean:

$$t \equiv \frac{x - \bar{x}}{s} \; where \quad \bar{x} = \text{sample average}, \qquad s = \text{sample standard deviation}.$$

We use t, and t-tables, to establish confidence intervals [ref??].

Uncertainty of Uncertainty: How Big Is Infinity?

Sometimes, we need to know the uncertainty in our estimate of the population variance (or standard deviation), i.e., we need dev(s). We start by looking more closely at the uncertainty in our estimate s^2 of the population variance σ^2. The random variable $\frac{(n-1)s^2}{\sigma^2}$ has chi-squared distribution with $n - 1$ degrees of freedom [W&M Thm 6.16 p201], or equivalently:

$$s^2 \in \frac{\sigma^2}{n-1}\chi^2(n-1) \qquad \Rightarrow \qquad \text{var}\left(s^2\right) = \left(\frac{\sigma^2}{n-1}\right)^2 2(n-1) = \frac{2\sigma^4}{n-1},$$

$$\text{dev}\left(s^2\right) = \sqrt{\frac{2}{n-1}}\cdot\sigma^2 .$$

However, usually we're more interested in the uncertainty of the standard deviation estimate *s*, rather than s^2. For that, we must approximate, using the fact that *s* is function of s^2: $s \equiv (s^2)^{1/2}$. For moderate or bigger sample sizes, and confidence ranges up to 95% or so, we can use the approximate formula for the deviation of a function of a random variable (see "Functions of Random Variables," elsewhere):

$$Y = f(X) \quad \Rightarrow \quad \text{dev}(Y) \approx f'\left(\langle X \rangle\right)\text{dev}(X) \qquad \text{for small dev}(X).$$

$$s \equiv \left(s^2\right)^{1/2} \quad \Rightarrow \quad \text{dev}(s) \approx \frac{1}{2}\left(\sigma^2\right)^{-1/2}\text{dev}\left(s^2\right) = \frac{1}{2\sigma}\sqrt{\frac{2}{n-1}}\sigma^2 = \frac{1}{\sqrt{2(n-1)}}\sigma \approx \frac{1}{\sqrt{2(n-1)}}s .$$

This allows us to address the rule of thumb: "$n > 30$" is statistical infinity.

This rule is most often used in estimating the standard error of the mean, $\text{dev}(\bar{x})$ (see above), given by $\text{dev}\left(\bar{x}\right) = \frac{\sigma}{\sqrt{n}} \approx \frac{s}{\sqrt{n}}$. For small samples, this approximation isn't so good. Then, as noted above, the uncertainty $\text{dev}(\bar{x})$ needs to include both the true sampling uncertainty in $\bar{x}$ and the uncertainty in *s*. To be confident that our $\bar{x}$ is within our claim, we need to expand our confidence limits, to allow for the chance that *s* happens to be low. The Student T-distribution exactly handles this correction to our confidence limits on $\bar{x}$ for all sample sizes.

However, when can we ignore this correction? In other words, how big should *n* be for the gaussian (as opposed to *T*) distribution be a good approximation. The uncertainty in *s* is:

$$\text{dev}(s) = \frac{1}{\sqrt{2(n-1)}}\sigma .$$

This might seem circular, because we still have σ (which we don't know) on the right hand side. However, it's effect is now reduced by the fraction multiplying it. So the uncertainty in σ is also reduced by this factor, and we can neglect it. Thus to first order, we have:

$$\text{dev}(s) = \sigma\frac{1}{\sqrt{2(n-1)}} \approx \frac{1}{\sqrt{2(n-1)}}s .$$

So long as this dev(*s*) << *s*, we can ignore it. In other words:

$$\text{dev}(s) << s \qquad \Rightarrow \qquad \frac{1}{\sqrt{2(n-1)}} << 1, \text{ for } \bar{x} \text{ to be approximately gaussian, and } s \approx \sigma.$$

(You may notice that dev(*s*) is correlated with *s*: bigger *s* implies bigger (estimated) dev(*s*), so the contribution to $\text{dev}(\bar{x})$ from dev(*s*) does *not* add in quadrature to $s/\sqrt{n}$.) When $n = 30$:

$$\frac{1}{\sqrt{2(30-1)}} = 0.13 << 1 .$$

13% is pretty reasonable for the uncertainty of the uncertainty, i.e. $\text{dev}\left(\text{dev}(\bar{x})\right)$, and $n = 30$ is the generally agreed upon bound for good confidence that $s \approx \sigma$ [ref??].

Functions of Random Variables

It follows from the definition of probability that the average value of any function of a random variable is:

$$\langle f(X) \rangle = \int_{-\infty}^{\infty} dx \; f(x) \, \mathrm{pdf}_X(x) \, .$$

We can apply this to our definitions of population average and population variance:

$$\bar{X} \equiv \langle X \rangle = \int_{-\infty}^{\infty} dx \, x \, \mathrm{pdf}_X(x), \qquad \text{and} \qquad \mathrm{var}(X) = \int_{-\infty}^{\infty} dx \left(x - \bar{X} \right)^2 \mathrm{pdf}_X(x) \, .$$

Statistically Speaking: What Is The Significance of This?

Before we compute any uncertainties, we should understand what they mean. Statistical significance interprets uncertainties. It is one of the most misunderstood, and yet most important, concepts in science. It underlies virtually all experimental and simulation results. Beliefs (correct and incorrect) about statistical significance drive experiment, research, funding, and policy.

Understanding statistical significance is a prerequisite to understanding science.

This cannot be overstated, and yet many (if not most) scientists and engineers receive no formal training in statistics. The following few pages describe statistical significance, surprisingly using almost no math.

Overview of Statistical Significance

The term "statistically significant" has a precise meaning which is, unfortunately, different than the common meaning of the word "significant."

Many experiments compare quantitative measures of two populations, e.g. the IQs of ferrets vs. gophers. In any real experiment, the two measures will almost certainly differ. How should we interpret this difference?

We can use statistics to tell us the meaning of the difference. A difference which is not "statistically significant" in some particular experiment may, in fact, be quite important. But we can only determine its importance if we do another experiment with finer resolution, enough to satisfy our subjective judgment of "importance." For this section, I use the word **importance** to mean a *subjective* assessment of a measured result.

The statement "We could not measure a difference" is very different from "There is no important difference." Statistical significance is a quantitative comparison of the magnitude of an effect and the resolution of the statistics used to measure it.

This section requires an understanding of probability and uncertainty.

Statistical significance can be tricky, so we start with several high level statements about what statistical significance is, and is not. We then give more specific statements and examples.

Statistical significance is many things:

Statistical significance is a measure of an experiment's ability to resolve its own measured result.
It is *not* a measure of the importance of a result.

"Statistically significant" means "measurable by this experiment." "Not statistically significant" means that we cannot fully trust the result *from this experiment alone*; the experiment was too crude to have confidence in its own result.

Statistical significance is closely related to uncertainty.

Statistical significance is a quantitative statement of the probability that a result is real, instead of a measurement error or the random result of sampling that just happened to turn out that way (by chance).

Statistical significance is a one-way street: if a result is statistically significant, it is (probably) real. However, it may or may not be important. In contrast, if a result is *not* statistically significant, then we don't know if it's real or not. However, we will see that even a not significant result can sometimes provide meaningful and useful information.

If the difference between two results in an experiment is not statistically significant, that difference may still be very real and important.

Details of Statistical Significance

A meaningful measurement must contain two parts: the magnitude of the result, and the confidence limits on it, both of which are quantitative statements. When we say, "the average IQ of ferrets in our experiment is 102 ± 5 points," we mean that there is a 95% chance that the *actual* average IQ is between 97 and 107. We could also say that our 95% **confidence limits** are 97 to 107. Or, we could say that our 95% **uncertainty** is 5 points. The confidence limits are sometimes called **error bars**, because on a graph, confidence limits are conventionally drawn as little bars above and below the measured values.

Suppose we test gophers and find that their average IQ is 107 ± 4 points. Can we say "on average, gophers have higher IQs than ferrets?" In other words, is the difference we measured significant, or did it happen just by chance? To assess this, we compute the difference, and its uncertainty (recall that uncorrelated uncertainties add in quadrature):

$$\Delta IQ = (107 - 102) \pm \sqrt{4^2 + 5^2} = 5 \pm 6 \qquad \text{(gophers – ferrets)}$$

This says that the difference lies within our uncertainty, so we are not 95% confident that gophers have higher IQs. Therefore, we still don't know if either population has higher IQs than the other. Our experiment was not precise enough to measure a difference. This does *not* mean that there is no difference. However, we can say that there is a 95% chance that the difference is between –1 and 11 (5 ± 6). A given experiment measuring a difference can produce one of two results of statistical significance: (1) the difference is statistically significant; or (2) it is not. In this case, the difference is not (statistically) significant at the 95% level.

In addition, confidence limits yield one of three results of "importance:" (1) confirm that a difference is important; or (2) not important, or (3) be inconclusive. But the *judgment* of how much is "important" is outside the scope of the experiment. For example, we may know from prior research that a 10 point average IQ difference makes a population a better source for training pilots, enough better to be "important." Note that this is a subjective statement, and its precise meaning is outside our scope here.

Five of the six combinations of significance and importance are possible, as shown by the following examples.

Example 1, not significant, and inconclusive importance: With the given numbers, $\Delta IQ = 5 \pm 6$, the "importance" of our result is inconclusive, because we don't know if the average IQ difference is more or less than 10 points.

Example 2, not significant, but definitely not important: Suppose that prior research showed (somehow) that a difference needed to be 20 points to be "important." Then our experiment shows that the difference is not important, because the difference is very unlikely to be as large as 20 points. In this case, *even though the results are not statistically significant, they are very valuable*; they tell us something meaningful and worthwhile, namely, the difference between the average IQs of ferrets and gophers is not important for using them as a source for pilots. The experimental result is valuable, even though not significant, because it establishes an *upper bound* on the difference.

Example 3, significant, but inconclusive importance: Suppose again that a difference of 10 points is important, but our measurements are: ferrets average 100 ± 3 points, and gophers average 107 ± 2 points. Then the difference is:

$$\Delta IQ = (107 - 100) \pm \sqrt{2^2 + 3^2} = 7 \pm 4 \qquad \text{(gophers – ferrets)}$$

These results are statistically significant: there is better than a 95% chance that the average IQs of ferrets and gophers are different. However, the importance of the result is still inconclusive, because we don't know if the difference is more or less than 10 points.

Example 4, significant and important: Suppose again that a difference of 10 points is important, but we measure that ferrets average 102 ± 3 points, and gophers average 117 ± 2 points. Then the difference is:

$$\Delta IQ = (117 - 102) \pm \sqrt{2^2 + 3^2} = 15 \pm 4 \qquad \text{(gophers – ferrets)}$$

Now the difference is both statistically significant, and important, because there is a 95% chance that the difference is > 10 points. We are better off choosing gophers to go to pilot school.

Example 5, significant, but not important: Suppose our measurements resulted in

$$\Delta IQ = 5 \pm 4$$

Then the difference is significant, but not important, because we are confident that the difference < 10. This result established an upper bound on the difference. In other words, our experiment was precise enough that if the difference were important (i.e., big enough to matter), then we'd have measured it.

Finally, note that we cannot have a result that is not significant, but important. Suppose our result was:

$$\Delta IQ = 11 \pm 12$$

The difference is unmeasurably small, and possibly zero, so we certainly cannot say the difference is important. In particular, we can't say the difference is greater than anything.

Thus we see that stating "there is a statistically significant difference" is (by itself) not saying much, because the difference could be tiny, and physically unimportant.

We have used here the common confidence limit fraction of 95%, often taken to be ~2σ. The next most common fraction is 68%, or ~1σ. Another common fraction is 99%, taken to be ~3σ. More precise gaussian fractions are 95.45% and 99.73%, but the digits after the decimal point are usually meaningless (i.e., not statistically significant!) Note that we cannot round 99.73% to the nearest integer, because that would be 100%, which is meaningless in this context. Because of the different confidence fractions in use, you should always state your fractions explicitly. You can state your confidence fraction once, at the beginning, or along with your uncertainty, e.g. 10 ± 2 (1σ).

Caveat: We are assuming **random errors**, which are defined as those that average out with larger sample sizes. **Systematic errors** do not average out, and result from biases in our measurements. For example, suppose the IQ test was prepared mostly by gophers, using gopher cultural symbols and metaphors unfamiliar to most ferrets. Then gophers of equal intelligence will score higher IQs because the test is not fair. This bias changes the meaning of all our results, possibly drastically.

Ideally, when stating a difference, one should put a lower bound on it that is physically important, and give the probability (confidence) that the difference is important. E.g. "We are 95% confident the difference is at least 10 points" (assuming that 10 points on this scale matters).

Examples

Here are some examples of meaningful and not-so-meaningful statements:

Meaningless Statements (appearing frequently in print)	**Meaningful Statements, possibly subjective (not appearing enough in print)**
The difference in IQ between groups A and B is not statistically significant. (Because your experiment was bad, or because the difference is small?)	Our data show there is a 99% likelihood that the IQ difference between groups A and B is less than 1 point.
We measured an average IQ difference of 5 points. (With what confidence?)	Our experiment had insufficient resolution to tell if there was an important difference in IQ.

Group A has a statistically significantly higher IQ than group B. (How much higher? Is it important?)	Our data show there is a 95% likelihood that the IQ difference between groups A and B is greater than 10 points.

Statistical significance summary: "Statistical significance" is a quantitative statement about an experiment's ability to resolve its own result. We use "importance" as a subjective assessment of a measurement that may be guided by other experiments, and/or gut feel. Statistical significance says nothing about whether the measured result is important or not.

Predictive Power: Another Way to Be Significant, but Not Important

Suppose that we have measured IQs of millions of ferrets and gophers over decades. Suppose their population IQs are gaussian, and given by (note the use of 1σ uncertainties):

$$\text{ferrets: } 101 \pm 20 \qquad \text{gophers: } 103 \pm 20 \qquad (1\sigma)\ .$$

The average difference is small, but because we have millions of measurements, the uncertainty in the average is even smaller, and we have a statistically significant difference between the two groups.

Suppose we have only one slot open in pilot school, but two applicants: a ferret and a gopher. Who should get the slot? We haven't measured these two individuals, but we might say, "Gophers have 'significantly' higher IQs than ferrets, so we'll accept the gopher." Is this valid?

To quantitatively assess the validity of this reasoning, let us suppose (simplistically) that pilot students with an IQ of 95 or better are 20% more likely (1.2×) to succeed than those with IQ < 95. From the given statistics, 61.8% of ferrets have IQs > 95, vs. 65.5% of gophers. That is, 61.8% of ferrets get the 1.2× boost in likelihood of success, and similarly for the gophers. Then the *relative* probabilities of success are:

$$\text{ferrets: } 0.382 + 0.618(1.2) = 1.12 \qquad \text{gophers: } 0.345 + 0.655(1.2) = 1.13\ .$$

Thus a random gopher is 113/112 times (less than 0.7% more) likely to succeed than a random ferret. This is pretty unimportant. In other words, species (between ferrets and gophers) is not a good predictor of success. Species is *so* bad that many, many other facts will be better predictors of success. Height, eyesight, years of schooling, and sports ability are probably all better predictors. The key point is this:

Differences in average between two populations, that are much smaller than the deviations *within* the populations, are poor predictors of individual outcomes.

Unbiased vs. Maximum-Likelihood Estimators

In experiments, we frequently have to estimate parameters from data. There is a very important difference between "unbiased" and "maximum likelihood" estimates, even though *sometimes* they are the same. Sadly, two of the most popular experimental statistics books confuse these concepts, and their distinction.

[A common error is to try to "derive" unbiased estimates using the principle of "maximum likelihood," which is impossible since the two concepts are very different. One popular text goes through the exercise of "deriving" the formula for unbiased sample variance from the principle of maximum likelihood, and necessarily gets the wrong answer! Hand waving is then applied to wiggle out of the mistake.]

Everything in this section applies to *arbitrary* distributions, not just gaussian. We follow these steps:

1. Terse definitions, which won't be entirely clear at first.
2. Example of estimating the variance of a population (things still fuzzy).
3. Examples of the need for maximum-likelihood in both single and repeated trials.
4. Real-world physics examples of different situations leading to different choices between unbiased and maximum-likelihood.
5. Closing comments.

Terse definitions: In short:

> An unbiased statistic is one whose average is exactly right: in the limit of an infinite number of estimates, the average of an unbiased statistic is exactly the population parameter.

For example, the average of many samples of a population average is likely closer to the right answer than a single sample of the population average is.

A **maximum likelihood** statistic is one which is most likely to have produced the given the data. Note that if it is biased, then the average of many maximum likelihood estimates does *not* get you closer to the right answer. In other words, given a fixed set of data, maximum-likelihood estimates have some merit, but biased ones can't be combined well with other sets of data (perhaps future data, not yet taken). This concept should become more clear below.

Which is better, an unbiased estimate or a maximum-likelihood estimate? It depends on what you goals are.

Example of population variance: Given a sample of values $\{y_i\}$ from a population, an *unbiased* estimate of the population variance is

$$\sigma^2 \approx s^2 \equiv \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n-1} \qquad \text{(unbiased estimate)}.$$

If we take several samples of the population, compute an unbiased estimate of the variance for each sample, and average those estimates, we'll get a better estimate of the population variance. Usually, unbiased estimators are those that minimize the sum-squared-error from the true value (**principle of least-squares**).

However, what if we only get *one* shot at estimating the population variance?

Examples: Maximum likelihood vs. unbiased: Suppose Monty Hall says "I'll give you a zillion dollars if you can estimate the variance (to within some tolerance)"? What estimate should we give him? Since we only get one chance, we don't care about the average of many estimates being accurate. We want to give Mr. Hall the variance estimate that is *most likely* to be right. One can show that the most likely estimate is given by using n in the denominator, instead of $(n - 1)$:

$$\sigma_{ML}^2 = \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n} \qquad \text{(maximum-likelihood estimate)}.$$

This is the estimate most likely to win the prize. Perhaps more physically, if you need to choose how long to fire a retro-rocket to land a spacecraft on the moon, do you choose (a) the burn time that, averaged over many spacecraft, reaches the moon, or (b) the burn time that is most likely to land your one-and-only craft on the moon?

In the case of variance, the maximum-likelihood estimate is smaller than the unbiased estimate by a factor of $(n - 1)/n$. If we were to make many maximum-likelihood estimates, each one would be small by the same factor. The average would then also be small by that factor. No amount of averaging would ever fix this error. Our average estimate of the population variance would *not* get better with more estimates.

You might conclude that maximum-likelihood estimates are only good for situations where you get a single trial. However, we now show that maximum-likelihood estimates can be useful even when there are many trials of a statistical process: you are a medieval peasant barely keeping your family fed. Every morning, the benevolent king goes to the castle tower overlooking the public square, and tosses out a gold coin to the crowd. Whoever catches it, keeps it.

Being better educated than most medieval peasants, each day you record how far the coin goes, and generate a PDF (probability distribution function) for the distance from the tower. It looks like Figure 7.5.

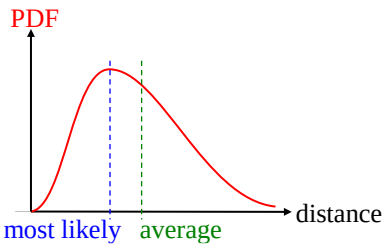


Figure 7.5 Gold coin toss distance PDF.

The most-likely distance is notably different than the average distance. Given this information, where do you stand each day? Answer: At the most-likely distance, because that maximizes your payoff not only for one trial, but across many trials over a long time. The "best" estimator is in the eye of the beholder: as a peasant, you don't care much for least squares, but you do care about most money.

Note that the previous example of landing a spacecraft is the same as the gold coin question: even if you launch many spacecraft, for each one you would give the burn *most-likely* to land the craft. The average of many failed landings has no value.

Real physics examples: Example 1: Suppose you need to generate a beam of ions, all moving at very close to the same speed. You generate your ions in a plasma, with a Maxwellian thermal speed distribution (roughly the same shape as the gold coin toss PDF). Then you send the ions through a velocity selector to pick out only those very close to a single speed. You can tune your velocity selector to pick any speed. Now ions are not cheap, so you want your velocity selector to get the most ions from the speed distribution that it can. That speed is the most-likely speed, not the average speed. So here again, we see that most-likely has a valid use even in repeated trials of random processes.

Example 2: Suppose you are tracing out the orbit of the moon around the earth by measuring the distance between the two. Any given day's measurement has limited ability to trace out an entire orbit, so you must make many measurements over several years. You have to fit a model of the moon's orbit to this large set of measurements. You'd like your fit to get better as you collect more data. Therefore, each day you choose to make *unbiased* estimates of the distance, so that on-average, over time, your estimate of the orbit gets better and better. If instead you chose each day's maximum-likelihood estimator, you'd be off of the average (in the same direction) every day, and no amount of averaging would ever fix that.

Wrap up: When you have a symmetric, unimodal distribution for a parameter estimate (symmetric around a single maximum), then the unbiased and maximum-likelihood estimates are identical. This is true, for example, for the average of a gaussian distribution. For asymmetric or multi-modal distributions, the unbiased and maximum-likelihood estimates are different, and have different properties. For gaussian populations, the least-squares estimators (which minimize the sum-squared error from the true value) are both unbiased and minimum variance (aka most "efficient") [Meyers 1996 p??].

Correlation and Dependence

To take a sample of a random variable X, we get a value of X_i for each sample point i, $i = 1 \ldots n$. Sometimes when we take a sample, for each sample point we get not one, but two, random variables, X_i and Y_i. The two random variables X_i and Y_i may or may not be related to each other. We define the joint probability distribution function of X and Y such that:

$$\Pr(x < X < x + dx \text{ and } y < Y < y + dy) \equiv \text{pdf}_{XY}(x, y) .$$

This is just a 2-dimensional version of a typical pdf. Since X and Y are random variables, we could look at either of them and find its individual pdf: $\text{pdf}_X(x)$, and $\text{pdf}_Y(y)$. If X and Y have nothing to do with each other (i.e., X and Y are **independent**), then a fundamental axiom of probability says that the probability density of finding $x < X < x + dx$ *and* $y < Y < y + dy$ is the product of the two pdfs:

$$X \text{ and } Y \text{ are independent} \quad \Rightarrow \quad \text{pdf}_{XY}(x, y) = \text{pdf}_X(x)\,\text{pdf}_Y(y)$$

The above equation is the *definition* of statistical independence:

Two random variables are independent if and only if
their joint distribution function is the product of the individual distribution functions.

A very different concept is "correlation." Correlation is a measure of how *linearly* related two random variables are. We discuss correlation in more detail later, but it turns out that we can define correlation mathematically by the correlation coefficient. We start by defining the **covariance**:

$$\text{cov}(X,Y) \equiv \left\langle \left(X - \bar{X}\right)\left(Y - \bar{Y}\right) \right\rangle, \qquad \rho(X,Y) \equiv \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \quad \text{correlation coefficient}.$$

The correlation coefficient $\rho(X, Y)$ is proportional to the covariance cov(X, Y). If ρ (or the covariance) = 0, then X and Y are uncorrelated. If ρ (or the covariance) $\neq 0$, then X and Y are **correlated**. For a discrete random variable:

$$\rho \propto \sum_{i=1}^{\textit{population}} \left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right).$$

Note that ρ and covariance are symmetric in X and Y:

$$\text{cov}(X,Y) = \text{cov}(Y,X), \qquad \rho(X,Y) = \rho(Y,X) \qquad (\text{symmetric}).$$

Two random variables are uncorrelated if and only if
their covariance, defined above, is zero.

Being independent is a stronger statement than uncorrelated. Random variables which are independent are necessarily uncorrelated (proof below). But variables which are uncorrelated can be highly dependent. For example, suppose we have a random variable X, which is uniformly distributed over [–1, 1]. Now define a new random variable Y such that $Y = X^2$. Clearly, Y is dependent on X, but Y is uncorrelated with X. Y and X are dependent because given either, we know a lot about the other. They are uncorrelated because for every Y value, there is one positive and one negative value of X. So for every value of $\left(X - \bar{X}\right)\left(Y - \bar{Y}\right)$, there is its negative, as well. The average is therefore 0; hence, cov(X, Y) = 0.

A crucial point is:

Variances add for uncorrelated variables, even if they are dependent.

This is easy to show. Given that X and Y are uncorrelated:

$$\begin{aligned}
\text{var}(X+Y) &= \left\langle \left[X + Y - \left(\bar{X} + \bar{Y}\right)\right]^2 \right\rangle = \left\langle \left[\left(X - \bar{X}\right) + \left(Y - \bar{Y}\right)\right]^2 \right\rangle \\
&= \left\langle \left(X - \bar{X}\right)^2 + 2\left(X - \bar{X}\right)\left(Y - \bar{Y}\right) + \left(Y - \bar{Y}\right)^2 \right\rangle \\
&= \left\langle \left(X - \bar{X}\right)^2 \right\rangle + 2\cancel{\left\langle \left(X - \bar{X}\right)\left(Y - \bar{Y}\right) \right\rangle} + \left\langle \left(Y - \bar{Y}\right)^2 \right\rangle \\
&= \text{var}(X) + \text{var}(Y).
\end{aligned}$$

All we needed to prove that variances add is that cov(X, Y) = 0.

Independent Random Variables are Uncorrelated

It is extremely useful to know that independent random variables are necessarily uncorrelated. We prove this now, in part to introduce some methods of statistical analysis, and to emphasize the distinction between "uncorrelated" and "independent." Understanding analysis methods enables you to analyze a new system reliably, so learning these methods is important for research.

Two random variables are **independent** if they have no relationship at all. Mathematically, the definition of **statistical independence** of two random variables is that the joint density is simply the product of the individual densities:

$$\text{pdf}_{x,y}(x,y) = \text{pdf}_x(x)\,\text{pdf}_y(y) \qquad \text{statistical independence .}$$

The definition of **uncorrelated** is that the covariance, or equivalently the correlation coefficient, is zero:

$$\text{cov}(x,y) \equiv \big\langle (x-\mu_x)(y-\mu_y) \big\rangle = 0 \qquad \text{uncorrelated random variables .} \tag{7.1}$$

These definitions are all we need to prove that independent random variables are uncorrelated. First, we prove a slightly simpler claim: independent *zero-mean* random variables are uncorrelated:

$$\text{Given:} \quad \mu_x \equiv \int_\infty dx\ \text{pdf}_x(x) = 0, \qquad \mu_y \equiv \int_\infty dy\ \text{pdf}_y(x) = 0\,,$$

then the integral factors into x and y integrals, because the joint density of independent random variables factors:

$$\text{cov}(x,y) = \langle xy \rangle = \iint_\infty dx\,dy\ \ \text{pdf}_{x,y}(x,y)\ xy = \left(\int_\infty dx\ \text{pdf}_x(x) \right) \left(\int_\infty dy\ \text{pdf}_y(x) \right) = 0\,.$$

For non-zero-mean random variables, $(x - \mu_x)$ *is* a zero-mean random value, as is $(y - \mu_y)$. But these are the quantities that appear in the definition of covariance (7.1). Therefore, the covariance of *any* two independent random variables is zero.

Note well:

> Independent random variables are necessarily uncorrelated, but the converse is *not* true: uncorrelated random variables may still be dependent.

For example, if $X \in$ uniform(–1,1), and $Y \equiv X^2$, then X and Y are uncorrelated, but highly dependent.

r You Serious?

Ask the average person on the street, "Is a correlation coefficient of 0.4 important?" You're likely to get a response like, "Wow, 40%. That's a lot." In fact, it's almost nothing. Racing through a quick calculation (that we explain more carefully below): $\rho = 0.4$ means the variance can be reduced by a fraction of $\rho^2 = 0.16$, to 0.84 of its original value. The standard deviation is then $(0.84)^{1/2} = 0.92$ of its original value, for a decrease of only 8%! Pretty paltry from a correlation coefficient of $\rho = 0.4$.

To see why, we first note that the standard deviation, σ, of a data set is a reasonable measure of its variation: σ has the same units as the measurements and the average, so it's easy to compare with them. (In contrast, the variance, σ^2, is an important and useful measure of variation, but it cannot be directly compared to measurements or averages.)

We now address the correlation coefficient, ρ (rather than r, which is an estimate of ρ). For definiteness, consider a set of measurements y_i, and their predictors (perhaps independent variables) x_i. ρ tells us what fraction of the *variance* of y is accounted for by the x_i. In other words, if we subtract out the values of y that are predicted by the x_i, by what fraction is the variance reduced?

$$\rho^2 = 1 - \frac{\text{var}\left(y_i - y_{pred,i}\right)}{\text{var}\left(y_i\right)} \quad \Rightarrow \qquad 1 - \rho^2 = \frac{\text{var}\left(y_i - y_{pred,i}\right)}{\text{var}\left(y_i\right)}\,.$$

But the important point is by what fraction is σ reduced? Since $\sigma = (\text{variance})^{1/2}$:

$$1 - \rho^2 = \left(\frac{\sigma_{new}}{\sigma} \right)^2, \qquad \frac{\sigma_{new}}{\sigma} = \sqrt{1-\rho^2}\,, \qquad \sigma\text{-reduction} = 1 - \frac{\sigma_{new}}{\sigma} = 1 - \sqrt{1-\rho^2}\,.$$

For even moderate values of ρ, the reduction in σ is small (Figure 7.6). In fact, it's not until $\rho \approx 0.5$, where the reduction in σ is about 13%, that the correlation starts to become import. Don't sweat the small stuff.

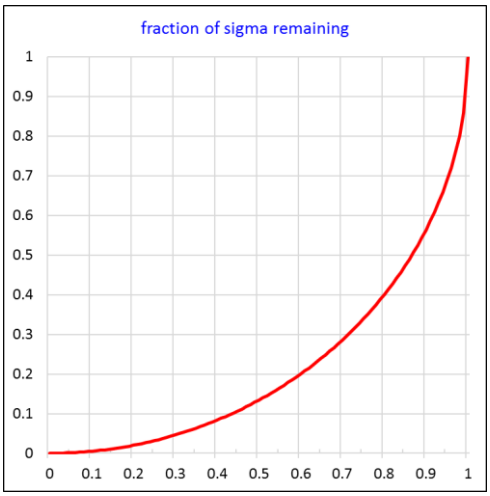


Figure 7.6 Fractional reduction in σ vs. ρ, for a predictor with correlation coefficient ρ. The curve is an arc of a circle.

Statistical Analysis Algebra

Statistical analysis relies on a number of basic properties of combining random variables (RVs), which define an algebra of statistics. This algebra of RV interaction relates to distributions, averages, variances, and other properties. Within this algebra, there is much confusion about which results apply universally, and which apply only conditionally: e.g., gaussian distributions, independent RVs, uncorrelated RVs, etc. We explicitly address all conditions here. We will use all of these methods later, especially when we derive the lesser-known results for uncertainty weighted data.

We first derive three important rules of statistical algebra. For random variable x and y:

1. $\langle x+y \rangle = \langle x \rangle + \langle y \rangle$, for *arbitrary* dependence between x and y.
2. $\langle xy \rangle = \langle x \rangle \langle y \rangle + \text{cov}(x,y)$.
3. $\text{var}(x+y) = \text{var}(x) + \text{var}(y) + 2\,\text{cov}(x,y)$.

[For other interesting results, see
https://www.probabilitycourse.com/chapter5/5_3_1_covariance_correlation.php .]

The Average of a Sum: Easy?

We all know that <x + y> = <x> + <y>. But is this true even if x and y are *dependent* random variables (RVs)? Let's see. We can find <x + y> for dependent variables by integrating over the joint density:

$$\langle x+y \rangle \equiv \iint_{\infty} dx\,dy\ \text{pdf}_{x,y}(x,y)\,(x+y) = \iint_{\infty} dx\,dy\ \text{pdf}_{x,y}(x,y)\,x + \iint_{\infty} dx\,dy\ \text{pdf}_{x,y}(x,y)\,y$$
$$= \langle x \rangle + \langle y \rangle .$$

Therefore, the result *is* easy, and essential for all further analyses:

The average of a sum equals the sum of averages, even for RVs of *arbitrary* dependence.

The Average of a Product

Life sure would be great if the average of a product were the product of the averages ... but it's not, in general. Although, sometimes it is. As scientists, we need to know the difference. Given x and y are random variables (RVs), what is <xy>?

In statistical analysis, it is often surprisingly useful to break up a random variable into its "varying" part plus its average; therefore, we define:

$$x \equiv \delta x + \mu_x, \qquad y \equiv \delta y + \mu_y \qquad \Rightarrow \qquad \langle \delta x \rangle = \langle \delta y \rangle = 0 .$$

Note that μ_x and μ_y are constants. Then we can evaluate:

$$\langle xy \rangle = \Big\langle (\delta x + \mu_x)(\delta y + \mu_y) \Big\rangle = \langle \delta x \delta y \rangle + \mu_y \cancel{\langle \delta x \rangle} + \mu_x \cancel{\langle \delta y \rangle} + \mu_x \mu_y$$

$$= \mu_x \mu_y + \Big\langle (x - \mu_y)(y - \mu_y) \Big\rangle \equiv \mu_x \mu_y + \mathrm{cov}(x, y) .$$

The average of the product is the product of the averages *plus* the covariance.

Only if x and y are uncorrelated, which is implied if they are independent (see earlier), then the average of the product is the product of the averages.

This rule provides a simple corollary: the average of an RV squared:

$$\langle x^2 \rangle = \mu_x{}^2 + \mathrm{cov}(x, x) = \mu_x{}^2 + \sigma_x{}^2 . \tag{7.2}$$

Variance of a Sum

We frequently need the variance of a sum of possible *dependent* RVs. We derive it here for RVs x, y:

$$\mathrm{var}(x + y) = \Big\langle \big(x + y - \mu_x - \mu_y\big)^2 \Big\rangle = \Big\langle \big[(x - \mu_x) + (y - \mu_y) \big]^2 \Big\rangle$$

$$= \Big\langle (x - \mu_y)^2 \Big\rangle + \Big\langle (y - \mu_y)^2 \Big\rangle + 2 \Big\langle (x - \mu_x)(y - \mu_y) \Big\rangle = \mathrm{var}(x) + \mathrm{var}(y) + 2\,\mathrm{cov}(x, y) .$$

Covariance Revisited

The covariance comes up so frequently in statistical analysis that it merits an understanding of its properties as part of the statistical algebra. Covariance appears directly in the formulas for the variance of a sum, and the average of a product, of RVs. (You might remember this by considering the units. For a sum $x + y$: [x] = [y] and [var($x + y$)] = [x^2] = [y^2] = [cov(x, y)]. For a product xy: [xy] = [cov(x, y)].) Conceptually, the covariance of two RVs, a and b, measures how much a and b vary together linearly from their respective averages. If positive, it means a and b tend to go up together; if negative, it means a tends to go up when b goes down, and vice-versa. Covariance is defined as a population average:

$$\mathrm{cov}(a, b) \equiv \Big\langle (a - \mu_a)(b - \mu_b) \Big\rangle .$$

From the definition, we see that cov( ) is a bilinear, commutative operator:

Given: a, b, c, d are random variables; $k \equiv$ constant:

$$\mathrm{cov}(a, b) = \mathrm{cov}(b, a)$$

$$\mathrm{cov}(ka, b) = \mathrm{cov}(a, kb) = k\,\mathrm{cov}(a, b)$$

$$\mathrm{cov}(a + c, b) = \mathrm{cov}(a, b) + \mathrm{cov}(c, b), \qquad \mathrm{cov}(a, b + d) = \mathrm{cov}(a, b) + \mathrm{cov}(a, d) .$$

Occasionally, when expanding a covariance, there may be constants in the arguments. We can consider a constant as a random variable which always equals its average, so:

$$\text{cov}(a,k) = 0$$

$$\text{cov}(a+k,b) = \text{cov}(a,b+k) = \text{cov}(a,b)\,.$$

From the definition, we find that the covariance of an RV with itself is the RV's variance:

$$\text{cov}(a,a) = \text{var}(a)\,.$$

Capabilities and Limits of the Sample Variance

The following developments yield important results, and illustrate some methods of statistical algebra that are worth understanding. We wish to determine an unbiased estimator for the population variance, σ^2, from a sample (set) of n independent values $\{y_i\}$, in two cases: (1) we already know the population average μ; and (2) we don't know the population average. The first case is easier. We proceed in detail, because we need this foundation of process to be rock solid, since so much is built upon it.

σ^2 from sample and known μ: We must start with the *definition* of population variance as an average over the population:

$$\sigma^2 \equiv \left\langle (y-\mu)^2 \right\rangle \quad \textit{where} \quad \mu \equiv \langle y \rangle \equiv \text{average over population of } y \equiv \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} y_i\,. \tag{7.3}$$

A simple guess for the estimator of σ^2, motivated by the definition, might be:

$$g^2 \equiv \frac{1}{n}\sum_{i=1}^{n} (y_i - \mu)^2 \qquad \text{(a guess)}\,.$$

We now analyze our guess over many samples of size n, to see how it performs. By definition, to be unbiased, the average of g^2 over an ensemble of samples of size n must equal σ^2:

$$\text{unbiased:} \qquad \left\langle g^2 \right\rangle_{ensemble} = \sigma^2 \equiv \left\langle (y-\mu)^2 \right\rangle_{population}\,.$$

Mathematically, we find an ensemble average by letting the number of ensembles go to infinity, and the definition of population average is given by letting the number of individual values go to infinity. Let M be the number of ensembles. Then:

$$\left\langle g^2 \right\rangle_{ensemble} \equiv \lim_{M\to\infty} \frac{1}{M}\sum_{m=1}^{M} {g_m}^2 = \lim_{M\to\infty} \frac{1}{M}\sum_{m=1}^{M} \frac{1}{n}\sum_{i=1}^{n} (y_i - \mu)^2\,.$$

Since all the y_i above are distinct, we can combine the summations. Effectively, we have converted the ensemble average on the RHS to a population average, whose properties we know:

$$\left\langle g^2 \right\rangle_{ensemble} = \lim_{M\to\infty} \frac{1}{Mn}\sum_{i=1}^{Mn} (y_i - \mu)^2 \equiv \left\langle (y-\mu)^2 \right\rangle_{population} \equiv \sigma^2\,.$$

We have proved that our guess is an unbiased estimator of the population variance, σ^2.

(In fact, since we already know that the sample average is an unbiased estimate of the population average, and the variance σ^2 is defined as a population average, then we can conclude immediately that the *sample* average of $\langle (y_i - \mu)^2 \rangle$ is an unbiased estimate of the *population* average $\langle (y_i - \mu)^2 \rangle \equiv \sigma^2$. Again, we took the long route above to illustrate important methods that we will use again.)

Note that the denominator is n, and not $n - 1$,
because we started with separate knowledge of the population average μ.

For example, when figuring the standard deviation of grades in a class, one uses n in the denominator, since the class average is known exactly.

σ^2 from sample alone: A harder case is estimating σ^2 when μ is not known. As before, we must start with a guess at an estimator, and then analyze our guess to see how it performs. A simple guess, motivated by the definition, might be:

$$s^2 \propto \sum_{i=1}^{n} (y_i - \bar{y})^2 \qquad \text{(a guess)} \qquad \textit{where} \quad \bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i \,.$$

By definition, to be unbiased, the average of s^2 over an ensemble of samples of size n must equal σ^2. We now consider the sum in s^2. We first show a failed attempt, and then how to avoid it. If we try to analyze the sum directly, we get :

$$\left\langle \sum_{i=1}^{n} (y_i - \bar{y})^2 \right\rangle = \sum_{i=1}^{n} \left\langle {y_i}^2 - 2\bar{y}y_i + \bar{y}^2 \right\rangle = \sum_{i=1}^{n} \left\langle {y_i}^2 \right\rangle - 2\sum_{i=1}^{n} \langle \bar{y}y_i \rangle + n\left\langle \bar{y}^2 \right\rangle .$$

In the equation above, angle brackets mean ensemble average. By tradition, we don't explicitly label our angle brackets to say what we are averaging over, and we make you figure it out. Even better, as we saw earlier, sometimes the angle brackets mean ensemble average, and sometimes they mean population average. (This is a crucial difference in definition, and a common source of confusion in statistical analysis: just what are we averaging over, anyway?) However, on the RHS, the first ensemble average is the same as the population average. However, further analysis of the ensemble averages at this point is messy (more on this later).

To avoid the mess, we note that definition (7.4) requires us to somehow introduce the population average into the analysis, even though it is unknown. By trial and error, we find it is easier to start with the population average, and write it in terms of $\bar{y}$:

$$\sum_{i=1}^{n} (y_i - \mu)^2 = \sum_{i=1}^{n} \big((y_i - \bar{y}) + (\bar{y} - \mu)\big)^2 = \sum_{i=1}^{n} (y_i - \bar{y})^2 + 2(\bar{y} - \mu)\underbrace{\sum_{i=1}^{n} (y_i - \bar{y})}_{0} + \sum_{i=1}^{n} (\bar{y} - \mu)^2 \,. \quad (7.5)$$

In the second term, $(\bar{y} - \mu)$ does not depend on *i*, so it comes out of the summation. Then the second term is identically zero, because:

$$\sum_{i=1}^{n} (y_i - \bar{y}) = \sum_{i=1}^{n} y_i - n\bar{y} = n\bar{y} - n\bar{y} = 0 \,.$$

Now we can take the ensemble average of the remains of the sum-of-squares equation:

$$\left\langle \sum_{i=1}^{n} (y_i - \mu)^2 \right\rangle = \left\langle \sum_{i=1}^{n} (y_i - \bar{y})^2 \right\rangle + \left\langle \sum_{i=1}^{n} (\bar{y} - \mu)^2 \right\rangle$$

$$\sum_{i=1}^{n} \left\langle (y_i - \mu)^2 \right\rangle = \left\langle \sum_{i=1}^{n} (y_i - \bar{y})^2 \right\rangle + n\left\langle (\bar{y} - \mu)^2 \right\rangle .$$

All the ensemble averages in the sum on the LHS are the same, and equal the population average, which is the definition of σ^2. On the RHS, we use the known properties of $\bar{y}$:

$$\langle \bar{y} \rangle = \mu, \qquad \left\langle (\bar{y} - \mu)^2 \right\rangle \equiv \text{var}(\bar{y}) = \sigma^2 / n \,.$$

Then we have:

$$n\sigma^2 = \left\langle \sum_{i=1}^{n} (y_i - \bar{y})^2 \right\rangle + \sigma^2$$

$$\left\langle \sum_{i=1}^{n} (y_i - \bar{y})^2 \right\rangle = (n-1)\sigma^2 .$$

Thus we see our guess for s^2 is correct. The last equation implies that the unbiased sample estimator is:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2 .$$

We made no assumptions at all about the distribution of y, therefore:

s^2 is an unbiased estimator of population variance σ^2 for *any* distribution.

How to Do Statistical Analysis Wrong, and How to Fix It

The following example development contains one error that illustrates a common mistake in statistical analysis: failure to account for dependence between random values. We then show how to correct the error using our statistical algebra. This example re-analyzes an earlier goal: to determine an unbiased estimator for the population variance, σ^2, from a sample of n values $\{y_i\}$.

As before, we start with a guess that our unbiased estimator of σ^2 is proportional to the sum squared deviation from the average (similar to the messy attempt we gave up on earlier). Since we know we must introduce μ into the computation, we choose to expand the sum by adding and subtracting μ:

$$\sum_{i=1}^{n} (y_i - \bar{y})^2 = \sum_{i=1}^{n} \left[(y_i - \mu) + (\mu - \bar{y}) \right]^2 = \sum_{i=1}^{n} \left[(y_i - \mu)^2 + 2(y_i - \mu)(\mu - \bar{y}) + (\mu - \bar{y})^2 \right].$$

Now we take ensemble averages, and bring them inside the summations:

$$\left\langle \sum_{i=1}^{n} (y_i - \bar{y})^2 \right\rangle = \sum_{i=1}^{n} \left\langle (y_i - \mu)^2 \right\rangle + 2\sum_{i=1}^{n} \left\langle (y_i - \mu)(\mu - \bar{y}) \right\rangle + n\left\langle (\mu - \bar{y})^2 \right\rangle . \tag{7.6}$$

All the ensemble averages on the RHS now equal their population averages. We consider each of the three terms in turn:

- $\left\langle (y_i - \mu)^2 \right\rangle_{ensemble} = \left\langle (y_i - \mu)^2 \right\rangle_{population} \equiv \sigma^2$, and the summation in the first term on the right is n times this.
- In the 2nd term on the RHS, the averages of both factors, $(y_i - \mu)$ and $(\mu - \bar{y})$, are zero, so we drop that term.
- $\left\langle (\mu - \bar{y})^2 \right\rangle = \left\langle (\bar{y} - \mu)^2 \right\rangle = \text{var}(\bar{y}) = \sigma^2 / n$.

Then:

$$\left\langle \sum_{i=1}^{n} (y_i - \bar{y})^2 \right\rangle = n\sigma^2 + \sigma^2 = (n+1)\sigma^2 \qquad \Rightarrow \qquad s^2 = \frac{1}{n+1} \sum_{i=1}^{n} (y_i - \bar{y})^2 \quad \text{(wrong!)} . \tag{7.7}$$

Clearly, this is wrong: the denominator should be $(n - 1)$. What happened? See if you can figure it out before reading further.

Really, stop reading now, and figure out what went wrong. Apply our statistical algebra.

The error is in the second bullet above: just because two RVs both average to zero doesn't mean their product averages to zero (see the average of a product, earlier). In fact, the average of the product must include their covariance. In this case, any given y_i correlates (positively) with $\bar{y}$ because $\bar{y}$ includes each y_i. Since the $\bar{y}$ is negated in the 2nd factor, the final correlation is negative. Then for a given k, using the bilinearity of covariance (μ is constant):

$$\operatorname{cov}\big((y_k-\mu),(\mu-\bar{y})\big) = -\operatorname{cov}\big(y_k,\bar{y}\big) = -\operatorname{cov}\left(y_k,\frac{1}{n}\sum_{j=1}^{n} y_j\right).$$

By assumption, the y_i are *independent* samples of y, and therefore have zero covariance between them:

$$\operatorname{cov}(y_k,y_j)=0,\quad k\neq j, \qquad \text{and} \qquad \operatorname{cov}(y_k,y_k)=\sigma^2.$$

Thus the only term in the summation over j that survives the covariance operation is when $j = k$:

$$\operatorname{cov}\big((y_k-\mu),(\mu-\bar{y})\big) = -\operatorname{cov}\left(y_k,\frac{1}{n}y_k\right) = -\frac{\sigma^2}{n}.$$

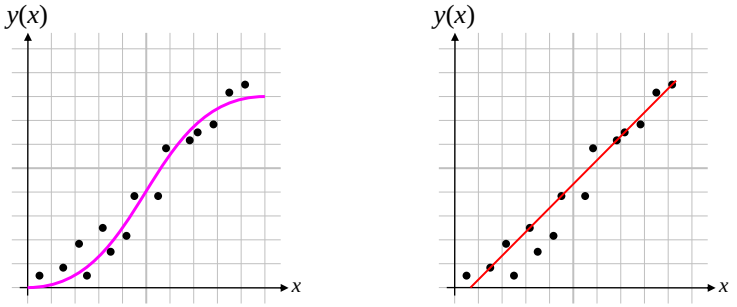
Therefore, equation (7.7) should include the summation term from (7.6) that we incorrectly dropped. The ensemble average of each term in that summation is the same, which we just computed, so the result is n times ($-\sigma^2/n$):

$$\left\langle \sum_{i=1}^{n}\big(y_i-\bar{y}\big)^2 \right\rangle = n\sigma^2 - 2n\frac{\sigma^2}{n} + \sigma^2 = (n-1)\sigma^2 \qquad \Rightarrow \qquad s^2 = \frac{1}{n-1}\sum_{i=1}^{n}\big(y_i-\bar{y}\big)^2 \quad \text{(right!)}.$$

Order is restored to the universe.

Introduction to Data Fitting (Curve Fitting)

Suppose we have an ideal process, with an ideal curve $y_{ideal}(x)$ mapping an independent variable x to a dependent variable y. Now we take a set of measurements of this process, that is, we measure a set of data pairs (x_i, y_i), Figure 7.7a.



(a) ideal curve, with non-ideal data (b) same data, with straight line guess

Figure 7.7 (a) Ideal curve with non-ideal data. (b) The same data with a straight line fit.

Suppose further we don't *know* the ideal curve, but we have to guess it. Typically, we make a guess (a **model**) of the general form of the curve from theoretical or empirical information, but we leave the exact parameters of the curve "free." For example, we may guess that the form of the curve is a straight line (Figure 7.7b):

$$y_{\text{mod}}(x) = mx + b,$$

but we leave the slope and intercept (m and b) of the curve as-yet unknown. (We might guess other forms, with other, possibly more parameters.) Then we **fit** our curve to the data, which means we compute the

values of *m* and *b* which "best" fit the data. "Best" means that the values of *m* and *b* minimize some measure of "error," called the **figure of merit**, compared to *all* other values of *m* and *b*. For data with constant uncertainty, the most common figure of merit is the sum-squared residual:

$$\text{sum-squared-residual} \equiv SSE \equiv \sum_{i=1}^{n} residual_i^2$$

$$= \sum_{i=1}^{n} \left(measurement_i - model_i \right)^2 = \sum_{i=1}^{n} \left(measurement_i - y_{\text{mod}}(x_i) \right)^2$$

$$where \quad y_{\text{mod}}(x) \text{ is our fitting function .}$$

The (*measurement – curve*) is often written as (*O – C*) for (*observed – computed*). In our example of fitting to a straight line, for given values of *m* and *b*, we have:

$$SSE \equiv \sum_{i=1}^{n} residual_i^2 = \sum_{i=1}^{n} \left(y_i - (mx_i + b) \right)^2 .$$

Curve fitting is the process of finding the values of all our unknown parameters such that they minimize some error function (for constant uncertainty, the sum-squared residual from our data).

The purpose of fitting, in general, is to estimate parameters, some of which may not have simple, closed-form estimators.

We discuss data with varying uncertainty later; in that more general case, we adjust parameters to minimize the χ^2 parameter.

An important *special case* of curve fitting is "linear fitting" (aka "linear regression"), which we discuss in much more detail later. Linear regression is *not* necessarily fitting to a straight line.

Goodness of Fit

Chi-Squared Distribution

Notational problem: in the literature, χ^2_ν sometimes means χ^2 with ν degrees of freedom, and sometimes means *reduced* χ^2 with ν degrees of freedom. I'm inclined to use $\chi^2(\nu; x)$ for the full χ^2 distribution, and $\chi^2_{red}(\nu; x)$ to denote explicitly the reduce χ^2 distribution, and χ^2_{red} for the statistic.

You don't really need to understand the χ^2 distribution to understand the χ^2 statistic, but we start there because it's helpful background.

Notation: $X \in D(x)$ means X is a random variable with probability distribution function (PDF) = $D(x)$.

$X \in kD(x)$ means X is an RV which is k (a constant) times an RV which is $\in D$.

Chi-squared (χ^2) distributions are a family of distributions characterized by one parameter, called ν (Greek nu), the "degrees of freedom." (Contrast with the gaussian distribution, which has two parameters, the mean, μ, and standard deviation, σ.) So we say "chi-squared is a 1-parameter distribution." ν is almost always an integer. The simplest case is $\nu = 1$: if we define a new random variable X from a gaussian random variable χ, as:

$$X = \chi^2, \qquad where \quad \chi \in gaussian(\mu = 0, \sigma^2 = 1), \quad \text{i.e.} \quad \text{avg} = 0, \text{variance} = 1,$$

then X has a χ^2_1 distribution. I.e., $\chi^2(\nu=1; x)$ is the probability distribution function of the square of a zero-mean unit-variance gaussian.

For general ν, $\chi^2(\nu; x)$ is the PDF of the *sum* of the squares of ν *independent* gaussian random variables:

$$Y = \sum_{i=1}^{\nu} \chi_i^2, \qquad \textit{where} \quad \chi_i \in \textit{gaussian}(\mu = 0, \sigma^2 = 1), \quad \text{i.e.} \quad \text{avg} = 0, \text{std deviation} = 1.$$

Thus, the random variable Y above has a $\chi^2(\nu)$ distribution (Figure 7.8). Chi-squared random variables are always ≥ 0, since they are the sums of squares of gaussian random variables. Since the gaussian distribution is continuous, the chi-squared distributions are also continuous.

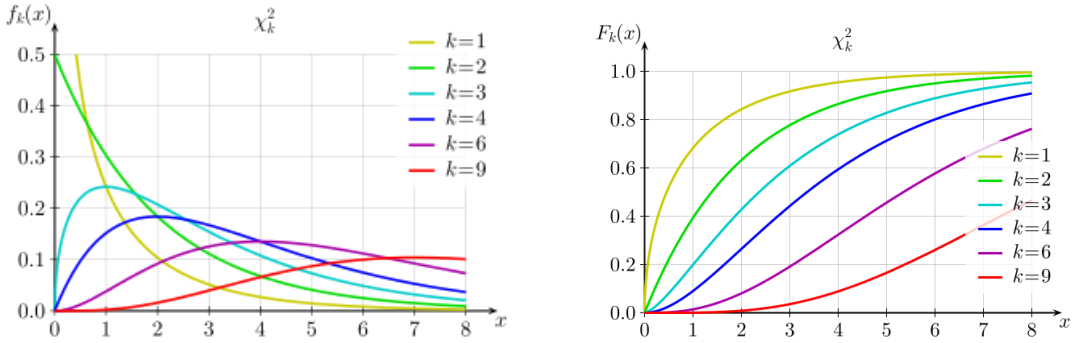


Figure 7.8 PDF (left) and CDF (right) of some $\chi^2(k)$ distributions. $\chi^2(1; 0) \to \infty$. $\chi^2(2; 0) = ½$. [http://en.wikipedia.org/wiki/Chi-squared_distribution]

From the definition, we can also see that the sum of two chi-squared random variables is another chi-squared random variable:

$$\text{Let} \qquad A \in \chi^2(n),\ B \in \chi^2(m), \qquad \text{then} \qquad A + B \in \chi^2(n+m).$$

By the central limit theorem, this means that for large ν, chi-squared itself approaches gaussian. However, a χ^2 random variable (RV) is always positive, whereas any gaussian PDF extends to negative infinity.

We can show that:

$$\left\langle \chi^2(1) \right\rangle = 1, \qquad \text{var}\left(\chi^2(1)\right) = 2$$

$$\Rightarrow \qquad \left\langle \chi^2(\nu) \right\rangle = \nu, \qquad \text{var}\left(\chi^2(\nu)\right) = 2\nu \ \Leftrightarrow\ \text{dev}\left(\chi^2(\nu)\right) = \sqrt{2\nu}$$

We don't usually need the analytic form, but for completeness [WMMY 2007 p200b]:

$$\text{PDF}: \quad \chi^2(\nu; x) = \frac{x^{\nu/2-1} e^{-x/2}}{\Gamma(\nu/2)\, 2^{\nu/2}}. \qquad \text{For } \nu \geq 3, \text{ there is a maximum at } \nu - 2.$$

For ν = 1 or 2, there is no maximum, and the PDF is monotonically decreasing.

Chi-Squared Statistic

As seen above, χ^2 is a continuous probability distribution. However, there is a goodness-of-fit test which computes a *statistic* also called "chi-squared." This statistic is from a distribution that is often close to a χ^2 distribution, but be careful to distinguish between the *statistic* χ^2 and the *distribution* χ^2.

The chi-squared statistic is not *required* to be from a chi-squared distribution, though it may be. All the chi-squared statistic really requires is that the variances of our residuals add, which is to say that our residuals are uncorrelated (not necessarily independent, though independent implies uncorrelated).

The χ^2 statistic is valid for any distribution of uncorrelated residuals.
The χ^2 statistic has a χ^2 distribution only if the residuals are gaussian.

However, for large ν, the χ^2 distribution approaches gaussian, as does the sum of many values of any distribution. Therefore:

The χ^2 distribution is a reasonable approximation to the distribution of any χ^2 statistic with $\nu >\sim 20$, even if the residuals are not gaussian [ref??].

To illustrate, consider a set of measurements, each with uncertainty u. Then if the set of {(*measurement* – *model*)/u} has zero mean, it has standard-deviation = 1, even for non-gaussian residuals:

$$\text{Define:}\quad \text{dev}(X) \equiv \text{standard deviation of random variable } X\text{, also written } \sigma_X\text{,}$$

$$\text{var}(X) \equiv \big(\text{dev}(X)\big)^2 = \text{variance of random variable } X\text{, also written } \sigma_X^2\text{.}$$

$$\text{dev}\left(\frac{residual}{u}\right) = 1 \qquad \Rightarrow \qquad \text{var}\left(\frac{residual}{u}\right) = 1\text{.}$$

As a special case, but *not* required for a χ^2 statistic, *if* our residuals are gaussian:

$$\frac{residual}{u} \in gaussian(0,1) \quad \Rightarrow \qquad \left(\frac{residual}{u}\right)^2 \in \chi^2(1)\text{.}$$

When fitting a curve to data, the uncertainties often vary from measurement to measurement (heteroskedastic data). In that case, we are fitting a curve to data triples: (x_i, y_i, u_i). Still, the error divided by uncertainty for any single measurement is unit deviation (for *any* distribution of residuals):

$$\text{dev}\left(\frac{residual_i}{u_i}\right) = 1, \qquad \text{and} \qquad \text{var}\left(\frac{residual_i}{u_i}\right) = 1, \qquad \text{for all } i\text{.}$$

If we have n measurements, with uncorrelated residuals, then because variances add:

$$\text{var}\left(\sum_{i=1}^{n} \frac{residual_i}{u_i}\right) = n. \qquad \text{For gaussian residuals:} \quad \sum_{i=1}^{n}\left(\frac{residual_i}{u_i}\right)^2 \in \chi^2(n)\text{.}$$

Returning to our ideal process from Figure 7.7a, with an ideal curve mapping an independent variable x to a dependent variable y, we now take a set of measurements y_i with known uncertainties u_i. Then our dimensionless statistic χ^2 for the *ideal* curve is defined as:

$$\chi^2 \equiv \sum_{i=1}^{n}\left(\frac{residual_i}{u_i}\right)^2 = \sum_{i=1}^{n}\left(\frac{measurement_i - ideal_i}{u_i}\right)^2 \qquad \left[\begin{array}{l}\text{If gaussian residuals and}\\ u_i \text{ are accurate, } \chi^2 \in \chi^2(n)\end{array}\right]\text{.}$$

For zero-mean errors, this χ^2 has an average of n:

$$\left\langle \chi^2 \right\rangle = \left\langle \sum_{i=1}^{n}\left(\frac{residual_i}{u_i}\right)^2 \right\rangle = n\text{.}$$

Now suppose we have **fit** a curve to our data, i.e. we guessed a functional form, and found the parameters which minimize the χ^2 statistic for that form with our data. If our fit is good, then our curve is very close to the ideal (or "real") dependence curve for y as a function of x, and our errors will be essentially random (no systematic error). We now compute the χ^2 statistic for our *fit*:

$$\chi^2 \equiv \sum_{i=1}^{n}\left(\frac{residual_i}{u_i}\right)^2 = \sum_{i=1}^{n}\left(\frac{measurement_i - fit_i}{u_i}\right)^2\text{.}$$

If our fit is good, the number χ^2 will likely be close to n. (We will soon modify the distribution of the χ^2 statistic, but for now, it illustrates our principle.)

If our fit is bad, there will be significant systematic fit error in addition to our random noise error, and our χ^2 statistic will be much larger than n. Summarizing:

If χ^2 is close to n, then our fit residuals are no worse than our measurement uncertainties, and the fit is "good." If χ^2 is much larger than n, then our fit residuals are worse than our measurement uncertainties, so our fit must be "bad."

Degrees of freedom: So far we have ignored the "degrees of freedom" of the fit, which we now motivate. (We prove this in detail later.) Consider again a straight-line fit to the data (Figure 7.7b). We are free to choose our parameters m and b to define our "fit-line." But in a set of n data points, we *could* (if we wanted) choose our m and b to *exactly* go through two of the data points. This guarantees that two of our fit residuals are zero. If n is large, it won't significantly affect the other residuals, and instead of χ^2 being the sum of n squared-residuals, it is approximately the sum of $(n - 2)$ squared-residuals. In this case, $\left\langle \chi^2 \right\rangle \approx n-2$. A rigorous analysis (given later) shows that for the best fit line (which probably doesn't go through *any* of the data points), and gaussian residuals, then $\left\langle \chi^2 \right\rangle = n-2$, *exactly*. This concept generalizes quite far:

- Even if we don't fit the line exactly to any 2 points;
- Even if our fit-curve is not a line;
- Even if we have more than 2 fit parameters;

the effect is to reduce the χ^2 statistic to be a sum of less than n squared-residuals. The effective number of squared-residuals in the χ^2 sum is called the **degrees of freedom (dof)**, and is given by:

$$dof \equiv n - (\#\text{ fit parameters}).$$

Thus for gaussian residuals, and p linear fit parameters, the parameters of our **χ^2 statistic** are really:

$$\left\langle \chi^2 \right\rangle = dof = n - p, \qquad \text{dev}\left(\chi^2\right) = \sqrt{2(dof)} = \sqrt{2(n-p)}\,. \tag{7.8}$$

For nonlinear fits, we usually use the same formula as an *approximation*.

Reduced Chi-Squared Statistic

Since it is awkward for everyone to know n, the number of points in our fit, it is convenient to define a "goodness-of-fit" parameter that is less dependent of n. We simply divide our chi-squared statistic by *dof*, to get the **reduced chi-squared statistic**. Then it has these properties:

$$\text{reduced } \chi^2 \equiv \frac{\chi^2}{dof} = \frac{1}{dof}\sum_{i=1}^{n}\left(\frac{measurement_i - fit_i}{u_i}\right)^2 \qquad \Rightarrow$$

$$\left\langle \text{reduced } \chi^2 \right\rangle = \frac{\left\langle \chi^2 \right\rangle}{dof} = \frac{dof}{dof} = 1,$$

$$\text{dev}\left(\text{reduced } \chi^2\right) = \frac{\text{dev}\left(\chi^2\right)}{dof} = \frac{\sqrt{2(dof)}}{dof} = \sqrt{\frac{2}{dof}}\,.$$

If reduced χ^2 is close to 1, the fit is "good." If reduced χ^2 is much larger than 1, the fit is "bad." By "much larger" we mean several deviations away from 1, and the deviation gets smaller with larger *dof* (larger n).

Of course, our confidence in χ^2 or reduced-χ^2 depends on how many data points went into computing it, and our confidence in our measurement uncertainties, u_i. Remarkably, one reference on χ^2 [which I don't remember] says that our estimates of measurement uncertainties, u_i, should come from a sample of at least *five*! That seems to me to be quite small to have much confidence in u.

The *reduced* χ^2 statistic is a convenient measure of the "misfit + noise" to "noise:"

$$\text{reduced } \chi^2 \equiv \frac{\chi^2}{\nu} = \frac{1}{\nu}\sum_{i=1}^{n}\left(\frac{y_i - y_{\text{mod},i}}{u_i}\right)^2 \sim \frac{\text{misfit} + \text{noise}}{\text{noise}} \qquad \textit{where} \quad \nu \equiv \text{degrees of freedom}$$

Each term of χ^2 is normalized to the noise of that term. If your fit is perfect, reduced χ^2 will be around 1. If you have misfit *and* noise, then reduced χ^2 is greater than 1.

Guidance Counselor: Computer Code to Fit Data

Finding model parameters from data is called **fitting** the model to data. The "best" parameters are chosen by minimizing some figure-of-merit function. For example, this function might be the sum-squared error (between the data and the model), or the χ^2 fit parameter. Generic fitting (or "optimization") algorithms are available off-the-shelf, e.g. [Numerical Recipes]. However, they are sometimes simplistic, and in the real world, often fail with arithmetic faults (overflow, underflow, domain-error, etc). The fault (no pun intended) lies not in their algorithm, but in their failure to tell you what you need to do to avoid such failures:

Your job is to write a bullet-proof figure-of-merit function.

This is harder than it sounds, but quite do-able with proper care.

As an example, I once wrote code to fit a 3-parameter sinusoid (frequency, amplitude, phase) to astronomical data: measures of a star's brightness at irregular times. That seems pretty simple, yet it was fraught with problems. The measurements were very noisy, which leads to lots of local minima. In some cases, the optimizer would choose an amplitude for the sinusoid that had a higher sum-of-squares than the sum-of-squares of the data! This amplitude is clearly "too big," but it is hard to know ahead of time how big is "too big." Furthermore, the "too big" threshold varies with the frequency and phase parameters, so you cannot specify ahead of time an absolute "valid range" for amplitude. Therefore, I had to provide "guiding errors" in my figure-of-merit function to "guide" the optimizer to a reasonable fit under *all* conditions.

Computer code for finding the best-fit parameters is usually divided into two pieces, one piece you buy, and one piece you have to write yourself:

- You buy a generic optimization algorithm, which varies parameters without knowledge of what they mean, looking for the minimum figure-of-merit (FOM). For each trial set of parameters, it calls your FOM function to compute the FOM as a function of the current trial parameters.
- You write the FOM function which computes the FOM as a function of the given parameters.

Generic optimizers usually minimize the figure-of-merit, consistent with the FOM being a "cost" or "error" that we want reduced. (If instead, you want to maximize a FOM, return its negative to the minimizer.)

If your optimizer allows you to specify valid ranges for parameters, *and* if your fit parameters have valid ranges that are independent of each other, then you don't need the methods here for your FOM function. If your optimizer (like many) does *not* allow you to limit the range of parameters, or if your parameters have valid ranges that depend on each other, then you need the following methods to make a bullet-proof FOM. In either case, this section illustrates how many seemingly simple calculations can go wrong in unexpected ways.

A bullet-proof FOM function requires only two things:

- Proper validation of all parameters.
- A properly "bad" FOM for invalid parameters (a "guiding error").

Guiding errors are similar to penalty functions, but they operate *outside* the valid parameter space, rather than inside it.

Parameter interdependence: Here is a simple example of a model where independent limits *don't* work; the parameters are necessarily interdependent. Consider a parametrized function of time $f(t; a, b)$, where t is the argument, and a and b are the two parameters to be varied for optimization, with $t, a, b \in [0, 1]$. Perhaps they represent the times of a first event at $t = a$, and a second event at $t = b$. The function f requires

that a and b are limited to the range [0, 1], but *furthermore*, a must be < b. The valid values of a and b are shown in Figure 7.9a.

Generic optimizers know nothing about your figure-of-merit (FOM) function, or its behavior,
and your FOM usually knows nothing about the optimizer, or its algorithms.
No standard optimizer allows for interdependence of the valid ranges of parameters.

The example below shows only a single parameter, and its guiding error values, but the concept is readily applied to the 2D example here, or any interdependence of parameters to be optimized. We will return to this 2D example after the 1D one below.

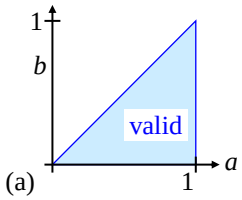


Figure 7.9 (a) Valid region of parameters a and b.

A simple 1D example: Suppose you wish to numerically find the minimum of the 1-parameter figure-of-merit function Figure 7.10a. Suppose the physics is such that only $p > 1$ is sensible.

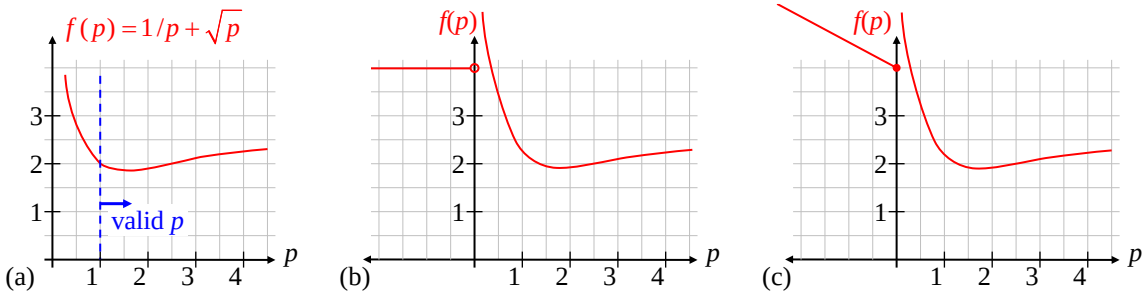


Figure 7.10 (a and b) Bad figure-of-merit (FOM) functions. (c) A bullet-proof FOM.

Your optimization-search algorithm will try various values of p, evaluating $f(p)$ at each step, looking for the minimum. You might write your FOM function like this:

```
fom(p) = 1./p + sqrt(p)
```

But the search function knows nothing of p, or which values of p are valid. It may well try $p = -1$. Then your function crashes with a domain-error in the sqrt( ) function. You fix it with (Figure 7.10b):

```
float fom(p)
      if(p < 0.)  return 4.
      return 1./p + sqrt(p)
```

Since you know 4 is much greater than the true minimum, you hope this will fix the problem. You run the code again, and now it crashes with divide-by-zero error, because the optimizer tried $p = 0$. Easy fix:

```
float fom(p)
      if(p <= 0.)  return 4.
      return 1./p + sqrt(p)
```

Now the optimizer crashes with an overflow error, $p < -(\text{max\_float})$. The big flat region to the left confuses the optimizer. It searches negatively for a value of p that makes the FOM increase, but it never finds one, and gets an overflow trying. The FOM's flat value for $p \le 0$ is no good. It needs to grow upward to the left to provide guidance to the optimizer (Figure 7.10c):

```
float fom(p)
      if(p <= 0.)  return 4. + fabs(p - 1.)    // fabs() = absolute value
      return 1./p + sqrt(p)
```

Now the optimizer says the minimum is 4 at $p = -10^{-6}$. It found the local "minimum" just to the left of zero. Your function is still ill-behaved. Since only $p > 1$ is sensible, you make yet another fix (Figure 7.11a):


```
float fom(p)
      if(p <= 1.)  return 4. + fabs(p - 1.)
      return 1./p + sqrt(p)
```

Finally, the optimizer returns the minimum FOM of 1.89 at $p = 1.59$. After 5 tries, you have made your FOM function bullet-proof:

A bullet-proof FOM has only one minimum, which it monotonically approaches from all sides of all parameters, even with *invalid* parameters, and it *never* crashes on any parameter set.

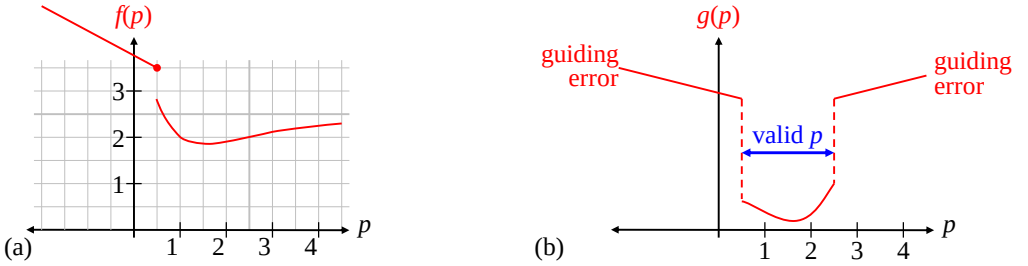


Figure 7.11 (a) Finally, a bullet-proof FOM. (b) "Guiding errors" lead naturally to a valid solution, and are better than traditional penalty functions.

In this example, the FOM is naturally bullet-proof from the right. However, if it weren't, you would include the absolute value of $(p - 1)$ on the error return value, to provide a V-shape which guides the optimizer into the valid range from either side. Such "guiding errors" are analogous to so-called penalty functions, but better, because they take effect only for *invalid* parameter choices, thus leaving the valid parameter space completely free for full optimization.

Multi-parameter FOMs: Most fit models use several parameters, p_i, and the optimizer searches over all of them iteratively to find a minimum. Your FOM function must be bullet-proof over *all* parameters: it must check each parameter for validity, and *must* return a large (guaranteed unoptimal) result for invalid inputs. It must also *slope the FOM toward valid values*, i.e. provide a "restoring force" to the invalid parameters toward the valid region (Figure 7.11b). Typically, with multiple parameters p_i, one uses:

$$\text{guiding\_bad\_FOM} = big\,\# + \sum_{i=1}^{M} \left| p_i - valid_i \right| \qquad where \quad valid_i \equiv \text{a valid value for } p_i \,.$$

This guides the minimization search when any parameter is outside its valid range.

We return now to the multi-parameter function of Figure 7.9, where the valid ranges of a and b are interdependent. We can say that b is allowed to be anywhere in [0, 1], and $valid_a$ is (say) b/2. Then our "guiding_bad_FOM" is:

$$\text{guiding\_bad\_FOM} = big\,\# + \left| b - 0.5 \right| + \left| a - b/2 \right| .$$

Alternatively, we could just as well say that $a \in [0, 1]$, and b must be between a and 1: $valid_b = (1 + a)/2$.

A final note:

The "big #" for invalid parameters may need to be much bigger than you think.

In my dissertation research, I used reduced χ^2 as my FOM, and the true minimum FOM is near 1. I started with 1,000,000 as my "big #", but it wasn't big enough! I was fitting to histograms with nearly a thousand counts in several bins. When the trial model bin count was small, the error was about 1,000, and the sum-squared-error over several bins was > 1,000,000. This caused the optimizer to settle on an invalid set of parameter values as the minimum! I had to raise "big #" to 10^9.

8 Multiple Linear Regression

Review of Multiple Linear Regression

Most intermediate statistics texts cover multiple linear regression, e,g, [W&M p353], but we remind you of some basic concepts here. Linear regression is also called "linear fitting." **Linear regression** is a special case of curve fitting where the model is a linear combination of basis functions, and the fit parameters are the coefficients. A simple example is:

$$y_{\text{mod}}(x) = b_0 + b_1 f_1(x) + b_2 f_2(x) + \ldots + b_k f_k(x) = b_0 + \sum_{k=1}^{p-1} b_k f_k(x), \qquad \{b_k\} \equiv \text{fit parameters}. \qquad (8.1)$$

For your data, you measure n observable y_i vs. an independent variable x_i, i.e. you measure $y_i \equiv y(x_i)$ for some set of $x = \{x_i\}$, and each y_i has an uncertainty u_i. You use multiple linear regression to find the best-fit coefficients b_k of the basis functions f_k which compose the model, $y_{mod}(x)$. The basis functions *need not* be orthonormal. The independent variable might be position, time, or anything else. Note that:

Linear regression is *not* limited to fitting data to a straight line.

Fitting data to a line is often called "fitting data to a line" (seriously). (Unfortunately, some references use the term "linear model" to mean a straight-line model; this different use of the term "linear" adds confusion.) We now show that there is no mathematical difference between fitting to a line and *linear* fitting to an arbitrary function.

The quirky part is understanding what are the "predictors" (which may be random variables) to which we perform the regression. As above, the predictors can be arbitrary functions of a single independent variable, but they may also be arbitrary functions of *multiple* independent variables, called **multiple linear regression**. For example, a model for the speed of light in air varies with 3 independent variables: temperature, pressure, and humidity:

$$c = c(T, P, H)\ .$$

Suppose we take n measurements of c at various combinations of T, P, and H. Then our data consists of quintuples: (T_i, P_i, H_i, c_i, u_i), i = 1, ... n, where u_i is the uncertainty in c_i. We might propose a linear model with p = 5 parameters:

$$c(T, P, H) = b_0 + b_1 T + b_2 P + b_3 H + b_4 TP\ .$$

The model is linear because it is a linear combination of arbitrary functions of T, P, and H. The last term above handles an interaction between temperature and pressure. Our linear regression model has 3 independent variables, and 4 fit functions: T, P, H, and TP (the product of T and P). Written in terms of (8.1):

$$f_1(T, P, H) = T, \quad f_2(T, P, H) = P, \quad f_3(T, P, H) = H, \quad f_4(T, P, H) = TP\ .$$

For simplicity, this chapter illustrates only one independent variable, but all the results work straightforwardly with multiple independent variables, as above. This is because:

It's the predictors that matter, not the independent variables that go into them.

[Units of [b] and [x], predictors in x_{mi} form.??]

We Fit to the Predictors, *Not* the Independent Variable

In general, we fit to multiple functions of the independent variables, usually not directly to the independent variables themselves. Figure 8.1 shows an example fit to a model, with t as the lone independent variable. The example model is $f_1(t) = \sin(\omega t)$ for some given, fixed ω. Then the predictors f_{1i} are:

$$y_{\text{mod}}(t) = b_1 f_1(t) = b_1 \sin(\omega t) \qquad \Rightarrow \qquad f_{1i} \equiv \sin(\omega t_i) \ \ \text{(predictors)}.$$

There is only 1 fit-function in this example; the predictors are the set of $\{f_{1i}\}$, and given them, there is no longer any reference to the independent variable t. The fit is to the predictors, not to the independent values t_i. Furthermore, the fit knows nothing of the *functions* f_k, it only knows their values f_{ki} at the measurement points t_i. In some cases, there is no independent variable; there are only predictors.

> The existence of a "good" fit of a model to measurements does *not* imply any sort of causation by the predictors on the measurement.

For example, the hands of a clock predict very well the brightness of the sky. But the hands do not *cause* the sky brightness, and changing their position will not change the sky brightness.

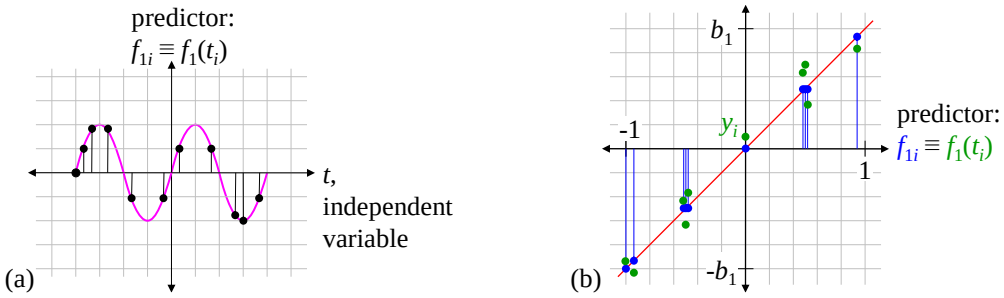


Figure 8.1 (a) Example predictor: an arbitrary function of independent variable t. (b) Linear fit to the predictor is a straight line. The fit is *not* to t itself. Even if the t_i are evenly spaced, the predictors are not. Note that the predictor values of –0.5 and +0.5 each occur 3 times. This shows a good fit: the measured values (green) are close to the model values.

We provide a brief overview of linear fitting here, and introduce our notation. A linear fit uses a set of p coefficients, b_1, ... b_p, as fit parameters in a model with *arbitrary* fit functions. The "model" fit may be defined as:

$$y_{\text{mod}}(x) \equiv b_1 f_1(x) + b_2 f_2(x) + \ldots + b_p f_p(x) = \sum_{k=1}^{p} b_k f_k(x) \qquad (\text{without } b_0)\ .$$

Note that a linear fit does *not* require that y is a straight-line function of x.

There is an important reason (discussed later) to include a fit-function that is just $f_0(x) = 1$, which gives a coefficient b_0 that is the constant offset beyond all the other fit-functions. In this case, there are p–1 *remaining* fit functions, since p is always the *total* number of fit parameters:

$$y_{\text{mod}}(x) \equiv b_0 + b_1 f_1(x) + b_2 f_2(x) + \ldots + b_{p-1} f_{p-1}(x) = b_0 + \sum_{k=1}^{p-1} b_k f_k(x)\ .$$

This is just a different notation than the first model given above. Therefore, the first form is completely general, and includes the second. However:

> Anything true of the first form is also true of the second,
> but the second form has important properties beyond those of the first form.

We use both forms, depending on whether our model includes b_0 or not.

Summarizing:

1. Multiple linear regression predicts the values of some random variable y from p (possibly correlated) sets of **predictors**, f_{ki}, k = 1, 2, ... p, by fitting for the coefficients b_k. The predictors may or may not be random variables.
2. In some cases, the predictors are arbitrary functions of a single independent variable, say t_i: $f_{ki} \equiv f_k(t_i)$. We assume that all the t_i, y_i, and all the f_k are *given*, which means all the $f_{ki} \equiv f_k(t_i)$ are given. In other cases, there are multiple independent variables, and multiple functions of those variables.

3. It's *linear* prediction, so our prediction model is that y is a *linear combination* of the predictors, $\{f_{ki}\}$:

$$y_{\text{mod}} = b_0 + b_1 f_1 + b_2 f_2 + \ldots + b_{p-1} f_{p-1} = b_0 + \sum_{k=1}^{p-1} b_k x_k \quad \left(\text{or} \quad \sum_{k=1}^{p} b_k x_k \right).$$

In the first form, we have included b_0 as a fitted constant, and there are p fit parameters: b_0 ... b_{p-1}. This is quite common, in practice, but not always necessary. The fit functions could just as well be labeled with k = 1, ... p. Note that this *prediction model* has no subscripts of i , because the model applies to *all* f_k and y values.

4. Linear regression assumes that our *measurements* includes noise (this is our "measurement model"):

$$y_i = y_{true,i} + noise_i$$

For a given set of measurements, the $noise_i$ are fixed, but unknown. Over an ensemble of many sets of measurements, the $noise_i$ are random variables. The **measurement uncertainty** is defined as the 1-sigma deviation of the noise (which *need not* be gaussian):

$$u_i \equiv \text{dev}\left(noise_i\right).$$

Note that the measurement model assumes *additive* noise (as opposed to, say, multiplicative noise). By definition, the *noise* cannot be modeled.

5. The residuals ε_i are the sum of measurement noise + unmodeled behavior. Thus our measurements can be written:

$$y_i = b_0 + b_1 f_{1i} + b_2 f_{2i} + \ldots + b_{p-1} f_{p-1,i} + \varepsilon_i = \underbrace{b_0 + \left(\sum_{k=1}^{p-1} b_k f_{ki} \right)}_{\text{model } y_{\text{mod},i}} + \varepsilon_i, \qquad i = 1, 2, \ldots n.$$

6. Multiple linear regression determines the unknown regression coefficients b_0, b_1, ... b_{p-1} from n samples of the y and each of the f_{ki}. The b_k are chosen to optimize some figure-of-merit, which is most often minimizing the sum-squared-residual:

$$\arg\min_{\{b_k\}} \sum_{i=1}^{n} \varepsilon_i^{\,2}.$$

Examples: For fitting to a line, in our notation, our model is:

$$y_{\text{mod}}(x) = b_0 + b_1 x \qquad \text{i.e.,} \qquad f_1(x) = x.$$

There are p = 2 parameters: b_0 and b_1.

For a sinusoidal periodogram analysis, we typically have a set of measurements y_i at a set of times t_i. Given a trial frequency ω, we wish to find the least-squares cosine and sine amplitudes that best fit our data. Without a b_0 (a bad idea), we would have:

$$p = 2: \quad f_1 = \cos, \quad f_2 = \sin, \qquad f_{1i} = \cos(\omega t_i), \quad f_{2i} = \sin(\omega t_i), \quad i = 1, 2, \ldots n,$$

and our fit model is:

$$y_{\text{mod}}(t) = b_1 \cos(\omega t) + b_2 \sin(\omega t).$$

(More on omitting b_0 below).

Fitting to a Polynomial is Multiple Linear Regression

Fitting a polynomial to data is a simple example of multiple linear regression [W&M p357] (see also the Numerical Analysis section for exact polynomial "fits"). We are predicting y_i from powers of (say) t_i. As

such, we let $f_{ki} = (t_i)^k$, and proceed with standard multiple linear regression by solving the p simultaneous standard equations (derived later) for b_k:

$$k = 0: \qquad b_0 n + b_1 \sum_{i=1}^{n} t_i + b_2 \sum_{i=1}^{n} t_i^2 + \ldots + b_{p-1} \sum_{i=1}^{n} t_i^{p-1} = \sum_{i=1}^{n} y_i \ .$$

$$k = 1, \ldots p-1: \qquad b_0 \sum_{i=1}^{n} t_i^k + b_1 \sum_{i=1}^{n} t_i^{k+1} + b_2 \sum_{i=1}^{n} t_i^{k+2} + \ldots + b_{p-1} \sum_{i=1}^{n} t_i^{k+p-1} = \sum_{i=1}^{n} t_i^k y_i \ .$$

Note that the second equation works for $k = 0$ as well, so we could have omitted the first equation.

Homoskedastic Case: All Measurements Have the Same Uncertainty

There are two major categories of uncertainty in multiple regression: all measurements have the same uncertainty (homoskedastic), and each measurement has its own uncertainty (heteroskedastic). In science, the heteroskedastic case is far more common, and only slightly more complicated than the homoskedastic case. Nonetheless, some 800+ page statistics books completely omit the heteroskedastic case of linear regression. We here follow tradition, and derive the slightly simpler homoskedastic case first, to illustrate the concepts. We then rederive nearly all the results for the heteroskedastic case, which is more useful in practice. Furthermore, there is a transformation from heteroskedastic measurements into an equivalent set of homoskedastic measurements, which are then subject to all of the following homoskedastic results.

We proceed along these steps:

- The raw sum of squares identity.
- The geometric view of a least-squares fit.
- The ANOVA sum of squares identity.
- The failure of the ANOVA sum of squares identity.
- Later, we provide the equivalent formulas for data with individual uncertainties.

Nowhere in this section do we make any assumptions at all about the residuals;
we do not assume they are gaussian, nor independent, nor even random.

For a set of n pairs (x_i, y_i), the "fit" means finding the values of b_k that together minimize the sum-squared residual (appropriate if all measurements have the same uncertainty):

$$\text{define:} \qquad y_{\text{mod},i} \equiv y_{\text{mod}}(x_i) = \sum_{k=1}^{p} b_k f_k(x_i) \ .$$

$$\text{minimize:} \qquad SSE \equiv \sum_{i=1}^{n} \left(y_i - y_{\text{mod},i} \right)^2 \equiv \sum_{i=1}^{n} \varepsilon_i^2 \ , \qquad \varepsilon_i \equiv y_i - y_{\text{mod},i} \ .$$

Note that the fit residuals ε_i include both unmodeled behavior (aka "misfit"), as well as noise (which, by definition, cannot be modeled). So even if your *noise* is gaussian, your residuals are not.

For least-squares fitting, we show later that we must simultaneously solve the following p linear equations in p unknowns for the b_k [W&M p355]:

$$b_0 n + b_1 \sum_{i=1}^{n} f_{1i} + b_2 \sum_{i=1}^{n} f_{2i} + \ldots + b_{p-1} \sum_{i=1}^{n} f_{p-1,i} = \sum_{i=1}^{n} y_i \ .$$

And for each $k = 1, 2, \ldots p-1$:

$$b_0\sum_{i=1}^{n} f_{ki} + b_1\sum_{i=1}^{n} f_{ki}f_{1i} + b_2\sum_{i=1}^{n} f_{ki}f_{2i} + \ldots + b_{p-1}\sum_{i=1}^{n} f_{ki}f_{p-1,i} = \sum_{i=1}^{n} f_{ki}y_i \ .$$

Again, all the y_i and f_{ki} are given. Therefore, all the sums above are constants, on both the left and right sides. In matrix form, we solve for column vector $\mathbf{b} \equiv (b_0, b_1, \ldots b_{p-1})^T$ from:

$$\mathbf{Xb} = \mathbf{y} \quad \text{or} \quad \begin{bmatrix} n & \sum f_{1i} & \cdots & \sum f_{p-1,i} \\ \sum f_{1i} & \sum (f_{1i})^2 & \cdots & \sum f_{1i}f_{p-1,i} \\ \vdots & \vdots & \ddots & \vdots \\ \sum f_{p-1,i} & \sum f_{p-1,i}f_{1i} & \cdots & \sum (f_{p-1,i})^2 \end{bmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{p-1} \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum f_{1i}y_i \\ \vdots \\ \sum f_{p-1,i}y_i \end{pmatrix}.$$

Algorithms for solving simultaneous linear equations are well-known, so we do not address them here.

The Raw Sum-of-Squares Identity

The sum of squares (SSQ) identity is a crucial tool of linear fitting (aka linear regression). It underlies many of the basic statistics of multiple linear regression and Analysis of Variance (or ANOVA). For straight-line fitting, the sum of squares identity can be used to define the "coefficient of determination" (and the associated "correlation coefficient"). For all linear fitting, the SSQ identity provides the basis for the F-test and t-test of fit parameter significance. The SSQ identity leads to ANOVA, which is a standardized way of organizing multiple regression results.

There are two forms of SSQ identity: raw and ANOVA. Most references do not consider the *raw* sum of squares (SSQ) identity (aka the "uncorrected SSQ identity", or SSQ identity "uncorrected for the mean" [W&M8]). We present it first because it provides a basis for the more-common ANOVA SSQ identity, and it is sometimes useful in its own right. Consider a set of data (x_i, y_i), $i = 1, \ldots n$. Conceptually, the SSQ identity says the sum of the squares of the y_i can be partitioned into a sum of squares of model values plus a sum of squares of **residuals** (often called "errors"):

$$\text{(raw)} \quad SST = SSA + SSE: \quad \sum_{i=1}^{n} {y_i}^2 = \sum_{i=1}^{n} {y_{\text{mod},i}}^2 + \sum_{i=1}^{n} \underbrace{\left(y_i - y_{\text{mod},i}\right)^2}_{\text{residuals} \equiv \varepsilon_i} \equiv \sum_{i=1}^{n} {y_{\text{mod},i}}^2 + \sum_{i=1}^{n} {\varepsilon_i}^2 \ . \qquad (8.2)$$

(The term "errors" can be misleading, so in words we always use "residuals." However, we write the term as SSE, because that is so common in the literature.) The SSQ identity is *only* true for a least-squares linear fit to a parametrized model, and has some important non-obvious properties. We start with some examples of the identity, and provide simple proofs later.

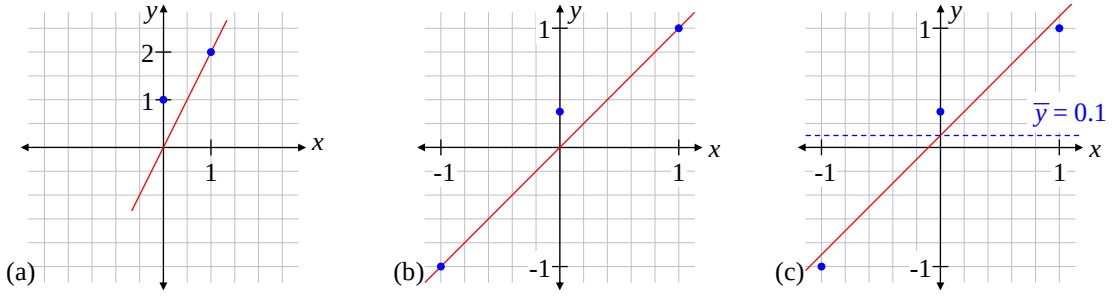


Figure 8.2 (a) Two data points, $n = 2$, and best-fit 1-parameter model. (b) Three data points, $n = 3$, and best-fit 1-parameter model. (c) Three data points, $n = 3$, and best-fit 2-parameter model.

Example: $n = 2$, $p = 1$: Consider a data set of two measurements (0, 1), and (1, 2) (Figure 8.2a). We choose a 1-parameter model:

$$y_{\text{mod}}(x) = b_1 x \ .$$

The best fit line is $b_1 = 2$, and therefore $y(x) = 2x$. (We see this because the model is forced through the origin, so the residual at $x = 0$ is fixed. Then the least squares residuals are those that minimize the error at $x = 1$, which we can make zero.) Our raw sum-of-squares identity (8.2) is:

$$\underbrace{1^2 + 2^2}_{SST} = \underbrace{\left(0^2 + 2^2\right)}_{SSA} + \underbrace{\left(1^2 + 0^2\right)}_{SSE} \qquad \rightarrow \qquad 5 = 4 + 1 .$$

Example: $n = 3$, $p = 1$: Consider a data set of three measurements (–1, –1), (0, 0.3), and (1, 1) (Figure 8.2b). We choose a 1-parameter model:

$$y_{\text{mod}}(x) = b_1 x .$$

The best fit line is $b_1 = 1$, and therefore $y(x) = x$. (We see this because the model is forced through the origin, so the residual at $x = 0$ is fixed. Then the least squares residuals are those that minimize the errors at $x = –1$ and $x = 1$, which we can make zero.) Our raw sum-of-squares identity (8.2) is:

$$\underbrace{1^2 + 0.3^2 + 1^2}_{SST} = \underbrace{\left((-1)^2 + 0^2 + 1^2\right)}_{SSA} + \underbrace{\left(0^2 + 0.3^2 + 0^2\right)}_{SSE} \qquad \rightarrow \qquad 2.09 = 2 + 0.09 .$$

Example: $n = 3$, $p = 2$: We consider the same data: (–1, –1), (0, 0.3), and (1, 1), but we now include a b_0 DC-offset parameter in the model:

$$y_{\text{mod}}(x) = b_0 + b_1 x .$$

The best fit line is $b_0 = 0.1$, $b_1 = 1$, and therefore $y(x) = 0.1 + x$, shown in Figure 8.2c. (We see this because the fit functions are orthogonal over the given $\{x_i\}$, and therefore the fit parameters $\{b_m\}$ can be found by correlating the data with the fit functions, normalized over the $\{x_i\}$. Trust me on this.)

$$\underbrace{1^2 + 0.3^2 + 1^2}_{SST} = \underbrace{\left((-1)^2 + 0^2 + 1^2\right)}_{SSA} + \underbrace{\left(0^2 + 0.3^2 + 0^2\right)}_{SSE} \qquad \rightarrow \qquad 2.09 = 2 + 0.09 .$$

The *raw* sum-of-squares identity holds for *any* linear least-squares fit,
even with non-gaussian (or non-random) residuals.

In general, the SSQ identities do not hold for nonlinear fits, as is evident from the following sections. This means that none of the *linear* regression statistics are strictly valid for a *nonlinear* fit. Nonetheless, they are sometimes approximately valid, and used quite often as such.

The Geometric View of a Least-Squares Fit

The geometric view of a least-squares fit requires defining a vector space: measurement space (aka "observation space"). This is an n-dimensional space, where $n \equiv$ the number of measurements in the data set. Our sets of measurements $\{y_i\}$, predictors $\{f_{ki}\}$, residuals $\{\varepsilon_i\}$, etc. can be viewed as vectors:

$$\mathbf{y} \equiv (y_1, y_2, \ldots y_n), \quad \boldsymbol{\varepsilon} \equiv \left(\varepsilon_1, \varepsilon_2, \ldots \varepsilon_n\right), \quad \text{etc.}$$

Thus, the entire set of measurements $\mathbf{y}$ is a *single point* in measurement space (Figure 8.3). We write that point as the displacement vector $\mathbf{y}$. If we have 1000 measurements, then measurement space is 1000-dimensional. Measurement space is the space of all possible data sets $\{y_i\}$, with the $\{x_i\}$ fixed.

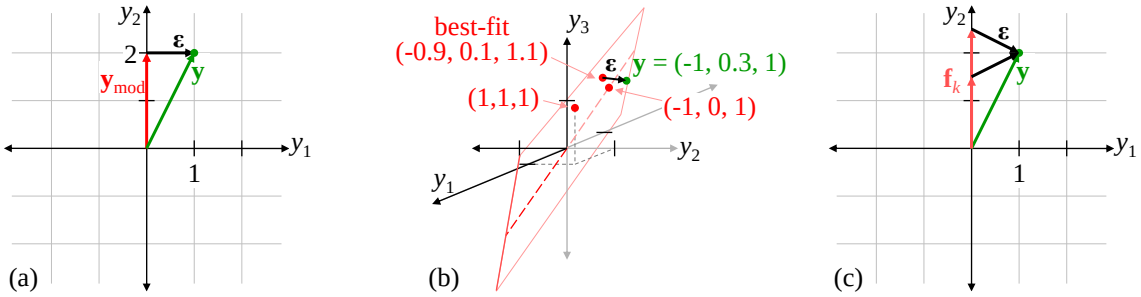


Figure 8.3 (a) Measurement space, $n = 2$, and best-fit 1-parameter model. (b) Measurement space, $n = 3$, and the 2-parameter model surface within it. (c) The shortest $\boldsymbol{\varepsilon}$ is perpendicular to every $\mathbf{f}_k$.

Given a set of parameters $\{b_k\}$ and the sample points $\{x_i\}$, the model (with no residuals) defines a set of measurements, $y_{\text{mod},i}$, which can also be plotted as a single point in measurement space. For example, Figure 8.3a shows our $n = 2$, $p = 1$ model $y = b_1 x$, taken at the two abscissa value $x_1 = 0$, and $x_2 = 1$, which gives $y_{mod,1} = 0$, $y_{mod,2} = b_1$. The least squares fit is $b_1 = 2$. Then the coordinates $(y_{mod,1}, y_{mod,2}) = (0, 2)$ give the model vector $\mathbf{y}_{\text{mod}}$ in Figure 8.3a. Note that the model vector $\mathbf{y}_{\text{mod}}$ is orthogonal to the residual vector $\boldsymbol{\varepsilon}$.

Note that by varying the b_k, the model points in measurement space define a p-dimensional subspace of it. In Figure 8.3a, different values of b_1 trace out a vertical line through the origin. In this case, $p = 1$, so the subspace is 1D: a line.

The $n = 3$ case is shown in Figure 8.3b. Here, $p = 2$, so the model subspace is 2D: a plane in the 3D measurement space. Different values of b_0 and b_1 define different model points in measurement space. For a linear fit, the origin is always on the model surface: when all the $b_k = 0$, all the model $y_i = 0$. Therefore, the plane goes through the origin. Two more points define the plane:

$$b_0 = 1, b_1 = 0 \qquad \Rightarrow \qquad \mathbf{y} = (1,1,1)$$

$$b_0 = 0, b_1 = 1 \qquad \Rightarrow \qquad \mathbf{y} = (-1,0,1)$$

As shown, the model plane passes through these points. Again using linearity, note that any model vector (point) lies on a ray from the origin, and the entire ray is within the model surface. In other words, you can scale any model vector by any value to get another model vector. To further visualize the plane, note that whenever $b_1 = -b_0$, $y_3 = 0$. Then $y_1 = -b_1 + b_0 = 2b_0$, and $y_2 = b_0$; therefore, the line $y_2 = 0.5\, y_1$ lies in the model surface, and is shown with a dashed line in Figure 8.3b.

The green dot in Figure 8.3b is the measurement vector $\mathbf{y}$ (in front of the model plane). The best-fit model point is (-0.9, 0.1, 1.1). The residual vector $\boldsymbol{\varepsilon}$ goes from the model to $\mathbf{y}$, and is perpendicular to the model plane.

The model surface is entirely determined by the model (the $f_k(x)$), and the sample points $\{x_i\}$.

The measured values $\{y_i\}$ will then determine the best-fit model, which is a point on the model surface.

In Figure 8.3a and b, we see that the residual vector is perpendicular to the best-fit linear model vector. Is this always the case? Yes. If the model vector were shorter (Figure 8.3c), $\boldsymbol{\varepsilon}$ would have to reach farther to go from there to the measurement vector $\mathbf{y}$. Similarly, if the model vector were longer, $\boldsymbol{\varepsilon}$ would also be longer. Therefore the shortest residual vector (least sum squared residual) must be perpendicular to the best-fit model vector. This is true in any number of dimensions. From this geometry, we can use the n-dimensional Pythagorean Theorem to prove the sum of squares identity immediately (in vector notation):

$$\boldsymbol{\varepsilon} \bullet \mathbf{y}_{\text{mod}} = 0 \qquad \Rightarrow \qquad \underbrace{\mathbf{y}^2}_{SST} = \underbrace{\mathbf{y}_{\text{mod}}{}^2}_{SSA} + \underbrace{\boldsymbol{\varepsilon}^2}_{SSE} \qquad \textit{where} \quad \mathbf{y}^2 \equiv \mathbf{y} \bullet \mathbf{y}, \text{ etc.}$$

Fit parameters as coordinates of the model surface: We've seen that each point in the model subspace corresponds to a unique set of $\{b_k\}$. Therefore, the b_k compose a new coordinate system for the model subspace, different from the y_i coordinates. For example, in Figure 8.3b, the b_0 axis is defined by setting $b_1 = 0$. This is the line through the origin and the model point $\mathbf{y} = (1, 1, 1)$. The b_1 axis is defined by setting b_0

= 0. This is the line through the origin and **y** = (–1, 0, 1). In general, the b_k axes need not be perpendicular, though in this example, they are.

In Figure 8.3b, **ε** is perpendicular to every vector in the model plane. In general, **ε** is perpendicular to every $\mathbf{f}_k$ vector (i.e. each of the *p* components of the best-fit model vector in the b_k coordinates):

$$\boldsymbol{\varepsilon}\bullet\mathbf{f}_k = 0 \qquad \forall k = 1, \ldots p \qquad \textit{where} \quad \mathbf{f}_k \equiv b_k \left(f_k(x_1),\, f_k(x_2), \ldots\, f_k(x_n) \right).$$

Again, this must be so to minimize the length of **ε**, because if **ε** had any component parallel to any $\mathbf{f}_m$, then we could make that $\mathbf{f}_m$ longer or shorter, as needed, to shrink **ε** (Figure 8.3c). We'll use this orthogonality in the section on the algebra of the sum of squares.

Algebra and Geometry of the Sum-of-Squares Identity

We now prove the sum of squares (SSQ) identity algebraically, and highlight its corresponding geometric features. We start by simply using the definition $y_i = y_{mod,i} + \varepsilon_i$:

$$\sum_{i=1}^{n} {y_i}^2 = \sum_{i=1}^{n} \left(y_{\mathrm{mod},i} + \varepsilon_i \right)^2 = \sum_{i=1}^{n} {y_{\mathrm{mod},i}}^2 + \sum_{i=1}^{n} {\varepsilon_i}^2 + \sum_{i=1}^{n} 2\varepsilon_i y_{\mathrm{mod},i} \,. \tag{8.3}$$

The last term is $\boldsymbol{\varepsilon}\cdot\mathbf{y}_{\mathrm{mod}}$, which we've seen geometrically is zero. We now easily show it algebraically: since SSE is minimized w.r.t. all the model parameters b_k, its derivative w.r.t. each of them is zero. I.e., for each *k*:

$$\frac{\partial SSE}{\partial b_k} = 0 = \frac{\partial}{\partial b_k} \sum_{i=1}^{n} {\varepsilon_i}^2 = \sum_{i=1}^{n} 2\varepsilon_i \frac{\partial}{\partial b_k} \varepsilon_i = 2 \sum_{i=1}^{n} \varepsilon_i \frac{\partial}{\partial b_k} \left(y_i - \sum_{m=1}^{p} b_m f_m(x_i) \right).$$

In this equation, all the y_i are constant. The only term that survives the partial derivative is where $m = k$. Dividing by –2, we get:

$$0 = \sum_{i=1}^{n} \varepsilon_i \frac{\partial}{\partial b_k} b_k f_k(x_i) = \sum_{i=1}^{n} \varepsilon_i f_k(x_i) \qquad \Leftrightarrow \qquad \boldsymbol{\varepsilon}\bullet\mathbf{f}_k = 0, \quad k = 1, \ldots p \,. \tag{8.4}$$

Therefore, the last term in (8.3) drops out, leaving the SSQ identity.

The ANOVA Sum-of-Squares Identity

It is often the case that the DC offset in a set of measurements is either unmeasurable, or not relevant. This leads to **ANalysis Of Variance (ANOVA)**, or analysis of how the data varies from its own average. In the ANOVA case, the sum-of-squares identity is modified: we subtract the data average $\bar{y}$ from both the y_i and the $y_{mod,i}$:

$$\text{(ANOVA)} \quad SST = SSA + SSE: \qquad \sum_{i=1}^{n} \left(y_i - \bar{y} \right)^2 = \sum_{i=1}^{n} \left(y_{\mathrm{mod},i} - \bar{y} \right)^2 + \sum_{i=1}^{n} {\varepsilon_i}^2 \,. \tag{8.5}$$

This has an important consequence which is often overlooked, and proved below: the ANOVA sum-of-squares identity *holds only if the sum of residuals (not squared) = 0.* This is most often achieved by including a constant offset fit parameter, which we call b_0. (See Phase Dispersion Minimization for another way to achieve this.)

Example: *n* = 3, *p* = 2: We again consider the data of Figure 8.2c: (–1, –1), (0, 0.3), and (1, 1). We now use the ANOVA sum-of-squares, which is allowed because we have a b_0 (constant offset) in the model:

$$y(x) = b_0 + b_1 x \,.$$

Our ANOVA sum-of-squares identity (8.5) is, using $\bar{y} = 0.1$:

$$\underbrace{(-1.1)^2 + 0.2^2 + 0.9^2}_{SST} = \underbrace{\left((-1)^2 + 0^2 + 1^2\right)}_{SSA} + \underbrace{\left((-0.1)^2 + 0.2^2 + (-0.1)^2\right)}_{SSE} \quad \rightarrow \quad 2.06 = 2 + 0.06 \,.$$

The ANOVA sum-of-squares identity holds
for any linear least-squares fit that includes a DC offset fit parameter
(and also in the special case that the sum of residuals (not squared) = 0).

With no DC offset parameter in the model, in general,
the ANOVA sum-of-squares identity fails.

We prove the ANOVA SSQ identity (often called just "the sum of squares identity") similarly to our proof of the raw SSQ identity. We start from the definition of ε_i:

$$\begin{aligned}
\sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2 &= \sum_{i=1}^{n} \left(\left(y_{\text{mod},i} - \bar{y}\right) + \varepsilon_i\right)^2 \\
&= \sum_{i=1}^{n} \left(y_{\text{mod},i} - \bar{y}\right)^2 + \sum_{i=1}^{n} {\varepsilon_i}^2 + \sum_{i=1}^{n} 2\varepsilon_i \left(y_{\text{mod},i} - \bar{y}\right) \\
&= \sum_{i=1}^{n} \left(y_{\text{mod},i} - \bar{y}\right)^2 + \sum_{i=1}^{n} {\varepsilon_i}^2 + \cancel{\sum_{i=1}^{n} 2\varepsilon_i y_{\text{mod},i}} + 2\bar{y}\sum_{i=1}^{n} \varepsilon_i \,.
\end{aligned}$$

Compared to the raw SSQ proof, there is an extra 4th term. The 3rd term is zero, as before, because $\boldsymbol{\varepsilon}$ is shortest when it is orthogonal to the model. The 4th term is zero when the sum of the residuals is zero. This might happen by chance (but don't count on it). However, it is guaranteed if we include a DC offset parameter b_0 in the model. Recall that the constant b_0 is equivalent to a fit function $f_0(x_i) = 1$. We know from the raw SSQ proof that for every k:

$$\boldsymbol{\varepsilon}\bullet\mathbf{f}_k \equiv \sum_{i=1}^{n} \varepsilon_i f_k(x_i) = 0 \qquad \Rightarrow \qquad \boldsymbol{\varepsilon}\bullet\mathbf{f}_0 = \sum_{i=1}^{n} \varepsilon_i \, b_0 f_0(x_i) = b_0 \sum_{i=1}^{n} \varepsilon_i = 0 \,.$$

QED.

The necessary and sufficient condition for the ANOVA SSQ identity to hold is that the sum of the residuals is zero. A sufficient condition (and the most common) is that the fit model contains a constant (DC offset) fit parameter b_0.

The Failure of the ANOVA Sum-of-Squares Identity

The ANOVA sum-of-squares identity fails when the sum of the residuals is not zero:

$$\sum_{i=1}^{n} \varepsilon_i \neq 0 \qquad \Rightarrow \qquad \text{(ANOVA) } SST \neq SSA + SSE \,.$$

(We proved this when we proved the ANOVA SSQ identity.) This usually mandates including a b_0 parameter, which guarantees the sum of the residuals *is* zero. You might think this is no problem, because everyone probably already has a b_0 parameter; however, the traditional (now deprecated) Lomb-Scargle algorithm [Sca 1982] fails to include a b_0 parameter, and the cos and sin components generally have nonzero DC; therefore all of its statistics are incorrect. The error is worse for small sample sizes, and better for large ones.

As an example of the failure of the sum-of-squares identity, consider again the data of Figure 8.2a: n = 2 measurements, (0, 1), and (1, 2). As before, we fit the raw data to $y = b_1 x$, and the best-fit is still $b_1 = 2$. We now incorrectly try the ANOVA sum-of-squares identity, with $\bar{y} = 1.5$, and find it fails:

$$\underbrace{\left(-\frac{1}{2}\right)^2+\left(\frac{1}{2}\right)^2}_{SST} \overset{?}{=} \underbrace{\left((-1.5)^2+0.5^2\right)}_{SSA}+\underbrace{\left(1^2+0^2\right)}_{SSE} \quad\rightarrow\quad \frac{1}{2}\neq 2.5+1\,.$$

For another example, consider again the n = 3 data from earlier: (–1, –1), (0, 0.3), and (1, 1). If we fit with just $y = b_1x$, we saw already that $b_1 = 1$ (Figure 8.2b). As expected, because there is no constant fit parameter b_0, the sum of the residuals is not zero:

$$\sum_{i=1}^{n}\varepsilon_i \equiv \sum_{i=1}^{n}\left(y_i - y_{\text{mod},i}\right) = 0+0.3+0\neq 0\,.$$

Therefore, the ANOVA sum-of-squares identity fails:

$$\underbrace{(-1.1)^2+0.2^2+0.9^2}_{SST} \overset{?}{=} \underbrace{\left((-1.1)^2+(-0.1)^2+0.9^2\right)}_{SSA}+\underbrace{\left(0^2+0.3^2+0^2\right)}_{SSE} \quad\rightarrow\quad 2.06\neq 2.03+0.09\,.$$

In the above two examples, the fit function had no DC component, so you might wonder if including a fit function *with* a DC component would restore the ANOVA SSQ identity. It doesn't, because the condition for the ANOVA SSQ identity to hold is that the sum of residuals is zero. To illustrate, we add a fit function, $(x^2 + 1)$ with a nonzero DC (average) value, so our model is this:

$$y_{\text{mod}}(x) = b_1x+b_2\left(x^2+1\right).$$

The best fit is $b_1 = 1$ (as before), and $b_2 = 0.0333$ (from correlation). Then $y_{\text{mod},i}$ = (–0.933, 0.0333, 1.0667), and:

$$\underbrace{(-1.1)^2+0.2^2+0.9^2}_{SST} \overset{?}{=} \underbrace{\left((-1.033)^2+(-0.0667)^2+0.967^2\right)}_{SSA}+\underbrace{\left((-0.0667)^2+0.267^2+(0.0667)^2\right)}_{SSE}$$

$$\rightarrow\quad 2.06\neq 2.007+0.08\,.$$

Subtracting DC Before Analysis: Just Say No

A common method of trying to avoid problems of DC offset is to simply subtract the average of the data before fitting to it. This generally fails to solve the DC problem (though it is sometimes advisable for other reasons, such as improved numerical accuracy in calculations). Subtracting DC makes $\bar{y} = 0$, so the ANOVA SSQ identity is the *same* as the raw SSQ identity, and the raw identity always holds. However, subtracting DC does *not* give an optimal fit when the fit functions have a DC offset over the $\{x_i\}$. The traditional (and deprecated) Lomb-Scargle analysis [Sca 1982] has this error. The only solution is to use a 3-parameter fit: a constant, a cosine component, and a sine component [Zechm 2009].

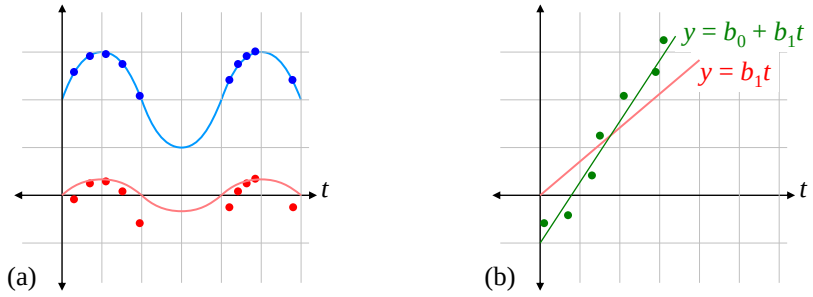


Figure 8.4 (a) The top curve (blue) shows a cosine whose amplitude is fit to data points. The bottom curve (red) shows the same frequency fit to DC-subtracted data, and is a much worse fit. (b) You would never fit a straight line without including DC. Why would you do it with a sinusoid?

Figure 8.4a shows an example of the failure of DC-subtraction to fix the problem, and how DC-subtraction can lead to a much worse fit. Therefore, for sinusoidal analysis (which excludes PDM):

> We must include the constant b_0 parameter both to enable the other parameters to be properly fit, and to enable Analysis of Variance with the ANOVA SSQ identity.

In general, any fit parameter that we must include in the model, but whose value we actually don't need, is called a **nuisance parameter**. b_0 is probably the most common nuisance parameter in data analysis.

Fitting to Orthonormal Functions

For *p orthonormal* fit functions, each b_k can be found by a simple inner product:

$$y_{\text{mod}}(x) \equiv \sum_{k=1}^{p} b_k f_k(x), \qquad \underbrace{\mathbf{f}_j \bullet \mathbf{f}_k = \delta_{jk}}_{\text{orthonormal}} \qquad \Rightarrow \qquad b_k \equiv \mathbf{f}_k \bullet \mathbf{y} \,. \tag{8.6}$$

As examples, this is how Fourier Transform coefficients are found (for uniformly spaced samples, eq. (12.2)), and usually how we find components of a ket in quantum mechanics.

Hypothesis Testing with the Sum of Squares Identity

A big question for some data analysts is, "Is there a signal in my data?" For example, "Is the star's intensity varying periodically?" One approach to answering this question is to fit for the signal you expect, and then test the probability that the fit is just noise. This is a simple form of **Analysis of Variance** (ANOVA). This type of hypothesis is widely used throughout science, e.g. astronomers use this significance test in Phase Dispersion Minimization periodograms, and (now deprecated) Lomb-Scargle.

To make progress in determining if a signal is present, we will test the hypothesis:

H_0: there is no signal, i.e. our data is pure noise.

This is called the **null hypothesis**, because we usually define it to be a hypothesis that nothing interesting is in our data, e.g. there is no signal, our drug doesn't cure the disease, the two classes are performing equally well, etc.

After our analysis, we make one of two conclusions: either we *reject* H_0, or we *fail to reject* it. It is crucial to be crystal clear in our logic here. If our analysis shows that H_0 is unlikely to be true, then we reject H_0, and take it to be false. We also quantify our confidence level in rejecting H_0, typically 95% or better. Rejecting H_0 means there *is* a signal, i.e. our data is *not* pure noise. Note that rejecting H_0, by itself, tells us nothing about the nature of the signal that we conclude is present. In particular, it may or may not match the model we fitted for (but it certainly must have some correlation with our model).

However, if our analysis says H_0 has even a fair chance of being true (typically > 5%), then we do not reject it.

> Failing to reject H_0 is *not the same* as accepting it. Failing to reject means either (a) H_0 is true; or (b) H_0 is false, but our data are insufficient to show that confidently.

This point cannot be over-emphasized.

Notice that scientists are a conservative lot: if we claim a detection, we want to be highly confident that our claim is true. It wouldn't do to have scientists crying "wolf" all the time, and being wrong a lot. The rule of thumb in science is, "If you are not highly confident, then don't make a claim." You can, however, say that your results are intriguing, and justify further investigation.

Introduction to Analysis of Variance (ANOVA)

ANOVA addresses the question: Why don't all my measurements equal the average? The "master equation" of ANOVA is the sum of squares identity (8.5):

$$SST = SSA + SSE \qquad \textit{where} \quad SST \equiv \text{total sum of squared variation}$$
$$SSA \equiv \text{modeled sum of squared variation}$$
$$SSE \equiv \text{residual sum of squares}$$

This equation says that in our data, the total of "squared-differences" from the average is the model differences from the average, plus the unmodeled residuals. Specifically, the total sum of squared differences (*SST*) equals the modeled sum of squared differences (*SSA*) plus the residual (unmodeled + noise) sum of squares (*SSE*).

As shown earlier, for a least-squares linear fit, the master equation (the SSQ identity) requires no statistics or assumptions of any kind (normality, independence, ...).

[ANOVA is identical to least-squares linear regression (fitting) to the "categorical variables." More later.]

To test a hypothesis, we must consider that our data is only one set of many possible sets that might have been taken, each with different noise contributions, ε_i. Recall that when considered over an ensemble of hypothetical data sets, all the fit parameters b_m, as well as *SST*, *SSA*, and *SSE* are random variables. It is in this sense that we speak of their statistical properties.

For concreteness, consider a time sequence of data, such as a light curve with pairs of times and intensities, (t_j, s_j). Why do the measured intensities *vary* from the average? There are conceptually three reasons:

- We have an accurate *model*, which predicts deviations from the average.
- The system under study is more complex than our model, so there are *unmodeled*, but systematic, deviations.
- There is noise in the measurement (which by definition, cannot be modeled).

However, mathematically we can distinguish only two reasons for variation in the measurements: either we predict the variation with a model, or we don't, i.e. modeled effects, and unmodeled effects. Therefore, in practice, the 2nd and 3rd bullets above are combined into **residuals**: unmodeled variations in the data, which includes both systematic physics and measurement noise.

This section requires a conceptual understanding of vector decomposition into both orthonormal and non-orthonormal basis sets.

The Temperature of Liberty

As prerequisite to hypothesis testing, we must consider a number of properties of the fit coefficients b_k that occur when we apply linear regression to measurements **y**. We then apply these results to the case when the "null hypothesis" is true: there is no signal (only noise). We proceed along these lines:

- A look ahead to our goal.
- The distribution of orthonormal fit coefficients, b_k.
- The non-correlation of orthonormal fit coefficients in pure noise.
- The model sum-of-squares (SSA).
- The residual sum-of-squares (SSE) in pure noise.

[Temperature of liberty? Get it? ("degrees of freedom").]

A Look Ahead to the Result Needed for Hypothesis Testing

To better convey where we are headed, the following sections will prove the degrees-of-freedom decomposition of the sum-of-squares (SSQ) identity:

$$\text{(raw)} \qquad SST = SSA + SSE \qquad \rightarrow \qquad \underbrace{\mathbf{y}^2}_{dof=n} = \underbrace{\mathbf{y}_{\text{mod},i}{}^2}_{dof=p} + \underbrace{\boldsymbol{\varepsilon}^2}_{dof=n-p} .$$

We already proved the SSQ identity holds for any least-squares linear fit (regardless of the distribution of *SSE*). To perform hypothesis testing, we must further know that for pure noise, the n degrees of freedom (dof) of *SST* also separate into p dof in SSA, and $n - p$ dof in SSE.

For the ANOVA SSQ identity, the subtraction of the average reduces the dof by 1, so the dof partition as:

$$\text{(ANOVA)} \qquad SST = SSA + SSE \qquad \rightarrow \qquad \underbrace{\left(\mathbf{y} - \bar{y}\right)^2}_{dof=n-1} = \underbrace{\left(\mathbf{y}_{\text{mod},i} - \bar{y}\right)^2}_{dof=p-1} + \underbrace{\boldsymbol{\varepsilon}^2}_{dof=n-p} .$$

Distribution of Orthogonal Fit Coefficients in the Presence of Pure Noise

We have seen that if a fit function is orthogonal to all other fit functions, then its fit coefficient is given by a simple correlation. I.e., for a given k:

$$\mathbf{f}_k \bullet \mathbf{f}_j = 0 \text{ for all } j \neq k \qquad \Rightarrow \qquad b_k = \frac{\mathbf{f}_k \bullet \mathbf{y}}{\mathbf{f}_k{}^2} \equiv \frac{\sum_{i=1}^{n} f_k(x_i) y_i}{\sum_{i=1}^{n} f_k(x_i)^2} . \tag{8.7}$$

We now further restrict ourselves to a normalized (over the $\{x_i\}$) fit-function, so that:

$$\sum_{i=1}^{n} f_k(x_i)^2 = 1 \qquad \Rightarrow \qquad b_k = \sum_{i=1}^{n} f_k(x_i) y_i .$$

We now consider an ensemble of sample sets of pure noise, each with the same set of $\{x_i\}$, and each producing a random b_k. In other words, the b_k are RVs over the set of possible sample-sets. Therefore, in the presence of pure noise, we can easily show that $\text{var}(b_k) = \text{var}(y) \equiv \sigma^2$. Recall that the variance of a sum (of uncorrelated RVs) is the sum of the variances, and the variance of k times an RV = k^2var(RV). All the values of $f_k(x_i)$ are constants, and $\text{var}(y_i) = \text{var}(y) \equiv \sigma^2$; therefore from (8.7):

$$\text{var}(b_k) = \underbrace{\left(\sum_{i=1}^{n} f_k(x_i)^2 \right)}_{1} \text{var}(y_i) = \sigma^2 .$$

This is a remarkable and extremely useful result:

> In pure noise, for a normalized fit-function orthogonal to all others, the variance of its least-squares linear fit coefficient is that of the noise, regardless of the noise PDF.

At this point, the noise need not be zero-mean. In fact:

$$\langle b_k \rangle = \left(\sum_{i=1}^{n} f_k(x_i) \right) \underbrace{\langle y_i \rangle}_{\mu_y} .$$

Since the sum has no simple interpretation, this equation is most useful for showing that if the noise is zero-mean, then b_k is also zero-mean: $\langle b_k \rangle = 0$. However, if the fit-function f_k taken over the $\{x_i\}$ happens to be zero mean, then the summation is zero, and even for non-zero mean noise, we again have $\langle b_k \rangle = 0$.

Similarly, any weighted sum of gaussian RVs is a gaussian; therefore, if the y_i are gaussian (zero-mean or not), then b_k is also gaussian.

Non-correlation of Orthogonal Fit Coefficients in Pure Noise

We now consider the correlation between two fit coefficients, b_k and b_m (again, over multiple samples (sample sets) of noise), when the fit-functions f_k and f_m are orthogonal to each other, and to all other fit-functions. We show that the covariance cov(b_k, b_m) = 0, and so the coefficients are uncorrelated. For convenience, we take f_k and f_m to be normalized: $\mathbf{f}_k^2 = \mathbf{f}_m^2 = 1$. We start with the formula for a fit-coefficient of a fit-function that is orthogonal to all others, (8.7), and use our algebra of statistics:

$$\text{cov}(b_k, b_m) = \text{cov}\left(\mathbf{f}_k \bullet \mathbf{y}, \mathbf{f}_m \bullet \mathbf{y}\right) = \text{cov}\left(\sum_{i=1}^{n} f_k(x_i) y_i, \sum_{j=1}^{n} f_m(x_j) y_j\right).$$

Again, all the f_k and f_m are constants, so they can be pulled out of the cov( ) operator:

$$\text{cov}(b_k, b_m) = \sum_{i=1}^{n}\sum_{j=1}^{n} f_k(x_i) f_m(x_j) \text{cov}\left(y_i, y_j\right).$$

As always, the y_i are independent, and therefore uncorrelated. Hence, when $i \neq j$, cov(y_i, y_j) = 0, so only the $i = j$ terms survive, and the double sum collapses to a single sum. Also, cov(y_i, y_i) = var(y_i) = σ^2, which is a constant:

$$\text{cov}(b_k, b_m) \propto \sigma^2 \underbrace{\sum_{i=1}^{n} f_k(x_i) f_m(x_i)}_{0} = 0 \qquad (f_k \ \& \ f_m \text{ are orthogonal}).$$

This is true for arbitrary distributions of y_i, even if the y_i are nonzero-mean.

> In pure noise of arbitrary distribution, for fit-functions orthogonal to all others, the $\{b_k\}$ are uncorrelated.

The Total Sum-of-Squares (SST) in Pure Noise

The total sum of squares is:

raw: $$SST = \mathbf{y} \bullet \mathbf{y} = \sum_{i=1}^{n} y_i^2$$

ANOVA: $$SST = \left(\mathbf{y} - \bar{y}\right)^2 = \sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2, \qquad \textit{where} \quad \bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i .$$

For zero-mean gaussian noise, the raw *SST* (taken over an ensemble of samples) satisfies the definition of a scaled χ^2 RV with n degrees of freedom (dof), i.e. SST/$\sigma^2 \in \chi^2_n$. As is well-known, the ANOVA SST, by subtracting off the sample average, reduces the dof by 1, so ANOVA *SST*/$\sigma^2 \in \chi^2_{n-1}$.

The Model Sum-of-Squares (SSA) in Pure Noise

We're now ready for the last big step: to show that in pure noise, the model sum-of-squares (*SSA*) has p degrees of freedom (p ≡ # fit parameters). The model can be thought of as a vector, $\mathbf{y}_{\text{mod}} = \{y_{\text{mod},i}\}$, and the basis functions for that vector are the fit-functions evaluated at the sample points, $\mathbf{f}_m \equiv \{f_m(x_i)\}$. Then:

$$\mathbf{y}_{\text{mod}} = \sum_{m=1}^{p} b_m \mathbf{f}_m .$$

The $\mathbf{f}_m$ may be oblique (non-orthogonal), and of arbitrary normalization. However, for any model vector space spanned by $\mathbf{y}_{\text{mod}}$, there exists an orthonormal basis in which it may be written:

$$\mathbf{y}_{\text{mod}} = \sum_{m=1}^{p} c_m \mathbf{g}_m \quad \textit{where} \quad \mathbf{g}_m \equiv \text{orthonormal basis}, \quad c_m \equiv \text{coefficients in the } \mathbf{g} \text{ basis} . \tag{8.8}$$

We've shown that since the $\mathbf{g}_m$ are orthonormal, the c_m are uncorrelated, with var(c_m) = σ^2. Now consider $\mathbf{y}_{\text{mod}}{}^2$ written as a summation:

$$\mathbf{y}_{\text{mod}}{}^2 = \sum_{i=1}^{n} \left(\sum_{m=1}^{p} c_m g_m(x_i) \right)^2 .$$

Since the g_m are orthogonal, all the cross terms in the square are zero. Then reversing the order of summation gives:

$$\mathbf{y}_{\text{mod}}{}^2 = \sum_{m=1}^{p} \sum_{i=1}^{n} \left(c_m g_m(x_i) \right)^2 = \sum_{m=1}^{p} c_m{}^2 \underbrace{\sum_{i=1}^{n} \left(g_m(x_i) \right)^2}_{1} = \sum_{m=1}^{p} c_m{}^2 . \tag{8.9}$$

Therefore, $\mathbf{y}_{\text{mod}}{}^2$ is the sum of p uncorrelated RVs (the $c_m{}^2$). Using the general formula for the average of the square of an RV (7.2):

$$\left\langle c_m{}^2 \right\rangle = \left\langle c_m \right\rangle^2 + \text{var}(c_m) = \left\langle c_m \right\rangle^2 + \sigma^2 \qquad \Rightarrow \qquad \left\langle \mathbf{y}_{\text{mod}}{}^2 \right\rangle = \left(\sum_{m=1}^{p} \left\langle c_m \right\rangle^2 \right) + p\sigma^2 .$$

This is true for any distribution of noise, even non-zero-mean. In general, there is no simple formula for var($\mathbf{y}_{\text{mod}}{}^2$).

If the noise is zero-mean, then each <c_m> = 0, and the above reduces to:

$$\left\langle \mathbf{y}_{\text{mod}}{}^2 \right\rangle = p\sigma^2 \qquad \text{(zero-mean noise)} .$$

If the noise is zero-mean gaussian, then the c_m are zero-mean uncorrelated joint-gaussian RVs. This is a well-known condition for independence [ref ??], so the c_m are independent, gaussian, with variance σ^2. Then (8.9) tells us that $\mathbf{y}_{\text{mod}}{}^2$ is a scaled chi-squared RV with p degrees of freedom:

$$\text{(raw)} \quad \frac{\mathbf{y}_{\text{mod}}{}^2}{\sigma^2} \equiv \frac{SSA}{\sigma^2} \in \chi^2_p \qquad \text{(zero-mean gaussian noise)} .$$

We developed this result using the properties of the orthonormal basis, but our model $\mathbf{y}_{\text{mod}}$, and therefore $\mathbf{y}_{\text{mod}}{}^2$, are identical in *any* basis. Therefore, the result holds for any p fit-functions that span the same model space, even if they are oblique (i.e. overlapping) and not normalized.

For the ANOVA SSQ identity, a similar analysis shows that the constraint of $\bar{y}$ removes one degree of freedom from SSA, and therefore, for zero-mean noise:

$$\left\langle \left(\mathbf{y}_{\text{mod}} - \bar{y} \right)^2 \right\rangle = (p-1)\sigma^2 \qquad \text{(zero-mean noise)} .$$

For zero-mean gaussian noise, then:

$$\frac{\left(\mathbf{y}_{\text{mod}} - \bar{y} \right)^2}{\sigma^2} \equiv \frac{SSA}{\sigma^2} \in \chi^2_{p-1} \qquad \text{(ANOVA SSQ, zero-mean gaussian noise)} .$$

If instead of pure noise, we have a signal that correlates to some extent with the model, then $\left(\mathbf{y}_{\text{mod}} - \bar{y} \right)^2$ will be bigger, on average, than $(p-1)\sigma^2$. That is, the model will explain some of the variation in the data, and therefore the model sum-of-squares will (on average) be bigger than just the noise (even non-gaussian noise):

$$\left\langle \left(\mathbf{y}_{\text{mod}} - \bar{y}\right)^2 \right\rangle \equiv \langle SSA \rangle > (p-1)\sigma^2 \qquad \text{(signal + zero-mean noise)} .$$

The Residual Sum-of-Squares (SSE) in Pure Noise

We determine the distribution of *SSE* in pure noise from the following:

- For least-squares linear fits: $SST = SSA + SSE$.
- From our analysis so far, in pure gaussian zero-mean noise:
  $SST/\sigma^2 \in \chi^2_{n-1}, \quad SSA/\sigma^2 \in \chi^2_{p-1}$.
- From the definition of χ^2_ν, the sum of independent χ^2 RVs is another χ^2 RV, and the dof add.

These are sufficient to conclude that SSE/σ^2 must be χ^2_{n-p}, and must be independent of *SSA*. [I'd like to show this separately from first principles??]:

$$SSE/\sigma^2 \in \chi^2_{n-p} \qquad \text{(for pure gaussian zero-mean noise)} .$$

The F-test: The Decider for Zero Mean Gaussian Noise

In the sections on linear fitting, our results are completely general, and we made no assumptions at all about the nature of the residuals. In the more recent results under hypothesis testing, we have made the minimum assumptions possible, to have the broadest applicability possible. However:

To do quantitative hypothesis testing,
we must know something about the residual distribution in our data.

One common assumption is that our noise is zero-mean gaussian. Then we can quantitatively test if our data are pure noise, and establish a level of confidence (e.g., 98%) in our conclusion. Later, we show how to use simulations to remove the restriction to gaussian noise, and establish confidence bounds for any distribution of residuals.

For zero-mean pure gaussian noise only: we have shown that the *raw* $(SSA/\sigma^2) \in \chi^2_p$. We have also indicated that for ANOVA:

$$\left.\begin{aligned} SST/\sigma^2 &\in \chi^2_{n-1} \\ SSA/\sigma^2 &\in \chi^2_{p-1} \\ SSE/\sigma^2 &\in \chi^2_{n-p} \end{aligned}\right\} \quad \Rightarrow \quad \begin{aligned} \sigma^2 &= \langle SST/(n-1) \rangle \\ \sigma^2 &= \langle SSA/(p-1) \rangle \\ \sigma^2 &= \langle SSE/(n-p) \rangle \end{aligned}$$

Furthermore, *SSA* and *SSE* are statistically independent, and each provides an estimate of the noise variance σ^2.

[Note that the *difference* between two *independent* χ^2 RVs has no simple distribution. This means that SST is *correlated* with SSA in just the right way so that (*SST* – *SSA*) = *SSE* is $\sigma^2\chi^2$ distributed with $p - 1$ dof; similarly SST is correlated with SSE such that (*SST* – *SSE*) = *SSA* $\in \sigma^2\chi^2$ with $n - p$ dof.]

We can take the ratio of the two independent estimates of σ^2, and in pure noise, we should get something close to 1:

$$f \equiv \frac{SSA/(p-1)}{SSE/(n-p)} \approx 1 \qquad \text{(in pure noise)} .$$

Of course, this ratio is itself a (dimensionless) random variable, and will vary from sample set to sample set. The distribution of the RV *f* is the **Fisher-Snedecor F-distribution** [ref??]. It is the distribution of the ratio of two *independent* reduced-χ^2 parameters. Its closed-form is not important, but its general properties are.

First, the distribution depends on both the numerator and denominator degrees of freedom, so F is a two-parameter family of distributions, denoted here as $F(\text{dof num, dof denom; } f)$. (Some references use F to denote the CDF, rather than PDF.)

If our test value f is much larger than 1, we might suspect that H_0 is false: we actually have a signal. We establish this quantitatively with a one-sided F-test, at the α level of significance (Figure 8.5):

$$f > \text{critial\_value}\left[F_{p-1,n-p};\alpha\right] \quad \Rightarrow \quad \text{reject } H_0 \,.$$

If $f >$ critical value, then it is unlikely to be the result of pure noise. We therefore reject H_0 at the α level of significance, or equivalently, at the $(1 - \alpha)$ level of confidence.

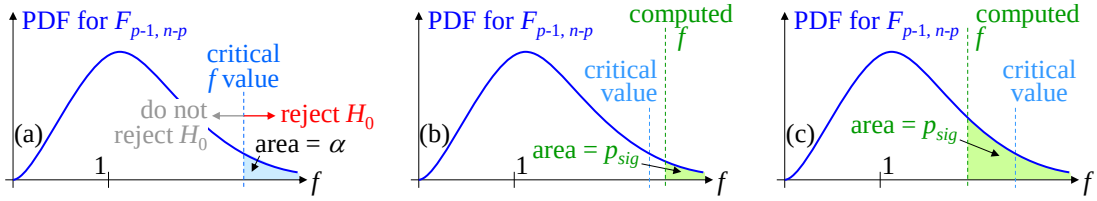


Figure 8.5 One-sided F-test for the null hypothesis, H_0. (a) Critical f value; (b) a statistically significant result; (c) not statistically significant result.

Large-*n* behavior of *F*: It is often the case that n is large, and therefore $\nu_{den} = n - p$ is large. Then the distribution $F(\nu_{num},\nu_{den};\, f)$ is dominated by the numerator, since the denominator distribution is very tight. Then the F distribution is approximately that of the numerator alone, which is χ^2 with ν_{num} degrees of freedom.

Coefficient of Determination and Correlation Coefficient

We hear a lot about the correlation coefficient, ρ, but it's actually fairly useless. However, its square (ρ^2) is the **coefficient of determination**, and is much more meaningful: it tells us the fraction of measured variation "explained" by a fit to the predictor $f_1(x)$. This is sometimes useful as a measure of the effectiveness of the model. ρ^2 is a particular use of the linear regression we have already studied. (We mention a slightly different use for ρ^2 at the end.)

First consider a (possibly infinite) *population* of (x, y) pairs. Typically, x is an independent variable, and y is a measured dependent variable. We often think of the fit function as $f_1(x) = x$ (which we use as our example), but as with all linear regression, the fit-function is *arbitrary*. Recall the sum-of-squares definitions of *SST*, *SSA*, and *SSE* (8.5). We *define* the coefficient of determination in linear-fit terms, as the fraction of SST that is determined by the best-fit model. This is also the ratio of population variances of a least-squares fit:

$$\rho^2 \equiv \frac{SSA}{SST} \equiv \frac{\text{var}(y_{\text{mod}})}{\text{var}(y)} \qquad \text{(population)}$$

.

Note that for the variance of the model y_{mod} to be defined, the domain of x must be finite, i.e. x must have finite lower and upper bounds. For experimental data, this requirement is necessarily satisfied.

Now consider a *sample* of n (x, y) pairs. It is a straightforward application of our linear regression principles to estimate ρ^2. We call the estimate the **sample coefficient of determination**, r^2, and define it analogously to the population parameter:

$$r^2 \equiv \frac{SSA}{SST} \qquad \text{(sample coefficient of determination)} \qquad \text{[Myers 1986 2.20 p28]}$$

$$\textit{where} \quad SSA \equiv \sum_{i=1}^{n}\left(y_{\text{mod},i} - \bar{y}\right)^2, \quad SST \equiv \sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2.$$

Note that the number of fit parameters is $p = 2$ (b_0 and b_1). Therefore *SSA* has $p - 1 = 1$ degree of freedom (dof), and SST has $n - 1$ dof.

[The sample correlation coefficient is just r (with a sign given below):

$$|r| \equiv \sqrt{r^2} \equiv \sqrt{SSA/SST} \qquad \text{(sample correlation coefficient)}.$$

For multiple regression (i.e., with multiple "predictors", where $p \geq 3$ but one is the constant b_0), we define r always ≥ 0. In the case of single regression to one predictor (call it x, $p = 2$ but still one is the constant b_0), $r > 0$ if y increases with the predictor x, and $r < 0$ if y *decreases* with *increasing* x.]

For simplicity, we start with a sample where $\bar{x} = \bar{y} = 0$. At the end, we easily extend the result to the general case where either or both averages are nonzero. If $\bar{x} = 0$, then f_1 is orthogonal to the constant b_0, and we can find b_1 by a simple correlation, including normalization of f_1 (see linear regression to orthogonal fit functions, earlier):

$$b_1 = \frac{\sum_{i=1}^{n} f_1(x_i) y_i}{\sum_{i=1}^{n} f_1(x_i)^2} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2} = \frac{n\langle xy \rangle}{n\sigma_x^2} = \frac{\langle xy \rangle}{\sigma_x^2}.$$

With b_1 now known, we can compute SSA (recalling that $\bar{y} = b_0 = 0$ for now):

$$SSA = \sum_{i=1}^{n} \left(y_{\text{mod},i} - \bar{y} \right)^2 = \sum_{i=1}^{n} \left(b_1 x_i \right)^2 = b_1^2 \underbrace{\sum_{i=1}^{n} x_i^2}_{n\sigma_x^2} = \left(\frac{\langle xy \rangle}{\sigma_x^2} \right)^2 n\sigma_x^2 = n \frac{\langle xy \rangle^2}{\sigma_x^2}.$$

SST is, with $\bar{y} = 0$:

$$SST = \sum_{i=1}^{n} y_i^2 = n\sigma_y^2.$$

Then:

$$r^2 \equiv \frac{SSA}{SST} = \frac{n\langle xy \rangle^2 / \sigma_x^2}{n\sigma_y^2} = \frac{\langle xy \rangle^2}{\sigma_x^2 \sigma_y^2} \qquad \Rightarrow \qquad r \equiv \frac{\langle xy \rangle}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\left(\sum_{i=1}^{n} x_i^2 \right) \left(\sum_{i=1}^{n} y_i^2 \right)}}.$$

Since $\bar{y}$ was known exactly, and not estimated from the sample, SST has n dof.

To generalize to nonzero $\bar{x}$ and $\bar{y}$, we note that we can transform $x \to x - \bar{x}$, and $y \to y - \bar{y}$. These are simple shifts in (x, y) position, and have no effect on the fit line slope or the residuals. These new random variables are zero-mean, so our simplified derivation applies, with one small change: $\bar{y}$ is estimated from the sample, so that removes 1 dof from SST: SST has $n - 1$ dof. Then:

$$r^2 \equiv \frac{SSA}{SST} = \frac{\langle (x - \bar{x})(y - \bar{y}) \rangle^2}{\sigma_x^2 \sigma_y^2} \qquad \textit{where} \quad \langle (x - \bar{x})(y - \bar{y}) \rangle^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

$$\sigma_x^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2, \quad \sigma_y^2 \text{ similar}.$$

Note that another common notation is:

$$r^2 \equiv \frac{SSA}{SST} = \frac{S_{xy}{}^2}{S_{xx}S_{yy}} \qquad where \qquad S_{xy} \equiv \left\langle (x-\bar{x})(y-\bar{y}) \right\rangle, \quad S_{xx} \equiv \sigma_x{}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x-\bar{x})^2, \quad S_{yy} \text{ similar}.$$

Distribution of r^2: Similarly to what we have seen with testing other fit parameters, to test the hypothesis that $r^2 > 0$, we first consider the distribution of r^2 in pure noise. For pure zero-mean gaussian noise, r^2 follows a beta distribution with 1 and n–1 degrees of freedom (dof) [ref ??]. We can use the usual one-sided test at the α significance threshold: if

$$p_{sig} = 1 - \text{cdf}_{beta}\left(r^2\right) > \text{critical\_value}\left[beta(1, n-1); \alpha\right] \qquad \text{(gaussian)}, \tag{8.10}$$

then we reject the null hypothesis H_0, and accept that ρ^2 is probably > 0, at the p_{sig} level of significance.

However:

> The beta distribution is difficult to use, since it crams up near 1, and many computer implementations are unstable in the critical region where we need it most [ref??]. Instead, we can use an equivalent F test, which is easy to interpret, and numerically stable.

Again applying our results from linear regression, we recall that:

$$\underbrace{SST}_{\substack{dof = \\ n-1}} = \underbrace{SSA}_{dof = 1} + \underbrace{SSE}_{\substack{dof = \\ n-2}} \qquad \Rightarrow \qquad f \equiv \frac{SSA / dof_A}{SSE / dof_E} = \frac{r^2 / 1}{\left(1-r^2\right)/(n-2)} \in F_{1,n-2}.$$

Then for pure noise, $f \approx 1$. If $f >> 1$, then ρ^2 is probably > 0, with significance given by the standard 1-sided F test (α is our threshold for rejecting H_0):

$$p_{sig} = 1 - \text{cdf}_F\left(f\right) > \text{critical\_value}\left[F(1, n-2; \alpha\right].$$

Note that the significance p_{sig} here is *identical* to the significance from the beta function (8.10), but using the F distribution is usually a more accurate and easier way to compute it.

Alternative interpretation of x and y: There is another way that ρ^2 can be used, depending on the nature of your data. Instead of x being an independent variable and y being corresponding measured values, it may be that *both* x and y are RVs, with some interdependence. Then, much like $\bar{y}$ is a population parameter of a single random variable y, ρ^2 is a population parameter of two *dependent* random variables, x and y, and their joint density function. Either way, we *define* the coefficient of determination in linear-fit terms, as a ratio of population variances of a least-squares fit of y to x. (We ignore here the question of the dof in $\sigma_x{}^2$.)

9 Uncertainty Weighted Data Analysis

(Needs a summary of common equations??)

When taking data, our measurements often have varying uncertainty (**heteroskedastic**): some measurements are "better" than others. We can still find an average of a set of such measurements, but what is the *best* average, and what is its uncertainty? These questions extend to almost all of the statistics we've covered so far: sample average and variance, fitting, etc. If you have a set of measurements, but each measurement has a *different* uncertainty, how do you combine the measurements for the most reliable estimate of a parameter? Intuitively, estimates with smaller uncertainty should be given more weight than estimates with larger uncertainty. But exactly how much?

Note that the uncertainty of a measurement may or may not be influenced by the value of the measurement itself. For example, for many systems, larger measured values have larger uncertainty. However, all of our results are *independent* of whether the uncertainty has a statistical dependence on the measured value or not. All we need is the uncertainty; we don't care how it comes about. (Many statistics references are unclear on this point.)

Each topic in this section assumes you thoroughly understand the unweighted case described earlier, before delving into the weighted case.

Throughout this section, we consider data triples of the form (x_i, y_i, u_i), where x_i are the independent variables, y_i are the measured variables, and u_i are the 1σ uncertainties of each measurement. We *define* the uncertainty as variations that *cannot* be modeled in detail, though their PDF or other statistics may be known.

> Formulas with uncertainties are *not* simply the unweighted formulas with weights thrown into the "obvious" places.

Two examples of the failure of the "obvious" adjustments to formulas for uncertainty-weighted data are the unbiased estimate of a population σ^2 from a sample (detailed below), and the (deprecated) Lomb-Scargle detection parameter.

Be Sure of Your Uncertainty

We must carefully define what we mean by "uncertainty" u_i. Figure 9.1 depicts a typical measurement, with two separate sources of noise: external (u_{ext}), and instrumental (u_{inst}). The model experiment could be an astronomical one, spread over millions of light-years, or it could be a table top experiment. The external noise might be background radiation, CMB, thermal noise, etc. The instrument noise is the inevitable variation in any measurement system. One can often calibrate the instrument, and determine its uncertainty u_{inst}. Sometimes, one can measure u_{ext}, as well. However, for purposes of this chapter, we *define* our **uncertainty** u_i as:

$u_i \equiv$ *all* of the noise outside of the desired signal, $s(t)$.

Our results depend on this.

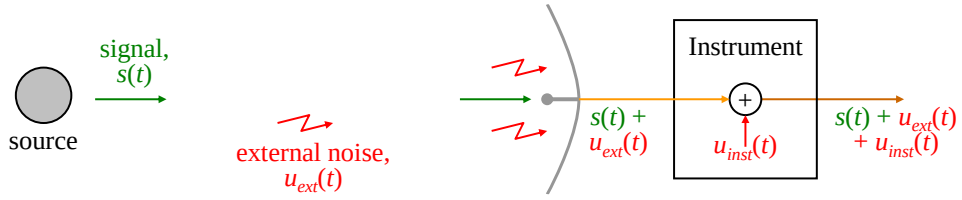


Figure 9.1 A typical measurement includes two sources of noise.

Average of Uncertainty Weighted Data

We give the formula for the uncertainty-weighted average of a sample, and the uncertainty of that average. Consider a sample of n uncertainty weighted measurements, say (x_i, y_i, u_i), where y_i is the measurement, and u_i is the 1σ uncertainty in y_i. How should we best estimate the population average from

this sample? If we *assume* the estimator is a weighted average (as opposed to RMS or something else), we now show that we should weight each y_i by u_i^{-2}. The general formula for a weighted average is:

$$\bar{y} \equiv \langle y_i \rangle = \frac{\sum_{i=1}^{n} w_i y_i}{\sum_{i=1}^{n} w_i} \qquad \textit{where} \quad w_i \text{ are to be determined .} \tag{9.1}$$

The variance (over an ensemble of samples) of this weighted average, where the weights are constants, is (recall that uncorrelated variances add):

$$\text{var}(\bar{y}) = \frac{\sum_{i=1}^{n} w_i^2 u_i^2}{\left(\sum_{i=1}^{n} w_i \right)^2} . \tag{9.2}$$

Note that because of the normalization factor in the denominator, both $\bar{y}$ and its variance are independent of any multiplicative constant in the weights (scale invariance): e.g., doubling all the weights has no effect on $\bar{y}$ or its variance. However, we want to choose the individual weights to give $\bar{y}$ the minimum variance possible. Therefore the derivative of the above variance with respect to any such weight, w_k, is zero. Using the quotient rule for derivatives:

$$\frac{\partial \, \text{var}(\bar{y})}{\partial w_k} = \frac{\left(\sum_{i=1}^{n} w_i \right)^2 \cancel{2} w_k u_k^2 - \left(\sum_{i=1}^{n} w_i^2 u_i^2 \right) \cancel{2} \left(\sum_{i=1}^{n} w_i \right)}{\cancel{\left(\sum_{i=1}^{n} w_i \right)^2}} = 0 \qquad \left(d(UV) = \frac{V\, dU - U\, dV}{V^2} \right)$$

$$w_k = \frac{\left(\sum_{i=1}^{n} w_i^2 u_i^2 \right) \left(\sum_{i=1}^{n} w_i \right)}{\left(\sum_{i=1}^{n} w_i \right)^2 u_k^2} .$$

Since the weights are scale invariant, the only dependence that matters is that $w_k \propto u_k^{-2}$. Therefore, we take the simplest form, and define:

$$w_i \equiv u_i^{-2} \qquad \text{(raw weights) .}$$

For a least-squares estimate of the population average, we weight each measurement by the inverse of the uncertainty squared (inverse of the *measurement* variance).

Some references use scaled weights proportional to u_i^{-2}, so we allows for other weight scalings below.

As expected, large uncertainty points are weighted less than small uncertainty points. Our derivation applies to *any* measurement error distribution; in particular, errors *need not* be gaussian. The least-squares weighted average is well-known [Myers 1986 p171t]. [Note that we have *not* proved that a weighted average, in general, is necessarily the optimum form for estimating a population average, but it is [ref??]. (I suspect this can be proved with calculus of variations, but I've not seen it done.)]

Given these optimum weights, we can now write the uncertainty of $\bar{y}$ more succinctly. For convenience, we define:

$$W \equiv \sum_{i=1}^{n} u_i^{-2}, \qquad V_1 \equiv \sum_{i=1}^{n} w_i \quad \text{(a normalization factor)}, \qquad V_2 \equiv \sum_{i=1}^{n} w_i^2 .$$

Note that W is defined to be independent of weight scaling, V_1 scales with the weights, and V_2 scales with the square of the weights. Then from eq. (9.2), the variance of $\bar{y}$ is:

$$\text{var}(\bar{y}) = \frac{\sum_{i=1}^{n} w_i^2 u_i^2}{V_1^2} \qquad \text{Use: } u_i^2 = w_i^{-1}: \qquad \text{var}(\bar{y}) = \frac{\sum_{i=1}^{n} w_i}{V_1^2} = \frac{V_1}{V_1^2} = \frac{1}{V_1} . \tag{9.3}$$

The variance must be scale invariant, but our result includes V_1, which scales. That's because we chose a scale when we used $u_i^2 = w_i^{-1}$, for which $V_1 = W$. We can infer the scale-invariant result as follows: W is scale invariant, so by substituting $V_1 = W$, the scale invariant result is:

$$\text{var}(\bar{y}) = \frac{1}{W}, \qquad \text{and} \qquad \text{uncertainty}(\bar{y}) \equiv \text{dev}(\bar{y}) = \sqrt{\text{var}(\bar{y})} = \frac{1}{\sqrt{W}} .$$

The raw weights, w_i, have units of [measurement]$^{-2}$.

Note that the weighted uncertainty of $\bar{y}$ reduces to the well-known unweighted uncertainty when all the uncertainties are equal, say u:

$$\text{var}(\bar{y}) = \frac{1}{W} = \left(\sum_{i=1}^{n} u^{-2} \right)^{-1} = \frac{u^2}{n} \qquad \Rightarrow \qquad \text{dev}(\bar{y}) = \frac{u}{\sqrt{n}} .$$

Variance and Standard Deviation of Uncertainty Weighted Data

Handy numerical identity: Recall that when computing *un*weighted standard deviations, we simplify the calculation using the handy identity:

$$\sum_{i=1}^{n} (y_i - \bar{y})^2 = \sum_{i=1}^{n} y_i^2 - n\bar{y}^2 \quad \text{or} \quad \sum_{i=1}^{n} y_i^2 - \frac{\left(\sum y_i\right)^2}{n} .$$

What is the equivalent identity for *weighted* sums of squared deviations? We derive it here:

$$\begin{aligned} \sum_{i=1}^{n} w_i (y_i - \bar{y})^2 &= \sum_{i=1}^{n} w_i \left(y_i^2 - 2y_i\bar{y} + \bar{y}^2 \right) \qquad \text{Use: } \sum_{i=1}^{n} w_i y_i = V_1 \bar{y} \\ &= \sum w_i y_i^2 - 2V_1\bar{y}^2 + V_1\bar{y}^2 \\ &= \sum w_i y_i^2 - V_1\bar{y}_1^2 \quad \text{or} \quad \sum w_i y_i^2 - \frac{\left(\sum w_i y_i\right)^2}{V_1} . \end{aligned} \tag{9.4}$$

We note a general pattern that in going from an unweighted formula to the equivalent weighted formula, the number n is often replaced by the number V_1, and all the summations include the weights.

Weighted sample variance: We now find an unbiased weighted sample variance; unbiased means that over many samples (sets of individual values), the sample variance averages to the population variance. We first state the result:

$$s^2 = \frac{\sum_{i=1}^{n} w_i (y_i - \bar{y})^2}{V_1 - V_2/V_1} .$$

We prove below that this is an unbiased estimator.

> Many references give incorrect formulas for the weighted sample variance;
> in particular, it is *not* just $(1/V_1)\sum w_i (y_i - \bar{y})^2$.

For computer code, we often use the weighted sum-of-squared deviations identity (9.4) to simplify the calculation:

$$s^2 = \frac{\sum_{i=1}^{n} w_i (y_i - \bar{y})^2}{V_1 - V_2/V_1} = \frac{\sum_{i=1}^{n} w_i y_i^2 - \left(\sum_i w_i y_i\right)^2 / V_1}{V_1 - V_2/V_1} \quad \text{or} \quad \frac{V_1 \sum_{i=1}^{n} w_i y_i^2 - \left(\sum_i w_i y_i\right)^2}{V_1^2 - V_2} .$$

We now prove that over many sample sets, the statistic s^2 averages to the true population σ^2. (We use our statistical algebra.) Without loss of generality, we take the population average to be zero, because we can always shift a random variable by a constant amount to make its (weighted) average zero, without affecting its variance. Then the population variance becomes:

$$\sigma^2 = \langle Y^2 \rangle - \langle Y \rangle^2 \quad \rightarrow \quad \sigma^2 = \langle Y^2 \rangle .$$

We start by guessing (as we did for unweighted data) that the *weighted* average squared-deviation is an estimate for σ^2. For a single given sample set, the simple weighted average of the squared-deviations from $\bar{y}$ is (again using (9.4)):

$$q^2 \equiv \frac{\sum_{i=1}^{n} w_i (y_i - \bar{y})^2}{\sum w_i} = \frac{\sum w_i y_i^2 - V_1 \bar{y}^2}{V_1} = \frac{\sum w_i y_i^2}{V_1} - \bar{y}^2 \tag{9.5}$$

Is this unbiased? To see, we average over the ensemble of all possible sample sets (using the same weights). I.e., the weights, and therefore V_1 and V_2, are constant over the ensemble average. The first term in (9.5) averages to:

$$\left\langle \frac{\sum_{i=1}^{n} w_i y_i^2}{V_1} \right\rangle = \frac{\sum_{i=1}^{n} w_i}{V_1} \langle y_i^2 \rangle = \langle Y^2 \rangle = \sigma^2 .$$

By separating the squared terms from the cross terms, we find the second term in (9.5) averages to:

$$\langle \bar{y}^2 \rangle = \left\langle \left(\frac{\sum w_i y_i}{V_1} \right)^2 \right\rangle = \frac{1}{V_1^2} \left[\left\langle \sum_{i=1}^{n} w_i^2 y_i^2 \right\rangle + \left\langle \sum_{i \neq j} w_i w_j y_i y_j \right\rangle \right] .$$

Recall that the covariance, or equivalently the correlation coefficient, between any two independent random variables is zero. The last term is proportional to $\langle y_i y_j \rangle$ (and therefore the covariance), which is zero for the independent values y_i and y_j. Thus:

$$\langle \bar{y}^2 \rangle = \frac{1}{V_1^2} V_2 \langle Y^2 \rangle = \frac{V_2}{V_1^2} \sigma^2 \qquad \Rightarrow \qquad \langle q^2 \rangle = \sigma^2 - \frac{V_2}{V_1^2} \sigma^2 = \left(1 - \frac{V_2}{V_1^2}\right) \sigma^2 .$$

Finally, the unbiased estimate of σ^2 simply divides out the prefactor:

$$\left\langle s^2 \right\rangle = \frac{\left\langle q^2 \right\rangle}{1 - V_2 / \left(V_1^2 \right)} \qquad \Rightarrow \qquad s^2 \equiv \frac{\sum_{i=1}^{n} w_i \left(y_i - \bar{y} \right)^2}{V_1 - V_2 / V_1}, \tag{9.6}$$

as above. Note that we have shown that s^2 is unbiased, but we have *not* shown that s^2 is the *least-squares* estimator, nor that it is the *best* (minimum variance) unbiased estimator. But it is [ref??].

Also, as always, the sample standard deviation $s \equiv \sqrt{s^2}$ is *biased*, because the square root of an average is not the average of the square roots. Since we are concerned most often with bias in the variance, and rarely with bias in the standard deviation, we don't bother looking for an unbiased estimator for σ, the population standard deviation.

Distribution of weighted s^2: If the uncertainties are exact, and the residuals gaussian, s^2 is $\chi^2(n-1)$ distributed. Realistically, the uncertainties are only estimates, so s^2 is *not exactly* χ^2 distributed, and therefore we cannot rigorously associate degrees of freedom with it. However, for large n, and/or good uncertainties u_i, we can *approximate* the weighted s^2 as having a $\chi^2(n-1)$ distribution (like the unweighted s^2 does).

Normalized weights

Some references normalize the weights so that they sum to 1, in which case they are dimensionless:

$$W \equiv \sum_{i=1}^{n} u_i^{-2}, \qquad \text{and} \qquad w_i \equiv \frac{u_i^{-2}}{W} \qquad \text{(normalized, dimensionless weights)}.$$

This makes $V_1 \equiv 1$ (dimensionless), and therefore V_1 does not appear in any formulas. (V_2 must still be computed from the normalized weights.) Both normalizations are found in the literature, so it is helpful to be able to switch between the two.

As an example of how formulas are changed, consider a chi-squared goodness-of-fit parameter. Its form is, in both raw and normalized weights:

$$\text{(raw)} \quad \chi^2 = \sum_{i=1}^{n} w_i \left(y_i - y_{\text{mod},i} \right)^2 \quad \rightarrow \quad \chi^2 = W \sum_{i=1}^{n} w_i \left(y_i - y_{\text{mod},i} \right)^2 \quad \text{(normalized)}.$$

Other similar modifications appear in other formulas. In general, we can say:

to go from raw to normalized: $\quad w_i^{raw} \rightarrow W w_i^{norm}, \quad V_1^{raw} \rightarrow W, \quad V_2^{raw} \rightarrow W^2 V_2^{norm}$

to go from normalized to raw: $\quad w_i^{norm} \rightarrow \frac{w_i^{raw}}{V_1}, \quad W \rightarrow V_1^{raw}, \quad V_2^{norm} \rightarrow \frac{V_2^{raw}}{V_1^2}.$

As another example, we transform the raw formula for s^2, eq. (9.6), to normalized:

$$\text{(raw)} \quad s^2 \equiv \frac{\sum_{i=1}^{n} w_i \left(y_i - \bar{y} \right)^2}{V_1 - V_2 / V_1} \quad \rightarrow \quad s^2 = \frac{W \sum_{i=1}^{n} w_i \left(y_i - \bar{y} \right)^2}{W - W^2 V_2 / W} = \frac{\sum_{i=1}^{n} w_i \left(y_i - \bar{y} \right)^2}{1 - V_2} \quad \text{(normalized)}.$$

To go back (from the normalized s^2 to raw), we take $W \rightarrow V_1$ (if W were there), $w_i \rightarrow w_i / V_1$, and $V_2 \rightarrow V_2 / V_1^2$.

For now, the raw, dimensionful weights give us a handy check of units for our formulas, so we continue to use them in most places.

Numerically Convenient Weights

It is often convenient to perform preliminary calculations by ignoring the measurement uncertainties u_i, and using unweighted formulas. We might even do such estimates mentally. Later, more accurate calculations may be done which include the uncertainties. It is often convenient to compare the preliminary unweighted values with the weighted values, especially for intermediate steps in the analysis, e.g. during debugging of analysis code. However, unnormalized weights, $w_i = u_i^{-2}$, have arbitrary magnitudes that lead to intermediate values with no simple interpretation, and that are not directly comparable to the unweighted estimates. Therefore, it is often convenient to scale the weights so that intermediate results have the same scale as unweighted results. The unweighted case is equivalent to all weights being 1, with a sum of n. We can scale our uncertainty weights to the same sum, i.e. n, or equivalently, we scale our weights to an *average* of 1:

$$\sum_{i=1}^{n} 1 = n \text{ (unweighted)} \quad \Rightarrow \quad \text{(weighted)} \quad \sum_{i=1}^{n} w_i = n \quad \text{and therefore} \quad w_i = \frac{n}{W} u_i^{-2}.$$

With this weight scaling, "quick and dirty" calculations are easily compared to more accurate fully-weighted intermediate (debug) results.

In general, for arbitrary weight scaling, we have:

$$w_i^{raw} = W w_i^{norm} = \frac{W}{V_1^{arb}} w_i^{arb} \qquad where \quad W \equiv \sum_{i=1}^{n} u_i^{-2}, \;\; V_1^{arb} \equiv \sum_{i=1}^{n} w_i^{arb}.$$

Uncertainty Weighted Straight-Line Fit

The common case of an uncertainty-weighted straight-line fit is worth noting for reference [Strutz 7.16 p177]:

$$y_{\text{mod}}(x) = b_0 + b_1 x.$$

$$V_1 \equiv \sum_{i=1}^{n} w_i, \qquad S_x \equiv \sum_{i=1}^{n} w_i x_i, \qquad S_y \equiv \sum_{i=1}^{n} w_i y_i, \qquad S_{xx} \equiv \sum_{i=1}^{n} w_i x_i^2, \quad S_{xy} \equiv \sum_{i=1}^{n} w_i x_i y_i,$$

$$b_0 = \frac{S_{xx} S_y - S_x S_{xy}}{V_1 S_{xx} - S_x^2}, \qquad b_1 = \frac{V_1 S_{xy} - S_x S_y}{V_1 S_{xx} - S_x^2}.$$

These are scale-free (i.e., they work with all weight scalings), because all terms scale as the square of the weights, so the scaling cancels.

Transformation to Equivalent Homoskedastic Measurements

We expect that the homoskedastic case (all measurements have the same uncertainty, σ) is simpler, and possibly more powerful than the heteroskedastic case (each measurement has its own uncertainty, u_i). Furthermore, many computer regression libraries cannot handle heteroskedastic data(!). Fortunately, for the purpose of linear regression, there is a simple transformation from heteroskedastic measurements to an equivalent set of homoskedastic measurements. This not only provides theoretical insight, but is very useful in practice: it allows us to use many (but not all) of the homoskedastic libraries by transforming to the equivalent homoskedastic measurements, and operating on the transformed data.

To perform the transformation, we choose an arbitrary uncertainty to act as our new, equivalent *homoskedastic* uncertainty σ. As a convenient choice, we might choose the smallest of all the measurement uncertainties u_{min} to be our equivalent homoskedastic uncertainty σ, or perhaps the RMS(u_i). (Recall that u_i is defined as *all* of the measurement error, both internal and external.) Then we define a new set of equivalent "measurements" $(x_i, y_i, u_i) \rightarrow (x'_i, y'_i, \sigma)$ according to:

$$y'_i = \frac{\sigma}{u_i} y_i, \qquad x'_{mi} = x_{mi} \frac{\sigma}{u_i}.$$

We can now use all of the homoskedastic procedures and calculations for linear regression on the new, equivalent "measurements." Note that we have scaled both the predictors x_{mi}, and the measurements y_i, by the ratio of our chosen σ to the original uncertainty u_i. Measurements with smaller uncertainties than σ get scaled "up" (bigger), and measurements with larger uncertainties than σ get scaled "down" (smaller).

If the original noise added into each sample was independent (as we usually assume), then multiplying the y_i by constants also yields independent noise samples, so the property of independent noise is preserved in the transformation.

Figure 9.2 shows an example transformation graphically, and helps us understand why it works. Consider 3 heteroskedastic measurements:

(1.0, 0.5, 0.1), (1.6, 0.8, 0.2), (2.0, 1.0, 0.3) (original measurements).

We choose our worst uncertainty, 0.3, as our equivalent homoskedastic σ. Then our equivalent measurements become:

(3.0, 1.5, 0.3), (2.4, 1.2, 0.3), (2.0, 1.0, 0.3) (equivalent measurements).

Figure 9.2 illustrates that an uncertainty of 0.3 at $x'_1 = 3.0$ is equivalent to an uncertainty of 0.1 at $x_1 = 1.0$, because the x' point "tugs on" the slope of the line with the same contribution to χ^2, the square of $(y_{\text{mod},i} - y_i)/u_i$. In terms of sums of squares, the transformation equates *every* term of the sum:

$$\frac{y_{\text{mod}}(x_i) - y_i}{u_i} = \frac{y_{\text{mod}}(x'_i) - y'_i}{\sigma}, \quad \forall i \quad \Rightarrow \quad \sum_{i=1}^{n} \left(\frac{y_{\text{mod}}(x_i) - y_i}{u_i} \right)^2 = \sum_{i=1}^{n} \left(\frac{y_{\text{mod}}(x'_i) - y'_i}{\sigma} \right)^2.$$

The transformation coefficients are dimensionless, so the units of the transformed quantities are the same as the originals. Note that:

> The regression coefficients b_k, and their covariances, are *unchanged* by the transformation to equivalent homoskedastic measurements, but the model *values* $y'_{\text{mod},i} = y_{\text{mod}}(x'_i)$ change because the predictors x'_i are transformed from the original x_i.

Equivalently, the *predictions* of the transformed model are *different* than the predictions of the original model. The uncertainties in the b_m are given by the standard homoskedastic formulas with σ as the measurement uncertainties, and the covariance matrix var(**b**) is also preserved by the transformation. These considerations show that *SST*, *SSA*, and *SSE* are *not* preserved in the transformation.

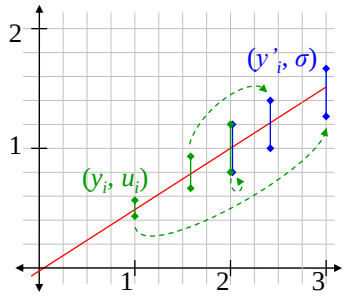


Figure 9.2 The model y_{mod} vs. the original and the equivalent homoskedastic measurements.

In matrix form, the transformation is:

$$\underbrace{\mathbf{T}}_{n\times n} \equiv \begin{bmatrix} \frac{\sigma}{u_1} & & & \\ & \frac{\sigma}{u_2} & & \\ & & \ddots & \\ & & & \frac{\sigma}{u_n} \end{bmatrix}, \qquad \mathbf{y}' = \mathbf{T}\mathbf{y}, \qquad \underbrace{\mathbf{x}'}_{n\times p} = \mathbf{T}\,\underbrace{\mathbf{x}}_{n\times p}.$$

The transformed data are *only* equivalent for the purpose of linear regression, and its associated capabilities, such as prediction, correlation coefficients, etc.

To illustrate the equivalence, recall that the standard sample average is a linear fit to a constant function $f_0(t) = 1$. Therefore, the weighted sample average *is* given by the unweighted average of the transformed measurements. Proof TBS??. Note that the transformed function, $f'_0(t)$ is *not* constant.

In contrast, note that the heteroskedastic population variance estimate (9.6),

$$s^2 \equiv \frac{\sum_{i=1}^{n} w_i \left(y_i - \bar{y}\right)^2}{V_1 - V_2/V_1},$$

is *not* a linear fit. That's why it requires this odd-looking formula, and is *not* given by the common homoskedastic variance estimate applied to the transformed data: $s^2 \neq \sum \left(y_i' - \bar{y}'\right)^2 / (n-1)$.

As another counter-example, the standard Lomb-Scargle algorithm (now deprecated) doesn't work on transformed data. Although Lomb-Scargle is *essentially* a simultaneous linear fit to a cosine and a sine, it relies on a nonlinear computation of the orthogonalizing time offset, τ; essentially, it relies on the fit-functions being cos and sin, and satisfying trigonometric identities. These properties do *not* hold for the transformed data.

Orthogonality is preserved: If two predictors are orthogonal w.r.t. the weights, then the transformed predictors are also orthogonal. Proof: with predictors x_{ki} and x_{mi}:

$$\sum_{i=1}^{n} w_i x_{ki} x_{mi} = 0 \quad \Rightarrow \qquad \sum_{i=1}^{n} x'_{ki}\, x'_{mi} = \sigma^2 \sum_{i=1}^{n} \frac{x'_{ki}}{u_i}\frac{x'_{mi}}{u_i} = \sigma^2 \underbrace{\sum_{i=1}^{n} w_i x_{ki} x_{mi}}_{0} = 0.$$

Linear Regression with Individual Uncertainties

We have seen that for data with *constant* uncertainties, we fit it to a model using the criterion of least-squared residual. If instead we have individual uncertainties (y_i, u_i), we commonly use a least-chi-squared criterion. That is, we fit the model coefficients (b_k) to minimize:

$$SSE \equiv \chi^2 \equiv \sum_{i=1}^{n} \frac{\varepsilon_i^{\,2}}{u_i^{\,2}} \qquad \textit{where} \quad \varepsilon_i \equiv \text{residual} \equiv y_i - y_{\text{mod},i}\,.$$

For gaussian residuals, least-chi-squared fits yields maximum likelihood fit coefficients. For non-gaussian residuals, least-chi-squared is usually as good a criterion as any.

However, there are many statistical formulas that need updating for uncertainty-weighted data. Often, we need an exact closed-form formula for a weighted-data statistical parameter. For example, computing an iterative approximate fit to data can be prohibitively slow, but a closed-form formula may be acceptable (e.g., periodgrams). Finding such exact formulas in the literature is surprisingly hard.

Even though we've described the transformation to linear equivalent measurements, it is often more convenient to compute results directly from the original measurements and uncertainties.

We discuss and analyze some direct weighted-regression computations here. As in the earlier unweighted analysis, we clearly identify the scope of applicability for each formula. And as always, understanding the methods of analyzing and deriving these statistics is essential to developing your own methods for processing new situations.

This section assumes a thorough understanding of the similar unweighted sections. Many of our derivations follow the unweighted ones, but may be briefer here.

The first step of linear regression with individual uncertainties is summarized in [Bev p117-118], oddly in the chapter "Least-Squares Fit to a Polynomial," even though it applies to *all* fit functions (*not* just polynomials). We summarize here the results. The linear model is the same as the unweighted case: given p functions we wish to fit to n data points, the model is:

$$y_{\text{mod}}(x) = \sum_{k=1}^{p} b_k f_k(x) = b_1 f_1(x) + b_2 f_2(x) + ...b_p f_p(x) \qquad \text{[Bev 7.3 p117]}.$$

Each measurement is a triple of independent variable, dependent variable, and measurement uncertainty, (x_i, y_i, u_i). As before, the predictors do not have to be functions of an independent variable (and in ANOVA, they are not); we use such functions only to simplify the presentation. We find the b_k by minimizing the χ^2 parameter:

$$SSE \equiv \chi^2 = \sum_{i=1}^{n} \frac{\left(y(x_i) - y_{\text{mod}}(x_i)\right)^2}{u_i^2} = \sum_{i=1}^{n} \frac{\left(y(x_i) - \sum_{m=1}^{p} b_m f_m(x_i)\right)^2}{u_i^2} \qquad \text{[Bev 7.5 p117]}.$$

For each k from 1 to p, we set the partial derivative, $\partial\chi^2/\partial b_k = 0$, to get a set of simultaneous linear equations in the b_k:

$$\frac{\partial \chi^2}{\partial b_k} = 0 = \sum_{i=1}^{n} 2 \frac{\left(y_i - \sum_{m=1}^{p} b_m f_m(x_i)\right)\left(-f_k(x_i)\right)}{u_i^2}, \qquad k = 1, 2, ... p.$$

Dividing out the –2, and simplifying:

$$0 = \sum_{i=1}^{n} \frac{\left(y_i - \sum_{m=1}^{p} b_m f_m(x_i)\right) f_k(x_i)}{u_i^2}, \qquad k = 1, 2, ... p.$$

Moving the constants to the LHS, we get a linear system of equations in the sought-after b_k:

$$\sum_{i=1}^{n} y_i \frac{f_k(x_i)}{u_i^2} = \sum_{i=1}^{n} \sum_{m=1}^{p} \frac{b_m f_m(x_i)}{u_i^2} f_k(x_i) = \sum_{m=1}^{p} b_m \sum_{i=1}^{n} \frac{f_m(x_i)}{u_i^2} f_k(x_i) \qquad k = 1, 2, ... p.$$

Linear Regression With Uncertainties and the Sum-of-Squares Identity

As with unweighted data, the weighted sum-of-squares (SSQ) identity is the crucial underpinning of **weighted linear regression** (aka "generalized linear regression"). Before considering uncertainties, recall our *unweighted* sum-of-squares identity in vector form:

$$\text{(raw, unweighted)} \quad SST = SSA + SSE \qquad \text{or} \qquad \mathbf{y}^2 = \mathbf{y}_{\text{mod}}{}^2 + \boldsymbol{\varepsilon}^2$$
$$\textit{where} \quad \boldsymbol{\varepsilon} \equiv \text{residual vector}, \quad \mathbf{y}^2 \equiv \mathbf{y}\bullet\mathbf{y}, \text{ etc}, \quad \mathbf{y}_{\text{mod}} \equiv \sum_{k=1}^{p} b_k \mathbf{f}_k . \tag{9.7}$$

Recall that the dot products are real numbers. Also, by construction, $\boldsymbol{\varepsilon}$ is orthogonal to $\mathbf{f}_k$, $\boldsymbol{\varepsilon}\cdot\mathbf{f}_k = 0$, and the SSQ identity hinges on this.

We derive the weighted theory almost identically to the unweighted case. All of our vectors remain the same as before, and we need only redefine our dot product. The weighted dot product weights each term in the sum by w_i:

$$\mathbf{a}\bullet\mathbf{b} \equiv \sum_{i=1}^{n} w_i a_i b_i , \qquad w_i \propto u_i^{-2} . \qquad \mathbf{a}^2 \equiv \mathbf{a}\bullet\mathbf{a} \qquad \text{(weighted dot-product)} .$$

Such generalized inner products are common in mathematics and science. They retain all the familiar, useful properties; in particular, they are bilinear, and in this case, commutative. Then the weighted SSQ identity has exactly the same form as the unweighted case (proved shortly):

$$\text{(raw)} \quad SST = SSA + SSE : \quad \mathbf{y}^2 = \mathbf{y}_{\text{mod}}{}^2 + \boldsymbol{\varepsilon}^2 . \tag{9.8}$$

Note that *SSE* is the χ^2 parameter we minimize when fitting. Written explicitly as summations, the weighted SSQ identity is:

$$\text{(raw)} \quad \underbrace{\sum_{i=1}^{n} w_i y_i{}^2}_{SST} = \underbrace{\sum_{i=1}^{n} w_i \left(b_k f_k(x_i)\right)^2}_{SSA} + \underbrace{\sum_{i=1}^{n} w_i \left(y_i - b_k f_k(x_i)\right)^2}_{SSE} \qquad \text{[Schwa 1998, eq 4 p832]} .$$

If this identity still holds in the weighted case, then most of our previous (unweighted) work remains valid. We now show that it *does* hold. We start by noting that even in the weighted case, $\boldsymbol{\varepsilon}\cdot\mathbf{f}_k = 0$. The proof comes from the fact that *SSE* is a minimum w.r.t. all the b_k:

$$\frac{\partial SSE}{\partial b_k} = 0 = \frac{\partial}{\partial b_k} \sum_{i=1}^{n} w_i \varepsilon_i{}^2 = \sum_{i=1}^{n} 2 w_i \varepsilon_i \frac{\partial}{\partial b_k} \varepsilon_i = 2 \sum_{i=1}^{n} w_i \varepsilon_i \frac{\partial}{\partial b_k} \left(y_i - \sum_{m=1}^{p} b_m f_{mi} \right) .$$

The only term surviving the derivative is when $m = k$:

$$0 = \sum_{i=1}^{n} w_i \varepsilon_i f_{ki} \equiv \boldsymbol{\varepsilon}\bullet\mathbf{f}_k , \quad k = 1, \ldots p .$$

Therefore, per (8.3), the *weighted* raw sum-of-squares identity holds.

The identity is a little simpler in terms of y_{mod}:

$$y_{\text{mod}}(x) \equiv \sum_{m=1}^{p} b_m f_m(x) \quad \Rightarrow \quad \text{(raw)} \quad \sum_{i=1}^{n} w_i y_i{}^2 = \sum_{i=1}^{n} w_i y_{\text{mod},i}{}^2 + \sum_{i=1}^{n} w_i \varepsilon_i{}^2 .$$

Also as before, if we include a constant b_0 fit parameter, then the ANOVA SSQ identity holds:

$$\text{ANOVA:} \qquad \sum_{i=1}^{n} w_i \left(y_i - \bar{y}\right)^2 = \sum_{i=1}^{n} w_i \left(y_{\text{mod},i} - \bar{y}\right)^2 + \sum_{i=1}^{n} w_i \varepsilon_i{}^2 .$$

Recall that $\bar{y}$ is the *weighted* average (9.1). With a b_0 fit parameter, we have:

$$\boldsymbol{\varepsilon}\bullet\mathbf{f}_0 = 0 = \sum_{i=1}^{n} w_i \varepsilon_i \qquad \textit{where} \quad f_{0i} = 1 \ \ \forall i \ .$$

Thus the weighted sum of residuals is zero. Then the ANOVA SSQ proof is only slightly more involved than the unweighted case:

$$\sum_{i=1}^{n} w_i \left(y_i - \bar{y}\right)^2 = \sum_{i=1}^{n} w_i \left(\left(y_{\text{mod},i} - \bar{y}\right) + \varepsilon_i\right)^2$$

$$= \sum_{i=1}^{n} w_i \left(y_{\text{mod},i} - \bar{y}\right)^2 + \sum_{i=1}^{n} w_i {\varepsilon_i}^2 + 2\underbrace{\sum_{i=1}^{n} y_{\text{mod},i}\varepsilon_i}_{0} - 2\bar{y}\underbrace{\sum_{i=1}^{n} w_i \varepsilon_i}_{0}$$

Distribution of Weighted Orthogonal Fit Coefficients in Pure Noise

As in the unweighted case, in hopes of hypothesis testing, we need the distribution of the b_k in pure noise (no signal). Here again, if a fit function is orthogonal (w.r.t the weights) to all other fit functions, then its (least-chi-squared) fit coefficient is given by a simple correlation. I.e., for a given k:

$$\mathbf{f}_k \bullet \mathbf{f}_j = 0 \text{ for all } j \neq k \qquad \Rightarrow \qquad b_k = \frac{\mathbf{f}_k \bullet \mathbf{y}}{{\mathbf{f}_k}^2} \equiv \frac{\sum_{i=1}^{n} w_i f_k(t_i) y_i}{\sum_{i=1}^{n} w_i f_k(t_i)^2} \ .$$

For convenience, we now further restrict ourselves to a normalized (over the $\{t_i\}$) fit-function, though this imposes no real restriction, since any function is easily normalized by a scale factor. Then:

$$\sum_{i=1}^{n} w_i f_k(t_i)^2 = 1 \ \Rightarrow \qquad b_k = \sum_{i=1}^{n} w_i f_k(t_i) y_i \ . \tag{9.9}$$

Now consider an ensemble of samples (sets) of noise, each with the same set of $\{(t_i, u_i)\}$, and each producing a random b_k. In other words, the b_k are RVs over the set of possible samples. We now find var(b_k) and $\langle b_k \rangle$. Recall that the variance of a sum (of uncorrelated RVs) is the sum of the variances, and the variance of k times an RV = k^2var(RV). All the values of w_i and $f_k(t_i)$ are constants, and var(y_i) $\equiv$ $u_i^2 = w_i^{-1}$; therefore taking the variance of (8.7):

$$\text{var}(b_k) = \sum_{i=1}^{n} {w_i}^2 f_k(t_i)^2 \underbrace{\text{var}(y_i)}_{w_i^{-1}} = \sum_{i=1}^{n} w_i f_k(t_i)^2 = 1 \, . \tag{9.10}$$

This is different than the unweighted case, because the noise variance σ^2 has been incorporated into the weights, and therefore into the normalization of the f_k.

In pure noise, for a normalized fit-function orthogonal to all others, using raw weights, the variance of its least-chi-squared linear fit coefficient is 1, regardless of the noise PDF.

We now find the average $\langle b_k \rangle$. Taking the ensemble average of (8.7):

$$\langle b_k \rangle = \left(\sum_{i=1}^{n} w_i f_{ki}\right) \langle y_i \rangle = \left(\sum_{i=1}^{n} w_i f_{ki}\right) \langle noise \rangle \, .$$

Since the sum has no simple interpretation, this equation is most useful for showing that if the noise (measured in y_i) is zero-mean, then b_k is also zero-mean: $\langle b_k \rangle = 0$. However, if the summation happens to be zero (i.e., the predictor is zero-mean), then even for non-zero mean noise, we again have $\langle b_k \rangle = 0$.

Furthermore, any weighted sum of gaussian RVs is a gaussian; therefore, if the y_i are gaussian (zero-mean or not), then b_k is also gaussian.

Non-Correlation of Weighted Fit Coefficients in Uncorrelated Noise

We now consider the correlation between two fit coefficients, b_k and b_m (again, over multiple samples (sets) of noise), when the fit-functions f_k and f_m are orthogonal to each other, and to all other fit-functions. (From the homoskedastic equivalent measurements, we already know that b_k and b_m are uncorrelated. However, for completeness, we now show this fact directly from the weighted data.) For convenience, we take f_k and f_m to be normalized: $\mathbf{f}_k^2 = \mathbf{f}_m^2 = 1$ (recall that our dot products are weighted).

As in the unweighted case, we derive the covariance of b_k and b_m from the bilinearity of the cov( ) operator. We start with the formula for a fit-coefficient of a normalized fit-function that is orthogonal to all others, (8.7), and use our algebra of statistics:

$$\text{cov}(b_k, b_m) = \text{cov}\left(\mathbf{f}_k \bullet \mathbf{y}, \mathbf{f}_m \bullet \mathbf{y}\right) = \text{cov}\left(\sum_{i=1}^{n} w_i f_{ki} y_i, \sum_{j=1}^{n} w_j f_{mj} y_j\right).$$

Again, all the w_i, w_j, f_k, and f_m are constants, so they can be pulled out of the cov( ) operator:

$$\text{cov}(b_k, b_m) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_i f_{ki} w_j f_{mj} \,\text{cov}\left(y_i, y_j\right).$$

It is quite comment that the noise between any two samples is uncorrelated (and usually independent). Then the y_i are uncorrelated. Hence, when $i \neq j$, $\text{cov}(y_i, y_j) = 0$, so only the $i = j$ terms survive, and the double sum collapses to a single sum:

$$\text{cov}(b_k, b_m) = \sum_{i=1}^{n} w_i^2 f_{ki} f_{mi} \underbrace{\text{cov}(y_i, y_i)}_{w_i^{-1}}.$$

Now $\text{cov}(y_i, y_i) = \text{var}(y_i) = u_i^2 = w_i^{-1}$, so:

$$\text{cov}(b_k, b_m) = \sum_{i=1}^{n} w_i f_{ki} f_{mi} = \mathbf{f}_k \bullet \mathbf{f}_m = 0. \tag{9.11}$$

This is true for arbitrary distributions of y_i, even if the y_i are nonzero-mean.

> In pure noise of arbitrary distribution, even for *weighted* fit-functions orthogonal to all others, the $\{b_k\}$ are uncorrelated.

The Weighted Total Sum-of-Squares (SST) in Pure Noise

The weighted total sum of squares is:

raw: $$SST = \mathbf{y} \bullet \mathbf{y} = \sum_{i=1}^{n} w_i y_i^2$$

ANOVA: $$SST = \left(\mathbf{y} - \bar{y}\right)^2 = \sum_{i=1}^{n} w_i \left(y_i - \bar{y}\right)^2, \qquad \textit{where} \quad \bar{y} \equiv \frac{1}{V_1}\sum_{i=1}^{n} w_i y_i.$$

For gaussian noise, if the uncertainties were perfect, then each term in the sum above would be $\in \chi^2_1$. But real uncertainties are just estimates, so in contrast to the unweighted case, the weighted *SST* (taken over an ensemble of samples) is *not* exactly a χ^2 RV. It has no general PDF. However, if the uncertainties are fairly reliable, we can often approximate *SST*'s distribution as χ^2 with n dof (raw), or $n - 1$ dof (ANOVA), especially when n is large.

The Weighted Model Sum-of-Squares (SSA) in Pure Noise

Recall that the model can be thought of as a vector, $\mathbf{y}_{\text{mod}} = \{y_{\text{mod},i}\}$, and the basis functions for that vector are the fit-functions evaluated at the sample points, $\mathbf{f}_m \equiv \{f_{mi}\}$. Then:

$$\mathbf{y}_{\text{mod}} = \sum_{m=1}^{p} b_m \mathbf{f}_m \ .$$

The $\mathbf{f}_m$ may be oblique (non-orthogonal), and of arbitrary normalization. However, as in the unweighted case, there exists an orthonormal basis in which $\mathbf{y}_{\text{mod}}$ may be written (just like eq. (8.8)):

$$\mathbf{y}_{\text{mod}} = \sum_{m=1}^{p} c_m \mathbf{g}_m \quad \textit{where} \quad \mathbf{g}_m \equiv \text{orthonormal basis}, \quad c_m \equiv \text{coefficients in the } \mathbf{g} \text{ basis} \ .$$

We've shown that the c_m are uncorrelated (see (9.11)), with var(c_m) = 1 (using raw weights, see (9.10)). Then (recall that the dot products are weighted):

$$SSA \equiv \mathbf{y}_{\text{mod}}{}^2 = \left(\sum_{m=1}^{p} c_m \mathbf{g}_m \right)^2 \ .$$

Since the g_m are orthogonal, all the cross terms in the square are zero. The $\mathbf{g}_m$ are normalized, so:

$$\mathbf{y}_{\text{mod}}{}^2 = \sum_{m=1}^{p} \left(c_m \mathbf{g}_m \right)^2 = \sum_{m=1}^{p} c_m{}^2 \underbrace{\mathbf{g}_m{}^2}_{1} = \sum_{m=1}^{p} c_m{}^2 \ . \tag{9.12}$$

Therefore, $\mathbf{y}_{\text{mod}}{}^2$ is the sum of p uncorrelated RVs (the $c_m{}^2$). We find SSA $\equiv \mathbf{y}_{\text{mod}}{}^2$ using the general formula for the average of the square of an RV (7.2):

$$\left\langle c_m{}^2 \right\rangle = \left\langle c_m \right\rangle^2 + \text{var}(c_m) = \left\langle c_m \right\rangle^2 + 1 \Rightarrow \qquad SSA \equiv \left\langle \mathbf{y}_{\text{mod}}{}^2 \right\rangle = \left(\sum_{m=1}^{p} \left\langle c_m \right\rangle^2 \right) + p \ .$$

where var(c_m) comes from (9.10). This is true for any distribution of uncorrelated noise, even non-zero-mean. In general, there is no simple formula for var($\mathbf{y}_{\text{mod}}{}^2$).

If the noise is zero-mean, then each $\langle c_m \rangle = 0$, and the above reduces to:

$$\left\langle \mathbf{y}_{\text{mod}}{}^2 \right\rangle = p \qquad \text{(zero-mean noise)} \ .$$

If the noise is zero-mean and gaussian, then the c_m are zero-mean uncorrelated gaussian RVs. This is a well-known condition for independence [ref ??], so the c_m are independent, gaussian, with variance 1 (see (9.10)). Then (9.12) tells us that $\mathbf{y}_{\text{mod}}{}^2$ is a chi-squared RV with p degrees of freedom:

$$\text{(raw)} \quad \mathbf{y}_{\text{mod}}{}^2 \equiv SSA \in \chi^2_p \qquad \text{(zero-mean gaussian noise)} \ .$$

We developed this result using the properties of the orthonormal basis $\mathbf{g}_m$, but our model $\mathbf{y}_{\text{mod}}$, and therefore $\mathbf{y}_{\text{mod}}{}^2$, are identical in *any* basis. Therefore, the result holds for any p fit-functions that span the same model space, even if they are oblique (i.e. overlapping) and not normalized.

The Residual Sum-of-Squares (SSE) in Pure Noise

For zero-mean gaussian noise, in the weighted case, we've shown that *SSA* is χ^2_p distributed, but *SST* is not exactly χ^2. Therefore, *SSE* is not, either. However, for large n, *SST* and *SSE* are *approximately* χ^2 distributed, with the usual (i.e. equal uncertainty case) degrees of freedom assigned:

raw: $$\underbrace{\sum_{i=1}^{n} w_i y_i^{\,2}}_{SST\ dof \approx n} = \underbrace{\sum_{i=1}^{n} w_i y_{\mathrm{mod},i}^{\,2}}_{SSA\ dof = p} + \underbrace{\sum_{i=1}^{n} w_i \varepsilon_i^{\,2}}_{SSE\ dof \approx n-p} \qquad \text{(zero-mean gaussian)}$$

ANOVA: $$\underbrace{\sum_{i=1}^{n} w_i \left(y_i - \bar{y}\right)^2}_{SST\ dof \approx n-1} = \underbrace{\sum_{i=1}^{n} w_i \left(y_{\mathrm{mod},i} - \bar{y}\right)^2}_{SSA\ dof = p-1} + \underbrace{\sum_{i=1}^{n} w_i \varepsilon_i^{\,2}}_{SSE\ dof \approx n-p} \qquad \text{(zero-mean gaussian)}$$

Hypothesis Testing a Model in Linear Regression with Uncertainties

The approximation that *SST* and *SSE* are almost χ^2 distributed allows the usual F-test as an *approximate* test for detection of a signal, i.e. testing whether the fit actually matches the presence of the model in the data. However, the F critical values will be approximate, and therefore so will the *p*-value. In fact, the approximation is usually worst in the tails of the distribution, right around where your *p*-value is, so precise *p*-values cannot be determined from an F-test.

In many cases, numerical simulations (shuffle simulations) can provide more reliable critical values than the theoretical gaussian F critical values, for 2 reasons: even the gaussian-noise F-values are only approximate (as described), and because the noise itself is often significantly non-gaussian.

We recommend numerical simulations (e.g., shuffling) to determine critical values, instead of the approximate (and often inapplicable) gaussian theory.

10 Practical Considerations for Data Analysis

Rules of Thumb

We present here some facts and theory, and also some suggestions, for processing data. Some of these suggestions might be called "rules of thumb." They will be appropriate for many systems, but not all systems. Rules of thumb can often help avoid pitfalls, but in the end, there is no substitute for understanding the details of your own system.

This chapter is more subjective than most others. Generally, there are no hard-and-fast rules for "optimum" data processing. The better you understand your system, the better choices you will be able to make.

Note that:

> Data analysis *is* signal processing. Much of signal processing theory and practice applies to data analysis, as well.

Signal to Noise Ratio (SNR)

Some systems lend themselves to simple parameters describing how "clean" the measurements (or signal) are, or equivalently, how "noisy." For example, communication systems, or a set of measurements, with additive noise can often be represented (to varying degrees of accuracy) by a Signal-to-Noise ratio, or SNR. In contrast, some other systems cannot be reasonably reduced to such a single parameter. We consider here systems with *additive* noise. We define noise as random, though it may have a wide variety of statistical properties, e.g. gaussian, uniform, zero-mean, or biased.

In addition to noise, which is random, measurements are often distorted by deterministic effects, such as nonlinearities in the system. If you know the distortion operation, you can sometimes correct for it. Any residual (uncorrected) distortion usually ends up being treated as if it were noise. (I once consulted for a communication company that was working to correct for nonlinear distortion that had previously been essentially ignored, and so treated as if it were noise. By correcting for the deterministic part, we were able to get a higher signal-to-noise ratio, and therefore better performance, than other systems.)

The term "signal to noise ratio" is widely used, and often abused. In data analysis, "SNR" has many definitions, so SNR quotes, by themselves, cannot be interpreted. At best, SNR is always an estimate; one can never perfectly separate signal and noise. If you could, you would recover the signal perfectly, and at that point, you have eliminated all noise, and your SNR is infinite.

By far the most widely used definition of SNR, and we think the most appropriate, is signal "energy" divided by noise "energy". In this context, "energy" simply means "sum of squares" (SSQ):

$$SNR \equiv \frac{SSQ(signal)}{SSQ(noise)}.$$

For *zero-mean* signals or noise, the sum of squares is proportional to the variance, so sometimes you'll see SNR written as the ratio of two variances.

SNR lies between 0 and ∞: 0 means no signal (all noise), and ∞ means all signal (no noise). For many systems, their performance can be well determined from SNR alone. This computation can be harder than it looks, though, because there is not always a clear definition of "signal" and "noise." However, in many common cases, there are generally accepted definitions that scientists and engineers should adhere to. We describe some of those cases below.

SNR is fundamentally a dimensionless quantity, but is often quoted in decibels (dB), a logarithmic scale for dimensionless quantities:

$$SNR_{dB} \equiv 10\log_{10} SNR = 10\log_{10} \frac{SSQ(signal)}{SSQ(nosie)} = 20\log_{10} \frac{RMS(signal)}{RMS(noise)}.$$

An increment of 3 dB corresponds to a factor of 2 change in SNR.

Any computation of SNR is necessarily an *estimate*,
because SNR is itself somewhat corrupted by noise.

Computing SNR From Data

To directly apply the above definition, we need two sets of data: one we call "signal" or "model," and another we call "noise" or "residuals." Then the computations of SSQ are straightforward. In all cases, we start with a set of n measurements, y_i, $i = 1 \ldots n$. If we can somehow separate the data into two sequences, model and noise, then we compute the SNR above as:

$$\text{Define: } y_i = y_{\text{mod},i} + \varepsilon_i. \quad \text{Then: } SNR = \frac{SSQ(y_{\text{mod},i})}{SSQ(\varepsilon_i)} \sim \frac{\text{model}}{\text{noise}}$$

$$\textit{where } SSQ(y_{\text{mod},i}) \equiv \sum_{i=1}^{n} \left(y_{\text{mod},i}\right)^2, \qquad SSQ(\varepsilon_i) \equiv \sum_{i=1}^{n} \varepsilon_i{}^2 .$$

One simple way to estimate a "signal" is to filter the data, either with analog filters, or the digital equivalent thereof. Digitally, one can also use more specialized filters, such as a "median filter." In all cases, one takes the filtered output as the "signal." Another way to estimate a signal is to fit a model to the data.

SNR for Linear Fits

When we fit a model to data with a *linear least-squares* fit (i.e., minimum χ^2), the total SSQ in the measurements partitions cleanly into "model" and "noise." This is a form of the sum-of-squares identity (see elsewhere for details of linear fitting):

$$SSQ(y_i) = SSQ(y_{\text{mod},i}) + SSQ(\varepsilon_i) .$$

Then SNR is well-defined, and simple. However, note that any "misfit" is counted as noise.

An important factor in estimating SNR is over what range you take the fit, and necessarily then, measure the χ^2. If you include regions you don't care about, it will make the χ^2 less relevant to you.

For linear least-squares fits, we can use the SSQ identity to write SNR in other ways:

$$SNR \equiv \frac{SSQ(signal)}{SSQ(noise)} = \frac{SSQ(signal)}{SSQ(data) - SSQ(signal)}$$

SNR for Nonlinear Fits

As shown elsewhere, for a nonlinear fit, the model values are not orthogonal to the residuals, and the SSQ identity does not hold:

$$SSQ(y_i) \neq SSQ(y_{\text{mod},i}) + SSQ(\varepsilon_i) \qquad \text{(nonlinear fit)} .$$

However, the above definition for SNR is still useful as an approximation (remember: all estimates of SNR are corrupted by noise). A "reasonable" fitting procedure will produce both a "signal" and "noise" that each have less SSQ than the original data. Then the SNR still lies between 0 and ∞.

Other Definitions of SNR

We discourage any other uses of the term SNR, but rarely, it is computed as the ratio of RMS values (or standard deviations), rather than SSQ (or variances). Still other more specialized definitions also exist. However:

SNR should always be dimensionless, and lie in the interval [0, ∞].

Spectral Method of Estimating SNR

In some cases, there is a way to estimate the SNR without explicitly separating a "signal" from the data. As before, suppose you have *n* points of measured data, which consist of an unknown signal plus noise:

$$\text{Define: } y_i = \underset{\text{signal}}{s_i} + \underset{\text{noise}}{\varepsilon_i}\ , \qquad i = 1, \ldots n\ .$$

If you know the approximate Fourier *amplitude* (or equivalently, power) spectrum of the signal, and the noise amplitude spectrum, you can estimate the SNR. This Fourier amplitude often exists in an abstract Fourier space, with little or no physical meaning. Note that you don't need the phase part of the Fourier spectrum, and (infinitely) many different signals have the same amplitude spectrum.

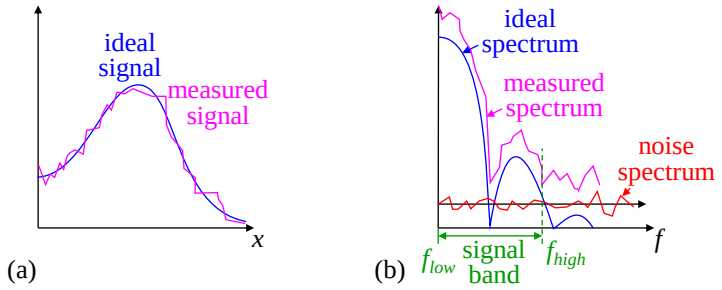


Figure 10.1 Estimating SNR from an amplitude spectrum. (a) Ideal and measured signal. (b) Fourier transform, with white noise. The noise spectrum is assumed known from other sources.

Figure 10.1 shows an example of a measured signal, and it discrete Fourier transform (DFT). The signal is $s(x)$, but it is measured at discrete intervals x_i, so:

$$s_i = s(x_i), \qquad \text{and} \qquad y_i = s_i + \varepsilon_i\ .$$

Any measurement includes noise. Suppose we know our noise spectrum (say, from other measurements). E.g., the noise spectrum is often white (constant). (In fact, in the common case where all the noise contributions, ε_i, are uncorrelated, then the noise is white.) From the measurement spectrum, we find the band (in this abstract Fourier space) where the signal "energy" resides. This band is wherever the measurements are significantly above the noise floor (Figure 10.1b). The energy in this band is signal + noise.

Knowing the "band of interest" of our signal, we can, in principle, filter our measurements to keep only (abstract) frequencies in that band. That will clean up our measurements somewhat, improving their signal-to-noise ratio. Then in terms of signal-energy and noise-energy:

$$SNR = \frac{signal}{noise} = \frac{signal + noise}{noise} - 1\ .$$

The signal + noise energy is the sum of the squares of the discrete Fourier component amplitudes:

$$signal + noise = \sum_{\substack{k \in \text{signal} \\ \text{band}}} |S_k|^2 \qquad where \quad S_k \equiv \text{Fourier coefficients of transformed data}\ .$$

The noise energy is found from our outside source of noise power spectral density. For white noise, the spectrum is constant, so:

$$noise = (signal\ bandwidth)\left(\underbrace{noise\ power\ spectral\ density}_{\text{constant for white noise}}\right) = \left(f_{high} - f_{low}\right) N_0 \qquad (\text{"}energy\text{"})\ .$$

For frequency-dependent noise spectra, $N_0(f)$ in "energy" per Hz, we must integrate to find the noise energy:

$$noise = \int_{f_{low}}^{f_{high}} N_0(f)\, df \qquad ("energy").$$

Tip: be careful how you think about the abstract Fourier space. In one real-world example, a physicist measured the transfer function $H(f)$ of an analog filter, and estimated the SNR of that *measurement* of $H(f)$. A transfer function is a function of frequency, however, we must think of it as just a function of an independent variable (say, a Lorentzian function of f). Now, we take the Fourier transform of $H(f)$, call it $\eta(g)$. This transform exists in an abstract Fourier space: it is a function of abstract frequency g. We must distinguish the abstract frequency g of the *transform* of $H(f)$ from the *real* frequency f of the transfer function $H(\,)$ itself. In this example, the noise floor in g-space was a known property of the measurement device (a network analyzer), so he could estimate his signal-band noise-energy from the noise floor.

Fitting Models To Histograms (Binned Data)

Data analysis often requires fitting a function to binned data, for example, fitting a probability distribution (PDF) to a histogram of measured values. While such fitting is very commonly done, it is much less commonly understood. There are important subtleties often overlooked. This section assumes you are familiar with the binomial distribution, the χ^2 "goodness of fit" parameter (described earlier), and some basic statistics.

The general method for fitting a model to a histogram of data is this:

- Start with n data points (measurements), and a parameterized model for the PDF of those data.
- Bin the data into a histogram.
- Find the model parameters which "best fit" the data histogram.

For example, suppose we have n measurements that we believe should follow a gaussian distribution. A gaussian distribution is a 2-parameter model: the average, μ, and standard deviation, σ. To find the μ and σ that "best fit" our data, we might bin the data into a histogram, and then fit the gaussian PDF to it (Figure 10.2). (Of course, for a gaussian distribution, there are better ways to estimate μ and σ, but the example illustrates the point of fitting to a histogram. Often, a realistic model is more complicated, and there are no simple formulas to compute the model parameters. We use the gaussian as an example because it is familiar to many.)

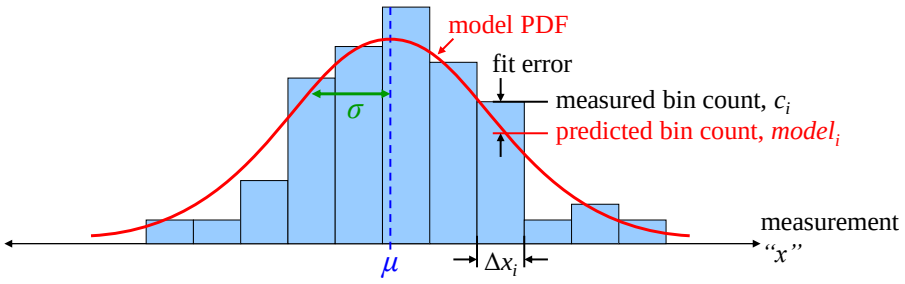


Figure 10.2 Sample histogram with a 2-parameter model PDF (μ and σ). The fit model is gaussian in this example, but could be any PDF with any parameters.

We must define "best fit." Usually, we use the χ^2 (chi-squared) "goodness of fit" parameter as the figure of merit (FOM). The smaller χ^2, the better the fit. Fitting to a histogram is a special case of general χ^2 fitting. Therefore, we need to know two things for each bin: (1) the predicted (model) count, and (2) the uncertainty to use in the χ^2. We now find these two quantities.

Chi-squared For Histograms: Theory

We develop here the χ^2 figure of merit for fitting to a histogram. A **sample** is a set of n **measurements** (data points). In principle, we could take many samples of data. For each sample, there is one histogram,

i.e., there is an infinite population of samples, each with its own histogram. But we have only *one* sample: the one we measured. The question is, how well does our *one* histogram agree with a given population (PDF) of data measurements.

Before the χ^2 figure of merit for the fit, we must first understand the statistics of a single histogram *bin*, from the population of all histograms that we might have produced from different samples. The key point is this: given a sample of n data points, and a particular histogram bin numbered i, each data point in the sample is either in the bin (with probability p_i), or it's not (with probability $(1 - p_i)$). Therefore, the count in the i^{th} histogram bin is binomially distributed, with some probability p_i, and n "trials." (See standard references on the binomial distribution if this is not clear.) Furthermore, this is true of *every* histogram bin:

The number of counts in each histogram bin is a binomial random variable.
Each bin has its own probability, p_i, but all bins share the same number of trials, n.

When fitting a PDF model to a histogram, the bin count is *not* Poisson distributed.

[Aside: when counting events in a fixed time interval, one gets a Poisson distribution of counts. That is not our case, here.]

Recall that a binomial distribution is a discrete distribution, i.e. it gives the probability of finding values of a natural-number random variable (a count of something); in this case, it gives the probability for finding a given number of counts in a given histogram bin. The binomial distribution has two parameters:

$p \equiv$ the probability of a given data point being in the bin

$n \equiv$ the number of data points in the sample, and therefore the number of "trials" in the binomial distribution.

The binomial distribution has average, a, and variance, σ^2 given by:

$$a = np, \qquad \sigma^2 = np(1-p) \qquad \text{(binomial distribution)}. \tag{10.1}$$

A general χ^2 indicator is given by:

$$\chi^2 \equiv \sum_{i=1}^{N_{bins}} \frac{\left(c_i - model_i\right)^2}{\sigma_i^2} \quad \textit{where} \quad c_i \equiv \text{the measured count in the } i^{th} \text{ bin},$$

$$model_i \equiv \text{the model average count in the } i^{th} \text{ bin}$$

$$\sigma_i^2 \equiv \text{the model variance of the } i^{th} \text{ bin}$$

Chi-squared For Histograms: Practice

Computing the χ^2 figure of merit for the fit typically comprises these steps:

- Given the trial fit parameters, compute a (usually unnormalized) PDF for the parameters.
- Normalize the model PDF to the data: compute the scale-factor required to match the number of measured data points.
- Compute the model variance of each bin (using the above two results).
- Compute the final χ^2.

We now consider each of these steps.

Compute the unnormalized model PDF: We find the model average bin-count for a bin from the model PDF: typically, the bins are narrow, and the absolute probability of a single measurement being in a bin is just:

$$\Pr(\text{being in bin } i) \equiv p_i \approx S' \,\text{pdf}_X(x_i)\,\Delta x_i,$$

where $x_i \equiv$ bin-center (narrow bins)

$S' \equiv$ possibly unknown normalization factor

$\text{pdf}_X(x_i) \equiv$ unnormalized model pdf at bin center

If the approximation $pdf_X(x)\Delta x$ is too crude, one can use any more sophisticated method to better integrate the PDF over the bin width to find p_i.

Then the model average bin-count for n measurements is Pr(being in bin) times n, so we define a scale factor S to include both the (possibly unknown) normalization factor for the PDF, as well as the scaling for the number of measurements n:

$$model_i \approx S\,\text{pdf}_X(x_i)\,\Delta x_i \qquad \text{(narrow bins)}$$

where $S \equiv$ as-yet unknown scale factor

$x_i \equiv$ bin center, $\Delta x_i \equiv$ bin width

$\text{pdf}_X(x_i) \equiv$ unnormalized model pdf at bin center

Note that Δx_i need not be constant across the histogram; in fact, it is often useful to have Δx_i wider for small values of the PDF, so the bin-count is bigger (more on this later).

Normalize the model PDF: The PDF computed above is often unnormalized, for two reasons: First, some models are hard to analytically normalize, but easy to calculate as *relative*, probability densities, and therefore we use an unnormalized PDF. Second, most histograms of measured data cover only a subset of all possible measured values, and so even a normalized PDF is not normalized over the restricted range of the histogram.

Normalizing the model PDF is trivial: scale the unnormalized PDF such that the sum of the model bin-counts equals the actual number of data points in the histogram:

$$n = \sum_{i=1}^{N_{bins}} model_i = \sum_{i=1}^{N_{bins}} S\; pdf_X(x_i)\,\Delta x_i \qquad \Rightarrow \qquad S = \frac{n}{\sum_{i=1}^{N_{bins}} pdf_X(x_i)\,\Delta x_i}.$$

(Note that for this step, we don't need to know the actual normalization factor for the model PDF.) The model value for each bin is then:

$$model_i = S\; pdf_X(x_i)\,\Delta x_i.$$

A common mistake is to include the scale parameter S as a *fit* parameter, instead of computing it exactly, as just described. Fitting for S makes your fits less stable, less accurate, and slower. (S would be a "nuisance parameter.") In general:

The fewer the number of fit parameters, the more stable, accurate, and faster your fits will be.

Compute the model variance for each bin: When computing χ^2, we are considering how likely are the *data* to appear for the given trial *model*. For some applications of χ^2, one uses the measurement uncertainties in the denominators of the χ^2 terms. However, when fitting PDFs to histograms, the model itself tells you the uncertainty (variance) in the bin count. Therefore, the "uncertainty" in the denominator of the χ^2 terms is that of the *model*. The exact variance of bin i comes from the binomial distribution (10.1):

$$\sigma_i^2 = np_i(1-p_i) \quad \textit{where} \quad p_i = model_i / n\,.$$

For a large number of histogram bins, N_{bins}, the probability of being in a given bin is of order $p_i \sim 1/N_{bins}$, which is small. Therefore (though we don't agree with it), people often approximate the model variance of the bin-count as:

$$\sigma_i^2 = np_i\left(1-p_i\right) \approx np_i = model_i \qquad (N_{bins} >> 1 \quad \Rightarrow \quad p_i << 1)\,.$$

However, conceptually, and for quick estimates, it is important to know that $\sigma_i^1 \approx model_i$.

Compute the final χ^2: We now have, for each histogram bin, (1) the model average count, $model_i$, and (2) the model variance of the measured count σ_i^2. We compute χ^2 for the model PDF (given a set of model parameters) in the usual way:

$$\chi^2 \equiv \sum_{i=1}^{N_{bins}} \frac{\left(c_i - model_i\right)^2}{\sigma_i^2} \quad where \quad \begin{array}{l} c_i \equiv \text{the measured count in the } i^{th} \text{ bin,} \\ model_i, \sigma_i^2 \equiv \text{the model average \& variance in the } i^{th} \text{ bin} \end{array}$$

If your model predicts any variance of zero (happens when $model_i = 0$ for some i), then χ^2 blows up. This is addressed below.

Reducing the Effect of Noise

To find the best-fit parameters, we take our given sample histogram, and try different values of the pdf(x) parameters (in our gaussian example above, μ and σ) to find the combination which produces the minimum χ^2.

Notice that the low count bins carry more weight than the higher count bins: χ^2 weights the terms by $1/\sigma_i^2 \approx 1/model_i$. This reveals a common misunderstanding about fits to histograms:

> An uncertainty-weighted fit to a histogram is driven by the tails, not by the central peak.
> This is usually bad.

Tails are often the worst part of the model (theory), and often the most contaminated (percentage-wise) by noise: background levels, crosstalk, etc. Three methods help reduce these problems:

- Limiting the weight of low-count bins
- Truncating the histogram
- Rebinning

Limiting the weight: The tails of the model distribution are often less than 1, and often approach zero. This gives them extremely high weights compared to other bins. Since the model is probably inadequate at these low bin counts (due to noise, etc.), one can limit the denominator in the χ^2 sum to at least (say) 1; this also avoids division-by-zero:

$$\chi^2 \equiv \sum_{i=1}^{N_{bins}} \frac{\left(c_i - model_i\right)^2}{d_i} \quad where \quad c_i \equiv \text{the measured count in the } i^{th} \text{ bin}$$

$$d_i \equiv \begin{cases} \sigma_i^2 & \text{if } \sigma_i^2 > 1 \\ 1 & \text{otherwise.} \end{cases}$$

This is an ad-hoc approach, and the minimum weight can be anything; it doesn't have to be 1. Notice, though, that each modified χ^2 term is still a continuous function of $model_i$. This means χ^2 is a continuous function of the fit parameters, which is critical for stable fits (it avoids local minima; see other considerations below).

Note that even if the best-fit model has reasonable values for all the bin-counts, during the optimization algorithm, the optimizer may explore *unreasonable* model parameters on its way to the best fit. Therefore:

> Even if the best-fit model has reasonable values for all the bin-counts,
> limiting the bin weight is often necessary for the optimizer to find the best-fit.

Truncating the histogram: In addition to limiting the bin weight, we can truncate the histogram on the left and right sides to those bins with a reasonable number of *measurements* (not model counts), substantially

above the noise (Figure 10.3a). [Bev p110] recommends a minim bin count of 10, based on a desire for gaussian errors. I don't think that matters much, since the χ^2 parameter works reasonably well, even with non-gaussian errors. In truth, the minimum count completely depends on the noise level: to be meaningful, the bin-count must be dominated by the model over the noise.

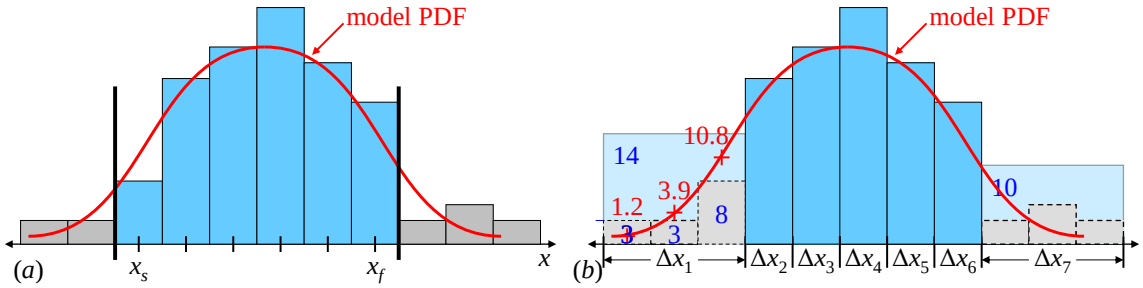


Figure 10.3 Avoiding noisy tails by (a) truncating the histogram, or (b) rebinning. The left 3 bins are combined into a single bin, as are the right 3 bins.

Truncation requires renormalizing (adjusting the number of measurements, n): we normalize the model within the truncated limits to the measured data count within those same limits. That is, we redefine n:

$$n_{norm} \equiv \sum_{i=s}^{f} c_i \qquad \textit{where} \quad s \equiv \text{start bin \#}; \quad f \equiv \text{end bin \#}; \; c_i \equiv \textit{measured} \text{ bin count}.$$

You might think that we should use the model, not the data histogram, to choose our truncation limits. After all, why should we let sampling noise affect our choice of bins? However, using the model fails miserably, because our bin choices change as the optimizer varies our parameters in the hunt for the optimum χ^2. Changing which bins are included in the FOM during the search causes unphysical jumps in χ^2 as we vary our parameters, making many local minima. These minima make the fit unstable, and generally unusable. For stability: truncate your histogram based on the data, and keep it fixed during the parameter search.

Rebinning: Instead of truncating, you can re-bin your data. Bins don't have to be of uniform width [Bev p175], so combining adjacent bins into a single, wider bin with higher count can help improve signal-to-noise ratio (SNR) in that bin (Figure 10.3b). Note that when rebinning, we evaluate the theoretical count as the sum of the original (narrow) bin theoretical counts. In the example of Figure 10.3b, the theoretical and measured counts for the new (wider) bin 1 are:

$$model_1 = 1.2 + 3.9 + 10.8 = 15.9 \qquad \textit{and} \qquad c_1 = 3 + 3 + 8 = 14\,.$$

Other Histogram Fit Considerations

Slightly correlated bin counts: Bin counts are binomially distributed (a measurement is either in a bin, or it's not). However, there is a small negative correlation between any two bins, because the fact that a measurement lies in one bin means it *doesn't* lie in any other bin. Recall that the χ^2 parameter relies on *uncorrelated* errors between bins, so a histogram slightly violates that assumption. With a moderate number of bins (> ~15 ??), this is usually negligible.

Overestimating the low count model: If there are a lot of low-count bins in your histogram, you may find that the fit tends to overestimate the low-count bins, and underestimate the high-count bins (Figure 10.4). When properly normalized, the sum of overestimates and underestimates must be zero: the sum of the model predicted counts is fixed at n. But since low-count bins weigh more than high-count bins, and since an overestimated model reduces χ^2 (the model value $model_i$ appears in the denominator of each χ^2 term), the overall χ^2 is reduced if low-count bins are overestimated, and high-count bins are underestimated.

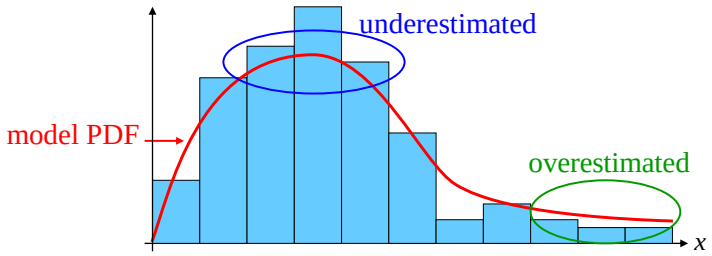


Figure 10.4 χ^2 is artificially reduced by overestimating low-count bins, and underestimating high-count bins.

This effect can only happen if your model has the freedom to "bend" in the way necessary: i.e., it can be a little high in the low-count regions, and simultaneously a little low in the high-count regions. Most realistic models have this freedom. This effect biases your model parameters. If the model is reasonably good, it can cause the reduced-χ^2 to be consistently less than 1 (which should be impossible).

I don't know of a simple fix for this. It helps to limit the weight of low-count bins to (say) 1, as described above. However once again, a better approach is to minimize the number of low-count bins in your histogram.

Noise not zero mean: In histograms, all bin counts are zero or positive. Any noise will add positive counts, and therefore noise cannot be zero-mean. If you know the PDF of the noise, then you can put it in the model, and everything should work out fine. However, if you have a lot of unmodeled noise, you should see that your reduced-χ^2 is significantly greater than 1, indicating a poor fit. Some people have tried playing with the denominator in the χ^2 sum to try to get more "accurate" fit parameters in the presence of noise, but there is little theoretical justification for this, and it lends itself to ad-hoc tweaking to get the answers you *want*. Better to model your noise, and be objective about it.

Non-χ^2 figure of merit: One does not have to use χ^2 as the fit figure of merit. If the model is not very good, or if there are problems as mentioned above, other FOMs might work better. The most common alternative is probably "least-squares," which means minimizing the sum-squared-residual:

$$SSE \equiv \sum_{i=1}^{N_{bins}} \left(c_i - model_i\right)^2 \qquad \text{(sum-squared-residual)}.$$

This is like χ^2 where the denominator in each term in the sum is always 1.

Data With a Hard Cutoff: When Zero Just Isn't Enough

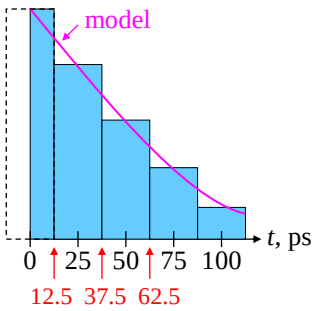


Figure 10.5 Binning data with a lower bound of zero creates a zero-bin of only half width.

Consider a measured quantity with zero as an absolute lower bound. For example, the parameter might be the delay time from cause to effect. Suppose we measure it in ps, and (for some reason) we want to bin the measurements into 25 ps bins. Following the advice above, we'd put our bin centers at 0 ps, 25 ps, 50 ps, etc., so our bin boundaries are 12.5 ps, 37.5 ps, 62.5 ps, etc. (Figure 10.5). However, in this case, the zero-bin is really only 12.5 ps wide. Therefore, when computing the model bin-count for the zero bin from

the model PDF, you must use only half the standard bin-width. That's all there is to it. Despite this slight accommodation, we think it's still worth it to keep the bin *centered* at zero.

Filtering and Data Processing for Equally Spaced Samples

Equally spaced samples (e.g., in time or space) are often "filtered" in some way as part of data reduction or processing. We present here some general guidelines that usually give the best results. We recommend that before deviating from these guidelines, you clearly justify the need to do so, and discuss the justification in any presentation of your data analysis (e.g., in your paper or presentation).

Finite Impulse Response Filters (aka Rolling Filters) and Boxcars

Finite Impulse Response (FIR) filters take a sequence of input data, and produce a (slightly shorter) sequence of output data which is "filtered" or "smoothed" in a particular way. Their primary uses are:

- In real-time processing (or a simulation of indefinite length), where an indefinitely long sequence of data must be continuously processed.
- To crudely smooth data for a visual graph, where the smoothed data are not to be used for further analysis.

Note that fitting procedures should usually be done on the original data, without any pre-filtering. Most fitting procedures inherently filter the noise, and pre-filtering usually degrades the results. Therefore, in data post-processing, where the entire data set is available at once, FIR filters (including "boxcar" filters) are rarely needed or used.

FIR Example: Consider a sequence of data u_j, $j = 1 \ldots n$. In an FIR filter, the output sample at index i is a weighted sum of nearby input samples (Figure 10.6). A simple, widely used filter is:

$$y_j = \frac{1}{4}u_{j-1} + \frac{1}{2}u_j + \frac{1}{4}u_{j+1}\,.$$

The coefficients, or **taps**, are the three weights ¼, ½, and ¼. This is a 3-tap filter. Most FIR filters will be symmetric in their coefficients (same backwards as forwards), because asymmetric filters not only give an erratic spectrum, but also introduce phase shifts that further distort the data.

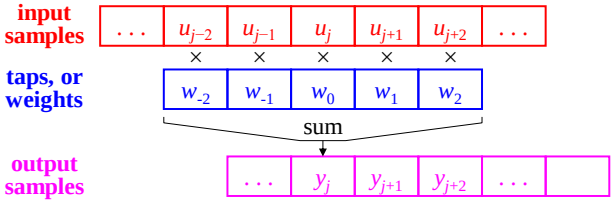


Figure 10.6 Example of a 5-tap FIR filter.

Definition of FIR filter: We define l to be distance to the farthest sample included in the weighted sum. In the 3-tap filter example above, $l = 1$. By definition, an FIR filter produces outputs y_j according to:

$$y_j = \sum_{k=-l}^{l} w_k u_{j+k} \qquad \textit{where} \quad w_k \equiv \text{weights, or "taps"}\,.$$

The number of taps is $2l + 1$. The weights are usually normalized so they sum to 1. We can think of an FIR as sequencing through the index j. Almost all FIR filters require input samples both ahead of and behind the current sample. Therefore, in real-time processing, and FIR filter introduces a delay of l samples before it produces its output. This is usually benign.

FIR (and IIR) filters are linear, so Fourier analysis is appropriate.

Use Smooth Filters (not Boxcars)

"Boxcar" filters are a special case of FIR filters where all the coefficients are the same.

> Boxcar filters are rarely appropriate, and we discourage their use.
> Far better filters are given here, so you can use them effortlessly.

A nice set of smooth filter coefficients turns out to be the odd rows of Pascal's Triangle (which are also the binomial coefficients). Figure 10.7a shows, as an example, the nice quality of the frequency response for the 9-tap filter. In the table below, the normalization factors are in parentheses before the integer coefficients:

$l = 3$: (¼) (1 2 1)

$l = 5$: (1/16) (1 4 6 4 1)

$l = 7$: (1/64) (1 6 15 20 15 6 1)

$l = 9$: (1/256) (1 8 28 56 70 56 28 8 1)

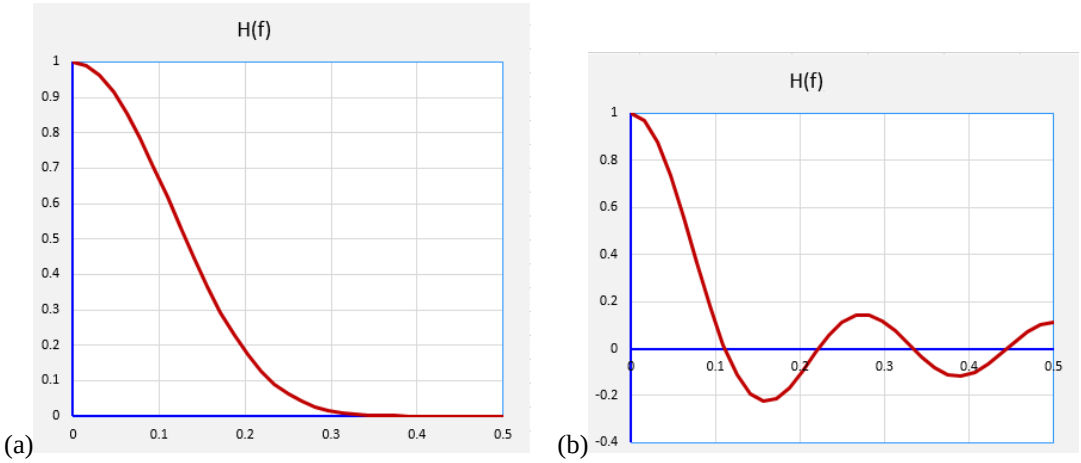


Figure 10.7 (Left) Frequency response of a 9-tap smooth filter is well behaved.
(Right) Frequency response of a 9-tap boxcar filter is erratic, and sometimes negative.

(To be supplied) Be careful to reproduce the tap coefficients exactly, and don't approximate with so-called "in-place" filters. Seemingly small changes to a filter can produce unexpectedly large degradations in behavior.

Problems With Boxcar Filters

Boxcar filters suffer from an erratic frequency response (Figure 10.7b). This colors the noise, which is sometimes harmful, and almost never useful. Furthermore, some frequencies are *inverted*, so they come out the *negative* of how they went in (between f = 0.11 – 0.21, and 0.33 – 0.44). Also, it's easy to mistakenly use an even number of taps in a boxcar, which makes the result even worse by introducing phase distortion.

11 Bayesian Methods

Statistical analysis methods for parameter estimation are roughly divided into two categories: so-called "frequentist" methods, and "Bayesian" methods. Both are widely used across countless fields. We can summarize the approaches as follows:

Frequentist methods estimate a parameter (including its uncertainty) from a set of data. To refine an estimate by combining it with other sets of data, frequentist methods use formulas for "pooling" results.

Bayesian methods start with an estimated PDF for the parameter, and then refine the entire PDF with the new data set. The "prior" estimated PDF carries forward information from prior experiments into the new result.

Sadly, there is an irrational "religious war" about which methods are "better." As always, "better" is subjective, and an analyst's choice of method is best driven by the goals and circumstances at hand. As with software coding, be a scientist, not a zealot.

Bayes' Rule

Bayes' Rule is an important formula to relate a conditional probability to its reverse: how does Pr(A | B) relate to Pr(B | A)? This relationship underlies the Bayesian method for parameter estimation that is widely used all through science and engineering. This section assumes a basic understanding of conditional probability (see Conditional Probability p96).

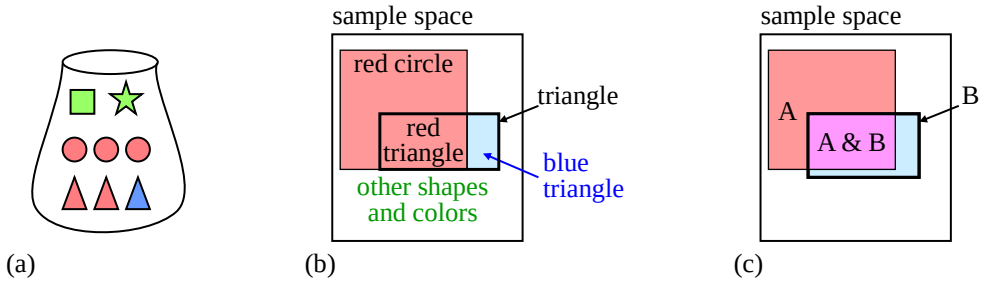


Figure 11.1 (a) Colored shapes in a bag. (b) Sample space diagram where areas are proportional to probabilities. (c) Areas $\propto$ probabilities; properties A and B are dependent.

To illustrate Bayes' rule, we consider a bag of 8 objects (Figure 11.1a). Each object has a shape and a color. If we pull an object at random from the bag, we can consider the probabilities of it having various properties. A priori (i.e., before we pull the shape), it has a 5/8 chance of being red. We write this as Pr(red) = 5/8. After pulling, without looking, we can feel that it's a triangle; therefore, it now has a 2/3 chance of being red. This is a conditional probability: what is the chance the object is red, given the condition that it is a triangle? We write this as Pr(red | triangle) = 2/3. In contrast, if we pull an object and see that it is red, what is the chance it is a triangle? By inspection, Pr(triangle | red) = 2/5. Clearly:

$$\Pr(\text{red} \mid \text{triangle}) \neq \Pr(\text{triangle} \mid \text{red}).$$

Figure 11.1b illustrates these relationships. The outer square represents the "sample space:" the set of all possible outcomes of trials (recall Figure 7.1). In this example, each point in the sample space represents two attributes: shape and color. Areas in the sample space diagram are relative probabilities. The Pr(red | triangle) is the fraction of the triangle region that is red: 2/3.

In general (Figure 11.1c), Pr(A | B) is the ratio of the magenta area (A & B) to the blue-and-magenta B-region. We see that most of the B's are also A's, but most of the A's are *not* also B's. Visually, Figure 11.1c shows that Pr(A | B) > ½, but (reversing the conditional) Pr(B | A) < ½. Thus, in general:

$$\Pr(A \mid B) \neq \Pr(B \mid A),$$

as before. Writing the above quantitative relations between areas, we see **Bayes rule**:

$$\Pr(A \,\&\, B) = \Pr(B \mid A)\Pr(A) = \Pr(A \mid B)\Pr(B) \qquad \Rightarrow$$
$$\Pr(B \mid A) = \frac{\Pr(A \mid B)\Pr(B)}{\Pr(A)} = \frac{\Pr(A \,\&\, B)}{\Pr(A)}. \tag{11.1}$$

This shows that Bayes' rule allows us to "turn around" a conditional probability: if A and B are binary events, then given Pr(A), Pr(B), and Pr(A | B), we can find Pr(B | A).

We can extend this rule to entire PDFs, by considering the probability that a random variable A will be in the range (a, $a + da$) (this constitutes an event, as in the preceding paragraph), and the probability that a random variable B will simultaneously be between b and $b + db$.

As a first step in extending Bayes' rule to PDFs, recall that when A and B are at all dependent, the conditional PDFs on A and B are different than the "overall" PDFs:

$$\text{pdf}_B(b) \neq \text{pdf}_B(b \mid a), \qquad \text{and} \qquad \text{pdf}_A(a) \neq \text{pdf}_A(a \mid b), \qquad A \text{ and } B \text{ dependent}.$$

Then Bayes' rule (11.1) becomes:

$$\text{pdf}_B(b \mid a)\,db = \frac{\text{pdf}_A(a \mid b)da\ \text{pdf}_B(b)db}{\text{pdf}_A(a)da} \qquad \text{or} \qquad \text{pdf}_B(b \mid a) = \frac{\text{pdf}_A(a \mid b)\ \text{pdf}_B(b)}{\text{pdf}_A(a)}. \tag{11.2}$$

In other words:

Bayes' rule (11.1) works for PDFs as well as discrete probabilities.

Bayesian Analysis

Bayesian parameter estimation doesn't just estimate a value and an uncertainty (two numbers); it estimates a full PDF for the parameter. From the PDF, one can readily calculate common point-estimates, such as most-likely value (maximum likelihood), average value, and standard deviation (uncertainty).

Refining an Estimate

The main purpose of Bayesian analysis is to refine estimates of previously estimated parameters. We wish to incorporate new measurements into our body of knowledge, i.e. incorporate new measurements along with our "prior information," typically to make a better estimate of some parameters. The refinement method is an approximation (a fact that I have never seen acknowledged in any reference on the subject).

Our prior estimate for θ is not a single value, but a whole PDF for θ: $\text{pdf}_{\text{prior}}(\theta)$. We introduce here a simplified notation, where we no longer use a subscript on the PDF to indicate which random variable it describes; instead, the argument to the PDF defines the applicable RV: thus $\text{pdf}_x(x) \rightarrow \text{pdf}(x)$. For our parameter θ (which is an RV), we have to manage two PDF's: the prior PDF that is information prior to new data, and the posterior PDF, that is our new estimate of θ's PDF. Thus we write $\text{pdf}_{\text{prior}}(\theta)$ and $\text{pdf}_{\text{post}}(\theta)$.

What has Bayes' Rule got to do with estimating a parameter from data? As a practical matter, it is often comparatively easy to compute the most-likely values to be *measured*, given some *trial* values of the parameter(s), i.e., given a model, it's fairly easy to compute pdf(data | parameters). But that is the reverse of what we want: we want pdf(parameters | data), and from this PDF, we can compute the parameter and its uncertainty (using our choice of criterion).

It is often fairly easy to compute pdf(data | parameters), but that is the reverse of what we want:
we want pdf(parameters | data), and from this PDF,
we can estimate each parameter and its uncertainty.

We use Baye's Rule to go from pdf(data | parameters) to pdf(parameters | data).

Consider a new dataset $D = \{x_1, x_2, \ldots x_n\}$, from which we wish to refine our prior estimate of a single parameter θ (we'll extend to multiple parameters as needed). From the new data, and the prior PDF, we will compute a new (one hopes better) posterior PDF: $\text{pdf}_{\text{post}}(\theta)$. We use an approximation: the new (posterior)

PDF of our parameter is approached by applying Bayes' rule and the *new* data to the *prior* PDF of our parameters. From (11.2), we could say:

$$\text{pdf}_{post}(\theta) \approx \frac{\text{conditional-pdf}\left(data \mid \theta\right)\ \text{pdf}_{prior}(\theta)}{\text{overall-pdf}\left(data\right)} .$$

More precisely:

$$\begin{aligned} &\text{pdf}_{post}\left(\theta \mid \{x_1, \ldots x_n\}\right) \approx \frac{\text{pdf}\left(\{x_1, \ldots x_n\} \mid \theta\right) \text{pdf}_{prior}(\theta)}{\text{pdf}\left(\{x_1, \ldots x_n\}\right)} \qquad \text{or} \\ &\text{pdf}_{post}\left(\theta \mid D\right) \approx \frac{\text{pdf}\left(D \mid \theta\right) \text{pdf}_{prior}(\theta)}{\text{pdf}\left(D\right)} \qquad \textit{where} \quad D \equiv \{x_1, \ldots x_n\} . \end{aligned} \tag{11.3}$$

(This is reminiscent of the Born approximation, where we use one approximation of a function to iterate to a better approximation of the function.) Note that the two PDFs for the data are *n*-dimensional PDFs, and the arguments are x_1, ... x_n. We will see that the above equation is harder to evaluate than it looks, because of the denominator.

We make a brief aside to consider the estimation of $p > 1$ parameters: Let our hypothesis $H \equiv \{\theta_1, \ldots \theta_p\}$ be a set of p parameters we wish to refine estimates of, using the n measurements. Typically, $n > p$, otherwise we have insufficient measurements to find p parameters. Then to explicitly label the dimensions, we use the form of the first equality in (11.2):

$$\text{pdf}_{post}(H \mid D)\, d^p\theta \approx \frac{\text{pdf}(D \mid H)\, d^n D\, \text{pdf}_{prior}(H)\, d^p H}{\text{pdf}(D)\, d^n D} \qquad \textit{where} \quad H \equiv \{\theta_1, \ldots \theta_p\} .$$

From now on, we use H to represent one or more parameters to be estimated (H is our *hypothesis*).

Let's return to the denominator in (11.3) (and (11.2)). It is the n-dimensional probability (density) of getting the specific data values measured, x_1, ... x_n, *averaged* (**marginalized**) over all the possible parameters. For the given x_i, it is a constant. This average is weighted by the probabilities (PDF) of the parameters, so more likely parameter values carry more weight than less likely ones. For a single parameter θ, it is:

$$\text{pdf}(D) \equiv \text{pdf}(x_1, \ldots x_n) = \int_{\substack{\text{parameter} \\ \text{space}}} \text{pdf}\left(x_1, \ldots x_n \mid \theta\right) \text{pdf}_{prior}(\theta)\, d\theta .$$

For p parameters θ_1, ... θ_p, the overall (i.e., marginalized) PDF of the data pdf(D) (the denominator in (11.3)) is simply a p-dimensional integral:

$$\text{pdf}(D) \equiv \text{pdf}(x_1, \ldots x_n) = \int_{\substack{\text{parameter} \\ \text{space}}} \text{pdf}\left(x_1, \ldots x_n \mid \theta_1, \ldots \theta_p\right) \text{pdf}_{prior}(\theta_1, \ldots \theta_p)\, d\theta_1 \ldots d\theta_p .$$

In practice, this integral usually cannot be evaluated analytically, and so must be numerically estimated. But direct numerical integration of even a few dimensions ($p >\sim 4$) is difficult and time consuming. One common method for feasibly estimating such integrations is Markov Chain Monte Carlo (MCMC), using the Metropolis-Hastings algorithm for importance-weighting.

Substituting pdf(D) above into (11.3), we see that the $\text{pdf}_{prior}(\theta)$ appears in both the numerator and denominator of pdf_{post}. In a slightly more compact notation:

$$\text{pdf}_{post}\left(H \mid D\right) \approx \frac{\text{pdf}\left(D \mid H\right) \text{pdf}_{prior}(H)}{\int_{\substack{\text{parameter} \\ \text{space}}} \text{pdf}\left(D \mid H\right) \text{pdf}_{prior}\left(H\right) d^p\theta} \qquad \textit{where} \quad H \equiv \{\theta_1, \ldots \theta_p\} .$$

This form shows explicitly that the denominator is simply a normalization constant for pdf_{post}. Importantly, pdf_{prior} can be unnormalized (i.e., *relative* probability), because any constant factor cancels. This is helpful, because analytically normalizing $\text{pdf}_{prior}(\theta)$ is sometimes infeasible. Conveniently, an unnormalized PDF also works for the Metropolis-Hastings numerical integration algorithm.

More TBS. In particular, discuss the approximation in computing the posterior distribution.

Bayesian vs. Frequentist

There are *not* two kinds of statistics; there is only one. In all of science and engineering, we frequently use legitimate approximations. *Both* frequentist and Bayesian methods are approximations. There is often no significant difference between the results of the two. In particular, when the prior information is weak, there is no significant benefit to including it in a statistical analysis. We discuss the Bayesian approximation below (TBS).

Bayesian analysis is often abused, as typified by this comment: "The intent here is that the likelihood is so sharply peaked that it dominates the shape of the posterior[,] rather than the prior [dominating the posterior]." [http://astrobites.org/2011/11/26/your-gateway-to-the-bayesian-realm/]. In effect, this statement says, "We don't have any prior information, so we *make something up* whose express purpose is to have *no effect* on our results." Why bother, then, with Bayesian analysis? Just do a simple fit, and be done with it; this yields a point estimate and its uncertainty, which can be used as prior information for a *future* experiment. In other words, just use the so-called "frequentist" approximation; that's what it's for.

12 Fourier Transforms and Digital Signal Processing

Signals, noise, and Fourier Transforms are an essential part of much data analysis. It is a deep and broad subject, so we can here establish only some foundational principles. The subject is, however, rife with misunderstandings and folklore. Therefore, we also here dispel some myths. For more specialized information, one must consult more specialized texts.

You may be familiar with Fourier Transforms as a method of analysis applied to functions, where the transform takes one function into another. Discrete Fourier Transforms are very different: they are used in numerical data analysis and signal processing, and there usually are no symbolic functions, just data.

NB: Fourier transforms do not take account of uncertainties in your data. They weight all data the same, in effect, *assuming* all the data have equal uncertainties. If your data points have widely varying uncertainties (or nonuniform sampling), other methods of periodicity analysis may be more appropriate. (See *Fourier Transforms, Periodograms, and (Deprecated) Lomb-Scargle* below.)

This section assumes you are familiar with complex arithmetic and exponentials, and with basic sampling and Fourier Transform principles. In particular, you must be familiar with decomposing a function into an orthonormal basis of functions. Understanding that a Fourier Transform is a phasor-valued function of frequency is very helpful, but not essential (see *Funky Electromagnetic Concepts* for a discussion of phasors).

We start with the most general (and simplest) case, then proceed through more specialized cases. We include some important (often overlooked) properties of Discrete Fourier Transforms. Topics:

- Complex sequences, and complex Fourier Transform (it's actually easier to start with the complex case, and specialize to real numbers later)
- Sampling and the Model of Digitization
- Even number of points vs. odd number of points
- Basis Functions and Orthogonality
- Real sequences: even and odd # points
- Normalization and Parseval's Theorem
- Continuous vs. discrete time and frequency; finite vs. infinite time and frequency
- Non-uniformly spaced samples

For definiteness, we call the independent variable "time, t". But it could be position, or almost anything.

Brief Definitions

Fourier Series represents a *periodic continuous* function as an infinite sum of sinusoids at discrete frequencies:

$$s(t) = \sum_{k=0}^{\infty} S_k e^{i2\pi k f_1 t}, \qquad \textit{where} \quad \begin{cases} S_k \text{ are complex (phasors), } t \equiv \textit{time} \\ f_1 = 1/\textit{ period} \text{ (in cycle/s or Hz), } \omega_1 \equiv 2\pi f_1 \text{ (in rad/s)} \end{cases}$$

f_1 = 1/period, the lowest nonzero frequency, is called the **fundamental frequency**. f_0 = 0, always.

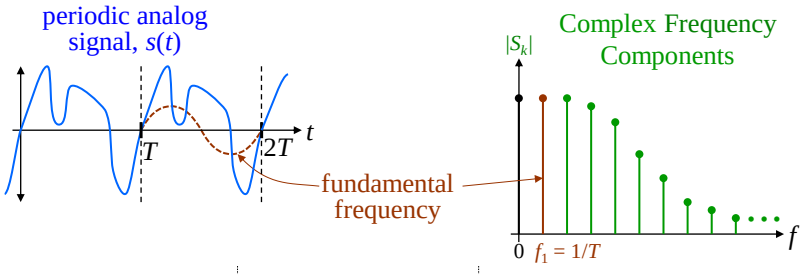


Figure 12.1 A (continuous) Fourier series represents a continuous, periodic "time-domain" function as an infinite sum of discrete frequencies.

Fourier Transform (FT)

represents a continuous function as an integral of sinusoids over continuous frequencies:

$$s(t) = \int_{-\infty}^{\infty} S(2\pi f) e^{i2\pi ft}\, df = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\omega) e^{i\omega t}\, d\omega, \qquad where \quad S(\cdot) \text{ is complex .}$$

We do not discuss this here. The function $s(t)$ is not periodic, so there is no fundamental frequency. $S(\omega)$ is a phasor-valued function of angular frequency. $s(t)$ must be well-behaved enough for the integral to converge. Otherwise, $s(t)$ does not have a Fourier Transform.

Discrete Fourier Transform (DFT)

represents a finite sequence of numbers as a finite sum of sinusoids:

$$s_j = \sum_{k=0}^{n-1} S_k e^{i2\pi(k/n)j}, \qquad where \quad \begin{cases} k \equiv \text{frequency index} \\ S_k \text{ are complex (phasors),} \\ j = 0, \ldots n-1 \equiv \text{the sample index,} \\ f_1 = 1/\,period \text{ (in cycle/s)}, \;\; \omega_1 = 2\pi f_1 \;\; \text{(in rad/s)} \end{cases}$$

The sequence s_j may be thought of as either periodic, or undefined outside the sampling interval. As in the Fourier Series, the fundamental frequency is 1/period, or equivalently 1/(sampled interval), and $f_0 = 0$, always.

[Since a DFT essentially treats the input as periodic, it might be better called a Discrete Fourier Series (rather than Transform), but Discrete Fourier Transform is completely standard.]

Contrary to common belief, a DFT can have an arbitrary number of samples n.
(It does *not* have to have a power of two number of samples.)

Fast Fourier Transform (FFT)

an algorithm for implementing special cases of DFT.

Computer library functions for FFTs are widely available. Contrary to common belief, an FFT does not have to have a power of two number of samples. It can have any number of samples n, though it is faster to compute if n has many small factors. Modern FFT functions accept any number of samples; if yours doesn't, get a new one that does (we recommend KISS-FFT). As described in the "Signal Processing" section, never pad your data; with modern libraries, there is no reason to, and good reason *not* to.

Inverse Discrete Fourier Transform (IDFT)

gives the sequence of numbers s_j from the DFT components.

The general digital Fourier Transform is a Discrete Fourier Transform (DFT).
An FFT is an algorithm for efficiently computing a DFT.

Model of Digitization and Sampling

All realistic systems which digitize analog signals must comprise at least the components in Figure 12.2.

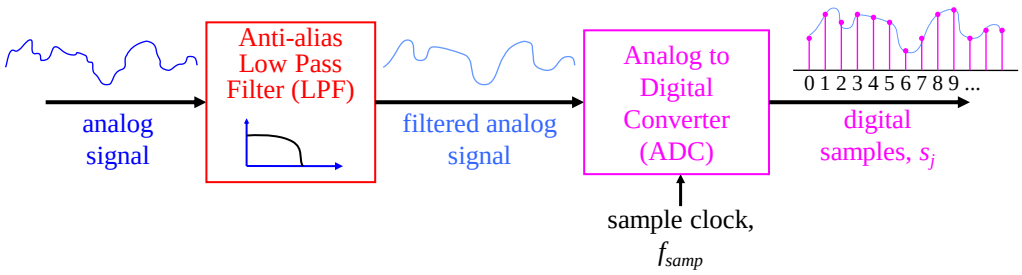


Figure 12.2 Minimum components of a Digital Signal Processing system, with uniformly spaced samples.

In this example, the output of the digitizer is a sequence of real numbers, s_j. Other systems (such as coherent quadrature downconverters) produce a sequence of complex numbers.

Sampling Does *Not* Produce Impulses

It is often said that sampling a signal is like setting it to zero everywhere except at the sample times, or like creating a series of impulses. It is not.

These notions are not true, and can be misleading [O&S p8b]. Note that a single impulse (in time) has infinite energy. Therefore, a sum (sequence) of such impulses has infinite power. In contrast, the original signal, and the sequence of samples, has finite power. This suggests immediately that samples are *not* equivalent to a series of impulses.

Nonetheless, there *is* an identity that involves impulse functions, which we discuss after introducing the DFT.

Complex Sequences and Complex Fourier Transform

It's actually easier to start with the complex case, and specialize to real numbers later. Given a sequence of n complex numbers s_j, we can write the sequence as a sum of sinusoids, i.e. complex exponentials:

Inverse Discrete Fourier Transform:

$$s_j = \sum_{k=0}^{n-1} S_k e^{i2\pi(k/n)j}, \qquad \text{where} \quad j = 0, \ldots n-1 \text{ is the sample index}$$

$$\frac{k}{n} \equiv \text{the frequency of the } k^{th} \text{ component, in cycle/sample}$$

$$S_k \equiv \text{the complex frequency component (phasor)}$$

Note that there are n original complex numbers, and n complex frequency components, so no information is lost.

The Discrete Fourier Transform is exact, unique, and invertible.

In other words, this is *not* a "fit."

The above equation forces all normalization conventions. We use the simple scheme wherein a function equals the sum of its components (with no factors of 2π or anything else).

Often, the index j is a measure of time or distance, and the sequence comprises samples of a signal taken at equal intervals. Without loss of generality, we will refer to j as a measure of "time," but it could be anything. Note that the equation above actually defines the **Inverse Discrete Fourier Transform** (IDFT), because it gives the original sequence from the Fourier components. [Mathematicians often reverse the definitions of DFT and IDFT, by putting a minus sign in the exponent of the IDFT equation above. Engineers and physicists usually use the convention given here.]

The sample index j, and the frequency index k, are dimensionless. Then the frequency amplitudes have the same units as the samples: $[S_k] = [s_j]$. If we think of the sample index as time, then the frequencies are in rad/s.

Each number in the sequence is called a **sample**, because such sequences are often generated by sampling a continuous signal $s(t)$. For n samples, there are n frequency components, S_k, each at normalized frequency k/n (defined soon); see Figure 12.3.

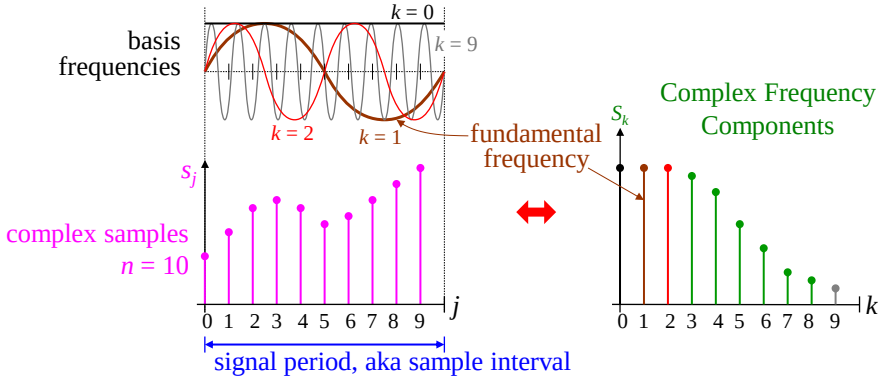


Figure 12.3 Samples in time, and their frequencies. For simplicity, the samples, sinusoids, and component amplitudes are shown as real, but in general, they are all complex valued.

Note that there are a full n sample times in the sample interval (aka *signal* period), not $(n - 1)$.

The above representation is used by many DFT functions in computer libraries.
We use it for some of our analyses, as well.

Also, there is no need for any other frequencies, because $k = 10$ has exactly the same values at all the sample points as $k = 0$. If the samples are from a continuous signal that had a frequency component at $k = 10$, then that component will be **aliased** down to $k = 0$, and added to the actual $k = 0$ component. It is forever lost, and cannot be recovered from the samples, nor distinguished from the $k = 0$ (DC) component. The same aliasing occurs for any two frequencies k and $k + n$.

The above definition is the *only* correct meaning for "aliasing."
Many people misuse this word to mean other things (e.g., "harmonics" or "sidebands").

To avoid a dependence on n, we usually label the frequencies as fractions. For n samples, there are n frequencies, measured in units of cycles/sample, and running from $f = 0$ to $f = (1 - 1/n)$ cycles/sample. The n **normalized frequencies** are:

$$f_k = \frac{k}{n}, \quad k = 0,1,\ldots n-1, \qquad \text{that is} \qquad \{f_k\} = \left\{0, \frac{1}{n}, \frac{2}{n}, \frac{3}{n}, \ldots \frac{n-1}{n}\right\}. \tag{12.1}$$

There is no $f = 1$, just as there is no $k = n$, because $f = 1$ is an alias of $f = 0$. Continuous Fourier components are written as $S(f)$, a function of f, so we re-label the above diagram with normalized frequencies:

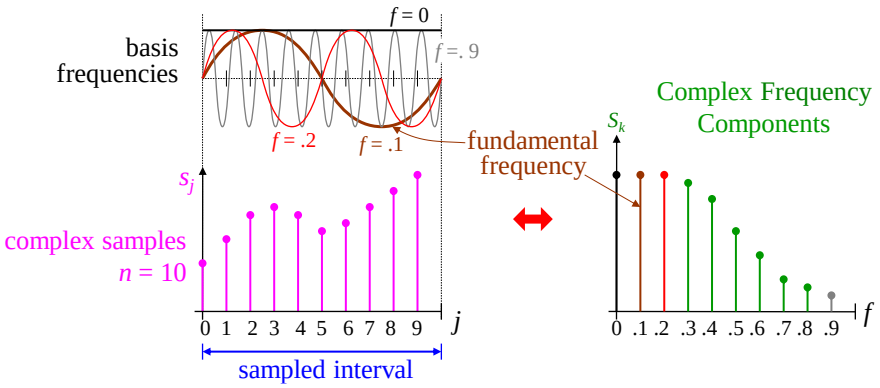


Figure 12.4 Samples in time, and their *normalized* frequencies. For simplicity, all values are shown as real, but in general, are complex.

Normalized frequencies are equivalent to measuring time in units of the sample time, and frequencies in cycles/sample.

For theoretical analysis, it is often more convenient to have the frequency range be $-0.5 < f \le 0.5$, instead of $0 \le f < 1$. Since any frequency f is equivalent to (an alias of) $f - 1$, we can simply move the frequencies in the range $0.5 < f < 1$ down to $-0.5 < f < 0$:

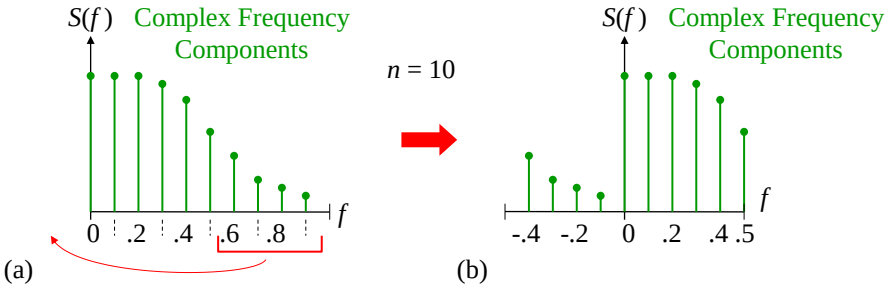


Figure 12.5 Even number of samples: two choices for the frequency set: (a) 0 to just-under 1; and (b) the set including negative frequencies, which is asymmetric.

Note that the lowest frequency is –0.4, and the highest is +0.5. For an even number of samples (and frequencies, diagram above), the resulting frequency set is necessarily asymmetric, because there is no $f = -0.5$, but there is an $f = +0.5$.

For an odd number of points (below), the frequency set *is* symmetric, and there is neither $f = -0.5$ nor $f = +0.5$:

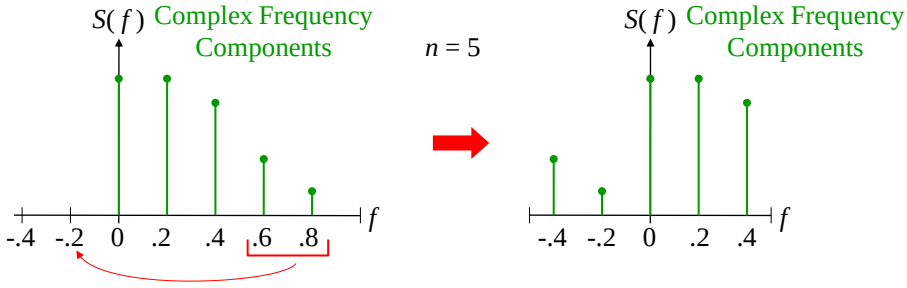


Figure 12.6 Odd number of samples: two choices for the frequency set. The set including negative frequencies (right) is symmetric.

Non-Equivalence of DFT and FT of Series of Time-Domain Impulses (Again)

As noted earlier, it is often said that sampling is like setting the function to zero between samples, or creating a series of impulse functions. This is a common misconception. It is well refuted by [Openheim and Schafer p8b, and dozens of other signal processing experts]. It is easy to show that that claim is not true, in several ways. One simple way is this: For a band-limited signal, I can reconstruct the signal between the sample times from just the samples alone. That makes no sense if sampling amounted to zeroing the signal between samples, because that would be a new function, which would lose information about the original. There would then be no way to recreate the original function from its DFT.

Furthermore, it is often said that the FT of a series of time-domain impulses is identical to the DFT of samples at those times. From the previous paragraph, this cannot be true, either. However, the following *is* true only at the (integer) k defined DFT frequencies of $k\omega_1$:

$$S(\omega_k) = S(k\omega_1) = \frac{1}{n}\sum_{j=0}^{n-1} s_j e^{-ik\omega_1 t_j} = \frac{1}{n}\int_{-\infty}^{\infty}\left[\sum_{j=0}^{n-1} s_j \delta(t - t_j)\right] e^{-ik\omega_1 t}\, dt \quad .$$

$\omega_1 \equiv$ fundamental frequency.

For frequencies in between the $k\omega_1$, the DFT is formally undefined, but can be taken as zero for the purpose of reconstructing the original samples. However, at those in-between frequencies the FT has some non-zero values which are usually of little interest. So we see that the FT of a series of weighted impulses (representing the sample values), *evaluated at the DFT frequencies* $k\omega_1$, is proportional to the DFT, but the full-spectrum of the FT is *different* from the spectrum of the DFT. Hence, the two transformations are *not* equivalent.

Basis Functions and Orthogonality

The basis functions of the DFT are the discrete-time exponentials, which are equivalent to sines and cosines:

$$b_k(j) = e^{i(2\pi k/n)j}$$

where $j \equiv$ sample index $= 0, 1, \ldots n-1,$

$$k \equiv \text{frequency index} = 0, 1, \ldots n-1 \quad \text{or} \quad \begin{cases} n \text{ even:} & -n/2+1, \ldots -1, 0, 1 \ldots n/2 \\ n \text{ odd}: & -\text{int}(n/2), \ldots -1, 0, 1, \ldots \text{int}(n/2). \end{cases}$$

Note that:

> The DFT and FT are simply decompositions of functions into basis functions, just like in ordinary quantum mechanics. The transform equations are just the inner products of the given functions with the basis functions.

The basis functions are orthogonal, normalized (in our convention) such that $\langle b_k | b_m \rangle = n\,\delta_{km}$. Proof:

$$\langle b_k | b_m \rangle \equiv \sum_{j=0}^{n-1} b_k^*(j) b_m(j) = \sum_{j=0}^{n-1} e^{-i(2\pi k/n)j} e^{i(2\pi m/n)j} = \sum_{j=0}^{n-1} e^{i(2\pi/n)(m-k)j} = \sum_{j=0}^{n-1} \left[e^{i(2\pi/n)(m-k)} \right]^j .$$

For $k = m$, we have $\langle b_k | b_m \rangle = \sum_{j=0}^{n-1} \left[e^0 \right]^j = n$.

For $k \neq m$, use $\sum_{j=0}^{n-1} r^j = \frac{1-r^n}{1-r}$ *where* $r = \left[e^{i(2\pi/n)(m-k)} \right]$. Then:

$$\langle b_k | b_m \rangle = \frac{1 - \left[e^{i(2\pi/n)(m-k)} \right]^n}{1 - \left[e^{i(2\pi/n)(m-k)} \right]} = \frac{1 - e^{i(2\pi)(m-k)}}{1 - \left[e^{i(2\pi/n)(m-k)} \right]} = \frac{1-1}{1 - \left[e^{i(2\pi/n)(m-k)} \right]} = 0 \quad \Rightarrow \quad \boxed{\langle b_k | b_m \rangle = n \, \delta_{km}} .$$

The orthogonality condition allows us to immediately write the DFT from the definition of the IDFT above:

Discrete Fourier Transform:

$$\boxed{S_k = \frac{1}{n} \sum_{j=0}^{n-1} s_j e^{-i2\pi(k/n)j}, \quad \textit{where} \quad S_k \equiv \text{the } k^{\text{th}} \text{ complex frequency component}, \quad \frac{k}{n} = \text{normalized frequency of the } k^{\text{th}} \text{ component, in cycles/sample}.} \tag{12.2}$$

Note that there are $2n$ independent real numbers in the complex sequence s_j, and there are also $2n$ independent real numbers in the complex spectrum S_k, as there must be (same number of degrees of freedom). For each k, S_k is the correlation of the data with the sampled sinusoid at frequency k/n cycles/sample, eq. (8.6).

Real Sequences

An important case of sequence s_j is a real-valued sequence, which is a special case of a complex-valued sequence. In this section, we use the positive and negative frequency DFT form, where k takes on both negative and positive integer values. Then for $s_j = \sum_{k \approx -n/2}^{\approx n/2} S_k e^{i2\pi(k/n)j}$ to be real, the S_k must occur in complex conjugate pairs, i.e., the spectrum S_k must be **conjugate symmetric**:

$$S_k = S_{-k}^* \qquad \text{for } s_j \text{ real, and } k \leq \text{int}(n/2) + 1 .$$

This implies that S_0 is always real, which is also clear since S_0 is just the average of the real sequence.

We now discuss the lower limit for k. (As discussed earlier, there is no $k = -n/2$). There are n independent real numbers in the real sequence s_j. We now consider two subcases: n is even, and n is odd.

For n even,

$$s_j = \sum_{k=-n/2+1}^{n/2} S_k e^{i2\pi(k/n)j} \qquad n \text{ even}, s_j \text{ real},$$

and we use the asymmetric frequency range $-0.5 < f \leq 0.5$, which corresponds to $-n/2 + 1 \leq k \leq n/2$ (Figure 12.7, left). For an even number of sample points, since there are no conjugates to $k = 0$ or $k = n/2$, we must have that S_0 and $S_{n/2}$ are real (actually, S_0 being real still satisfies conjugate symmetry). All other S_k may be complex, and are conjugate symmetric: $S_{-k} = S_k^*$.

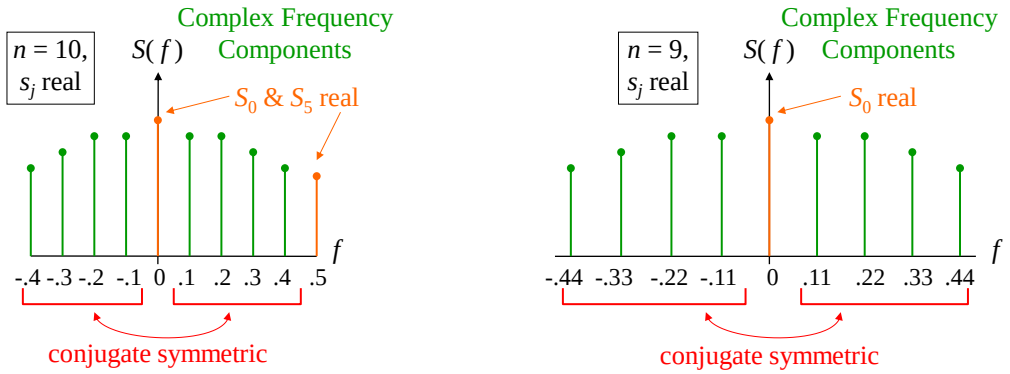


Figure 12.7 (Left) Sequence with even number of samples, $n = 10$. (Right) Sequence with odd number, $n = 9$.

Therefore, in the spectrum, there are $(n/2 - 1)$ independent complex frequency components, plus two real components, totaling n independent real numbers in the spectrum, matching the n independent real numbers in the sequence s_j.

In terms of sine and cosine components (rather than the complex components), there are $(n/2 + 1)$ independent cosine components, and $(n/2 - 1)$ independent sine components. All n_f frequencies are nonnegative:

$$s_j = A_0 + \sum_{k=1}^{n/2-1} \left\{ A_k \cos\left[2\pi (k/n) j\right] + B_k \sin\left[2\pi (k/n) j\right] \right\} + A_{n/2} \cos \pi j \qquad (12.3)$$

$$n \text{ even}, \; s_j \text{ real}, \; n_f = n/2 + 1 .$$

Note that in the last term, $\cos \pi j$ is just an alternating sequence of +1, –1, +1, ...

For an odd number of points (Figure 12.7, right),

$$s_j = \sum_{k=-(n-1)/2}^{(n-1)/2} S_k e^{i2\pi (k/n) j}, \qquad n \text{ odd},$$

there is no $k = n/2$ component, and again there is no conjugate to $k = 0$. Therefore, we must have that S_0 is real. All other S_k are conjugate symmetric. Therefore, in the spectrum, there are $(n - 1)/2$ independent complex frequency components, plus one real component (S_0), totaling n independent real numbers in the spectrum, matching the n independent real numbers in the sequence s_j.

In terms of sine and cosine components (rather than the complex components), there are $(n + 1)/2$ independent cosine components, and $(n - 1)/2$ independent sine components. All n_f frequencies are nonnegative:

$$s_j = A_0 + \sum_{k=1}^{(n-1)/2} \left\{ A_k \cos\left[2\pi \left(\frac{k}{n}\right) j\right] + B_k \sin\left[2\pi \left(\frac{k}{n}\right) j\right] \right\}, \qquad (12.4)$$

$$n \text{ odd}, \; s_j \text{ real}, \; n_f = (n+1)/2$$

Note that there is no final lone-cosine term, and no alternating sequence.

These examples illustrate how the notation is slightly more involved for the cosine/sine form than for the complex exponential form.

Normalization and Parseval's Theorem

When the original sequence represents something akin to samples of voltage over time, we sometimes speak of "energy" in the signal. The energy of the signal is the sum of the energies of each sample:

$$E_j = G s_j^{\,2} = s_j^{\,2}, \quad \textit{where} \quad G = \text{"conductance", choosen to be 1.}$$

$$E = \sum_{j=0}^{n-1} E_j = \sum_{j=0}^{n-1} s_j^{\,2}.$$

When the "conductance" is chosen to be 1, or some other reference value, the "energy" in the signal does *not* correspond to any physical energy (say, in joules).

The energies of the sinusoidal components in the DFT add as well, because the sinusoids are orthogonal (show why??):

$$E \propto \sum_{k=0}^{n-1} |S_k|^2 \ .$$

Parseval's Theorem equates the energy of the original sequence to the energy of the sinusoidal components, by providing the constant of proportionality. First, we evaluate the energy of a single sinusoid:

$$E_k = |S_k|^2 \sum_{j=0}^{n-1} \left| e^{i(2\pi k/n)j} \right|^2 = n|S_k|^2 \qquad \textit{where} \quad k \equiv \text{frequency index} = 0, 1, \ldots n-1 \ .$$

Then summing over all frequencies yields:

$$E = \sum_{k=0}^{n-1} E_k = n \sum_{k=0}^{n-1} |S_k|^2 \quad \Rightarrow \quad \boxed{E = n \sum_{k=0}^{n-1} |S_k|^2 = \sum_{j=0}^{n-1} s_j^{\,2}} \ . \tag{12.5}$$

Energy For Real Sequences: We derive Parseval's Theorem for real sequences in two ways. Since the s_j are real, the interesting form is the cosine/sine form of DFT, (12.3) and (12.4). We again consider separately the cases of n even, and n odd.

First, for n even, k runs from 0 to (n/2). We can deduce the equation for Parseval's Theorem by exploiting the conjugate symmetry of the S_k. Recall that S_0 has no conjugate term, nor does $S_{n/2+1}$. Therefore:

$$E_0 = n A_0^{\,2}, \qquad E_{n/2+1} = n \left(A_{n/2} \right)^2 .$$

For k = 1, ... n/2, we have:

$$A_k = 2\,\mathrm{Re}\{S_k\}, \quad B_k = 2\,\mathrm{Im}\{S_k\}, \quad |S_k|^2 + |S_{-k}|^2 = 2\,\mathrm{Re}\{S_k\}^2 + 2\,\mathrm{Im}\{S_k\}^2 \qquad \Rightarrow$$

$$E_k = \frac{n}{2} \left(A_k^{\,2} + B_k^{\,2} \right), \quad k = 1, \ldots n/2 \ .$$

We can derive this another way directly from the definition (12.3). Since A_0 is a constant added to each s_j, the energy contributed from this term is $E_0 = nA_0$. Since cos πj is just alternating +1, –1, ..., it's energy at each sample is 1, and $E_{n/2+1} = n(A_{n/2})^2$. Finally, the average value of cos^2 over a full period is ½, as is the average of sin^2. Therefore, for k = 1, ... n/2, $E_k = (n/2)(A_k^2 + B_k)^2$.

Second, for n odd, k runs from 0 to (n – 1)/2. The above arguments still apply, but there is no lone-cosine term at the end. Therefore the result is the same, without the lone-cosine term.

Summarizing:

$$n \text{ even:} \quad E = nA_0{}^2 + n\left(A_{n/2}\right)^2 + \frac{n}{2}\sum_{k=1}^{n/2-1}\left(A_k{}^2 + B_k{}^2\right)$$

$$n \text{ odd:} \quad E = nA_0{}^2 + \frac{n}{2}\sum_{k=1}^{(n-1)/2}\left(A_k{}^2 + B_k{}^2\right)$$

Other normalizations: Besides our normalization choice above, there are several other choices in common use. In general, between the DFT, IDFT, and Parseval's Theorem, you can choose a normalization for one, which then fixes the normalization for the other two. For example, some people choose to make the DFT and IDFT more symmetric by defining:

$$\left.\begin{aligned} \text{IDFT:} \quad s_j &= \frac{1}{\sqrt{n}}\sum_{k=0}^{n-1} S_k e^{(i2\pi k/n)j} \\ \text{DFT:} \quad S_k &= \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1} s_j e^{-i(2\pi k/n)j} \end{aligned}\right\} \Rightarrow \qquad \sum_{k=0}^{n-1}|S_k|^2 = \sum_{j=0}^{n-1} s_j{}^2 \qquad \text{(alternate normalizations)}.$$

Continuous and Discrete, Finite and Infinite

TBS: Finite length implies discrete frequencies; infinite length implies continuous frequencies. Discrete time implies finite frequencies; continuous time implies infinite frequencies. Finite length is equivalent to periodic.

White Noise, Correlation, and Gaussianity

White noise has, on average, all frequency components of equal *amplitude* (named by incorrect analogy with white light); samples of white noise are uncorrelated. Non-white noise has unequal frequency components (on average); samples of non-white noise are necessarily correlated. (Show this??).

White noise may or may not be gaussian. Gaussian noise may or may not be white.

For example, shot noise (aka "impulse noise") is white, but not at all gaussian. Pink noise may be gaussian, but is not white. Gaussian noise, after being sent through a linear filter, is still gaussian (the weighted sum of any number of gaussian RVs is still gaussian). Clearly though, the noise after filtering has the frequency transfer function of the filter impressed on it, and is not white.

Why Oversampling Does Not Improve Signal-to-Noise Ratio

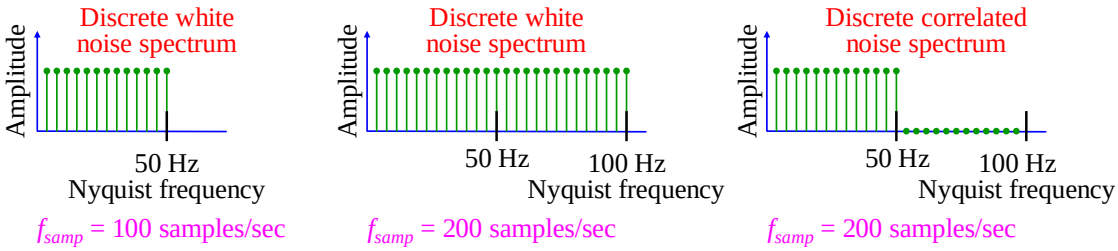
Sometimes it might seem that if I oversample a signal (i.e., sample above the Nyquist rate), the noise power stays constant (= noise variance is constant), but I have more samples of the signal which I can average. Therefore, by oversampling, it seems like I should be able to improve my SNR by averaging out more noise, but keeping all the signal.

This reasoning is wrong, of course, because it implies that by sampling arbitrarily fast, I can filter out arbitrarily large amounts of noise, and ultimately recover anything from almost nothing. So what's wrong with this reasoning? Let's take an example.

Suppose I sample a signal at 100 samples/sec, with white noise. Then my Nyquist frequency is 50 Hz, and I must use a 50 Hz Low Pass Filter (LPF) for anti-aliasing before sampling. This LPF leaves me with 50 Hz worth of noise power (= variance).

Now suppose I double the sampling frequency to 200 samples/sec. To maintain white noise, I must open my anti-alias filter cutoff to the new Nyquist frequency, 100 Hz. This doubles my noise power. Now I have twice as many samples of the signal, with twice as much noise power. I can run a LPF to reduce the noise (say, averaging adjacent samples). At best, I cut the noise by half, reducing it back to its 100 sample/sec

value, and reducing my sample rate by 2. Hence, I'm right back where I was when I just sampled at 100 samples/sec in the first place.



But wait! Why open my anti-alias LPF? Let's try keeping the LPF at 50 Hz, and sampling at 200 samples/sec. But then, my noise occupies only ½ of the sampling bandwidth: it occupies only 50 Hz of the 100 Hz Nyquist band. Hence, the noise is *not* white, which means adjacent noise samples are *correlated*! Hence, when I average adjacent samples, the noise variance does *not* decrease by a factor of 2. The factor of 2 gain *only* occurs with uncorrelated noise. In the end, oversampling buys me nothing.

Filters TBS??

FIR vs. IIR. Because the data set can be any size, and arbitrarily large:

The transfer function of an FIR or IIR is continuous.

Consider some filter. We must carefully distinguish between the filter in general, which can be applied to *any* data set (with any n), and the filter as applied to one particular data set. Any given data set has only discrete frequencies; if we apply the filter to the data set, the data set's frequencies will be multiplied by the filter's transfer function at those frequencies. But we can apply *any* size data set to the filter, with frequency components, $f = k/n$, anywhere in the Nyquist interval. For every data set, the filter has a transfer function at all its frequencies. Therefore, the filter in general has a *continuous* transfer function.

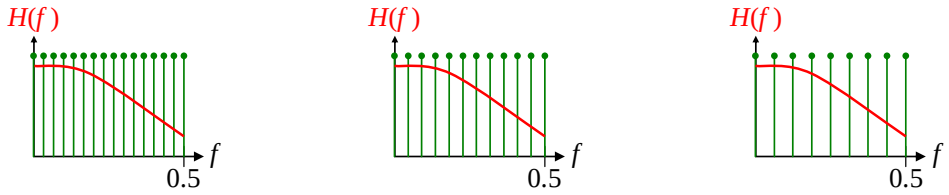


Figure 12.8 Data sets with different n sample the transfer function $H(f)$ at different points. $H(f)$, in general, is a continuous curve, defined at *all* points in the Nyquist interval $f \in [0, 1)$ or $(-0.5, +0.5]$.

What Happens to a Sine Wave Deferred?

"... Maybe it just sags, like a heavy load. Or does it explode?" [Sincere apologies to Langston Hughes.] You may ask, "The DFT has only a finite set of basis frequencies. Can I use a DFT to estimate the frequency of an unknown signal? What happens if I sample a sinusoid of a frequency *in between* the chosen DFT basis frequencies? What is its spectrum?" Good questions. We now demonstrate. The results here are important for generalizing the DFT, and spectral analysis in general, to non-uniformly sampled signals.

We choose $n = 40$ samples, which means the basis frequencies are $k(1/n)$, $k = -19, \ldots 0, \ldots 20$, measured in cycles per sample (or equivalently, in units of the sampling rate, f_{samp}). The frequency spacing is $1/n = 0.025$ cycle/sample. No other frequencies exist in the DFT.

First, we show the result of sampling an *existing*-frequency sinusoid of $f = 10/n = 0.25$ cycle/sample ($k = 10$). Since the signal is real, the spectrum is conjugate symmetric ($S_{-k} = S_k^*$); therefore, I show only the positive frequencies, and double their magnitudes:

$$s_j = \cos\left(\frac{2\pi k}{n} j\right), \qquad f = \frac{k}{n} \text{ cycle/sample} .$$

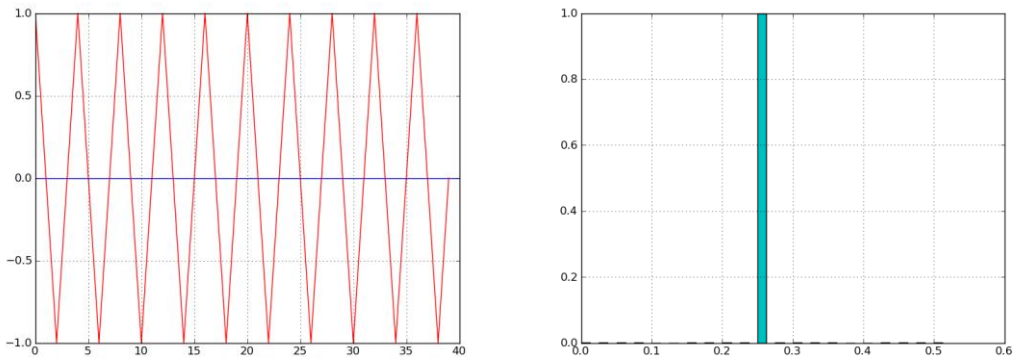
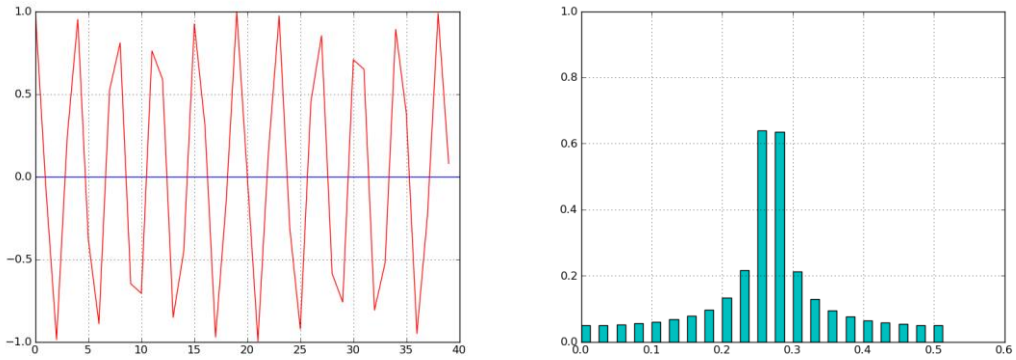


Figure 12.9 (Left) A sampled sinusoid of $f = 0.25$, $n = 40$. (Right) As expected, its magnitude spectrum (DFT) has exactly one component at $f = 0.25$, with magnitude 1.0.

[Notice that when the sample points are connected by straight segments, the sinusoid doesn't "look" sinusoidal, but recall that connecting with straight segments is *not* the proper way to interpolate between samples.]

The "energy" of the sample set is exactly (1/2)40 = 20, because there is an integral number of cycles in the sample set, and the average energy of a sinusoid is ½.

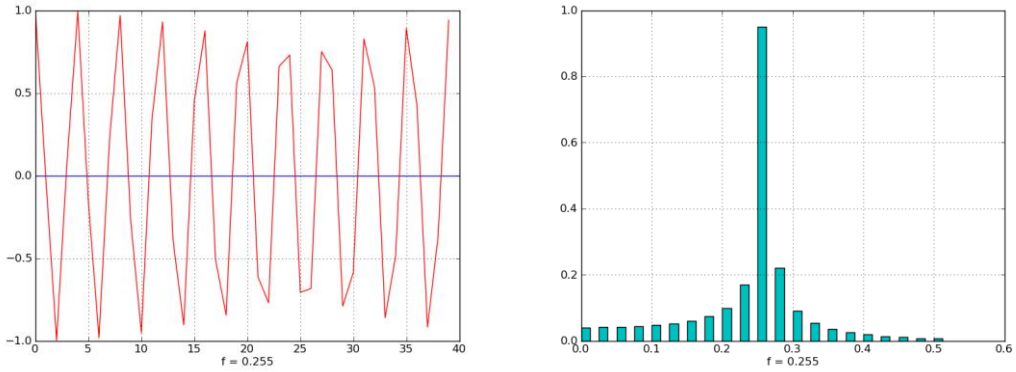
Now we take our signal off-frequency by half the frequency spacing: $f = 10.5/n = 0.2625$ cycle/sample, halfway between two chosen basis frequencies:



(Left) A sampled sinusoid of $f = 0.2625$, $n = 40$. (Right) Its magnitude spectrum (DFT) has components everywhere, but is peaked around $f = 0.2625$.

Not too surprisingly, the components are peaked at the two basis frequencies closest to the sinusoid frequency, but there are also components at *all other* frequencies. This is an artifact of sampling a pure sinusoid of a non-basis frequency for a finite time. Note also that the total energy in the *sampled* $f = 0.26$ signal is slightly *larger* than that of the $f = 0.25$ signal, even though both signals are the *same* amplitude. This is due to a few more of the samples being near a peak of the signal. This shift in total energy is another artifact of sampling a non-basis frequency. For other signal frequencies, or other time shifts, the energy could just as well be *lower* in the sampled signal. This energy shift also explains why the two largest components of the spectrum are not exactly equal, even though they are equally distant from the true signal frequency of $f = 0.2625$.

Finally, instead of being half-way between allowed frequencies, suppose we're only 0.2 of the way, $f = 10.2/n = 0.255$ cycle/sample:



(Left) A sampled sinusoid of $f = 0.255$, $n = 40$. (Right) Its magnitude spectrum (DFT) has components everywhere, is asymmetric, and peaked at $f = 0.25$.

The two largest components are still those surrounding the signal frequency, with the larger of the two being the one closer to the signal frequency.

These examples show that a DFT, with its fixed basis frequencies, can give only a rough estimate of an unknown sinusoid's frequency. The estimate gets worse if the unknown signal is not exactly a sinusoid, because then it has an even smaller spectral peak, with more components spread around the spectrum.

Other methods exist for estimating the frequency of an unknown signal (even one that is non-uniformly sampled in time). If the signal is fairly sinusoidal, one can correlate with a sinusoidal basis frequency, and numerically search for the frequency with maximum correlation. This avoids the discrete-frequency limitation of a DFT (see discussion around Figure 12.10). Other methods usually require many periods of data, e.g. phase dispersion minimization (PDM), or epoch folding [Leahy, Ap J, 1983, vol 266, p160??].

We discuss nonuniform sampling in more detail in the chapter "Period Finding".

Don't Pad Your Data, Even for FFTs

Old fashioned FFT implementations required you to have N = a power of 2 number of samples (64, 1024, etc.). Modern FFT implementations are general to any number of samples, and use the prime decomposition of N to provide the fastest and most accurate DFT known. The worst case is when N is prime, and no FFT optimization is possible: the DFT is evaluated directly from the defining summations, requiring compute time $O(n^2)$. But with modern computers, this is usually so fast that we don't care.

In the old days, if people had a non-power-of-2 number of data points, they used to "pad" their data, typically (and horribly) by just adding zeros to the end until they reached a power of 2. This introduced artifacts into the spectrum, which often obscured or destroyed the very information they sought: "Padding with zeros to cover the missing data will not give you reasonable results" [Ham p218m].

> Don't pad your data. It screws up the spectrum.
> With a modern FFT implementation, there is no need for it, anyway.

If for some reason, you absolutely must constrain N to some preferred values, it is much better to throw away some data points than to add fake ones.

Don't Pad Your Data, Even for Finer Frequency Resolution

Sometimes, people try to get finer frequency resolution by padding their data with zeros, since the number of frequencies (for real-valued data) is about $n/2$ (see (12.3) and (12.4)). We show here why this is usually a bad idea. (We show at the end a very limited case where padding with zeros can be useful.)

To see the problem with zero-padding, suppose we have 64 samples of an exact frequency f = 1/32 cyc/sample. It's spectrum is Figure 12.10a: exactly the one frequency.

Now let's pad the data with 16 zeros (n = 80) in a misguided attempt to get higher frequency resolution. The resulting spectrum is Figure 12.10b. The zeros have smeared out the true component, and introduced many spurious components that highly misrepresent the original data. If instead we add 32 zeros (n = 96), we get another bad spectrum: Figure 12.10c.

Don't pad your data! It corrupts the very content you're analyzing. If you want finer frequency resolution, just use a periodogram (Figure 12.10d). This plots the fraction of the data variation that is accounted for by a sinusoid at each trial frequency. In this no-noise example, the fraction approaches 1 as the trial frequency approaches the true signal frequency, 0.03125 cps. The resulting frequency estimate is dramatically more accurate (within 1.4%) than what you get from Figure 12.10b or (c). (See *Fourier Transforms, Periodograms, and (Deprecated) Lomb-Scargle* below.)

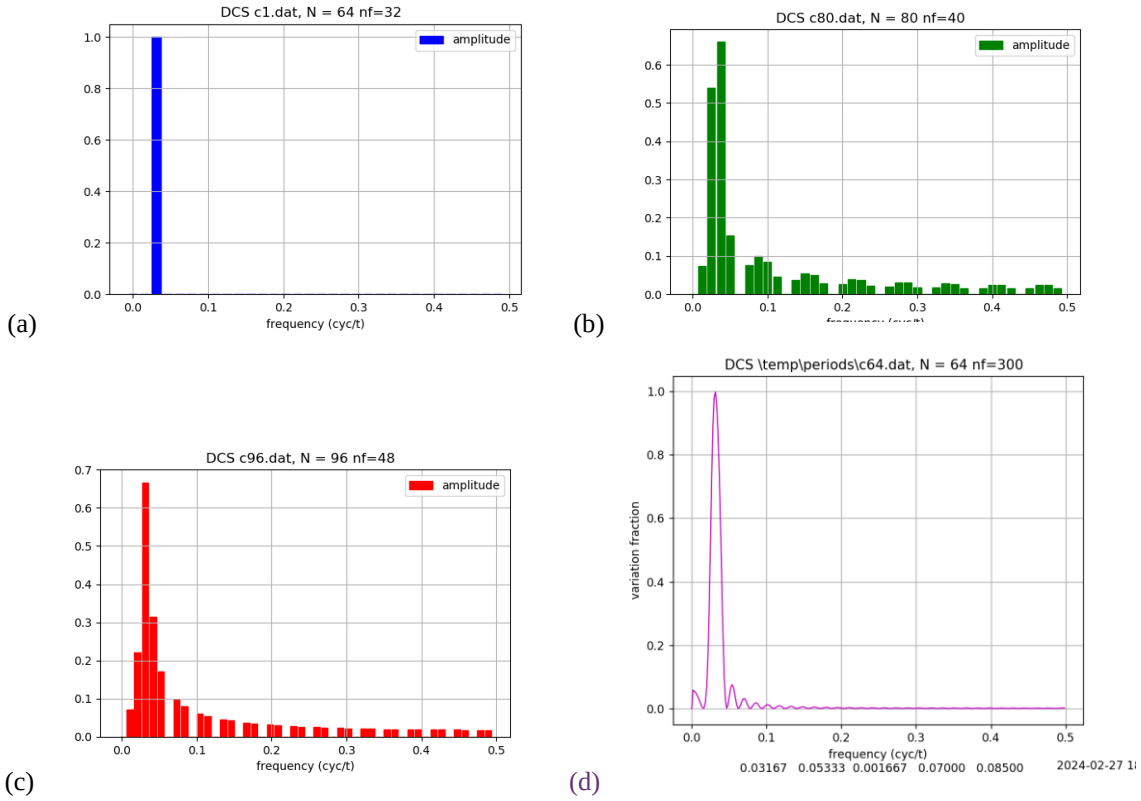


Figure 12.10 (a) Amplitude spectrum of f = 1/32 cyc/samp with 64 samples. (b) Amplitude spectrum with same data padded with 16 zeros. (c) Amplitude spectrum with same data padded with 32 zeros. (d) Original 64 samples, dense periodogram with 300 frequencies. Note that the "unknown" frequency (0.03125 cps) is unmistakable, and closely estimated as 0.03167 cps.

We can compute exactly how the padded DFT spectrum is distorted. Assuming a real sequence s_j, it is convenient to use the complex notation, and one-sided DFT. It's probably easiest to use frequency indexes from 0 to n–1 (normalized frequencies $0 \le f < 1.0$), and then we don't care if n is even or odd. (I find that many theoretical analyses are easiest with this convention, though in practical applications, I find that using frequencies only up to $f \le 0.5$ is easiest.) Then the correct DFT is:

$$S_k = \frac{1}{n}\sum_{j=0}^{n-1} s_j e^{-i2\pi(k/n)j}, \quad k = 0, \ldots n-1 .$$

If we pad with zeros, so there are M data points, the new spectrum (such as Figure 12.10b) is:

$$S_l = \frac{1}{M}\sum_{j=0}^{M-1} s_j e^{-i2\pi(l/M)j}, \quad l = 0, \ldots M-1 .$$

To see directly how the true spectrum is scrambled into the new spectrum, we write the samples s_j as the inverse transform of their true spectrum:

$$S_l = \frac{1}{M}\sum_{j=0}^{M-1}\left(\sum_{k=0}^{n-1} S_k e^{+i2\pi(k/n)j}\right) e^{-i2\pi(l/M)j}, \quad l = 0, \ldots M-1 .$$

We can combine the exponents, and then we can perform the sum over j explicitly, since it forms a geometric series. You can think of this as changing the order of summation:

$$S_l = \frac{1}{M}\sum_{k=0}^{n-1}\left(\sum_{j=0}^{M-1} S_k \exp\left[i2\pi\left(\frac{k}{n}-\frac{l}{M}\right)j\right]\right), \qquad l = 0, \ldots M-1.$$

$$\text{Let} \qquad r \equiv \exp\left[i2\pi\left(\frac{k}{n}-\frac{l}{M}\right)\right] \quad \Rightarrow \quad \sum_{j=0}^{M-1}(\bullet) \equiv sum_{k,l} = \frac{1-r^M}{1-r} = \frac{1-\exp\left[i2\pi\left(\frac{k}{n}-\frac{l}{M}\right)M\right]}{1-\exp\left[i2\pi\left(\frac{k}{n}-\frac{l}{M}\right)\right]}.$$

Notice that ($k/n - l/M$) is just the frequency difference between the original frequency $f_k = k/n$ and the new frequency $f_l = l/M$. Finally:

$$S_l = \frac{1}{M}\sum_{k=0}^{n-1} S_k sum_{k,l}\,.$$

Also note that *even if the frequency difference is zero*, i.e., even if $f_k = f_l$, the amplitude for f_l is still distorted, because the summation includes a bunch of zeros that are *not in the original data*! (In this case, the "geometric" sum is just M, and the geometric formula above does not apply.)

The Limited Case Where Zero-Padding Is Useful

Compared to DFTs, dense periodograms have the advantages of (1) allowing individual uncertainties, (2) allowing nonuniform sampling, (3) allowing nonuniform frequencies (e.g., uniform periods), and (4) a variety of detection algorithms. The complexity of most algorithms is $O(n \cdot n_f)$, where $n \equiv$ #samples, and $n_f \equiv$ #frequencies in the periodogram (~200). In the rare case where you have (1) a very large number of data points (millions), (2) uniform sampling, (3) uniform uncertainties, (4) uniform frequency spacing, (5) want sinusoidal correlation detections, *and* (6) computing time is *critical*, you can pad your data to a number $N > n$ where the FFT algorithm is very efficient. N should be a product of many small primes (2, 3, and 5), and often a power of 2. The complexity of the FFT is $O(n \log n)$, which beats a periodogram by a factor of $\sim n_f/(\log n)$, which is about a factor of 10 for n = 1,000,000, and about a factor of 20 for n = 1,000.

Note that sinusoidal correlation is often an undesirable detection algorithm;
if so, even this limited case does not provide a reason for padding.

To see that the FFT on N points is equivalent to a simple periodogram based on sinusoidal correlation with n points, consider the simple detection statistic θ for a periodogram that is just correlation against the arbitrary trial frequency cosine and sine:

$$\theta = \underbrace{\left(\sum_{j=0}^{n-1} s_j \cos(\omega j T)\right)^2}_{A_k} + \underbrace{\left(\sum_{j=0}^{n-1} s_j \sin(\omega j T)\right)^2}_{B_k} \qquad \textit{where} \quad \omega \equiv \text{trial frequency}, T \equiv \text{sample period}\,.$$

This is exactly proportional to a DFT/FFT power parameter: $A_k^2 + B_k^2$. If we pad our data s_j with $N - n$ zeros, then we can extend the summations from $n - 1$ to $N - 1$ with no effect. The new sum is exactly a DFT/FFT (12.2) on the padded data (to within the normalization constant).

Two Dimensional Fourier Transforms

One dimensional Fourier transforms often have time or space as the independent variable. Two dimensional transforms almost always have space, say (x, y), as the independent variables. The most common 2D transform is of pictures.

In the continuous world of light, lenses can physically project a Fourier transform of an image based on optics, with no computations. This allows for filtering the image with opaque masks, and re-transforming back to the original-but-filtered image, all at the speed of light with no computer. But digitized images store the image as pixels, each with some light intensity. These are computationally processed by computer.

Basis Functions

TBS. Not sines and cosines, or products of sines and cosines. Products of complex exponentials. Wave fronts at various angles, discrete k_x and k_y.

Note on Continuous Fourier Series and Uniform Convergence

The continuous Fourier Series is defined for a periodic signal $s(t)$ over a continuous range of times, $t \in [0, T)$:

$$s(t) = \sum_{k=0}^{\infty} S_k e^{i2\pi k\omega_0}, \qquad \textit{where} \quad k\omega_0 \text{ is the frequency of the } k^{th} \text{ component}$$

$$S_k \text{ is the complex amplitude of the component}$$

Note that the time interval is continuous, but the frequency components are discrete. In general, periodic signals lead to discrete frequency components.

The continuous Fourier Series for discontinuous functions is *not* uniformly convergent. Therefore, the order of integrations and summations cannot always be interchanged.

Non-uniform convergence is illustrated by the famous **Gibbs phenomenon**, frequently observed in digital electronics: when we transform a square wave to the frequency domain (aka Fourier space), then retain only a finite number of frequency components, and then transform back to the time domain, the square wave comes back with **overshoot**: wiggles that are large near the discontinuities:

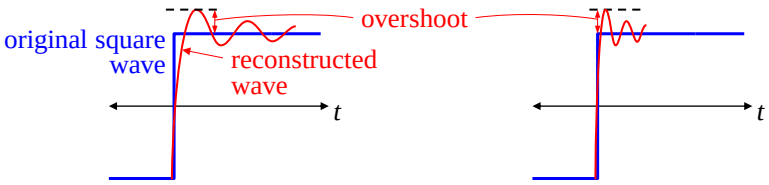


Figure 12.11 Gibbs phenomenon: (Left) After losing high frequencies, the reconstructed square wave has overshoot and wiggles. (Right) Retaining more frequencies reduces wiggle time, but not amplitude.

As we include more and more frequency components, the wiggles get narrower (damp faster), but do not get lower in amplitude. This means that there are always some time points for which the inverse transform does not converge to the original square wave. Such wiggles are commonly observed in many electronic systems, which must necessarily drop high frequency components above some cut-off frequency.

However:

Continuous signals have Fourier Series that converge uniformly. This applies to most physical phenomena, so interchanging integration and summation is valid [F&W p217+].

This is true even if the derivative of the signal is discontinuous.

Nonuniform Sampling and Arbitrary Basis Functions

So far, we have used a signal sampled uniformly in time. We now show that one can find a Fourier transform of a signal with *any* set of n samples, uniform or not. This has many applications: some experiments (such as lunar laser ranging) cannot sample the signal uniformly for practical, economic, or political reasons. Magnetic Resonance Imaging (MRI) often uses non-uniform sampling to reduce imaging time, which can be an hour or more for a patient.

We write the required transform as a set of simultaneous equations, with t_j as the arbitrary sample times, and keeping (for now) the uniformly spaced frequencies:

$$s(t_0) = \sum_{k=0}^{n-1} S_k \exp\left(i\left(2\pi k / n\right)t_0\right)$$

$$s(t_1) = \sum_{k=0}^{n-1} S_k \exp\left(i\left(2\pi k / n\right)t_1\right)$$

...

$$s(t_{n-1}) = \sum_{k=0}^{n-1} S_k \exp\left(i\left(2\pi k / n\right)t_{n-1}\right).$$

Or:

$$\begin{matrix} & S_0 & S_1 & \dots & S_{n-1} \end{matrix}$$
$$\begin{bmatrix} s(t_0) \\ s(t_1) \\ \vdots \\ s(t_{n-1}) \end{bmatrix} = \begin{bmatrix} 1.0 & \exp\left(2\pi f_1 t_0\right) & \dots & \exp\left(2\pi f_{n-1} t_0\right) \\ 1.0 & \exp\left(2\pi f_1 t_1\right) & \dots & \exp\left(2\pi f_{n-1} t_1\right) \\ \vdots & \vdots & \ddots & \vdots \\ 1.0 & \exp\left(2\pi f_1 t_{n-1}\right) & \dots & \exp\left(2\pi f_{n-1} t_{n-1}\right) \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{n-1} \end{bmatrix}.$$

How can we find the required coefficients, S_k?

The exponential functions are no longer orthogonal over the sample times;
they are only orthogonal over uniformly spaced samples.

Nonetheless, we have n unknowns (S_0, ... S_{n-1}), and n equations. So long as the basis functions are *linearly independent* over the sample times, we can (in principle) solve for the needed coefficients, S_k. We have now greatly expanded our ability to decompose arbitrary samples into basis functions:

We can decompose a signal over any set of sample times into any set of linearly independent
(not necessarily orthogonal) basis functions.

Note that Parseval's theorem does *not* apply to the coefficients, since the basis functions (evaluated at the non-uniform sample points) are no longer orthogonal. Also, S_0 is no longer the average of the signal values, since the sinusoids may have nonzero average over the sample points.

There is one more subtlety: what is the fundamental frequency f_1? Equivalently, what is the signal period? The two are related, because f_1 = 1/period. There is no unique answer to this. However, since a finite-time signal transforms as if it is periodic, the period cannot be ($t_{n-1} - t_0$), since the first and last samples would then have to be identical. The period must be longer than that. A convenient choice is to simply mimic what happens when the samples *are* uniform. In that case,

$$period = \left(t_{n-1} - t_0\right)\frac{n}{n-1}, \qquad f_1 = 1/\ period\ .$$

This choice for period reproduces the traditional DFT when the samples are uniform, and is usually adequate for non-uniform samples, as well.

Example: DFT of a real, non-uniformly sampled sequence: We can set up the matrix equation to be solved by recalling the frequency layout for even and odd n, and applying the above. We set $t_0 = 0$, and for illustration of the last two matrix columns, we take n odd:

$$s(t_0) = A_0 + \sum_{k=1}^{(n-1)/2} \left[A_k \cos\left(k\omega t_0\right) + B_k \sin\left(k\omega t_0\right) \right] \qquad \textit{where} \quad n \text{ odd}, \ \omega \equiv 2\pi / n$$

$$s(t_1) = A_0 + \sum_{k=1}^{(n-1)/2} \left[A_k \cos\left(k\omega t_1\right) + B_k \sin\left(k\omega t_1\right) \right]$$

$$\vdots$$

$$s(t_{n-1}) = A_0 + \sum_{k=1}^{(n-1)/2} \left[A_k \cos\left(k\omega t_{n-1}\right) + B_k \sin\left(k\omega t_{n-1}\right) \right].$$

Or:

$$\begin{matrix} & A_0 & A_1 & B_1 & \dots & A_{(n-1)/2} & B_{(n-1)/2} \end{matrix}$$

$$\begin{bmatrix} s(t_0) \\ s(t_1) \\ \vdots \\ s(t_{n-1}) \end{bmatrix} = \begin{bmatrix} 1.0 & 1.0 & 0.0 & \dots & 1.0 & 0.0 \\ 1.0 & \cos(\omega t_1) & \sin(\omega t_1) & \dots & \cos\left((n/2)\omega t_1\right) & \sin\left((n/2)\omega t_1\right) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1.0 & \cos(\omega t_{n-1}) & \sin(\omega t_{n-1}) & \dots & \cos\left((n/2)\omega t_{n-1}\right) & \sin\left((n/2)\omega t_{n-1}\right) \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ B_{(n-1)/2} \end{bmatrix}.$$

This gives us the sine and cosine components separately. For n even, the highest frequency component is $k = n/2$, or $\omega = 2\pi k/n = 2\pi(1/2) = \pi$ rad/sample, and the final column of sin(·) is not present.

Note that this is *not a fit*; it is an exact, invertible transformation. The matrix is the set of all the basis functions (across each row), evaluated at the sample points (down each column). The matrix has no summations in it, and depends on the sample points, but not on the sample values.

Example: basis functions as powers of *x*: In the continuous world, a Taylor series is a decomposition of a function into powers of $(x - a)$, which are a set of linearly independent (but not orthogonal) basis functions. Despite this lack of orthogonality, Taylor devised a clever way to evaluate the basis-function coefficients without solving simultaneous equations.

Example: sampled standard basis functions: We could choose a standard (continuous) mathematical basis set, such as Bessel functions, $J_n(t)$. For n sample points, t_1, ... t_n, the Bessel functions are (almost certainly) linearly independent, and we can solve for the coefficients, A_k. We need a scale factor α for the time (equivalent to $2\pi k/n$ in the Fourier transform). For example, we might use α = the $(n-1)^{th}$ zero of $J_{n-1}(t)$. Then:

$$s(t_0) = \sum_{k=0}^{n-1} A_k J_k\left(t_0 \frac{\alpha}{t_{n-1}} \right)$$

$$s(t_1) = \sum_{k=0}^{n-1} A_k J_k\left(t_1 \frac{\alpha}{t_{n-1}} \right)$$

...

$$s(t_{n-1}) = \sum_{k=0}^{n-1} A_k J_k\left(t_{n-1} \frac{\alpha}{t_{n-1}} \right)$$

We have n equations and n unknowns, A_0, ... A_{n-1}, so we can solve for the A_k.

Fourier Transforms, Periodograms, and (Deprecated) Lomb-Scargle

In some circles, one hears the terms "Fourier Transform," "periodogram," and "Lomb-Scargle" a lot. Each of these is distinct, but they are related. Understanding the differences can help you analyze your data. We provide here an overview of some common signal processing algorithms. Be warned:

Because spectral analysis can be tricky, its practice is rife with misunderstanding and mythology.

Throughout the text, I will occasionally note common misunderstandings, but there are too many for me to correct them all. We address the (deprecated) Lomb-Scargle algorithm in particular, since it is widely misunderstood.

Correspondingly, the terminology is also highly confused and abused. We define here some common terms in ways that are consistent with the majority of our (limited) experience in the literature. However, there appears to be little universal agreement on precise definitions, especially across different disciplines. (Words are the tools of communication; it is impossible to make fine points with dull tools.) In this work, we adhere to the following definitions:

- **Spectral analysis** is the examination of periodic components of a data sequence.
- The **energy** of a single data point is its squared magnitude, and is always ≥ 0 (the term "energy" derives from early applications where the squared magnitude was proportional to physical energy). The "energy" of a sample-set is the sum of the squares of the samples. The "energy" of a frequency component is the sum of the "energies" of that frequency taken over the sample times.
- The **power** in a frequency component is its squared magnitude, and is often normalized in some specified way. In some references, the term "energy" is used for "power." (As with "energy," the "power" in a component might be unrelated to physical power.) In this work, we occasionally write "energy" and "power" in quotes, as a reminder that they are not physical energy and power. For non-uniform sampling, the "energy" of a frequency component is *not* proportional to its "power."
- The **statistical significance** is the false alarm probability, often called alpha α. Experimenters usually choose α before analyzing the data. It is the probability that a pure noise signal will, by chance, suggest the presence of a signal. It is essentially the same as the **p-value**. Note that a *lower* significance means a result is *more* significant, i.e. more likely real, and less likely random. Nonetheless, authors often speak loosely of "higher" significance meaning "more significant" or a *lower* significance value. It is more clear to say "more significant" instead of "higher significance."
- A **detection parameter** is a statistic calculated for a trial signal that (roughly) tells how likely the trial is to be a real phenomenon, rather than a result of random chance. A higher significance means a result is less likely to be random. In (deprecated) Lomb-Scargle, the significance of a frequency tells how likely that frequency is to be a significant component of the signal. Note that "significance" is different than "power."
- A **DFT** (Discrete Fourier Transform) is a precisely defined decomposition of a sequence (uniformly spaced or not) into a set of sinusoidal components. (An "FFT" is just an efficient way to perform a DFT in some limited cases. We have limited use for "FFT" here.) DFTs use uniformly spaced frequencies, but are easily extended to non-uniformly spaced frequencies.
- A **periodogram** is some kind of graph of periodic components of a sample set. There are many methods for producing a periodogram, which produce different results [2]. Therefore, a "periodogram" is less well-defined than a DFT. Usually, the frequencies (or else periods) in a periodogram are uniformly spaced, but the periodogram frequency spacing may be tighter than the DFT.
- **Lomb-Scargle** (now deprecated) is a formula for finding the significance of a *given* sinusoidal frequency in data. It is usually used over a set of frequencies to produce a periodogram.
- A **Lomb-Scargle periodogram** is a graph of detection parameter vs. frequency, where each parameter is computed with the (deprecated) Lomb-Scargle algorithm. The "LS" in periodogram can stand for either "Lomb-Scargle" or "Least Squares", since the Lomb-Scargle algorithm produces the detection parameter for a least squares sinusoidal fit. Note that the LS algorithm produces a *detection parameter*, not power, despite common belief to the contrary.

Be careful to distinguish between uniformly spaced *samples* of data,
and uniformly spaced *frequencies* in the periodogram.

Caution: For orthogonal basis functions (as in a uniformly sampled DFT), the energy and power of every frequency are proportional, and therefore the terms are often interchanged. However, for non-uniform sampling, the "energy" of a frequency component is *not* proportional to its "power." This is the crux of the confusion about the LS periodogram. The LS result is essentially the "energy" of a given sinusoidal frequency in the data, used to help find significant sinusoids in the data. (Explain this more??)

The Discrete Fourier Transform vs. the Periodogram

The single biggest distinction between a DFT and a periodogram is that
the DFT *simultaneously* optimizes *all* the components to form an *exact* transformation.
A periodogram examines each frequency *by itself*, regardless of other frequencies.

The DFT is exact, and invertible, with no loss of information. At times, this can be a plus, but in many cases, this "exactness" results in anomalies. In particular, any set of physical measurements is only a subset of the *exact* representation of the physical phenomenon. In other words, a *sample* is an incomplete set, and so the information contained in it is limited. In addition, all measurements contain some noise. If we put such a noisy sample into a DFT, it gives us frequency components which *exactly* match the given incomplete samples, noise and all. To achieve this exactness, the DFT must sometimes contort the spectrum in unphysical ways. In particular, highly non-uniformly sampled signals often result in large DFT artifacts. By definition, a DFT produces a spectrum of precisely defined, uniformly spaced frequencies. [However, one can easily compute an exact decomposition onto an arbitrary set of frequencies, and furthermore, onto an arbitrary set of basis functions that need not be sinusoidal.]

As scientists, we often would rather see something less mathematically exact, and more physically meaningful. We combine all our knowledge of the system (and science in general) with our limited, noisy data, to reach new conclusions. A periodogram provides a way to look at frequency content of a signal, without some of the unphysical anomalies of an exact DFT. Also, a periodogram can plot results at an arbitrary set of frequencies, not just those defined in a DFT. In fact, periodograms usually choose a larger, and more densely packed, set of frequencies than a DFT produces. However, periodograms suffer from anomalies, as well.

In a DFT, the frequency components don't "overlap," in the sense that none of the "information" of one component appears in any other component. This is true even though the basis sinusoids are not orthogonal over the given sample times. There is no "extra" information in the DFT: e.g., a sample set of $n = 40$ points transforms into exactly 21 cosines and 19 sines, having exactly the same 40 degrees of freedom as the original data set.

In contrast, in a periodogram, the component amplitudes are themselves correlated, and the information from one component also shows up in some of the other components (especially nearby components). Furthermore, especially in small data sets [ref??], the non-orthogonality of the periodogram's sinusoids may cause a single component of the data to produce spikes at multiple widely-spaced frequencies in the periodogram (see Spectral Window Function below). This may mislead the user into believing there are multiple causes, one for each peak. Finally, for any sample size n, we can make a periodogram with any number of frequencies, even far more than n. This again shows that the periodogram contains redundant information.

Despite common belief, a (now deprecated) Lomb-Scargle periodogram is not a periodogram of sinusoidal "power." It is a graph of *detection parameter* vs. frequency, where each parameter is computed by a minimum least-squares residual fit of a single sinusoid at that frequency [3]. For large data sets, or well-randomized sample times, the parameter value approaches the power, so people often "get away with" confusing the two. However, for small data sets, or those where the sample times are clustered around a periodic event (say, nighttime), the significance of a frequency can be very different than its "power" estimate. Note that when the sample times are clustered around a frequency, say 1 cpd (cycle per day), it can affect many frequencies in the sample, especially near harmonics or sub-harmonics (e.g., 2 cpd, 3 cpd, 0.5 cpd, etc.).

When fitting a sinusoid of *given* frequency to data, there are two fit parameters. These may be taken as cosine and sine amplitudes, or equivalently as magnitude and phase of a single sinusoid. The true "power"

at that frequency (considered by itself) is the sum of the squares of the cosine and sine amplitudes, or equivalently, the square of the magnitude.

Practical Considerations

Here are a few possible issues with spectral analysis. Again, it is a highly involved topic, and these issues are only a tiny introduction to it.

Removing trends: Before using spectral analysis, it is common to remove simple trends in the data, such as a constant offset, or straight line trends [ref??]. A straight-line introduces a complicated spectral structure which often obscures the phenomena of interest. Thus, removing it before spectral analysis usually helps. A constant offset introduces spurious frequency detections, especially for bunched samples, as are typical astronomical data. Also, constant offsets may lead to worse round-off error. Furthermore, even though you should never pad your data (see below), padding with zeros when your data has a non-zero average only compounds your error.

Stepwise regression: Sometimes we have in our data a frequency component which is obscuring the phenomenon of interest. We may model (fit) that frequency, and subtract it from the data, in hopes of revealing the interesting data. Note that finding frequencies in our data, and subtracting them, one at a time, is simply the standard statistical method of *stepwise* multiple regression (not simultaneous multiple regression). We are "regressing" one frequency component at a time. Therefore, stepwise frequency subtraction has all the usual pitfalls of stepwise regression. In particular, the *single* biggest component may be completely subsumed by two (or more) smaller components. Therefore, when performing such stepwise frequency modeling, it may help to use the standard method of backward elimination to delete from the model any previously found component that is no longer useful in the presence of newer components.

Computational burden: Many decomposition algorithms rely on some form of orthogonality, e.g., this is the basis (wink) of Discrete Fourier Transforms. Orthogonality allows a basis decomposition to be done by correlation (aka using inner-products). Recall that such a correlation decomposition, including Lomb-Scargle periodograms, requires $O(n^2)$ operations. In contrast, a non-orthogonal decomposition, such as a DFT over non-uniform sample times, solves simultaneous equations requiring $O(n^3)$ operations, so can be *much* slower. For n = 1,000 samples, the non-orthogonal decomposition is about 1,000 times slower, and requires billions of operations. This may be a noticeable burden, even on modern computers (perhaps requiring many minutes).

Smoothed DFT: One surprisingly common approach to making a periodogram (not Lomb-Scargle) is to make a DFT, with its possible anomalies, and then try to disperse those anomalies by "smoothing" the resulting graph of power vs. frequency. I believe smoothing a DFT is like trying to invest wisely after you've lost all your money gambling. It's too late: you can't get back what's already lost. Likely much better is to make some other kind of periodogram in the first place, and don't use a DFT, or use it only as guidance for more appropriate analyses. In particular, with highly non-uniformly spaced samples, the DFT anomalies include large (but unphysical) amplitudes, which are not removed by smoothing. Furthermore, smoothing a DFT of nonuniformly spaced samples requires $O(n^3)$ operations, so it not only likely produces poor results, it does so slowly.

One possible advantage of the "smoothed DFT" approach is that for very large data sets (n > ~10,000), if n is amenable to a Fast Fourier Transform *and* your samples are uniformly spaced, then the DFT can be done in $O(n \log n)$ operations. A typical Lomb-Scargle periodogram requires $O(n^2)$ operations. However, [Press and Rybicki 1989] provide a way to use FFT-like methods to create a Lomb-Scargle periodogram, thus using $O(n \log n)$ operations. While still slower than a true FFT, this makes Lomb-Scargle periodograms of millions of data points feasible.

Bad information: As mentioned earlier, many references (seemingly most references, especially on the web), are wrong in important (but sometimes subtle) ways. E.g., some references actually *recommend* padding your data (almost always a terrible idea, discussed elsewhere in *Funky Mathematical Physics Concepts*). Many references incorrectly describe the differences between uniform and non-uniform sampling, the meaning of FFT, aliasing, and countless other concepts. In particular,

Some references say that sampling a signal is like
setting it to zero everywhere except the sample times. It is not.

This is a common misconception, which is discussed earlier in the section "Sampling."

The (Deprecated) Lomb-Scargle Algorithm

For historical reference, we here describe the now-deprecated Lomb-Scargle (L-S) algorithm; p275 explains how it works. The algorithm is interesting for the concepts it uses, even though in practice, it is much better to use modern, uncertainty-weighted 3-parameter sinusoidal fits.

The algorithm starts with n discrete measurements (samples), s_j, taken at times t_j, j = 0, ... n–1. The algorithm first finds the time offset that makes the cosine and sine orthogonal *over the given sample times*:

$$\exists\, \tau \text{ such that } \quad \sum_{j=0}^{n-1} \cos\omega t_j \sin\omega t_j = 0. \qquad \tau \text{ satisfies } \quad \tan(2\omega\tau) = \frac{\sum_{j=0}^{n-1} \sin 2\omega t_j}{\sum_{j=0}^{n-1} \cos 2\omega t_j}.$$

Note that τ depends on ω; so each ω has its own τ. Also, τ depends on the sample times, but not on the measurements, s_j. You can use a principal valued (2-quadrant) arctangent to find τ.

Next, L-S subtracts out the average signal, giving samples:

$$h_j \equiv s_j - \langle s_j \rangle \qquad \textit{where} \quad \langle s_j \rangle \equiv \text{average of } s_j .$$

Then the Lomb-Scargle normalized periodogram *detection statistic* is, in inner product notation:

$$D(\omega) = \frac{1}{2s^2}\left[\frac{\langle \cos|h\rangle^2}{\langle \cos|\cos\rangle} + \frac{\langle \sin|h\rangle^2}{\langle \sin|\sin\rangle}\right] \qquad \text{[Pres\&Ryb 1989 (3) p277].}$$

$$\textit{where} \quad s^2 \equiv \frac{1}{n-1}\sum_{j=0}^{n-1} h_j{}^2 = \text{sample variance}.$$

We deliberately use the non-standard notation $D(\omega)$, rather than $P(\omega)$, to emphasize that the L-S parameter is a detection statistic, *not* a power (despite widespread belief). In particular, $D(\omega)$ is dimensionless, and independent of measurement scaling, so it can't be a "power." Expanded in more conventional notation, the L-S normalized detection parameter at a given frequency ω is [Pres&Ryb 1989 (3)]:

$$D(\omega) = \frac{1}{2s^2}\left[\frac{\left(\sum_{j=0}^{n-1} h_j \cos\omega\left(t_j-\tau\right)\right)^2}{\sum_{j=0}^{n-1}\cos^2\omega\left(t_j-\tau\right)} + \frac{\left(\sum_{j=0}^{n-1} h_j \sin\omega\left(t_j-\tau\right)\right)^2}{\sum_{j=0}^{n-1}\sin^2\omega\left(t_j-\tau\right)}\right].$$

NB: The L-S algorithm demands *equal uncertainties* on the data, which is one reason it is deprecated. This is exactly the equation for the coefficient of determination one gets from a standard statistical fit which minimizes the residual signal in a least-squares sense (i.e., minimum residual energy) [4]. Such a fit is a simultaneous 2-parameter linear fit (for A and B) to the model:

$$S_{fit}(t) = A\cos\left(\omega\left(t-\tau\right)\right) + B\sin\left(\omega\left(t-\tau\right)\right), \qquad P_{true}(\omega) = A^2 + B^2 .$$

$P_{true}(\omega)$ is the true estimate of the "power" at ω, because it is proportional to the *squared* amplitude of the fitted sinusoid at frequency ω. For large data sets, or well-randomized sample times, $D(\omega)$ approaches

being proportional to $P_{true}(\omega)$ at all frequencies. Therefore, the parameter $D(\omega)$ is often used as a substitute for the spectral power estimate, $P_{true}(\omega)$. As with most hypothesis testing, the presence of a spectral line (frequency) is deemed likely if the line's parameter is substantially less likely than that expected from pure noise. Since both terms in the L-S formula are gaussian random variables (RVs), the Lomb-Scargle expression in brackets for pure gaussian noise is proportional to a $\chi^2_{\nu=2}$ distribution. The factor of 1/2 makes the probability distribution of $D(\omega)$ approach a unit-mean exponential [3], rather than a $\chi^2_{\nu=2}$. However, the normalization by s^2 means that $D(\omega)$ is exactly beta distributed (*not* F distributed, as thought for decades) [A. Schwarzenberg-Czerny, 1997].

Note that s^2 is (close to) the average "energy" (squared value) of the samples (remember that the average value of all the samples has been subtracted off, so the h_j have zero average). The $1/s^2$ in this equation makes the result independent of the signal amplitude, i.e. multiplying all the data by a constant has no effect on the periodogram. Also, for pure noise, $D(\omega)$ is roughly independent of the number of samples, n, since s^2 is independent of n, and the numerators and denominators both scale roughly like n. The numerator summations scale like $\sqrt{n}$, because they are sums of random variables (noise).

In contrast, if a signal is present at frequency ω, $D(\omega)$ grows like n, because then the numerator summation grows like n. Thus, if a signal is present, it becomes easier to detect with a larger sample set, consistent with our intuition.

Bandwidth Correction (aka Bandwidth Penalty)

Determining the significance of a signal detection requires some care, since most algorithms search for any *one* of many possible signals. For example, when searching for periodic signals in noisy data, one often searches many trial frequencies, and a "hit" on any frequency counts as a detection. How do we determine the significance (p value) of such a detection? p is also called the "false alarm" probability.

All the common periodic-signal detection algorithms require bandwidth correction, because if one makes enough attempts, even an unlikely outcome will eventually happen. If one tries many frequencies, the probability that *one* of them exceeds a threshold is much higher than the probability of a *single* given frequency exceeding that threshold. From elementary statistics, if the parameters for all frequencies are independent, the probability that they are *all* not false alarm (FA) is the product of the probabilities that each one is not false alarm. For M independent detection parameters at various frequencies, and a given p-value, then in our gaussian white noise case (i.e., the standard Null Hypothesis of no signal):

$$\Pr(\text{all not FA}) = \Pr(\text{one not FA})^M = (1-p_{1f})^M$$

$$\textit{confidence level} \equiv 1-p = \Pr(\text{all not FA}) = (1-p_{1f})^M .$$

Therefore, to achieve an overall p-value for all frequencies of p, we must choose the p-value for each trial frequency such that [Schwarzenberg-Czerny (1997)]:

$$1-p = (1-p_{1f})^M , \qquad \Rightarrow \qquad p_{1f} = 1-(1-p)^{1/M} .$$

Since p is usually small (<~ .05), we can often use the binomial theorem to approximate:

$$(1-p)^{1/M} \approx 1-p/M , \qquad p_{1f} \approx p/M .$$

For simulations, we may want to estimate M from p and p_{1f}. For example, we choose p_{1f}, measure p, and from that estimate M. Solving the above for M:

$$\ln(1-p) = M \ln(1-p_{1f}), \qquad M = \frac{\ln(1-p)}{\ln(1-p_{1f})} .$$

Larger M is more demanding on your data.
Being conservative on a claim of detection means favoring larger M.

In most period-searching methods (except for the DFT), we are free to search as many frequencies as we like, at as dense a trial frequency spacing as we like. We call our significance parameter $\theta = \theta(f)$, because it is a function of frequency. Intuitively, we expect that two very close frequencies will produce similar θ values, and indeed, such θ-values are correlated (in the precise statistical sense). So our problem reduces to determining M, the number of *independent* frequencies in our arbitrary set of frequencies.

The bottom line is that, for dense trial frequencies, M is approximately the same as if we had equally spaced samples, and therefore a simple DFT [Press 1988]. Such a DFT has independent frequency components. This simple-sounding result, however, requires understanding a few subtleties, especially when the trial frequencies are sparse.

We consider 3 cases, starting with the simplest:

- Uniformly space data points, uniformly spaced frequencies (i.e., a DFT).
- Arbitrarily spaced data points, uniformly spaced frequencies.
- Arbitrarily space data points, arbitrarily spaced frequencies.

Notation:

θ	significance parameter, such as Lomb-Scargle, Phase Dispersion Minimization, etc.
Δf	the independent frequency spacing.
N	number of data samples.
M	number of *independent* θ values over our chosen set of frequencies.
BW	the total range of frequencies tried: $BW \equiv f_{max} - f_{min}$.
T	the total duration of samples: $T \equiv t_{max} - t_{min}$.

Equally spaced samples: If our samples our equally spaced, we have the common case of a Discrete Fourier Transform (DFT). For white (i.e., uncorrelated) noise, each frequency component is independent of all the others. Furthermore, in the relevant notation (where all frequencies are positive), the maximum number of frequencies, and their spacing, is:

$$\text{max \# DFT frequencies} = N/2, \qquad \Delta f = 1/T \ .$$

Note that the maximum number of frequencies depends *only* on the number of data points, N, and not on T. However, we may be looking for frequencies only in some range (Figure 12.3).

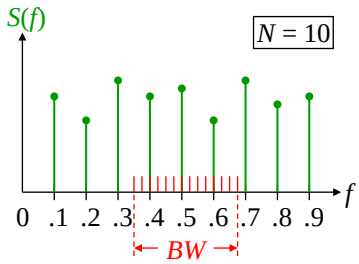


Figure 12.12 Sample frequency spectrum for uniformly spaced discrete time data (here $\Delta f = 0.1$). BW defines a subset of frequencies (here BW = 0.325).

Therefore, for dense trial frequencies ($\Delta f_{trial} < \Delta f$), the number of independent frequencies is approximately:

$$M \approx BW / \Delta f = (BW)T = 0.325/0.1 = 3.25 \rightarrow 4 \ .$$

We round M up to be conservative.

Arbitrarily spaced samples: In astronomy, the data times are rarely uniformly spaced. In such cases, we usually choose our trial frequency spacing, Δf_{trial}, to be dense, i.e., smaller than the independent frequency

spacing: $\Delta f_{trial} < \Delta f \approx 1/T$. Then, per [Press 1988], we use the same equations as above to approximate Δf and M:

$$\Delta f \approx 1/T, \qquad M \approx BW/\Delta f = (BW)T, \qquad (\Delta f_{trial} < \Delta f). \tag{12.6}$$

Note that this is true even if $BW >$ Nyquist frequency, which is perfectly valid for nonuniformly spaced time samples [Press 1988].

In the unusual case that our trial frequency spacing is large, $\Delta f_{trial} > \Delta f$, then we approximate that each frequency is independent:

$$M \approx \#\text{ trial frequencies}, \qquad (\Delta f_{trial} > \Delta f). \tag{12.7}$$

(In reality, even if θ values separated by Δf are truly independent, some θ values separated by more than Δf will be somewhat correlated. However, the correlation coefficient "envelope" mostly decreases with increasing frequency spacing. Nonetheless, these correlations imply that there are parasitic cases where this approximation, eq. (12.7), fails.)

Arbitrarily spaced trial frequencies: One common situation leading to nonuniformly spaced trial frequencies is that of *uniformly* space trial *periods*. If the ratio of highest to lowest period is large (say, > 2), then the frequency spacing is seriously nonuniform.

We may think of Δf as approximately the difference in frequency required to make the θ values independent. (In reality, even if θ values separated by Δf are truly independent, *some* θ values separated by more than Δf will be somewhat correlated. However, the correlation coefficient "envelope" decreases with increasing frequency spacing.) In such a case, we may break up the trial frequencies into (1) regions where $\Delta f_{trial} < \Delta f$, and (2) regions where $\Delta f_{trial} > \Delta f$ (Figure 12.13).

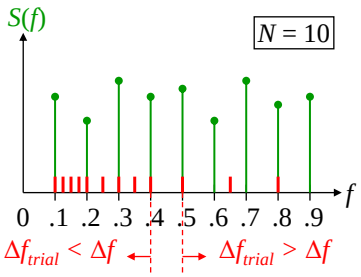


Figure 12.13 Nonuniformly spaced trial frequencies.

The region where $\Delta f_{trial} < \Delta f$ behaves as before, as does the region where $\Delta f_{trial} > \Delta f$. In the example of Figure 12.13, we have:

$$\Delta f = 0.1, \qquad M \approx [(0.5-0.1)/0.1] + 2 = 6.$$

Summary

Bandwidth correction requires estimating the number of *independent* frequencies. For uniformly spaced, dense trial frequencies (and arbitrarily spaced time samples), we approximate the number of independent frequencies, M, with eq. (12.6). We may think loosely of Δf as the difference in frequency required for θ to become independent of its predecessor. Therefore, for nonuniformly spaced trial frequencies, we must consider two types of region: (1) where the trial frequency spacing $\Delta f_{trial} < \Delta f$, we use eq. (12.6); (2) where the trial frequency spacing $\Delta f_{trial} > \Delta f$, we approximate M as the number of trial frequencies, eq. (12.7).

References

[Pres&Ryb 1989] Press, William H. and George B. Rybicki, *Fast Algorithm for Spectral Analysis of Unevenly Sampled Data*, Astrophysics Journal, 338:277-280, 1989 March 1.

[2] http://www.ltrr.arizona.edu/~dmeko/notes_6.pdf , retrieved 1/22/2012.

[3] Press, William H. and Saul A. Teukolsky, *Search Algorithm for Weak Periodic Signals in Unevenly Spaced Data*, Computers in Physics, Nov/Dec 1988, p77.

[4] Scargle, Jeffry, *Studies in Astronomical Time Series Analysis. II. Statistical Aspects of Spectral Analysis of Unevenly Spaced Data*, Astrophysical Journal, 263:835-853, 12/15/1982.

[Lom] Lomb, N. R., *Least Squares Frequency Analysis of Unequally Spaced Data*, Astrophysics and Space Science 39 (1976) 447-462.

[Schw 1997] Schwarzenberg-Czerny, A., *The Correct Probability Distribution for the Phase Dispersion Minimization Periodogram*, The Astrophysical Journal, 489:941-945, 1997 November 10.

Analytic Signals and Hilbert Transforms

Given some real-valued signal, $s(t)$, it is often convenient to write it in "phasor form", i.e. its complex-valued "analytic signal". Such uses arise in diverse signal processing applications from communication systems to neuroscience. We describe here the meaning of "analytic signals," and some practical computation considerations. This section relies heavily on phasor concepts, which you can learn from *Funky Electromagnetic Concepts*. We proceed along these lines:

- Mathematical definitions and review.
- The meaning of the analytic signal, $A(t)$.
- Instantaneous values.
- Finding $A(t)$ from the signal $s(t)$, theoretically.
- The special case of zero reference frequency, $\omega_0 = 0$; Hilbert Transform.
- A simple and reliable numerical computation of $A(t)$ without Hilbert Transforms.

Definitions, conventions, and review: There are many different conventions in the literature for normalization and sign of the Fourier Transform (FT). We define the Fourier Transform such that our basis functions are $e^{i\omega t}$, and our original (possibly complex) signal $z(t)$ is composed from them; this fully defines all the normalization and sign conventions:

$$\text{For } z(t) \text{ complex:} \qquad z(t) \equiv \int_{-\infty}^{\infty} Z(\omega) e^{+i\omega t}\, d\omega \qquad \Rightarrow \qquad Z(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} z(t)\, e^{-i\omega t}\, dt$$

$$\textit{where} \quad Z(\omega) \equiv \mathcal{F}\{z(t)\} \text{ is the Fourier Transform of } z(t).$$

Note that we can think of the FT as a phasor-valued function of frequency, and that we use the positive time dependence $e^{+i\omega t}$.

For real-valued signals we use $s(t)$ instead of $z(t)$. For real $s(t)$, the FT is **conjugate symmetric**:

$$S(-\omega) = S^*(\omega) \quad \text{for } s(t) \text{ real}.$$

This conjugate symmetry for real signals allows us to use a 1-sided FT, where we consider only positive frequencies, so that:

$$s(t) = 2\,\mathrm{Re}\left\{\int_0^{\infty} S(\omega) e^{i\omega t}\, d\omega\right\}, \quad \text{which is equivalent to} \quad s(t) = \int_{-\infty}^{\infty} S(\omega) e^{i\omega t}\, d\omega, \quad s(t) \text{ real}.$$

Note that a *complex* signal with no negative frequencies is very different from a *real* signal which we choose to write as a 1-sided FT. We rely on this distinction in the following discussion.

Analytic signals: Given a real-valued signal, $s(t)$, its phasor form is:

$$s(t) = \mathrm{Re}\left\{A(t)e^{i\omega_0 t}\right\} = |A(t)|\cos\left(\omega_0 t + \phi(t)\right)$$

$$\textit{where} \quad A(t) \equiv |A(t)|e^{i\phi(t)} \text{ is a (complex) phasor function of time} \qquad (12.8)$$

$$\omega_0 \equiv \text{somewhat arbitrary reference frequency}.$$

Recall that as a phasor, $A(t)$ is complex. The phasor form of $s(t)$ may be convenient when $s(t)$ is **band-limited** (exists only in a well-defined range of frequencies, Figure 12.14 left), or where we are only concerned with the components of $s(t)$ in some well-defined range of frequencies. Figure 12.14 shows two 1-sided Fourier Transform (FT) examples of $S(\omega)$, the FT of a hypothetical (real) signal $s(t)$.

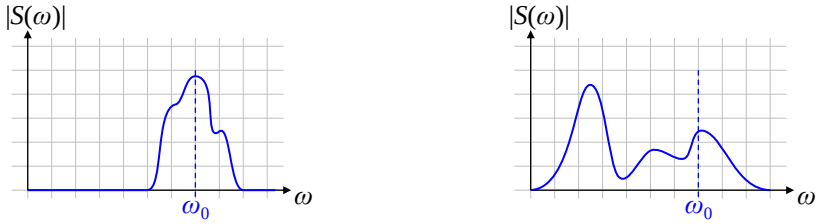


Figure 12.14 Example 1-sided FTs of a real signal $s(t)$: (Left) band-limited. (Right) Wideband. The ω axis points only to the right, because we need consider only positive frequencies for a 1-sided FT.

In communication systems, ω_0 is the carrier frequency. Note that even in the band-limited case, ω_0 may be different than the band center frequency. [For example, in vestigial sideband modulation (VSB), ω_0 is close to one edge of the signal band.] Keep in mind throughout this discussion that ω_0 is often chosen to be zero, i.e. the spectrum of $s(t)$ is kept "in place".

We start by considering the band-limited case, because it is somewhat simpler. From Figure 12.14 (left), we see that our signal $s(t)$ is not too different from a pure sinusoid at a reference frequency ω_0, near the middle of the band. $s(t)$ and $\cos(\omega_0 t)$ might look like Figure 12.15, left. $s(t)$ is a modulation of the pure sinusoid, varying somewhat (i.e. perturbing) the amplitude and phase at each instant in time. We define these variations as the complex function $A(t)$. When a signal $s(t)$ is real, and $A(t)$ satisfies eq. (12.8), $A(t)$ is called the **analytic signal** for $s(t)$.

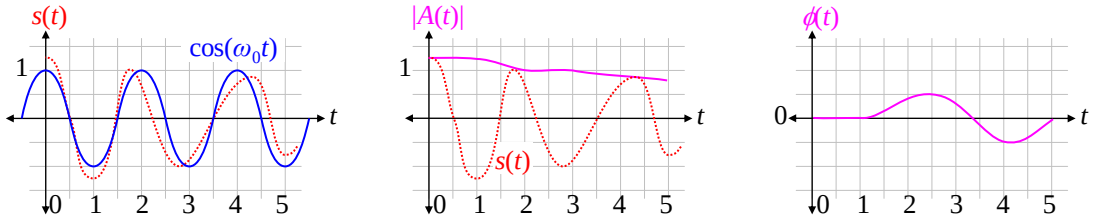


Figure 12.15 (Left) $s(t)$ (dotted), and the reference sinusoid. (Middle) Magnitude of the analytic signal $|A(t)|$. (Right) Phase of the analytic signal.

We can visualize $A(t)$ by considering Figure 12.15, left. At $t = 0$, $s(t)$ is a little bigger than 1, but it is in-phase with the reference cosine; this is reflected in the amplitude $|A(0)|$ being slightly greater than 1, and the phase $\phi(0) = 0$. At $t = 1$, the amplitude remains > 1, and ϕ is still 0. At $t = 2$, the amplitude has dropped to 1, and the phase $\phi(2)$ is now positive (early, or leading). This continues through $t = 3$. At $t = 4$, the amplitude drops further to $|A(4)| < 1$, and the phase is now negative (late, or lagging), i.e. $\phi(4) < 0$. At $t = 5$, the amplitude remains < 1, while the phase has returned to zero: $\phi(5) = 0$. Figure 12.15, middle and right, are plots of these amplitudes and phases.

Instantaneous values: When a signal has a clear oscillatory behavior, one can meaningfully define instantaneous values of phase, frequency, and amplitude. Note that the frequency of a sinusoid (in rad/s) is the rate of change of the phase (in rad) with time. A general signal $s(t)$, has a varying phase $\phi(t)$, aka an **instantaneous phase**. Therefore, we can define an **instantaneous frequency**, as well:

$$phase = \omega_0 t + \phi(t) \qquad \Rightarrow \qquad \omega(t) = \frac{d(phase)}{dt} = \omega_0 + \frac{d\phi}{dt} .$$

Such an instantaneous frequency is more meaningful when $|A(t)|$ is fairly constant over one or more periods. For example, in FM radio (frequency modulation), $|A(t)|$ is constant for all time, and *all* of the information is encoded in the instantaneous frequency.

Finally, we similarly define the **instantaneous amplitude** of a signal $s(t)$ as $|A(t)|$. This is more meaningful when $|A(t)|$ is fairly constant over one or more cycles of oscillation. The instantaneous amplitude is the "envelope" which bounds the oscillations of $s(t)$ (Figure 12.15, middle). By construction, $|s(t)| < |A(t)|$ everywhere.

Finding A(t) from s(t): Given an arbitrary $s(t)$, how do we find its (complex) analytic signal, $A(t)$? First, we see that the defining eq. (12.8), $s(t) = \text{Re}\left\{ A(t) e^{i\omega_0 t} \right\}$, is underdetermined, since $A(t)$ has two real components, but is constrained by only the one equation. Therefore, if $A(t)$ is to be unique, we must further constrain it.

As a simple starting point, suppose $s(t)$ is a pure cosine (we will generalize shortly). Then:

$$s(t) = \cos\omega_0 t = \text{Re}\left\{ 1\, e^{i\omega_0 t} \right\} \qquad \textit{where} \quad A(t) = 1 .$$

If instead, $s(t)$ has a phase offset θ, then:

$$s(t) = \cos\left(\omega_0 t + \theta\right) = \text{Re}\left\{ e^{i\theta} e^{i\omega_0 t} \right\} \quad \textit{where} \quad A(t) = e^{i\theta} = \cos\theta + i\sin\theta .$$

(Note that $\theta = 0$ reproduces the pure-cosine example above.) Thus, in the case of a pure sinusoid, $A \equiv A(t)$ is the (constant) phasor for the sinusoid $s(t)$, and the imaginary part of A is the same sinusoid delayed by ¼ cycle (90°):

$$\text{Re}\{A\} = \cos\theta, \qquad \text{Im}\{A\} = \cos\left(\theta - \pi/2\right) .$$

In Fourier space, the real and imaginary parts of A are simply related. Recall that delaying a sinusoid by ¼ cycle multiplies its Fourier component by $-i$ (for $\omega > 0$). Therefore:

$$\mathcal{F}\left\{ \text{Im}\left(A(t) \right) \right\} = -i\mathcal{F}\left\{ \text{Re}\left(A(t) \right) \right\}, \quad \text{1-sided FT, } \omega > 0 .$$

We now generalize our pure sinusoid example to an arbitrary signal, which can be thought of as a linear combination sinusoids. The above relation is linear, so it holds for any linear combination of sinusoids, i.e. it holds for any real signal $s(t)$. This means that, by construction, the imaginary part of $A(t)$ has exactly the same *magnitude* spectrum as the real part of $A(t)$. Also, the imaginary part has a phase spectrum which is everywhere ¼ cycle delayed from the phase spectrum of the real part. This is the relationship that uniquely defines the analytic signal $A(t)$ that corresponds to a given real signal $s(t)$ and a given reference frequency ω_0. From this relation, we can solve for $\text{Im}\{A(t)\}$ explicitly as a functional of $\text{Re}\{A(t)\}$:

$$\text{Im}\{A(t)\} = \mathcal{F}^{-1}\left\{ -i\mathcal{F}\left\{ \text{Re}\left(A(t) \right) \right\} \right\}, \qquad \text{1-sided FT, } \omega > 0 . \tag{12.9}$$

This relation defines the **Hilbert Transform** (HT) from $\text{Re}\{A(t)\}$ to $\text{Im}\{A(t)\}$.

> The Hilbert Transform of $s(t)$ is a function $H(t)$ that has all the Fourier components of $s(t)$, with the same amplitude but delayed in phase by ¼ cycle (90°).

Note that the Hilbert transform takes a function of time into another function of time (in contrast to the Fourier Transform that takes a function of time into a function of frequency). Since the FT is linear, eq. (12.9) shows that the HT is also linear. The Hilbert Transform can be shown to be given by the time-domain form:

$$\mathcal{H}\{s(t)\} \equiv H(t) = \frac{1}{\pi} \text{PV} \int_{-\infty}^{\infty} dt' \frac{s(t)}{t - t'} \qquad \textit{where} \quad \text{PV} \equiv \text{Principal Value} .$$

(The integrand blows up at $t' = t$, which is why we need the Principal Value to make the integral well-defined.) We now easily show that the inverse Hilbert transform is the negative of the forward transform:

$$\mathcal{H}^{-1}\{H(t)\} \equiv s(t) = -\frac{1}{\pi}\text{PV}\int_{-\infty}^{\infty} dt' \frac{H(t)}{t-t'} \qquad \textit{where} \quad \text{PV} \equiv \text{Principal Value}\,.$$

We see this because the Hilbert Transform shifts the phase of every sinusoid by 90°. Therefore, two Hilbert transforms shifts the phase by 180°, equivalent to negating every sinusoid, which is equivalent to negating the original signal. Putting in a minus sign then restores the original signal.

Equivalently, the HT multiplies each Fourier component ($\omega > 0$) by $-i$. Then $\mathcal{H}\{\mathcal{H}(\,)\}$ multiplies each component by $(-i)^2 = -1$. Thus, $\mathcal{H}\{\mathcal{H}[\,s(t)\,]\} = -s(t)$, and therefore $\mathcal{H}^{-1} = -\mathcal{H}$.

Analytic signal relative to $\omega_0 = 0$: Some analysts prefer not to remove a reference frequency $e^{i\omega_0 t}$ from the signal, and instead include all of the phase in $A(t)$; this is equivalent to choosing $\omega_0 = 0$:

$$s(t) = \text{Re}\{A(t)\} = |A(t)|\cos(\phi(t))\,.$$

Since $s(t) = \text{Re}\{A(t)\}$ is given, we can now find $\text{Im}\{A(t)\}$ explicitly from (12.9):

$$\text{Im}\{A(t)\} = \mathcal{F}^{-1}\{-i\mathcal{F}\{s(t)\}\} \equiv \mathcal{H}\{s(t)\} \qquad \text{1-sided FT, } \omega > 0\,.$$

In other words:

For $\omega_0 = 0$, $A(t)$ is just the complex phasor factors for $s(t)$, without taking any real part.

If $s(t)$ is dominated by a single frequency ω, then $\phi(t)$ contains a fairly steady phase ramp that is close to $\phi(t) \approx \omega t$ (Figure 12.16). We can use the phase function $\phi(t)$ to estimate the frequency ω by simply taking the average phase *rate* over our sample interval:

$$\omega_{est} = \frac{\phi(t_{end}) - \phi(0)}{t_{end}}\,.$$

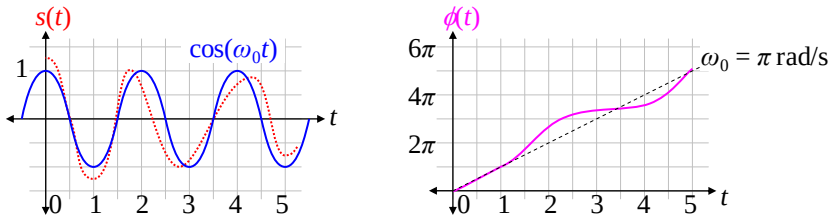


Figure 12.16 Phase ramp of a perturbed sinusoid, and the estimate of ω_0.

Efficient numerical computation of A(t): The traditional way to find $A(t)$ is to use a discrete Hilbert Transform to evaluate the defining integral. (This is a standard function in scientific software packages.) The discrete Hilbert Transform (DHT) is often implemented by taking a DFT, manipulating it, and then inverse FT back to the time domain. This can be seen by recasting our above (1-sided DFT) description of the Hilbert Transform (HT) into a 2-sided DFT form.

Recall that in the 1-sided DFT for a real signal $s(t)$, the frequencies are always positive, $\omega > 0$, and $S(\omega)$ is just a phasor-valued function of frequency. To recover the real signal from phasors, we must take a real-part, Re{ }. In the 2-sided DFT, we instead arrive at the real part by adding the complex conjugate of all the phasor factors:

$$s(t) = 2\int_0^{\infty} d\omega\, \text{Re}\{S(\omega)e^{+i\omega t}\} \qquad \rightarrow \qquad s(t) = \int_0^{\infty} d\omega \left[S(\omega)e^{+i\omega t} + S^*(\omega)e^{-i\omega t}\right].$$

However, to achieve a 2-sided FT, we rewrite the second term as a negative frequency. Then the integral spans both positive and negative frequencies:

$$s(t) = \int_{-\infty}^{\infty} S(\omega) e^{i\omega t}\, d\omega, \qquad where \quad S(-\omega) = S^*(\omega)\,.$$

For a complex signal, $z(t)$, only a 2-sided FT exists (a 1-sided FT is not generally possible). Then there is no symmetry or relation between positive and negative frequencies.

We now describe a simple, efficient, and stable, purely time-domain algorithm for finding $A(t)$ from a band-limited $s(t)$. This algorithm is sometimes more efficient than the DFT-based approach. It is especially useful when the data must be downsampled (converted to a lower sampling rate by keeping only every n^{th} sample, called **decimating**). Even though $s(t)$ is real, the algorithm uses complex arithmetic throughout.

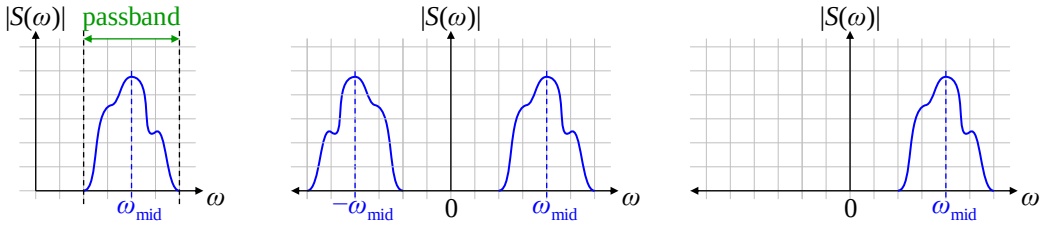


Figure 12.17 (Left) 1-sided FT of $s(t)$, and (middle) its 2-sided equivalent. (Right) 2-sided FT of $A(t)$.

Figure 12.17 shows a 1-sided FT for a real $s(t)$, along with its 2-sided FT equivalent, and the 2-sided FT of the desired complex $A(t)$. We define ω_{mid} as the midpoint of the signal band (this is *not* ω_0, which we take to be zero for illustration). The question is: how do we efficiently go from the middle diagram to the right diagram? In other words, how do we keep just the positive frequency half of the 2-sided spectrum? Figure 12.18 illustrates the simple steps to achieve this:

- Rotate the spectrum down by ω_{mid} (downconvert).
- Low-pass filter (LPF) around the downconverted signal band.
- (Optional) Decimate (downsample).
- Rotate the spectrum back up by ω_{mid} (upconvert).

This results in a complex function of time whose 2-sided spectrum has only positive frequencies; in other words, it is the analytic signal $A(t)$. The numerical error in this algorithm comes from the leakage through the LPF of the undesired part of the spectrum, which you can make arbitrarily small.

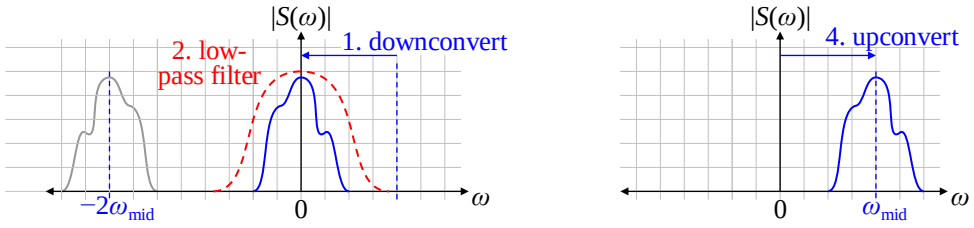


Figure 12.18 (Left) To find $A(t)$: 1. downconvert; 2. low-pass filter; (Right) 4. upconvert.

Step 1: Downconvert: Numerically, we downconvert in the time domain by multiplying by $\exp(-i\omega_{mid}t)$. This simply subtracts ω_{mid} from the frequency of each component in the spectrum:

$$\text{For every } \omega\text{:} \qquad S(\omega)e^{i\omega t}e^{-i\omega_{mid}\, t} = S(\omega)e^{i(\omega-\omega_{mid})t}\,.$$

Note that both positive and negative frequencies are shifted to the left (more negative) in frequency. In the time domain, we construct the complex downconverted signal for each sample time t_j as:

$$z_{down}(t_j) = s(t_j)\exp\left(-i\omega_{mid}t_j\right) = s(t_j)\cos\left(\omega_{mid}t_j\right) - i\sin\left(\omega_{mid}t_j\right).$$

Step 2: Low-pass filter: Low pass filters are easily implemented as Finite Impulse Response (FIR) filters, with symmetric filter coefficients [Ham chap. 6, 7]. In the time domain:

$$A_{down}(t_j) = 2\sum_{k=-m}^{m} c_k z_{down}(t_{j+k}) \qquad \textit{where} \quad 2m+1 \equiv \text{the number of filter coefficients}$$

$$c_k \equiv \text{filter coefficients}$$

The leading factor of 2 is to restore the full amplitude to $A(t)$ after filtering out half the frequency components.

Step 3: (Optional) Decimate: We now have a (complex) low-pass signal whose full (2-sided) bandwidth is just that of our desired signal band. If desired, we can now downsample (decimate), by simply keeping every n^{th} sample. In other words, our low-pass filter acts as both a pass-band filter for the desired signal, and an anti-aliasing filter for downsampling. Two for the price of one.

Step 4: Upconvert: We now restore our complex analytic signal to a reference frequency of $\omega_0 = 0$ by putting the spectrum back where it came from. The key distinction is that after upconverting, there will be no components of negative frequency because we filtered them out in Step 2. This provides our desired complex analytic signal:

$$A(t_j) = A_{down}(t_j)\exp\left(i\omega_{mid}t_j\right).$$

Note that the multiplications above are full complex multiplies, because both A_{down} and the exponential factor are complex. Also, if we want some nonzero ω_0, we would simply upconvert by $(\omega_{mid} - \omega_0)$ instead of upconverting by ω_{mid}.

Summary

The analytic signal for a real function $s(t)$ is $A(t)$, and is the complex phasor-form of $s(t)$ such that:

$$s(t) = \text{Re}\left\{A(t)\exp\left(i\omega_0 t\right)\right\} \qquad \textit{where} \quad \omega_0 \equiv \text{reference frequency}\ .$$

ω_0 is often chosen to be zero, so that $s(t) = \text{Re}\{A(t)\}$. This definition does *not* uniquely define $A(t)$, since $A(t)$ has real and imaginary components, but is constrained by only one equation. The Hilbert Transform of a real function $s(t)$ is $H(t)$, and comprises all the Fourier components of $s(t)$ phase-delayed by $\pi/4$ radians (90°). We uniquely define $A(t)$ by saying that its imaginary part is the Hilbert Transform of its real part. This gives the imaginary part the exact same magnitude spectrum as the real part, but a shifted phase spectrum.

Analytic signals allow defining instantaneous values of frequency, phase, and amplitude for almost-sinusoidal signals. Instantaneous values are useful in many applications, including communication and neuron behavior.

[Ham] Hamming, R. W., *Digital Filters*, 3ed, Dover Publications (1998), ISBN 0-486-65088-X.

13 Period Finding

A common experimental task is to examine a data sequence for a periodic component, or several of them. For example, in astronomy, periodic signals in a star's brightness point to physics in the star, or to planets orbiting the star. In model construction, periodicities in a model's residual point to improvements to the model.

Often, such a "periodicity" is obscured by noise and other components that are either aperiodic, or of vastly different period than the one(s) of interest. Usually, the experimenter does not know the frequency of the sought-after component, but has an idea of some range of possible frequencies, which may span several orders of magnitude. The experimenters job is then at least two-fold:

- Determine a statistically valid likelihood that a periodic component exists in the data; and
- If it exists, estimate the frequency of the component.

Other goals may include estimating the shape and the amplitude of such a component (or components). In this chapter, we consider some algorithms for making such determinations. We focus here on a single periodic component, with a few suggestions for multiple components at the end.

We discuss and compare periodogram algorithms: Discrete Fourier Transform (DFT), Lomb-Scargle (deprecated, but unfortunately still popular), Phase Dispersion Minimization (PDM), Epoch Folding (ToBeSupplied), what I call DCS (DC, cosine, sine), and briefly note why Discrete Fourier Transforms are often ineffective with non-uniformly sampled data (such as most astronomical data).

The literature is rife with conflicting definitions, and even misuse of terms.

Precise definitions are essential to understanding the data,
which point to physical mechanisms underlying them.

Therefore, we strictly follow these definitions:

alias	an artifact of uniform sampling. An alias is specifically *not* a sideband, though it can be considered a limiting case of a correlate.
amplitude	is the usual real amplitude of a basis function: for a sinusoid, it is simply the real amplitude of the sinusoid.
correlate	a sinusoidal frequency component that is correlated with another given sinusoidal frequency component. The spectral window function describes how different sinusoidal components are correlated for a given set of sample times (more later). The appearance of a true sinusoid at other periodogram frequencies is called **spectral leakage** [Scargle 1982 p837].
detection parameter	a *statistic* from the data that can be used to estimate the significance (or confidence level) of a potential detection of a signal.
DFT	Discrete Fourier Transform: a unique and invertible transform from the (typically) time-domain to the frequency domain: a DFT gives the frequency components comprising a signal.
FFT	Fast Fourier Transform: an algorithm for computing some DFTs. Not all DFTs benefit from an FFT.
periodogram	a set of detection statistics as a function of frequency. There are several ways to make a periodogram, with some choice in which detection statistic to use. Because of these choices, and unlike a DFT, a periodogram is neither unique nor invertible.
power or energy	is proportional to $(\text{amplitude})^2$, with normalization varying throughout the literature.
probability of false alarm (PFA)	the probability that pure noise will create an artifact looking like a signal.
sideband	a frequency component resulting from modulating (multiplying) a carrier sinusoid by a modulating function of time, such as a slowly changing envelope function.

spectral window function For sinusoidal detections, a function $W(\Delta\omega)$ that approximates the correlation of an amplitude at frequency ω to the amplitude at frequency $\omega + \Delta\omega$. $W(\ )$ depends only on the sample times, and not the measured values.

Sinusoids, Fourier, Nyquist, and All That

Many people are familiar with the basics of uniform sampling, DFTs, the Nyquist limit, and frequency resolution. This is described in more detail elsewhere in this work (*Funky Mathematical Physics Concepts*). However, most period finding is done on non-uniformly sampled data, is *not* based on a DFT, and the limitations of frequency resolution and Nyquist limit do not apply.

For example, with a DFT, the fundamental frequency is $f_1 = 1/T$, and all frequencies in the transform are multiples of that. Thus we say the "frequency resolution" is $1/T$. But with a periodogram, the frequency spacing is arbitrary, and the resolution is limited only by the signal-to-noise ratio (SNR). Frequencies are often determined with uncertainty $<< 1/T$.

Also, with uniform sampling, there is a **Nyquist limit** on the maximum frequency that can be detected: $f_{max} = f_{samp}/2$. Higher frequencies are "aliased" down to lower frequencies by sampling, and the higher frequencies are irretrievably lost. With non-uniform sampling, there is no sharp cutoff on the highest frequency, and f_{max} is on the order of the inverse of the minimum sample-time-intervals [Press and Rybicki 1989 p277].

Overview of Period Finding Algorithms

All the period-finding algorithms here rely on the statistical method of choosing a **probability of false alarm** α (say, 5%), and finding a critical value of the detection parameter such that pure noise will exceed the critical value only a fraction α of the time. Therefore, if a peak exceeds the critical value, we are confident at the $(1 - \alpha)$ level that the peak is a signal, and *not* a fluke of pure noise. Finding the critical value is complicated by the presence of many trial frequencies, and correlations between those trials. The presence of background signals, other than pure noise, complicates the detection probabilities. Ultimately, shuffle simulations can overcome these complications to provide a valid critical detection value for almost any background signal.

It is crucial to distinguish a period-finding detection parameter from a component "amplitude" or "power." A **detection parameter** is a *statistic* from the data that can be used to estimate the significance (or confidence level) of potential detection of a signal. All detection parameters of all algorithms are necessarily dimensionless and invariant to constant scale factors of the data. In contrast, a detected signal **amplitude** is the usual (real) amplitude of a basis function: for a sinusoid, it is simply the amplitude of the sinusoid. There is much confusion in the literature on this distinction, and many references use the term "power" to mean "detection parameter."

We can make a periodogram graph of the detection parameter vs. frequency, or of amplitude vs. frequency.

References

[Leahy 1983a] D. A. Leahy, et. al., "On Searches for Pulsed Emission with Application to Four Globular Cluster X-Ray Sources: NGC 1851, 6441, 6624, and 6712," Astrophysical Journal, 266:160-170, 1983 March 1.

[Scargle 1982] Jeffrey D. Scargle, "Studies in Astronomical Time Series Analysis. II. Statistical Aspects of Spectral Analysis of Unevenly Spaced Data," Astrophysical Journal, 263:835-853, 1982 December 15.

Phase Dispersion Minimization

Some periodic signals are far from sinusoidal. For example, a planet transiting across a star makes a small, nearly rectangular dip in the light curve (intensity vs. time) at regular intervals (Figure 13.1a). The units are arbitrary. A more realistic light curve would have more noise, obscuring the periodicity (Figure

13.1b). Zooming in on the data does not help much (Figure 13.1c). Phase Dispersion Minimization is an algorithm to identify arbitrary wave shape periodic functions (not just sinusoids), even when such periodicities are corrupted by noise and other signals. PDM creates a **periodogram**, a graph of a detection parameter vs. frequency. Peaks in the periodogram indicate periodic components of the signal (Figure 13.1d). The peak at f = .01 clearly indicates the nearly-invisible periodic component (the time units are arbitrary). (The subharmonic at f = .005, and the harmonics at f = .02, .03, and .04 are also clearly visible. We discuss sub/harmonics later.)

By its nature (as with epoch folding), PDM also picks up subharmonics of a periodic signal, though subharmonics weaken with their order. PDM is invariant to constant scale factors in either time or intensity, and also to DC (constant) offsets in time or intensity.

A straightforward extension of the basic PDM allows for uncertainties to be included in measurements. (To our knowledge, including uncertainties is not addressed in the literature.)

For illustration, we take the independent variable to be time, and the dependent variable to be intensity, but of course, any dependent variables can be used. The measurements are triples (t_i, y_i, u_i), where t_i are the independent variables, y_i are the measured quantities, and u_i are the uncertainties for each measurement. The sample times are arbitrary, and often nonuniform, however they should be dense enough to have several measurements in each period of the sought-after signal.

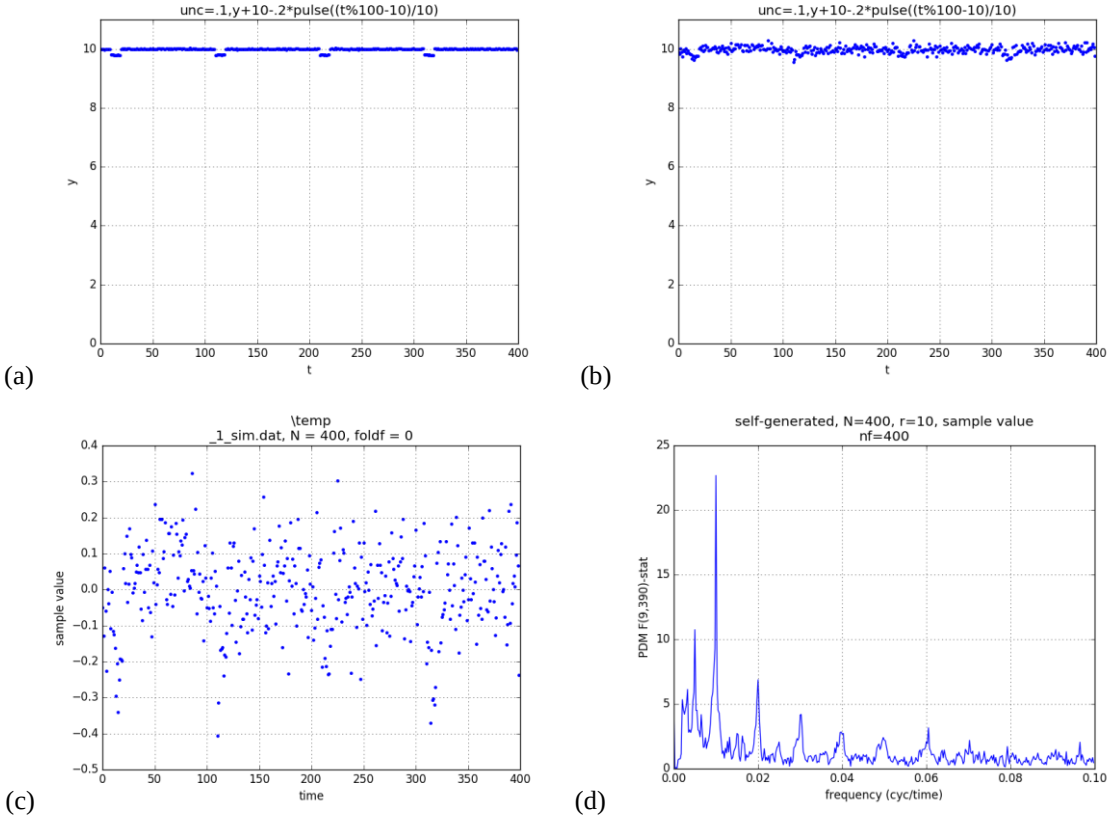


Figure 13.1 (a) Simulated nonsinusoidal periodic light curve. (b) Same as (a) but with more noise. (c) Zoom in on data. (d) PDM periodogram of (b) or (c).

PDM Algorithm

A PDM periodogram computes a detection parameter for each trial frequency. The trial frequencies are arbitrary, but are usually uniformly spaced over an interval of interest. The frequency spacing is also arbitrary, but periodograms typically have 100-400 frequencies. Sometimes, periodograms are graphed as a

function of period, instead of frequency. However, we think frequency plots provide better information, since physical phenomena such as signal modulation have readily apparent frequency structure.

For a given trial frequency, the data are folded at the trial frequency, and binned into r bins (typically 10-20). "Folding" means replacing each data sample time t_i with t_i mod T ($T \equiv 1/f$). The model value for each bin is the (weighted) average of the points in the bin. Figure 13.2a shows a significant trial frequency that reveals a periodicity, and (b) shows an insignificant one. Figure 13.3 shows the folded and fitted result for the data of Figure 13.1

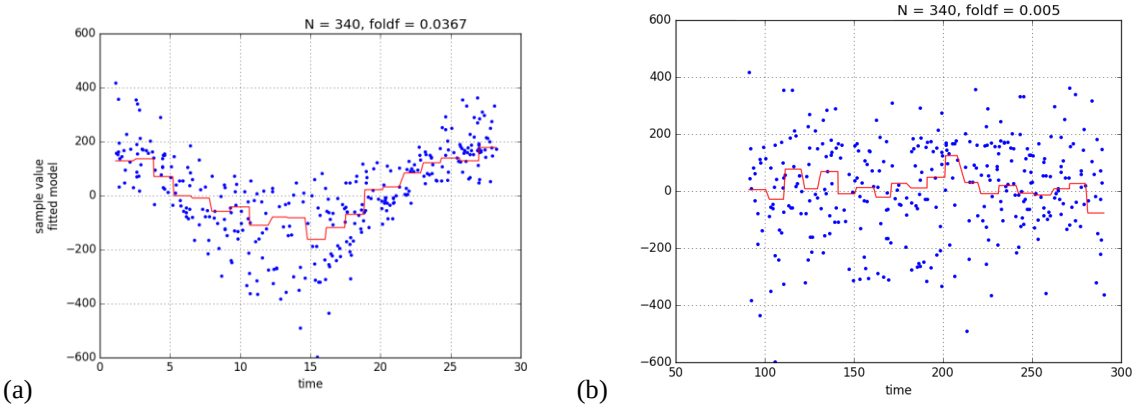


Figure 13.2 The red line is the best PDM model fit to the folded data. (a) PDM data points and a good fit, r = 20. (b) A bad fit shows wide dispersion in each bin, and a fairly flat "fit".

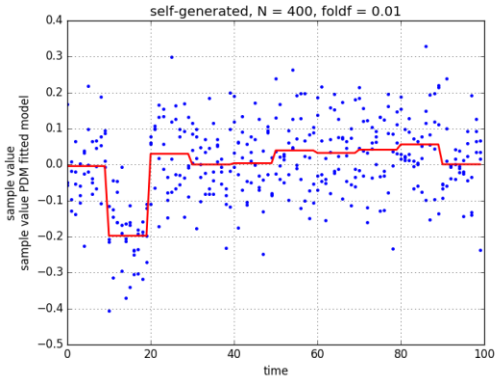


Figure 13.3 A fit to the folded data clearly shows the periodic component of the signal.

The PDM algorithm is exactly equivalent to a linear fit of an orthogonal set of r periodic rectangular pulses to the data (Figure 13.4). Therefore, the results are amenable to standard analysis of variance (ANOVA). Across the PDM literature, there are multiple detection parameters in use, but we recommend the standard ANOVA F-statistic (detailed below), which is dimensionless and well-understood. Critical F values depend on gaussian residuals, which most data do *not* have. However shuffle simulations allow for a wide variety of detection parameters, and also accommodate non-gaussian residuals. In practice, the F statistic is very effective on arbitrary residuals when its critical values are determined from shuffle simulations, rather than gaussian theory. [Somebody discusses shuffle simulations ??]. For many analyses, the periodogram peaks are so obvious that quantitative critical values are not needed.

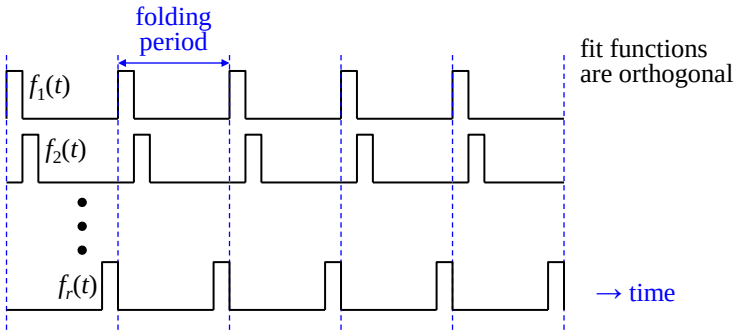


Figure 13.4 Basis functions for the equivalent multiple linear regression fit.

The regression model in PDM comprises the bin averages of the folded/binned data. One early detection parameter is the pooled variance of all the bins, s^2 [Stell 1978 eq. 2]; the best fit (most likely period) is that with the smallest s^2, meaning measurements in the same bin (similar "phase") are similar to each other.

ANOVA F-statistic: Instead of s^2, we recommend the standard ANOVA F-statistic as the detection parameter. The general theory of ANOVA proves the sum-of-squares identity: the total variation from the average (*SST*) equals the variation of the model (*SSA*), plus the variation in unmodeled residuals (*SSE*) [W&M8 Ch 13]. For the (deprecated) unweighted algorithm (no uncertainties):

$$\underbrace{SST}_{\text{total}} = \underbrace{SSA}_{\text{model}} + \underbrace{SSE}_{\text{residual}} \qquad \textit{where}$$

$$\text{total:} \qquad SST \equiv \sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2$$

$$\text{model:} \qquad SSA \equiv \sum_{b=1}^{r} n_b \left(\bar{y}_b - \bar{y}\right)^2, \qquad n_b \equiv \#\text{ points in bin } b \tag{13.1}$$

$$\text{residual:} \qquad SSE \equiv \sum_{b=1}^{r} \sum_{j=1}^{n_b} \left(y_{bj} - \bar{y}_b\right)^2 \qquad y_{bj} \equiv \text{the } j^{th} \text{ point in the } b^{th} \text{ bin}$$

For each bin b, the model value is $y_{\text{mod},\,j} = \bar{y}_b$. $SST = SSA + SSE$ is an exact mathematical identity for all linear fits with a zero residual sum: $\sum \varepsilon_i = 0$, with the residuals ε_i defined as:

$$\varepsilon_i \equiv y_i - y_{\text{mod},i} = y_i - \bar{y}_{b(i)} \qquad \textit{where} \quad b(i) \equiv \text{bin for point } i\,.$$

The PDM basis functions automatically remove any DC (constant offset) from the signal, so the residual sum is zero; it is *not* necessary to subtract out the average before computing the periodogram, and doing so has no effect. However, some applications subtract any straight-line trend from the data before processing (more later).

Many data sets have individual uncertainties (**heteroskedastic** data): the uncertainty of each point y_i is u_i. To take account of these unequal uncertainties, we compute the sums-of-squares *SST*, *SSA*, and *SSE* from standard formulas for weighted estimates (see chapter "Uncertainty Weighted Data Analysis"). For *SST*:

$$SST = \sum_{i=1}^{n} w_i \left(y_i - \bar{y}\right)^2 = \sum_{i=1}^{n} w_i \left(y_i^2 - 2y_i\bar{y} + \bar{y}^2\right) = \sum_{i=1}^{n} w_i y_i^2 - 2V_1\bar{y}^2 + V_1\bar{y}^2 = \sum_{i=1}^{n} w_i y_i^2 - V_1\bar{y}^2$$

$$\textit{where} \quad w_i \equiv u_i^{-2}, \;\; V_1 \equiv \sum_{i=1}^{n} w_i\,.$$

The last expression is efficient for computer evaluation.

For *SSA*:

$$SSA = \sum_{b=1}^{r} \frac{\left(\bar{y}_b - \bar{y}\right)^2}{\mathrm{var}\left(\bar{y}_b\right)} = \sum_{b=1}^{r} V_{1b}\left(\bar{y}_b - \bar{y}\right)^2, \qquad V_{1b} \equiv \sum_{j=1}^{n_b} w_{bj} \ .$$

As a check on this, we set $u_i = 1$ for all measurements, and we should recover the unweighted formula (13.1):

$$V_1 = 1^{-2} + 1^{-2} + \ldots 1^{-2} = n_b \ ,$$

as required.

For *SSE*, note that in each bin b, $y_{\mathrm{mod},\,j} = \bar{y}_b$. Then the SSE for bin b is:

$$SSE_b = \sum_{j=1}^{n_b} w_{bj}\left(y_{bj} - \bar{y}_b\right)^2 = \sum_{j=1}^{n_b} w_{bj}\left({y_{bj}}^2 - 2y_{bj}\bar{y}_b + {\bar{y}_b}^2\right) = \sum_{j=1}^{n_b} w_{bj}{y_{bj}}^2 - 2V_{1b}{\bar{y}_b}^2 + V_{1b}{\bar{y}_b}^2$$

$$= \sum_{j=1}^{n_b} w_{bj}{y_{bj}}^2 - V_{1b}{\bar{y}_b}^2 \ .$$

Then the full formulas, in efficient form for computing, are:

$$SST = \sum_{i=1}^{n} w_i {y_i}^2 - V_1 \bar{y}^2, \qquad SSA = \sum_{b=1}^{r} V_{1b}\left(\bar{y}_b - \bar{y}\right)^2, \qquad SSE = \sum_{b=1}^{r}\left(\sum_{j=1}^{n_b} w_{bj}{y_{bj}}^2 - V_{1b}{\bar{y}_b}^2\right)$$

$$\textit{where} \quad u_i \equiv \text{uncertainties}, \quad w_i \equiv u_i^{-2}, \quad V_{1b} \equiv \sum_{j=1}^{n_b} w_{bj}, \quad V_{2b} \equiv \sum_{j=1}^{n_b} {w_{bj}}^2, \quad \bar{y}_b \equiv \frac{1}{V_{1b}} \sum_{j=1}^{n_b} w_{bj} y_{bj} \ .$$

$$SST = SSA + SSE \ .$$

When all the u_i are equal, this reduces to (13.1). (Numerical experiments on real data, with millions of trial frequencies, confirm that the implementation satisfies this final equation, the weighted ANOVA sum-of-squares identity, to within my detection limit of 1 part in 10^{12}.)

If the uncertainties are accurate, then *SST* has degrees of freedom dof = $n - 1$, *SSA* has dof = $r - 1$, and *SSE* has dof = $n - r$. In pure gaussian noise (no signal), the detection statistic $D(f)$ follows an $F_{r-1,\ n-r}$ distribution:

$$D(f) \equiv \frac{SSA/(r-1)}{SSE/(n-r)} \in F_{r-1,n-r} \qquad \textit{where} \quad f = \frac{1}{T} \equiv \text{the trial frequency} \ .$$

As with all periodograms, there are multiple independent frequencies, so the false alarm probability (FAP ≡ α) for the entire periodogram is higher than that for a single frequency. As noted in the section "ANOVA with Uncertainties," this standard F-test can be used as an approximate test for detection of a signal. However, the F critical values will be approximate, and therefore so will the *p*-value.

Once again, shuffle simulations overcome all of these approximations numerically, and are often the most reliable way to determine critical values.

[Schwa 1998 p832]. However, we dispute his claim [Schwa 1998 p833] that the "accuracy" of random number generators is not well tested. Using a 64-bit uniform generator, with a recurrence time of ~10^{18}, I have verified some detailed statistics of bin counts in uniformly sized bins. From uniform random numbers, exact theoretical transforms produce other needed distributions.

[The weighted ANOVA sum-of-squares identity holds only when the weighted sum-of-residuals = 0:

$$\sum_{i=1}^{n} w_i \varepsilon_i = 0 \, .$$

In general regression, this is usually enforced with a fit parameter b_0 added as a constant to the model: $y_{mod} = b_0 + \ldots$. However, the Phase Dispersion Minimization fit functions are flat-topped rectangular pulses, so they already include any constant offset in the fit. Adding a b_0 parameter would be redundant, and therefore would make the fit unstable. Do not do this.]

PDM Computer Implementation Tips

It is highly inefficient to actually fold the data and sort it for each trial frequency in the PDM periodogram. The sorting is at least $O(n \log n)$ (and often implemented as $O(n^2)$), and would be used for each trial frequency. A small addition is the per-bin statistics calculation. The total computational cost is then, at best, $O((n \log_2 n + r) n_f)$, and often worse. For thousands of points and frequencies, this could be prohibitive.

Instead, for each trial frequency, make a single pass through the data, and keep statistical running sums for each of the r bins. There is no sorting. At the end, compute the final statistics for each bin from the running sums. This is $O(n + r)$, for a final cost of $O((n + r) n_f)$. This reduces computation by at least a factor of $\log_2 n$ (and maybe n, possibly thousands).

PDM On Data With Trends

Sometimes data includes a straight-line trend (Figure 13.5). Since the PDM fit follows such a trend, it distorts the detection statistic from that for truly *periodic* components. Figure 13.6a shows a periodogram for "untrended" data, i.e. with the straight-line trend first fitted and subtracted. The result is a reasonable measure of truly period components. In contrast, the raw periodogram (Figure 13.6b), made from the original data with the trend, is very different at most frequencies. Therefore, if you are looking for periodic signals, and have a significant trend in your data, we recommend you "untrend" your data before making the PDM periodogram.

For reference, here are the straight-line fit equations for uncertainty-weighted data [Strutz 7.16 p177]. We start with measurement triples (t_i, y_i, u_i), where t_i are the independent variables, y_i are the measured quantities, and u_i are the uncertainties for each measurement. Then:

$$y_{\text{mod}}(x) = b_0 + b_1 x \, , \qquad w_i \equiv u_i^{-2} .$$

$$V_1 \equiv \sum_{i=1}^{n} w_i \, , \qquad S_x \equiv \sum_{i=1}^{n} w_i x_i \, , \qquad S_y \equiv \sum_{i=1}^{n} w_i y_i \, , \qquad S_{xx} \equiv \sum_{i=1}^{n} w_i x_i^2 \, , \quad S_{xy} \equiv \sum_{i=1}^{n} w_i x_i y_i \, ,$$

$$b_0 = \frac{S_{xx} S_y - S_x S_{xy}}{V_1 S_{xx} - S_x^2} \, , \qquad b_1 = \frac{V_1 S_{xy} - S_x S_y}{V_1 S_{xx} - S_x^2} \, .$$

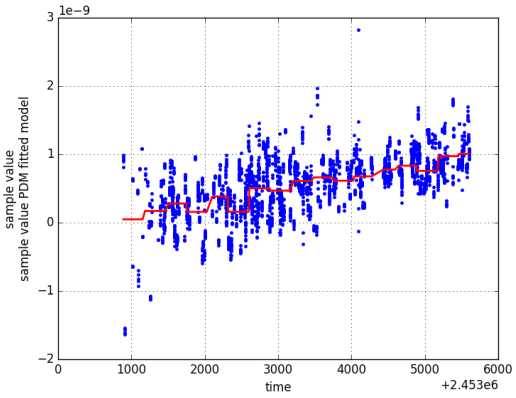


Figure 13.5 Exampe PDM fit (red, at the peak frequency) with folded data, when the data includes a straight-line trend. r = 20 bins. (The axes are not important.)

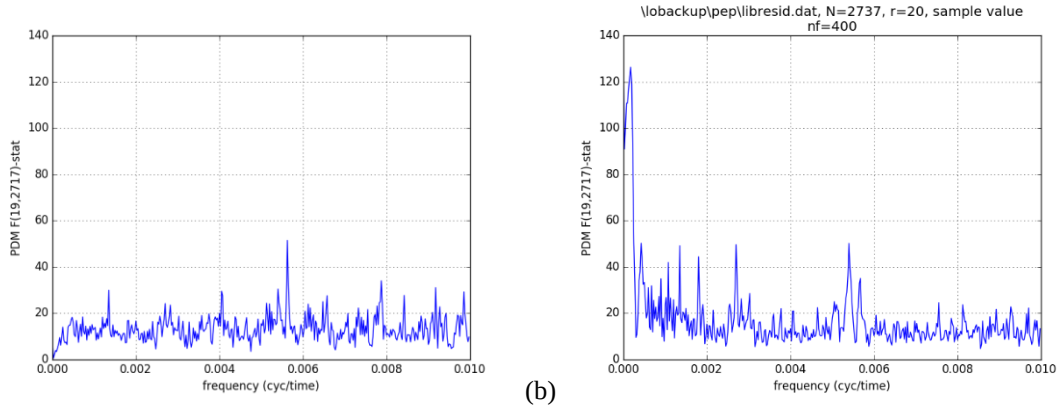


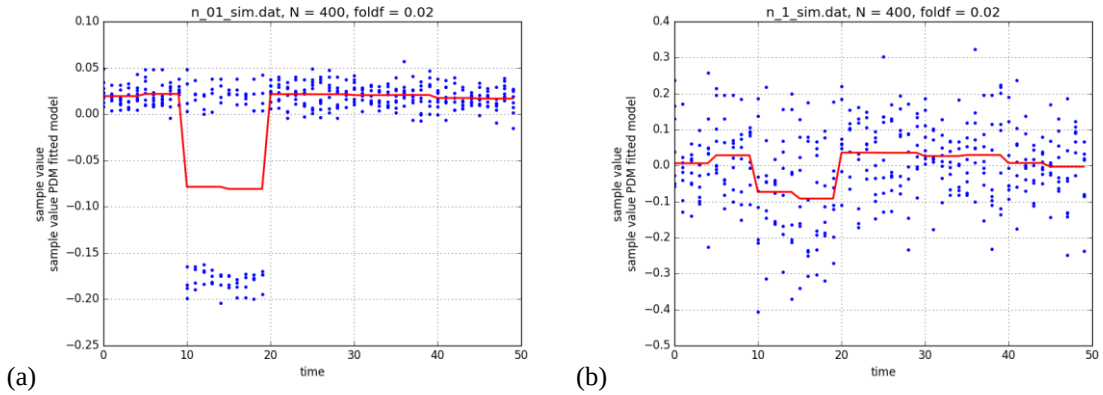
Figure 13.6 (a) PDM periodogram with the trend removed. (b) Periodogram with the trend intact is vastly different at most frequencies.

Subharmonic Response

Any signal periodic in $T = 1/f$ is also periods in $2T$, $3T$, ... Therefore, PDM produces a peak at each subharmonic of the periodicity. At least one subharmonic is visible in Figure 13.1d, and possibly 3. Often 5 or more subharmonics are visible. The response weakens slowly with each subharmonic order (increasing period) because data points from a wider range of "phase" (wider fraction of a period) are included in each bin as the subharmonic order increases. In effect, the bins become wider: they occupy a larger fraction of the true (fundamental) period.

Harmonic response

Harmonics of the fundamental frequency are sometimes visible as peaks in the periodogram, but they decay quickly with harmonic order. Harmonics are usually less visible than subharmonics. Figure 13.7a shows why: a fit to a harmonic must compromise between the two phases that are mapped to the same folded interval. The best fit is halfway between the two groups of points, and so not very good. Figure 13.7b is the 2nd harmonic fit to the data of Figure 13.1. It's amplitude is roughly half that of the fundamental in Figure 13.4.



(a) (b)

Figure 13.7 (a) Fit to the simulated low-noise transit data. (b) Fit to the simulated high-noise transit data.

Spectral window function

There is often confusion about whether the spectral window function (SWF) affects PDM periodograms. It does not. The SWF is computed explicitly for sinusoidal fits, such as the generalized sinusoidal periodogram [ref??], or the now-deprecated Lomb-Scargle periodogram.

NB: The "spectral window function" is *not* a windowing function for smoothing data, or for smoothing a spectrum. Those are unrelated to the "spectral window function," which is computed from the sample times of time-series data.

References

[Stell 1978] R. F. Stellingwerf, "Period Determination Use Phase Dispersion Minimization," Astrophysical Journal, 224:953-960, 9/15/1978.

Eric L. Michelsen, *Funky Mathematical Physics Concepts*, https://elmichelsen.physics.ucsd.edu/FunkyMathPhysics.pdf , 2/10/2020 or later.

14 Numerical Analysis

Round-Off Error, And How to Reduce It

Floating point numbers are stored in a form of scientific notation, with a **mantissa** and **exponent**. E.g.,

1.23×10^{45} has mantissa $m = 1.23$ and exponent $e = 45$.

Computer floating point stores only a finite number of digits. 'float' (aka single-precision) typically stores at least 6 digits; 'double' typically stores at least 15 digits. We'll work out some examples in 6-digit decimal scientific notation; actual floating point numbers are stored in a binary form, but the concepts remain the same. (See "IEEE Floating Point" in this document.)

Precision loss due to summation: Adding floating point numbers with different exponents results in **round-off error**:

$$\begin{array}{rcll} 1.234\,56 \times 10^{2} & \rightarrow & 1.234\,56 \times 10^{2} & \\ +\ 6.111\,11 \times 10^{0} & & +\ 0.061\,111\,1 \times 10^{2} & \\ & & = 1.295\,67 \times 10^{2} & \text{where } 0.000\,001\,1 \text{ of the result is lost,} \end{array}$$

because the computer can only store 6 digits. (Similar round-off error occurs if the exponent of the result is bigger than both of the addend exponents.) When adding many numbers of similar magnitude (as is common in statistical calculations), the round-off error can be quite significant:

```
float sum = 1.23456789;          // Demonstrate precision loss in sums
printf("%.9f\n", sum);           // show # significant digits
for(i = 2;  i < 10000;  i++)
     sum += 1.23456789;
printf("Sum of 10,000 = %.9f\n", sum);

1.234567881              8 significant digits
Sum of 10,000 = 12343.28 only 4 significant digits
```

We lost about 1 digit of accuracy for each power of 10 in n, the number of terms summed. I.e.:

$$\textit{digit-loss} \approx \log_{10} n\ .$$

When summing numbers of different magnitudes, you get a better answer by adding the small numbers first, and the larger ones later. This minimizes the round-off error on each addition, and shows that:

Finite-precision addition is *not* associative).

E.g., consider summing $1/n$ for 1,000,000 integers. We do it in both single- and double-precision, so you can see the error:

```
float sum = 0.;
double      dsum = 0.;
// sum the inverses of the first 1 million integers, in order
for(i = 1;  i <= 1000000;  i++)
sum += 1./i, dsum += 1./i;
printf("sum:  %f\ndsum: %f.  Relative error = %.2f %%\n",
                                sum, dsum, (dsum-sum)/dsum);

sum:  14.357358
dsum: 14.392727.  Relative error = 0.002457
```

This was summed in the worst possible order: largest to smallest, and (in single-precision) we lose about 5 digits of accuracy, leaving only 3 digits. Now sum in reverse (smallest to largest):

```
float sumb = 0.;
double      dsumb = 0.;
for(i = 1000000;  i >= 1;  i--)
```



```
      sumb += 1./i, dsumb += 1./i;
printf(" sumb:  %f\ndsumb:  %f.  Relative error = %.6f\n",
                          sumb, dsumb, (dsumb-sumb)/dsumb);
 sumb:  14.392652
dsumb:  14.392727.  Relative error = 0.000005
```

The single-precision sum is now good to 5 digits, losing only 1 or 2.

[In my research, I needed to fit a polynomial to 6000 data points, which involves many sums of 6000 terms, and then solving linear equations. I needed 13 digits of accuracy, which easily fits in double-precision ('double', 15-17 decimal digits). However, the precision loss due to summing was over 3 digits, and my results failed. Simply changing the sums to 'long double', then converting the sums back to 'double', and doing all other calculations in 'double' solved the problem. The dominant loss was in the sums, not in solving the equations.]

Summing from smallest to largest is very important for evaluating polynomials, which are widely used for transcendental functions. Suppose we have a 5th order polynomial, `f(t)`:

$$f(t) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 ,$$

which might suggest a computer implementation as :

```
f = a0 + a1*t + a2*t*t + a3*t*t*t + a4*t*t*t*t + a5*t*t*t*t*t
```

Typically, the terms get progressively smaller with higher order. Then the above sequence is in the *worst* order: biggest to smallest. (It also takes 15 multiplies.) It is more accurate (and faster) to evaluate the polynomial as:

```
f = ((((a5*t + a4)*t + a3)*t + a2)*t + a1)*t + a0
```

This form adds small terms of comparable size first, progressing to larger ones, and requires only 5 multiplies.

How To Extend Precision In Sums Without Using Higher Precision Variables

(Handy for statistical calculations): You can avoid round-off error in sums without using higher precision variables with a simple trick. For example, let's sum an array of *n* numbers:

```
sum = 0.;
for(i = 0;  i < n;  ++i)  sum += a[i];
```

This suffers from precision loss, as described above. The trick is to actually measure the round-off error of each addition, and save that error for the next iteration. This is called a **compensated add**:

```
sum = 0.;
error = 0.;                // the carry-in from the last add
for(i = 0;  i < n;  i++)
{   newsum = sum + (a[i] + error);     // include the lost part of prev add
    diff = newsum - sum;               // what was really added
    error = (a[i] + error) - diff;     // the round-off error
    sum = newsum;
}
```

The 'error' variable is the round-off error, and is always small compared to the 'sum'. Keeping track of it effectively doubles the number of accurate digits in the sum. After each addition, 'error' tells you how far off your sum is. For most practical purposes, this eliminates any precision loss due to sums. Let's try summing the inverses of integers again, in the "bad" order, but with this trick:

```
float  newsum, diff, sum = 0., error = 0.;
for(i = 1;  i <= 1000000;  i++)
{   newsum = sum + (1./i + error);
    diff = newsum - sum;               // what was really added
    error = (1./i + error) - diff;     // the round-off error
    sum = newsum;
```



```
}
printf(" sum:  %f\ndsumb:  %f.  Relative error = %.6f, error = %g\n",
                          sum, dsumb, (dsumb-sum)/dsumb, error);
  sum:  14.392727
dsumb:  14.392727.  Relative error = -0.000000, error = -1.75335e-07
```

As claimed, the sum is essentially perfect.

Numerical Integration

The above method of sums is extremely valuable in numerical integration. Typically, for accurate numerical integration, one must carefully choose an integration step size: the increment by which you change the variable of integration. E.g., in time-step integration, it is the time step-size. If you make the step size too big, accuracy suffers because the "rectangles" (or other approximations) under the curve don't follow the curve well. If you make the step size too small, accuracy suffers because you're adding tiny increments to large numbers, and the round-off error is large. You must "thread the needle" of step-size, getting it "just right" for best accuracy. This fact is independent of the integration interpolation method: trapezoidal, quadratic, Runge-Kutta ... .

By virtually eliminating round-off error in the sums (using the method above), you eliminate the lower-bound on step size. You can then choose a small step-size, and be confident your answer is right. It might take more computer time, but integrating 5 times slower and getting the right answer is vastly better than integrating 5 times faster and getting the wrong answer.

Sequences of Real Numbers

Suppose we want to generate the sequence 2.01, 2.02, ... 2.99, 3.00. A simple (but flawed) approach is this:

```
float s;
for(s = 2.01;  s <= 3.;  s += 0.01) ...
```

The problem with this is round-off error: 0.01 is inexact in binary (has round-off error). This error accumulates 100 times in the above loop, making the last value $100^{1/2}$ ~10 times more wrong than the first. In fact, the loop might run 101 times instead of 100. The fix is to use integers where possible, because they are exact:

```
float s;
int  i;
for(i = 201;  i <= 300;  ++i)
{   s = i/100.; ...
}
```

When the increment is itself a variable, note that multiplying a real by an integer incurs only a single round-off error:

```
real  s, base, incr;
int  i;
for(i = 1;  i <= max;  ++i)  s = base + i*incr;
```

Hence, every number in the sequence has only two round-off errors, from the multiply and the add.

Root Finding

In general, a **root** of a function $f(x)$ is a value of x for which $f(x) = 0$. It is often not possible to find the roots analytically, and it must be done numerically. [TBS: binary search]

Simple Iteration Equation

Some forms of $f(\ )$ make root finding easy and fast; if you can rewrite the equation in this form:

$$f(x) = 0 \qquad \rightarrow \qquad x = g(x)$$

then you may be able to iterate, using each value of $g(\,)$ as the new estimate of the root, r.

This is the simplest method of root finding, and generally the slowest to converge.

It may be suitable if you have only a few thousand solutions to compute, but may be too slow for millions of calculations.

You start with a guess that is close to the root, call it r_0. Then

$$r_1 = g(r_0), \qquad r_2 = g(r_1), \qquad \ldots \qquad r_{n+1} = g(r_n)$$

If $g(\,)$ has the right property (specifically, $|g'(x)| < 1$ near the root) this sequence will converge to the solution. We describe this necessary property through some examples. Suppose we wish to solve $\sqrt{x}/2 - x = 0$ numerically (exact is $x = 0.25$). First, we re-arrange it to isolate x on the left side: $x = \dfrac{\sqrt{x}}{2}$ (Figure 14.1a).

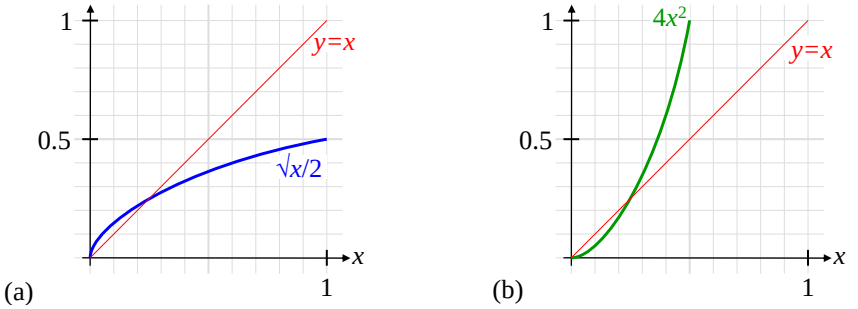


Figure 14.1 Two iteration equations for the same problem. (a) Sequence converges. (b) Sequence fails.

From the graph, we might guess $r_0 \approx 0.2$. Then we would find,

$$r_1 = \sqrt{0.2}/2 = 0.2236, \qquad r_2 = \sqrt{r_1}/2 = 0.2364, \qquad r_3 = 0.2431, \qquad r_4 = 0.2465, \qquad r_5 = 0.2483,$$

$$R_6 = 0.2491, \qquad r_7 = 0.2496$$

We see that the iterations approach the exact answer of 0.25.

But we could have re-arranged the equation differently: $2x = \sqrt{x}, \quad x = 4x^2$ (Figure 14.1b). Starting with the same guess $x = 0.2$, we get this sequence:

$$r_1 = \sqrt{0.2}/2 = 0.16, \qquad r_2 = \sqrt{r_1}/2 = 0.1024, \qquad r_3 = 0.0419, \qquad r_4 = 0.0070$$

It is not converging on the nearby root; the sequence diverges away from it. So what's the difference? Look at a graph of what's happening, magnified around the equality:

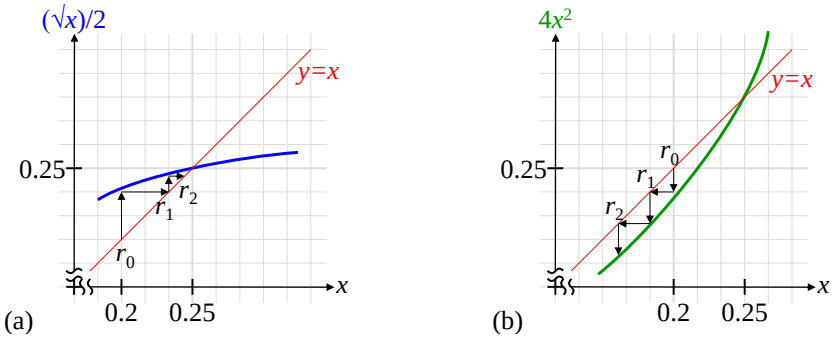


Figure 14.2 (a) Sequence converges. (b) Sequence fails.

When the curve is flatter than $y = x$ (above left), then trial roots that are too small get bigger, and trial roots that are too big get smaller. So iteration approaches the root. When the curve is steeper than $y = x$ (above right), trial roots that are too small get even smaller, too big get even bigger; the opposite of what we want. So *for positive slope curves*, the condition for convergence is

$$\frac{\Delta y}{\Delta r} = \frac{y(s)-y(r)}{s-r} < 1, \qquad \text{in the region} \qquad r - |r_0 - r| < s < r + |r_0 - r| \quad |r_i - r| < |r_0 - r|,$$

where r is the exact root; r_0 is the first guess.

Consider another case, where the curve has negative slope. Suppose we wish to solve $\cos^{-1} x - x = 0$, (x in radians). We re-write it as $x = \cos^{-1} x$. On the other hand, we could take the cosine of both sides and get an equivalent equation: $x = \cos x$. Which will converge? Again look at the graphs:

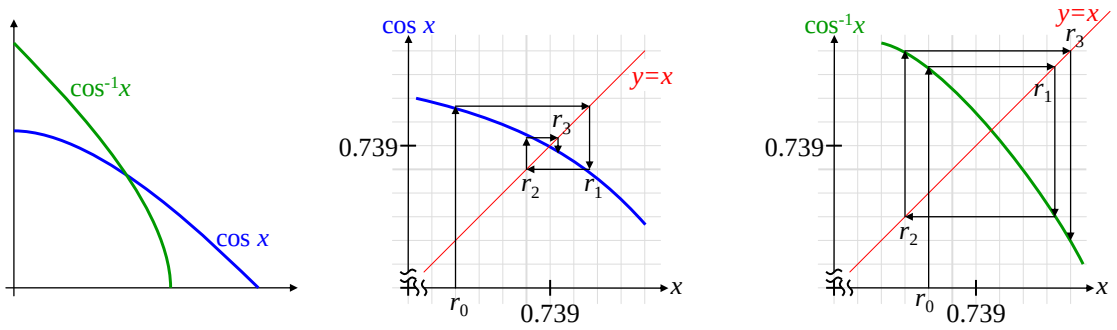


Figure 14.3 (Left) cos and $\cos^{-1}$ are superficially similar. (Middle) cos converges everywhere. (Right) $\cos^{-1}$ fails everywhere.

So long as the magnitude of the slope < 1 in the neighborhood of the solution, the iterations converge. When the magnitude of the slope > 1, they diverge. We can now generalize to all curves of any slope:

The general condition for convergence is

$$\left|\frac{\Delta y}{\Delta r}\right| = \left|\frac{y(s)-y(r)}{s-r}\right| < 1, \quad \text{in the region} \qquad r - |r_0 - r| < s < r + |r_0 - r|.$$

The flatter the curve, the faster the convergence.

Given this, we could have easily predicted that the converging form of our iteration equation is $x = \cos x$, because the slope of cos x is always < 1, and $\cos^{-1} x$ is always > 1. Note, however, that if the derivative (slope) is > 1/2, then the binary search will be faster than iteration.

Newton-Raphson Iteration

The above method of variable iteration is kind of "blind," in that it doesn't use any property of the given functions to advantage. **Newton-Raphson iteration** is a method of finding roots that uses the derivative of

the given function to provide more reliable and faster convergence. Newton-Raphson uses the original form of the equation: $f(x) = \sqrt{x}/2 - x = 0$. The idea is to use the derivative of the function to approximate its slope to the root (Figure 14.4a). We start with the same guess, $r_0 = 0.2$.

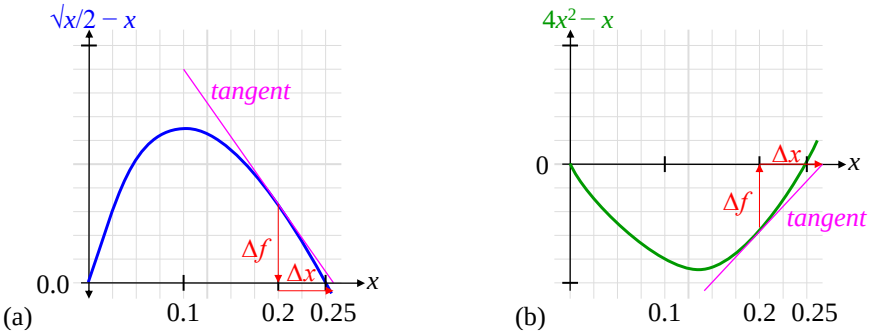


Figure 14.4 Illustration of Newton-Raphson for both forms of the root-finding equation.

$$\frac{\Delta f}{\Delta x} \approx f'(r_0) \qquad \Rightarrow \qquad \Delta r \approx -\frac{f(r_i)}{f'(r_i)} \qquad (\text{Note } f'(r_0) < 0)$$

$$f'(x) = \frac{1}{4}x^{-1/2} - 1 \qquad \Rightarrow \qquad \Delta r = -\frac{r_i^{1/2}/2 - r_i}{r_i^{-1/2}/4 - 1} \cdot \frac{4r_i^{1/2}}{4r_i^{1/2}} = -\frac{2r_i - 4r_i^{3/2}}{1 - 4r_i^{1/2}}$$

Here's a sample computer program fragment, and its output:

```
// Newton-Raphson iteration
r = 0.2;
for(i = 1;  i < 10;  i++)
{    r -= (2.*r - 4.*r*sqrt(r)) / (1. - 4.*sqrt(r));
     printf("r%d  %.16f\n", i, r);
}
```

r1 0.253 532 216 545 439 2

r2 0.250 012 217 175 258 8

r3 0.250 000 000 149 248 4

r4 0.250 000 000 000 000 0

In 4 iterations, we get essentially the exact answer, to double precision accuracy of 16 digits. This is much faster than the variable isolation method above. In fact, it illustrates a property of some iterative numerical methods called **quadratic convergence**:

> Quadratic convergence is when the fractional error (aka relative error) gets squared on each iteration, which doubles the number of significant digits on each iteration.

You can see this clearly above, where r_1 has 2 accurate digits, r_2 has 4, r_3 has 9, and r_4 has at least 16 (maybe more). Derivation of quadratic convergence??

Another advantage of Newton-Raphson over simple iteration is that Newton-Raphson does not have the restriction on the slope of any function, as does simple iteration. We can use it just as well on the reverse formula (Figure 14.4b):

$$f(x) = 4x^2 - x, \quad f'(x) = 8x - 1, \quad \Delta r = -\frac{f(r_i)}{f'(r_i)} = -\frac{4x^2 - x}{8x - 1}, \text{ with these computer results:}$$

r1 0.266 666 666 666 666 7

r2 0.250 980 392 156 862 7

r3 0.250 003 814 755 474 2

r4 0.250 000 000 058 207 7

r5 0.250 000 000 000 000 0

This converges essentially just as fast as the first form, and clearly shows quadratic convergence.

If you are an old geek like me, you may remember the iterative method of finding square roots on an old 4-function calculator: to find $\sqrt{a}$: divide a by r, then average the result with r. Repeat as needed:

$$r_{n+1} = \frac{a / r_n + r_n}{2}$$

You may now recognize that as Newton-Raphson iteration:

$$f(r) = r^2 - a = 0, \qquad f'(r) = 2r,$$

$$r_{n+1} = r_n + \Delta r = r_n - \frac{f(r)}{f'(r)} = r_n - \frac{r_n^2 - a}{2r_n} = r_n - \frac{r_n}{2} + \frac{a}{2r_n} = \frac{1}{2}\left(r_n + \frac{a}{r_n} \right)$$

If you are truly a geek, you tried the averaging method for cube roots: $r_{n+1} = \frac{a / r_n^2 + r_n}{2}$. While you found that it converged, it was very slow; cube-root(16) with $r_0 = 2$ gives only 2 digits after 10 iterations. Now you know that the proper Newton-Raphson iteration for cube roots is:

$$f(r) = r^3 - a = 0, \qquad f'(r) = 3r^2, \qquad r_{n+1} = r_n - \frac{r_n^3 - a}{3r_n^2} = r_n - \frac{r_n}{3} + \frac{a}{3r_n^2} = \frac{1}{3}\left(2r_n + \frac{a}{r_n^2} \right)$$

which gives a full 17 digits in 5 iterations for $r_0 = 2$, and shows (of course) quadratic convergence:

```
r1  2.6666666666666665
r2  2.5277777777777777
r3  2.5198669868999541
r4  2.5198421000355395
r5  2.5198420997897464
```

It is possible for Newton-Raphson to cycle endlessly, if the initial estimate of the root is too far off, and the function has an inflection point between two successive iterations (Figure 14.5):

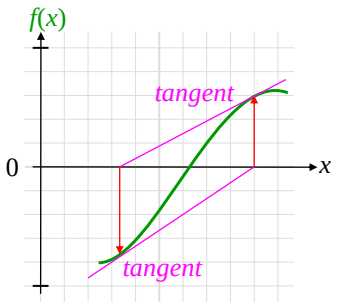


Figure 14.5 Failure of Newton-Raphson iteration.

It is fairly easy to detect this failure in code, and pull in the root estimate (say, with a midpoint method) before iterating again.

Pseudo-Random Numbers

We use the term "random number" to mean "pseudo-random number," for brevity. **Uniformly distributed** random numbers are equally likely to be anywhere in a range, typically (0, 1).

> Uniformly distributed random numbers are the starting point
> for many other statistical applications.

Computers can easily generate *uniformly distributed* random numbers. The best generators today are based on linear feedback shift registers (LFSR) [*Numerical Recipes*, 3rd ed.]. The old linear-congruential generator is:

```
// Uniform random value, 0 < v < 1, i.e. on (0,1) exclusive.
// Numerical Recipes in C, 2nd ed., p284
static  uint32  seed=1;         // starting point
vflt  rand_uniform(void)
{
    do seed = 1664525L*seed + 1013904223L;    // period 2^32-1
    while(seed == 0);
    rand_calls++;               // count calls for repetition check
    return seed / 4294967296.;
} // rand_uniform()
```

> Many algorithms that use random numbers fail on a "random" value of 0 or 1,
> so this generator never returns them.

After a long simulation with a large number of calls, it's a good idea to check 'rand_calls' to be sure it's < ~400,000,000 = 10% period. This confirms that the generated numbers are essentially random, and not predictable.

Arbitrary distribution random numbers: To generate any distribution from a uniform random number:

$$R = \text{cdf}_R^{-1}(U) \qquad \textit{where} \quad R \text{ is the random variable of the desired distribution}$$

$$\text{cdf}_R^{-1} = \text{inverse of the desired cumulative distribution function of } R$$

$$U \text{ is a uniform random number on (0,1)}$$

Figure 14.6 illustrates the process graphically. We can derive it mathematically as follows: recall that the **cumulative distribution function** gives the probability of a random variable being less than or equal to its argument:

$$\text{cdf}_X(a) \equiv \Pr(X \le a) = \int_{-\infty}^{a} dx\, \text{pdf}_X(x) \qquad \textit{where} \quad X \text{ is a random variable .}$$

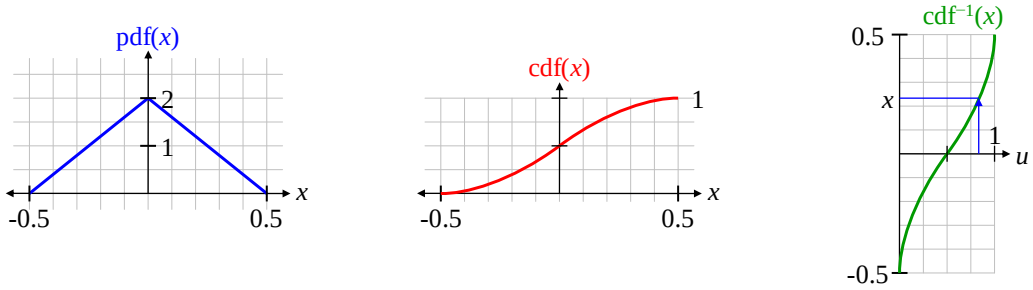


Figure 14.6 Steps to generating the probability distribution function (pdf) on the left.

Also recall that the pdf of a function of a random variable, say $F = f(u)$, is (see *Probability and Statistics* elsewhere in this document):

$$\text{pdf}_F(x) = \frac{\text{pdf}_X(x)}{f'(x)}, \qquad \textit{where} \quad f'(x) \equiv \text{derivative of } f(x).$$

Let $Q \equiv \text{cdf}_R^{-1}(U)$. Using $\text{pdf}_U(u) = 1$ on $[0,1]$

$$\text{pdf}_Q(r) = \frac{\text{pdf}_U(u)}{\frac{d}{du}\text{cdf}_R^{-1}(u)} = \frac{1}{\left(\frac{d}{dr} cdf_R(r)\right)^{-1}} \qquad \text{Using } \frac{d}{du} g^{-1}(u) = \left(\frac{d}{du} g(u)\right)^{-1}, \text{ and } u \to r$$

$$= \text{pdf}_R(r), \qquad \text{as desired.}$$

Generating Gaussian Random Numbers

The inverse CDF method is a problem for gaussian random numbers (any many others), because there is no closed-form expression for the CDF of a gaussian (or for the CDF^{-1}):

$$\text{CDF}(a) = \int_{-\infty}^{a} dx \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \qquad \text{(gaussian)}.$$

But [Knu] describes a clever way based on polar coordinates to use two uniform random numbers to generate a gaussian. He gives the details, but the final result is this:

$$\textit{gaussian} = \left(\sqrt{-2\ln u}\right)\cos\theta \qquad \textit{where} \quad \theta \text{ is uniform on } (0, 2\pi)$$

u is uniform on $(0,1)$

```
/* Gaussian random value, 0 mean, unit variance.  From Knuth, "The Art of
Computer Programming, Vol. 2: Seminumerical Algorithms," 2nd Ed., p. 117.
It is exactly normal if rand_uniform() is uniform.  */
PUBLIC  double     rand_gauss(void)
{   double  theta = (2.*M_PI) * rand_uniform();
    return sqrt( -2. * log(rand_uniform()) ) * cos(theta);
} // rand_gauss()
```

Note that u must exclude both 0 and 1.

Generating Poisson Random Numbers

Poisson random numbers are integers; we say the Poisson distribution is **discrete**:

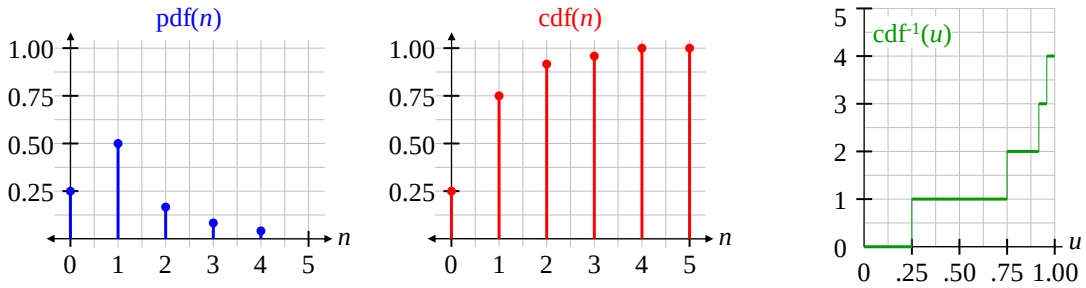


Figure 14.7 Example of generating the (discrete) Poisson distribution.

We can still use the inverse-cdf method to generate them, but in an iterative way. The code starts with a helper function, poisson( ), that compute the probability of exactly n events in a Poisson distribution with an average of *avg* events:

```
// ---------------------------------------------------------------------------
double poisson(          // Pr(exactly n events in interval)
      double avg,        // average events in interval
      int    n)          // n to compute Pr() of
```



```
{   double  factorial;
    int     i;

    if(n <= 20)  factorial = fact[n];
    else
    {   factorial = fact[20];
        for(i = 21;  i <= n;  ++i)  factorial *= i;
    }
    return exp(-avg) * pow(avg, n) / factorial;
} // poisson()

/*-----------------------------------------------------------------------------
Generates a Poisson random value (an integer), which must be <= 200.
Prefix 'irand_...' emphasizes the discreteness of the Poisson distribution.
-----------------------------------------------------------------------------*/
int   irand_poisson(      // Poisson random integer <= 200
      double avg)               // avg # "events"
{
    int           i;
    double  cpr;                // uniform probability

    // Use inverse-cdf(uniform) for Poisson distribution, where
    // inverse-cdf() consists of flat, discontinuous steps
    cpr = rand_uniform();
    for(i = 0;  i <= 200;  ++i) // safety limit of 200
    {   cpr -= poisson(avg, i);
        if(cpr <= 0)  break;
    }
    return i;                   // 201 indicates an error
} // irand_poisson()
```

Other example random number generators: TBS.

Generating Useful, But More Challenging, Random Numbers

Sometimes you need to generate more complex distributions, such as a combination of a gaussian with a uniform background of noise, a "raised gaussian" (Figure 14.8).

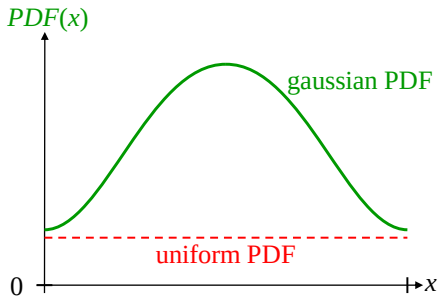


Figure 14.8 Construction of a raised gaussian random variable from a uniform and a gaussian.

Since this distribution has a uniform "component," it is only meaningful if it's limited to some finite "width." To generate distributions like this, you can compose two different distributions, and use the principle:

> The PDF of a random choice of two random variables is the weighted sum of the individual PDFs.

For example, the PDF for an RV (random variable) which is taken from X 20% of the time, and Y the remaining 80% of the time is:

$$\text{pdf}_z(z) = 0.2\,\text{pdf}_X(z) + 0.8\,\text{pdf}_Y(z)\,.$$

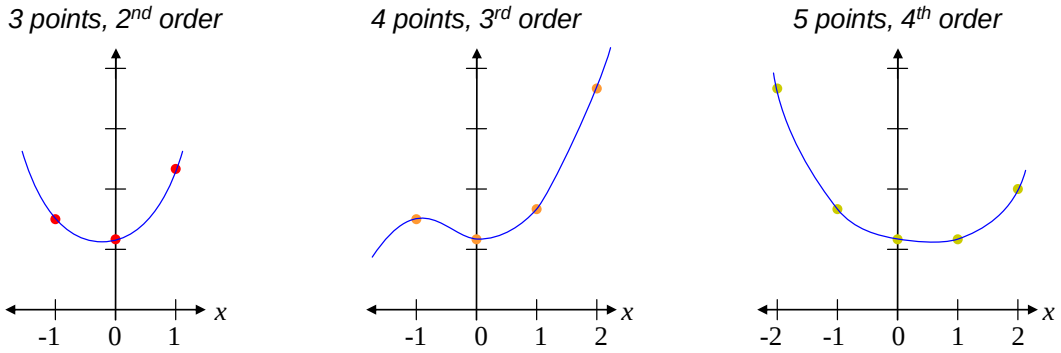
In this example, the two component distributions are uniform and gaussian. Suppose the uniform part of the pdf has amplitude 0.1 over the interval (0, 2). Then it accounts for 0.2 of all the random values. The remainder are gaussian, which we take to be mean of 1.0, and σ = 0.5. Then the random value can be generated from three more-fundamental random values:

```
// Raised Gaussian random value: gaussian part: mean=1, sigma=1
// Uniform part (20% chance): interval (0, 2)
if(rand_uniform() <= 0.2)
      random_variable = rand_uniform()*2.0;
else
      random_variable = rand_gauss()*0.5 + 1.0;       // mean = 1, sigma = 0.5
```

This isn't quite right, because the gaussian will have values < 0 and > 2, which will give weird "tails" to the generated PDF. More complete code would detect those, and regenerate until a valid value results.

Exact Polynomial Fits

It's sometimes handy to make an exact fit of a quadratic, cubic, or quartic polynomial to 3, 4, or 5 data points, respectively.



The quadratic case illustrates the principle simply. We seek a quadratic function:

$$y(x) = a_2 x^2 + a_1 x + a_0$$

which exactly fits 3 equally spaced points, at $x = -1$, $x = 0$, and $x = 1$, with value y_{-1}, y_0, and y_1, respectively (shown above). So long as your actual data are equally spaced, you can simply scale and offset to the x values –1, 0, and 1. We can directly solve for the coefficients a_2, a_1, and a_0:

$$\left.\begin{aligned} a_2(-1)^2 + a_1(-1) + a_0 &= y_{-1} \\ a_2(0)^2 + a_1(0) + a_0 &= y_0 \\ a_2(1)^2 + a_1(1) + a_0 &= y_1 \end{aligned}\right\} \quad \Rightarrow \quad \left.\begin{aligned} a_2 - a_1 + a_0 &= y_{-1} \\ a_0 &= y_0 \\ a_2 + a_1 + a_0 &= y_1 \end{aligned}\right\}$$

$$\Rightarrow \qquad a_2 = (y_{-1} + y_1)/2 - y_0, \qquad a_1 = (y_1 - y_{-1})/2, \qquad a_0 = y_0$$

Similar formulas for the 3rd and 4th order fits yield this code:

```
// ------------------------------------------------------------------------
// fit3rd() computes 3rd order fit coefficients.   4 mult/div, 8 adds
PUBLIC      void   fit3rd(
     double ym1, double y0, double y1, double y2)
{
    a0 = y0;
    a2 = (ym1 + y1)/2. - y0;
    a3 = (2.*ym1 + y2 - 3.*y0)/6. - a2;
    a1 = y1 - y0 - a2 - a3;
} // fit3rd()
```



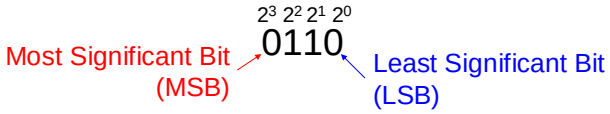
```
// ------------------------------------------------------------------------
// fit4th() computes 4th order fit coefficients.  6 mult/div, 13 add
PUBLIC      void   fit4th(
     double ym2, double ym1, double y0, double y1, double y2)
{
    b0 = y0;
    b4 = (y2 + ym2 - 4*(ym1 + y1) + 6*y0)/24.;
    b2 = (ym1 + y1)/2. - y0 - b4;
    b3 = (y2 - ym2 - 2.*(y1 - ym1))/12.;
    b1 = (y1 - ym1)/2. - b3;
} // fit4th()
```

TBS: Alternative 3rd order (4 point) symmetric fit, with x = {-3, -1, 1, 3}.

15 Computer Math Internals

Digital Integer Arithmetic: Two's Complement

Two's complement is a way of representing negative numbers in binary. It is universally used for integers, and rarely used for floating point. This section assumes the reader is familiar with positive binary numbers and simple binary arithmetic.



Two's complement uses the most significant bit (MSB) of an integer as a sign bit: zero means the number is ≥ 0; 1 means the number is negative. Two's complement represents non-negative numbers as ordinary binary, with the sign bit = 0. Negative numbers have the sign bit = 1, but are stored in a special way: for a b-bit word, a negative number n ($n < 0$) is stored as if it were unsigned with a value of $2^b - |n| = 2^b + n$. This is shown below, using a 4-bit "word" as a simple example:

bits	unsigned	signed
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

With two's complement, a 4-bit word can store integers from –8 to +7. E.g., –1 is stored as 16 – 1 = 15. The two's-complement rule is usually defined as follows (which completely obscures the purpose):

For $n < 0$, let $a \equiv \|n\|$	Example: $n = -4$, $a = 4$
Start with the bit pattern for a:	0100
Complement it (change all 0s to 1s and 1s to 0s):	1011
Add 1:	1100

Let's see how two's complement works in practice. There are 4 possible addition cases:

(1) Adding two positive numbers: so long as the result doesn't overflow, we simply add normally (in binary).

(2) Adding two negative numbers: Recall that when adding unsigned integers, if we overflow our 4 bits, the "carries" out of the MSB are simply discarded. This means that the result of adding $a + c$ is actually

(a + c) mod 16. Now, let n and m be negative numbers in twos complement, so their bit patterns are 16 + n, and 16 + m. If we add their bit patterns as unsigned integers, we get

$$(16+n)+(16+m)=\big[32+(n+m)\big]\bmod 16=16+(n+m), \qquad n+m<0$$

which is the 2's complement representation of $(n + m) < 0$.

E.g.,	–2	1110	16 + (–2)
	+ –3	+ 1101	+ 16 + (–3)
	–5	1011	16 + (–5)

So with two's complement, adding negative numbers uses the same algorithm as adding unsigned integers! That's why almost all computers use two's complement.

(3) Adding a negative and a positive number, with positive result:

$$(16+n)+a=\big[16+(n+a)\big]\bmod 16=n+a, \qquad n+a>0$$

E.g.,	–2	1110	16 + (–2)
	+ 5	0101	+ 5
	3	0011	3

(4) Adding a negative and a positive number, with negative result:

$$(16+n)+a=16+(n+a), \qquad n+a<0$$

E.g.,	–6	1010	16 + (–6)
	+ 3	0011	+ 3
	–3	1101	16 + (–3)

In all cases, with two's complement arithmetic, adding signed integers uses the same algorithm as adding unsigned integers! That's why almost all computers use two's complement.

The computer hardware need not know which numbers are signed, and which are unsigned: it adds the same way no matter what. It works the same with subtraction: subtracting two's complement numbers is the same as subtracting unsigned numbers.

It even works multiplying to the same word size:

$$-+: \quad (16+n)a=\big[16a+(na)\big]\bmod 16=16+na, \qquad n<0,\ a>0,\ na<0$$

$$--: \quad (16+n)(16+m)=\big[256+16(n+m)+nm\big]\bmod 16=nm, \qquad n<0,\ m<0,\ nm>0$$

In reality, word sizes are usually 32 (or maybe 16) bits. Then in general, we store b-bit negative numbers ($n < 0$) as $2^b + n$. E.g., for 16 bits, $(n < 0) \rightarrow 65536 + n$.

Two's complement notation explains why the ranger of signed integers is lopsided: from $–2^b$ to $+(2^b – 1)$.

Hexadecimal

Hexadecimal notation (base-16) is used as a shorthand for writing binary. Each group of 4 bits is written as a single hexadecimal digit:

Hexadecimal							
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Any binary bit pattern can be written in hexadecimal (or "hex"):

0110 1100 1010 0011 = 6CA3 .

Hex letters can be written as either upper or lower case, though upper case is a little more formal, and (we think) easier to read.

For unsigned integers, hexadecimal isn't just shorthand; it is a true base-16 number:

$$1010\,0011 = 2^7 + 2^5 + 2^1 + 2^0 = 163 \qquad \Leftrightarrow \qquad A3 = \underbrace{10\cdot 16^1 + 3\cdot 16^0 = 163}_{\text{base 10}}$$

Two's complement negative numbers, floating point, and many other bit patterns are often written in hex. Two's complement looks a little funny:

$$1111\,0011 = 243 - \underbrace{2^8}_{256} = -13 \qquad \Leftrightarrow \qquad F3 = \underbrace{243 - 16^2 = -13}_{\text{base 10}}.$$

In C, hexadecimal numbers start with a "0x" prefix, e.g. 0xA3. In Fortran code, hex uses the Z or X notation, e.g. z'A3' or x'A3'. However, for input/output, the FORMAT edit descriptor uses only Z (because X is already used for spaces).

In the old days, there was such a thing as "octal," which is groups of 3 bits, or base 8. This is clumsy because computer word lengths are usually divisible by 4, but not 3. Octal is obsolete.

Digital Floating Point

Computer floating point is essentially scientific notation, but in binary. Briefly, a binary floating point number is represented by 3 fields: S, E, and T: a sign bit, an exponent, and a significand (aka mantissa), Figure 15.1. The value v of the number is [IEEE-754-2008 p9]:

$$v = (-1)^{sign}\,(2^{exponent})\,(significand)\ .$$

For example, $v = -(2^{10})\ 1.625$. Details later.

How Far Can I Go?

What is the range, in decimal, of numbers that can be represented by the IEEE floating point formats? The answer is dominated by the number of bits in the binary exponent. This table shows it:

Range and Precision of Some Floating Point Formats

Format	Significant Bits	Smallest Normal Magnitude	Largest Number	Decimal Digits
IEEE binary32	24	$1.175... \times 10^{-38}$	$3.402... \times 10^{+38}$	6-9
IEEE binary64	53	$2.225... \times 10^{-308}$	$1.797... \times 10^{+308}$	15-17
x86 long double	64	$3.362... \times 10^{-4932}$	$1.189... \times 10^{+4932}$	18-21
IEEE binary128	113	$3.362... \times 10^{-4932}$	$1.189... \times 10^{+4932}$	33-36

How Many Digits Do I Get, 6 or 9?

How many decimal digits of accuracy do I get with a binary floating point number? You often see a range: 6 to 9 digits. Huh?

Wobble, but don't fall down: The idea of "number of digits of accuracy" is somewhat flawed. Six digits of accuracy near 100,000 is ~10 times worse than 6 digits of accuracy near 999,999. The smallest increment is 1 in the least-significant digit (1 ULP ≡ unit in the last place). One in 100,000 is accuracy of 10^{-5}; 1 in 999,999 is essentially 10^{-6}, or 10 times more accurate.

Aside: The **wobble** of a floating point number is the ratio of the lowest accuracy to the highest accuracy for a fixed number of digits in its representation (e.g., digits in base 2, 10, or 16). It is always equal to the base in which the floating point number is expressed, which is 10 in this example. The wobble of binary floating point is 2. The wobble of hexadecimal floating point (mostly obsolete now) is 16.

We assume IEEE-754 compliant numbers (see later section). To insure, say, 6 decimal digits of accuracy, the worst-case binary accuracy must exceed the best-case decimal accuracy. For IEEE binary32, there are 23 fraction bits (and one implied-1 bit), so the worst case accuracy is $2^{-23} = 1.2 \times 10^{-7}$. The best 6-digit accuracy is 10^{-6}; the best 7 digit accuracy is 10^{-7}. Thus we see that single-precision guarantees 6 decimal digits, but almost gets 7, i.e. most of the time, it actually achieves 7 digits. Though again, "significant digits" is a flawed concept; "relative error" is better.

The table in the next section summarizes precision in 4 common floating point formats.

How many digits do I need?

Often, we need to convert a binary number to decimal, write it to a file, and later read it back in, converting it back to binary. An important question is, how many decimal digits do we need to write to insure that we get back *exactly* the same binary floating point number we started with? In other words, how many binary digits do I get with a given number of decimal digits? (This is essentially the reverse of the preceding section.) We choose our number of decimal digits to insure full binary accuracy (assuming our conversion software is good, which is not always the case).

Our worst-case decimal accuracy has to exceed our best-case binary accuracy. For IEEE binary32 (single precision), the best accuracy is $2^{-24} = 6.0 \times 10^{-8}$. For 9 decimal digits, the worst case accuracy is 10^{-8}, so we need 9 decimal digits to fully represent binary32. Below is a table of precisions for 4 common formats; note that all but the x86 long double format use an implied leading '1' bit, so the precision is one more than the number of stored mantissa bits.

Format	Significant bits	Minimum decimal digits accuracy	Decimal digits for exact replication	Decimal digits range
IEEE binary32	24	$2^{-23} = 1.2 \times 10–7$ => 6	$2^{-24} = 6.0 \times 10–8$ => 9	6 – 9
IEEE binary64	53	$2^{-52} = 2.2 \times 10–16$ => 15	$2^{-53} = 1.1 \times 10–16$ => 17	15 – 17
x86 long double	64	$2^{-63} = 1.1 \times 10–19$ => 18	$2^{-64} = 5.4 \times 10–20$ => 21	18 – 21
IEEE binary128	113	$2^{-112} = 1.9 \times 10–34$ => 33	$2^{-113} = 9.6 \times 10–35$ => 36	33 – 36

These numbers of digits agree exactly with the quoted ranges in the "IEEE Floating Point" section below, and the ULP table in the underflow section. In C, then, to insure exact binary accuracy when writing, and then reading, in decimal, for binary64, use:

```
sprintf(dec, "%.17g", x);
```

Not all conversion software achieves the ideal accuracy. We recommend testing at least 100,000,000 bit patterns (logarithmic distribution) on your system.

Warning: it is wholly inadequate to test "uniformly" distributed real numbers on any interval, because that does not exercise the full range of exponents in floating point representations. It *is* sufficient to test uniformly distributed bit patterns.

Emulate vs. Simulate

(where to put??) There is substantial disagreement on the difference between emulation and simulation, and there is some genuine gray area between them. However, we think the best definitions are:

simulate — Use modeling to compute aspects of a system, without actually producing any of the physical interfaces or interactions of a real system. It's all done in the computer.

emulate — Mimic at least some of the physical interfaces or interactions of a system; this may include simulating some aspects of the system, but usually includes some hardware beyond a general purpose computer.

Emulators may sometimes run at full speed, but often must run at reduced speed due to physical limitations. Simulators have no timing constraints, because they aren't physically "doing" anything; they usually run orders of magnitude slower than the real system.

We illustrate the gray area with an example: consider a touchpad. It's a simple device, with a touch screen, and software to interact with a user. Suppose we write a program for a computer with a touchscreen that performs the functions of some touchpad program. Is the computer program a simulator, or an emulator? Since it reproduces the physical interactions of the user touch interface, it fits the definition of an emulator. On the other hand, it was all done on the computer, with no additional hardware, so it *feels* like a simulator.

Many computing systems do not have floating point hardware. The floating point operations are implemented in a software library. This is neither simulation nor emulation; it's just an implementation. However, we have worked on systems with a floating point *architecture* (instruction set), but with inexpensive versions of hardware that do not have built-in floating point. Nonetheless, they run code with floating point instructions by trapping on them, invoking software routines to perform the floating point operation, and returning to the mainline execution. The mainline code does not "know" whether the floating point is done is software or hardware, nor should it. Such trapping is usually called floating point emulation, because the software is actually executing the floating point instructions, albeit at slower speeds. The emulator is providing the instruction set interface to the application code.

Programming Guidance for Floating Point

Even in IEEE compliant systems, the same expression can evaluate to different results in different contexts or locations in the code. Most languages do not demand that two instance of the same expression always evaluate to the same result, and in many real cases, they do not. A trivial example is:

```
double x = 1./3.;
if(x == 1./3.)  ...       // ill-posed
```

Is the above comparison equal or not equal? You can't know. But all such code is ill-formed.

> It is fundamental to floating point that the unavoidable approximations in computations mostly prevent exact comparisons.

The reason the above code is ill-posed is that many architectures (e.g., x86) keep intermediate results in registers of higher precision than the constituent values. 'x' above is a binary64, and will be rounded to that precision. In the 'if' statement, 1./3. will likely be held in a register, which on the x86 is 80 bits. A 64-bit version of 1/3 is not the same as an 80-bit version of 1/3. Therefore, it is well-known that comparisons should generally test for equality within some allowable range (which depends on the application):

```
if( fabs(x - 1./3.) < epsilon)  ...    // well-posed
```

Sometimes you must check for some *relative* error, rather than an absolute error. For example, to compare 'x' and 'y' (assuming x > 0):

```
if( fabs((x - y)/x - 1.) < epsilon)  ...      // well-posed
```

It seems clunky, but it is widely used. It gets clunkier if there's a chance that $x = 0$. However, you can write a function to centralize it all.

One of the few times you *can* exactly compare floating point numbers is when they have been *directly assigned* integer values: they can be counted on to be exact, because there is no possibility of rounding:

```
float x = 27.;
if(x == 27.) ...          // well-posed
if(int(x) == 27) ...      // well-posed
```

(Of course, 'x' must be big enough to hold the integer.) *However*, *any* intermediate fractional computations break this assurance:

```
float x = 27./5*5;
```

results in an unpredictable value of 'x'. All we can rely on is that 'x' is *close to* 27.

Some other considerations are in [Unk 1999].

IEEE Floating Point

This section describes IEEE-754-2008, which updated the 1985 version. There is a 2019 update which makes only minor changes. Much of the information here comes directly from the standard. Much of the tabular information comes from Sun's *Numerical Computation Guide*, 2003 [Sun 2003]. Floating point computation is a large and intricate field, and we strongly recommend consulting more detailed references (including the sections above) before making assumptions about how it works.

Note: the standard uses the word "exception" to mean something different than the definition used by many computer languages [IEEE-754 p3b]. For example, in C/C++, IEEE "exceptions" are turned into "signals," and do *not* trigger C++ "exceptions."

Goals

IEEE-754 lists goals in both the introduction [p. iv], and the overview [p1]. Essentially, they want to make it easier to write portable numeric code, and have such code report errors consistently across all platforms. And they have succeeded well in both of these.

They also hope (in both the introduction and overview) that computation results will be identical across platforms. This is probably a pipe dream. There are many factors working against this goal, most of which are outside our scope here. Essentially though, the cost to achieve this goal is enormous, and most users are simply unwilling to pay such a price. However, the standard recommends languages define a "literal meaning" that, in principle, could allow identical results on disparate platforms [IEEE-754 p50], even at the cost of runtime efficiency. For many platforms, this recommendation is not feasible.

What Is IEEE Arithmetic?

In brief, IEEE 754 specifies exactly how floating point operations are to occur, and to what precision. Though it does *not* specify how the floating point numbers are stored in memory, it *does* specify bit-level interchange formats. Each computer makes its own choice for how to store floating point numbers. We give some popular formats later.

The original specification, IEEE-754-1985, defined binary floating point. IEEE-754-2008 added decimal formats. We discuss here only binary floating point.

IEEE 754-2008 specifies (among other things) a binary floating point standard with:

- Three basic binary floating-point formats: binary32 (32-bits), binary64, and binary128 [IEEE-754 p6]. (The 32 and 64-bit formats were called "single" and "double" in the 1985 version of IEEE-754.)
- Every representable number has a unique representation [IEEE-754 p9t].
- Recommendations for extending these formats to other lengths [IEEE-754 p6].
- Precise accuracy requirements for mathematical operations, including rounding. Though individual operations require bit-level agreement between implementations [IEEE-754 p. iv, top], other factors usually prevent such agreement on the whole.
- Four rounding modes for binary [IEEE-754 sec. 4.3.3 p16b], only one of which is widely used: "roundTiesToEven."
- Precise accuracy requirements for conversion between decimal and binary, including guarantees of exact recovery from binary->decimal->binary conversions [IEEE-754 p30].
- Representations for negative zero, ±infinity, and two forms of NaN (not a number).
- Five kinds of exceptions: invalid operation, division by zero, overflow, underflow, and inexact.

The standard does *not* provide:

- Function libraries (sin, cos, etc.), which often do not have bit-level accuracy, and therefore may differ between platforms.
- Precise handling of intermediate results: in a multi-step calculation, rounding may occur at different places in a full-language programming environment, even between conforming IEEE platforms. Therefore, "conforming" platforms may yield different results.

Note that *most* floating point programs work fine on multiple platforms, and even on non-compliant floating-point implementations, despite tiny differences in their results. Numerical computation existed long before the first standard in 1985, and was eminently useful.

High level languages have different names for floating point data types, which usually correspond to the IEEE formats as shown here [Sun 2003]:

IEEE Formats and Example Language Types

IEEE Precision	C, C++	Fortran
binary32	`float`	`REAL` or `REAL*4`
binary64	`double`	`DOUBLE PRECISION` or `REAL*8`
double extended (80-bit)*	`long double`	`REAL*10`
binary128		`REAL*16` [e.g., SPARC]. Note that in many implementations, REAL*16 is *different* than 'long double'

* 80-bit double-extended is not part of the standard, and does not comply with the interchange format for "extended" precisions, due to the explicit 'j' field. However, it is a widely used part of the x86 architecture.

[IEEE-754 2008 p13] defines a binary16 format, but it is rarely used.

Binary Numeric Formats

A binary floating point number is represented by 3 fields: S, E, and T: a sign bit, an exponent, and a significand (aka mantissa) (Figure 15.1). The value v of the number is:

$$v = (-1)^{sign}\ (2^{exponent})\ (significand)\ , \qquad \text{e.g. } -(2^{10})\ 1.625 \qquad \text{[IEEE-754-2008 p9].}$$

p is defined as the number of bits in the significand (the precision), e.g. for binary32, $p = 24$ [IEE-754 p8].

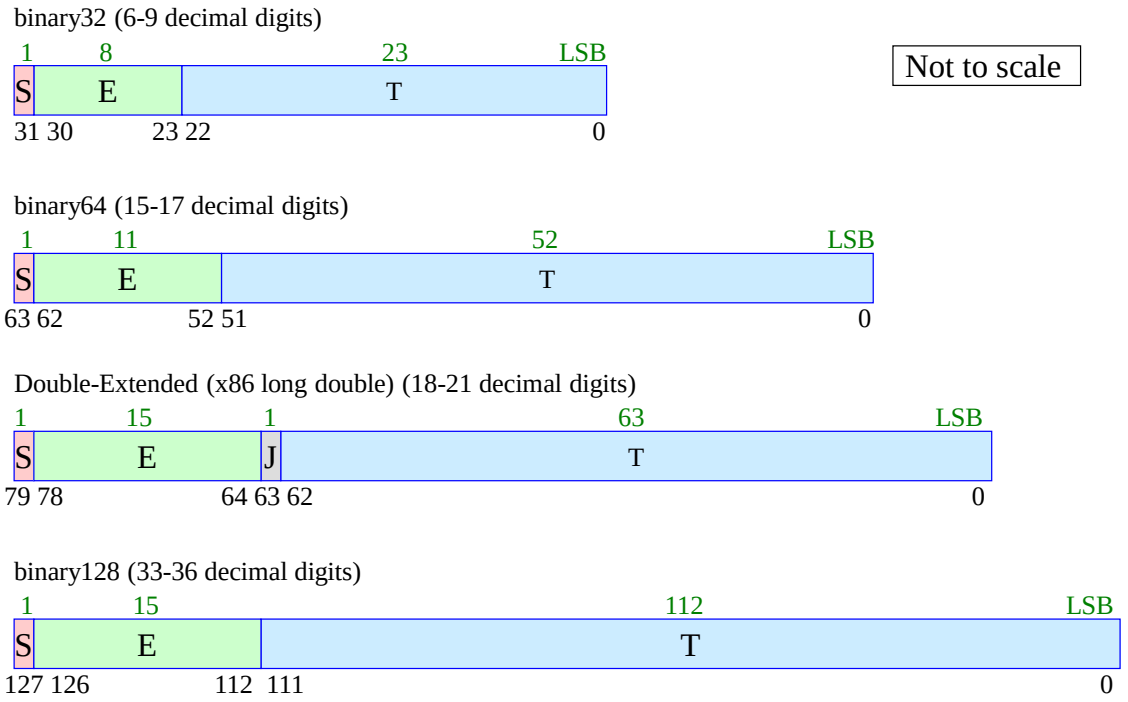


Figure 15.1 Common floating point formats, with S, E, T fields named as in [IEEE-754 p9]. (However, IEEE-754 numbers the bits from 0 on the left.)

As in the standard, we sometimes refer to the bit-fields E and T by their integer interpretation. Then in binary32, E = 128 corresponds to an exponent of 1 = 128 – *bias*.

When a Bias Is a Good Thing

IEEE floating point uses **biased exponents**, where the actual exponent is the unsigned value of the 'E' field minus a constant, called a bias:

exponent = E – bias *where* E ≡ biased exponent .

The bias makes the E field an *unsigned* integer, and the smallest numbers have the smallest E field (as well as the smallest exponent). The bias provides that (1) floating point numbers sort in the same order as if their bit patterns were integers; and (2) true floating point zero is naturally represented by an all-zero bit pattern. These might seem insignificant, but they are quite useful, and so biased exponents are nearly universal.

In IEEE floating point, two values of E are special: 0, and all 1's. Briefly: E = 0 is used for subnormal numbers (described later), and is the same exponent value as E = 1 (exponent = 1 – bias). For example, in binary32, E is an 8-bit field, and the bias is 127. The maximum exponent is then 254 (0xFE), and the minimum is –126. When E is all 1's (255 in binary32), the number is either an infinity or NaN (not a number). Full details to come.

Interchange Formats and Storage Formats

IEEE-754-2008 defines an "interchange format" to be used for exchanging data between systems. However, within a single system, each implementation defines its own storage format (how a number is stored in memory). Oddly, [IEEE-754 Table 3.5 p13] defines the "storage width" in bits, and the number of bits in each field, but this certainly describes the interchange format.

Note that two's complement is *not* used for negative numbers.

The IEEE floating-point formats define the fields that compose a floating-point number, the bits in those fields, and their arithmetic interpretation, but not how those formats are stored in memory.

Each computer defines its own storage formats, though they are obviously all related.

IEEE binary32 (Single) Format

The table below shows the constituent fields, S, E, and T, and the values represented [Sun 2003]:

Binary32 Format Fields	Value
$1 \le E \le 254$	$(-1)^S \times 2^{E-127} \times 1.T$ (normal numbers)
$E = 0;\ T \ne 0$	$(-1)^S \times 2^{-126} \times 0.T$ (subnormal numbers)
$E = 0;\ T = 0$	$(-1)^S \times 0.0$ (signed zero)
$E = 255;\ T = 0$	$(-1)^S \times \infty$ (± infinity)
S = either; $E = 255;\ T \ne 0$	NaN (Not-a-Number)

On x86, which is little-endian, the least significant byte is stored first.

Note that when $1 \le E \le 254$, the significand is found by prepending "1." to the fraction in T:

$$v = (-1)^S \times 2^{E-bias+1} \times 1.T\ .$$

The "1." is implied, and is called the "implicit bit." The result is that $1 \le significand < 2$. Thus the 23-bit T field yields 24 bits of precision.

Below, the hexadecimal numbers follow Figure 15.1, with the sign and exponent on the left. The decimal values are rounded [Sun 2003].

Important Bit Patterns in IEEE Binary32 Format

Common Name	Bit Pattern (Hex)	Approximate Value
+0	0000 0000	0.0
– 0	8000 0000	–0.0
1	3F80 0000	1.0
2	4000 0000	2.0
maximum normal number	7F7F FFFF	3.40282347e+38
minimum positive normal number	0080 0000	1.17549435e–38
maximum subnormal number	007F FFFF	1.17549421e–38
minimum positive subnormal number	0000 0001	1.40129846e–45
+ ∞	7F80 0000	+ ∞ (positive infinity)
– ∞	FF80 0000	– ∞ (negative infinity)
Not-a-Number (NaN)	7FCx xxxx	NaN*

* NaN requires only that the most significant bit of T = 1, so the 'C' hex digit can be C - F.

binary64 (Double) Format

The IEEE binary64 format is an obvious extension of the binary32 format: a 52-bit trailer, T; an 11-bit biased exponent, E; and a 1-bit sign, S. The 52-bit fraction combined with the implicit leading 1-bit provides 53 bits of precision. The table below shows the values represented [Sun 2003]:

binary64 Format Fields	Value
$1 \le E \le 2046$	$(-1)^S \times 2^{E-1023}$ x 1.T (normal numbers)
$E = 0; T \ne 0$	$(-1)^S \times 2^{-1022}$ x 0.T (subnormal numbers)
$E = 0; T = 0$	$(-1)^S \times 0.0$ (signed zero)
$E = 2047; T = 0$	$(-1)^S \times \infty$ (± infinity)
S = either; $E = 2047; T \ne 0$	NaN (Not-a-Number)

On SPARC, which is big-endian, the most significant 32 bits are stored as the first word. x86 is little-endian, so the least significant byte is stored first.

Below, the hexadecimal numbers follow Figure 15.1, with the sign and exponent on the left. The decimal values are rounded [Sun 2003].

Important Bit Patterns in IEEE binary64 Format

Common Name	Bit Pattern (Hex)	Approximate Value
+ 0	`000 00000 00000000`	0.0
– 0	`800 00000 00000000`	−0.0
1	`3FF 00000 00000000`	1.0
2	`400 00000 00000000`	2.0
max normal number	`7FE FFFFF FFFFFFFF`	1.797 693 134 862 315 7 e+308
min positive normal number	`001 00000 00000000`	2.225 073 858 507 201 4 e−308
max subnormal number	`000 FFFFF FFFFFFFF`	2.225 073 858 507 200 9 e−308
min positive subnormal number	`000 00000 00000001`	4.940 656 458 412 465 4 e−324
+ ∞	`7FF 00000 00000000`	+ ∞ (positive infinity)
− ∞	`FFF 00000 00000000`	– ∞ (negative infinity)
Not-a-Number	`7FF 8xxxx xxxxxxxx`	NaN*

* NaN requires only that the most significant bit of T = 1, so the '8' hex digit can be 8 - F.

binary128 (Double-Extended) Format

SPARC quadruple-precision conforms to IEEE binary128 format. Analogous to binary64, the first 32-bit word holds the sign, exponent, and most-significant bits of T. This table shows the values represented [Sun 2003]:

binary128 Format Fields	Value
$1 \le E \le 32766$	$(-1)^S$ x $2^{E-16383} \times 1.T$ (normal numbers)
$E = 0, T \ne 0$	$(-1)^S$ x $2^{-16382} \times 0.T$ (subnormal numbers)
$E = 0, T = 0$	$(-1)^S$ x 0.0 (signed zero)
$E = 32767, T = 0$	$(-1)^S \times \infty$ (± infinity)
S = either, $E = 32767, T \ne 0$	NaN (Not-a-Number)

On SPARC, which is big-endian, the most significant 32 bits are stored as the first word. x86 is little-endian, so the least significant byte is stored first.

Below, the hexadecimal numbers follow Figure 15.1, with the sign and exponent on the left. The decimal values are rounded [Sun 2003]:

Important Bit Patterns in IEEE binary128 Format

Name	Bit Pattern (hex)	Approximate Value
+0	0000 0000 00000000 00000000 00000000	0.0
−0	8000 0000 00000000 00000000 00000000	−0.0
1	3FFF 0000 00000000 00000000 00000000	1.0
2	4000 0000 00000000 00000000 00000000	2.0
max normal	7FFE FFFF FFFFFFFF FFFFFFFF FFFFFFFF	1.189 731 495 357 231 765 085 759 326 628 0070 e+4932
min normal	0001 0000 00000000 00000000 00000000	3.362 103 143 112 093 506 262 677 817 321 7526 e−4932
max subnormal	0000 FFFF FFFFFFFF FFFFFFFF FFFFFFFF	3.362 103 143 112 093 506 262 677 817 321 7520 e−4932
min pos subnormal	0000 0000 00000000 00000000 00000001	6.475 175 119 438 025 110 924 438 958 227 6466 e−4966
+ ∞	7FFF 0000 00000000 00000000 00000000	+ ∞
− ∞	FFFF 0000 00000000 00000000 00000000	− ∞
Not-a- Number	7FFF 8xxx xxxxxxxx xxxxxxxx xxxxxxxx	NaN*

* NaN requires only that the most significant bit of T = 1, so the ‘8’ hex digit can be 8 - F.

x86 Double-Extended Format

The x86 double-extended format is *not* part of the standard. The important difference in the x86 double-extended (aka long-double) format is the lack of an implicit leading 1-bit in the significand. Instead, the 1-bit is explicit in the j field, and always present in normalized numbers. The binary (radix) point is between j and T, which together compose the significand. This clearly violates the spirit of the IEEE standard. However, big companies carry a lot of clout with standards bodies, so Intel claims this double-extended format conforms to the IEEE definition of double-extended formats, because IEEE 754 does not specify how (or if) the leading 1-bit is *stored*. Thus, x86 long-double consists of *four* fields (rather than three): a 63-bit fraction, T; a 1-bit explicit leading significand bit, j; a 15-bit biased exponent, E; and a 1-bit sign, S (note the additional j field as the *explicit* leading bit).

Nonetheless, for all normalized numbers ($1 \le E \le 32766$), j *must* be 1. Operating on a value with $1 \le E \le 32766$ and j = 0 triggers an invalid operation. Only subnormal numbers can have j = 0.

In x86 architectures, a double-extended number occupies 10 bytes, in little-endian order. However, for parameter and result passing, the UNIX System V Application Binary Interface Intel 386 Processor Supplement (Intel ABI) specifies 12-bytes, with the most significant two bytes being unused (see below). (This is likely due to the run-time speed benefits of 32-bit aligned data in memory.)

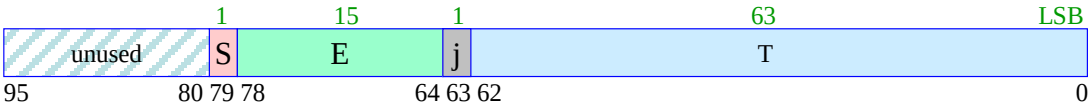


Figure 15.2 Unix ABI format for 80-bit double-extended (long double) includes two pad bytes.

The x86 architecture also specifies two kinds of NaN: a quiet NaN which always propagates through calculations without raising a hardware signal, and a signaling NaN, which raises a hardware signal if it is used in any operation.

Below shows the four constituent fields and the values they represented. x = don't care [Sun 2003].

Double-Extended Fields (x86)	Value
j = 0, 1 <= E <= 32766	Unsupported
j = 1, 1 <= E <= 32766	$(-1)^S$ x $2^{E-16383}$ x 1.T (normal numbers)
j = 0, E = 0; T ≠ 0	$(-1)^S$ x 2^{-16382} x 0.T (subnormal numbers)
j = 1, E = 0	$(-1)^S$ x 2^{-16382} x 1.T (pseudo-denormal numbers)
j = 0, E = 0, T = 0	$(-1)^S$ x 0.0 (signed zero)
j = 1; S = 0/1; E = 32767; T = 0	± ∞ (± infinity)
j = 1; S = x; E = 32767; T = .1xxx...xx	QNaN (quiet NaNs)
j = 1; S = x; E = 32767; T = .0xxx...xx ≠ 0	SNaN (signaling NaNs)

Below are some important bit patterns in the double-extended format [Sun 2003].

Important Bit Patterns in Double-Extended (x86) Format and their Values

Common Name	Bit Pattern	Approximate Value
+0	0000 00000000 00000000	0.0
−0	8000 00000000 00000000	−0.0
1	3FFF 80000000 00000000	1.0
2	4000 80000000 00000000	2.0
max normal	7FFE FFFFFFFF FFFFFFFF	1.189 731 495 357 231 765 05 e+4932
min positive normal	0001 80000000 00000000	3.362 103 143 112 093 506 26 e−4932
max subnormal	0000 7FFFFFFF FFFFFFFF	3.362 103 143 112 093 506 08 e−4932
min positive subnormal	0000 00000000 00000001	3.645 199 531 882 474 602 53 e−4951
+ ∞	7FFF 80000000 00000000	+ ∞
− ∞	FFFF 80000000 00000000	− ∞
quiet NaN with least fraction	7FFF Cxxxxxxx xxxxxxxx	QNaN*
signaling NaN with least fraction	7FFF 8xxxxxxx xxxxxxxx	SNaN*

All NaN's have E = E_{max}. A quiet NaN has the most significant trailer (T) bits = 11, so the leading hex digit may be C-F. A signaling NaN sets the most significant T bits to 10, so the leading hex digit may be 8-B.

Precision in Binary and Decimal

See the earlier section on *How Many Digits Do I Get?* for introductory information. Because decimal numbers are different than binary numbers, estimating the number of significant decimal digits corresponding to *b* significant binary bits requires some definition. Figure 15.3 illustrates that representable decimal base numbers are different than those in binary. Although all binary numbers can be represented exactly in decimal, this usually requires unreasonably many digits to do so. Therefore, *exact conversions are generally not possible*.

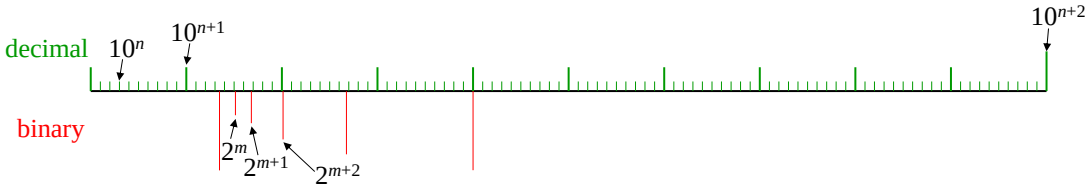


Figure 15.3 Comparison of numbers representable in decimal and binary.

However, as noted earlier, conversions from binary->decimal->binary can be done which preserve exactly the original binary value, and this is required by [IEEE-754 sec. 5.12]. For example, in a typical system, converting sqrt(2) to 19 decimal digits yields:

sqrt(2) = 1.414 213 562 373 095 146

The following calculations show that in this case, 17 decimal digits represent the value exactly, and 16 digits do not:

sqrt(2) – 1.414 213 562 373 095 1 = 0

sqrt(2) – 1.414 213 562 373 095 = 2.220 446 049 250 313 081 e-16

However, for some numbers, a rounded 16 digits is adequate:

sqrt(5) = 2.236 067 977 499 789 805

sqrt(5) – 2.236 067 977 499 790 = 0

We showed earlier that (for good conversion software), 17 digits is always enough to represent binary64 numbers.

The Big ULP

Before we discuss underflow, we need to understand concepts and terminology related to precision. Most floating point calculations have to be rounded. Just as in ordinary scientific notation, the round off error depends on the magnitude of the number being rounded. Using binary32 as an example, consider the number:

$$\mathbf{1}.000\ 0000\ 0000\ 0000\ 0000\ 0001 \times 2^0 .$$

The smallest increment we can make to such a number is to add one **unit in the last place** (**ULP**). In this case, that's 2^{-23}. Ideally, any roundoff will be less than this; in fact, roundoff should be $\le$ (1/2)ULP = $(1/2)2^{-23}$. A number with a bigger exponent will have a bigger ULP:

$$1.000\ 0000\ 0000\ 0000\ 0000\ 0001 \times 2^2 \qquad \Rightarrow \qquad \text{ULP} = 2^{-23}2^2 = 2^{-21} .$$

Since the ULP depends on the number, we can write ULP as a function of that number, ulp(*x*). This function is often available to programmers, such as in the Boost C++ libraries. Here are ulp(1.0) for some common precisions [Sun 2003]:

ulp(1.0) in Different Precisions	
binary32 (single)	ulp(1.) = 2^{-23} ~ 1.192 093 e–07
binary64 (double)	ulp(1.) = 2^{-52} ~ 2.220 446 e–16
double extended (x86)	ulp(1.) = 2^{-63} ~ 1.084 202 e–19
binary128	ulp(1.) = 2^{-112} ~ 1.925 930 e–34

Along these lines, there is a standard C++ library function `nextafter(x,y)` that returns the next floating point number after *x* in the direction of *y* (up or down). Here are several examples [Sun 2003]:

Gaps Between Representable binary32 Numbers		
x	**nextafter(x, +∞)**	**Gap**
0.0	1.401 298 5 e–45	1.401 298 5 e–45
1.1754944e-38	1.175 494 5 e–38	1.401 298 5 e–45
1.0	1.000 000 1	1.192 092 9 e–07
2.0	2.000 000 2	2.384 185 8 e–07
16.000 000	16.000 002	1.907 348 6 e–06
128.000 00	128.000 02	1.525 878 9 e–05
1.000 000 0e+20	1.000 000 1 e+20	8.796 093 0 e+12
9.999 999 7e+37	1.000 000 1 e+38	1.014 120 5 e+31

Round and Round

Unfortunately, there are several ways to round off numbers, and IEEE-754 specifies four different ways (for binary). The only reasonable rounding method is the default, called roundTiesToEven. This is essentially the way scientists learn to round off numbers: fractions >½ get rounded up, <½ get rounded down, and exactly =1/2 get rounded to make the preceding digit even. The standard requires exact conformance to this rounding, so care must be taken because multiply and add of 24-bit operands can produce 47 or 48 bit results. *Every bit* of the result must be taken into account, if necessary, to properly round. For example, in binary32, consider these 48-bit multiplication products:

1.000 0000 0000 0000 0000 0000 1000 0000 0000 0000 0000 0001 rounds to
1.000 0000 0000 0000 0000 0001

but

1.000 0000 0000 0000 0000 0000 1000 0000 0000 0000 0000 0000 rounds to
1.000 0000 0000 0000 0000 0000

Efficient implementations need not compute all 48 bits of the result to effect proper rounding. They can actually keep only 25 bits of result, along with a flag indicating that one or more 1-bits have been discarded so far [Knu_vol2 Ex. 5 p227]:

1.000 0000 0000 0000 0000 0001 1 + (discarded 1's flag)

To round:

- if the 25th bit is 0, round down;
- if it is 1 and there are any discarded 1's, round up;
- if it is 1 and no discarded 1's, round to preceding digit even.

Why roundTiesToEven? A long calculation accumulates more and more roundoff error as it progresses. Many operations round, and the error (on average) gets worse. With grade-school-type rounding, where ties (e.g., 0.5) always rounds up, the rounding error is *biased*: it has a non-zero average, because it slightly favors rounding up. The (average) cumulative rounding error is then proportional to n, the number of operations.

In contrast, roundTiesToEven is *unbiased*: half the time ties goes up, and half the time they goes down. The average magnitude of the rounding error is proportional to $n^{1/2}$, which is generally much smaller.

roundTiesToEven is also symmetric: round(x) = –round(–x).

Other rounding modes: The standard suggests that the other rounding modes might be useful for diagnosing numerical problems [IEEE-754 p55]. There are some rare calculations, such as interval computing, that can try to bound the errors on a given calculation. One approach is to run the calculation with "roundTowardNegative," and then run it again with "roundTowardPositive." Under some reasonable conditions, the true answer will be between the two, so they establish error bounds.

We have seen several software implementations that provide only the default rounding mode, roundTiesToEven.

Underflow

Normalized numbers have a minimum nonzero magnitude, given in the tables above for each precision. Before the standard, any result smaller than this minimum was often simply replaced by zero (an underflow method called "store 0"). This sometimes causes "jerkiness" in sensitive algorithms near zero. Instead, IEEE-754 specifies a "smoother" method, that actually represents numbers smaller than the minimum normal, called "gradual underflow." These unnormalized numbers are called **subnormal numbers**.

The idea is simple: subnormal numbers simply don't have an implied '1' as the most significant bit of the mantissa. Instead, their value is:

$$v = (-1)^S \times 2^{-bias\,+\,1} \times 0.\mathrm{T}\ .$$

Subnormal numbers always have the smallest allowed exponent, e.g. –126 in binary32, though it is stored as E = 0. Thus E = 1 and E = 0 are the *same* exponent, but the former has an implied 1-bit, and the latter does not. Therefore, the smallest representable magnitude is, in binary32 format:

$$0.000\ 0000\ 0000\ 0000\ 0000\ 0001 \times 2^{-126} = 2^{-23}\ 2^{-126} = 2^{-149} = 1.4 \times 10^{-45}\ .$$

The following table repeats the minimum normal magnitudes given earlier, and includes the next lower representable magnitude, which is the *largest* subnormal magnitude [Sun 2003]:

Underflow Thresholds in Each Precision

binary32 (single)	smallest normal number largest subnormal number	1.175 494 35 e–38 1.175 494 21 e–38
binary64 (double)	smallest normal number largest subnormal number	2.225 073 858 507 201 4 e–308 2.225 073 858 507 200 9 e–308
double extended (x86)	smallest normal number largest subnormal number	3.362 103 143 112 093 506 26 e–4932 3.362 103 143 112 093 505 90 e–4932
binary128	smallest normal number largest subnormal number	3.362 103 143 112 093 506 262 677 817 321 752 6 e–4932 3.362 103 143 112 093 506 262 677 817 321 752 0 e–4932

Subnormal numbers have fewer bits of precision than normal numbers, and the precision decreases as the subnormal numbers get smaller. This decreasing precision is called **gradual underflow**. The inclusion of gradual underflow into the standard was controversial at the time, but is now almost universally accepted.

Gradual underflow means the *absolute* roundoff error for subnormal numbers is the same as for the smallest normal numbers. Of course, the *relative* roundoff error is worse for subnormal numbers, because they are smaller magnitude than normal numbers.

Gradual underflow has several useful properties. An important one is:

$$x \neq y \quad \Leftrightarrow \quad x - y \neq 0 .$$

This is true mathematically, but with Store 0, $x - y$ might underflow and appear to be 0, when in fact $x \neq y$. With gradual underflow, $x \neq y$ is *always* equivalent to $x - y \neq 0$. This is because of the following: if x and y have the same exponent, then $x - y$ is exact, even with subnormal numbers. If x and y have different exponents, then at least one of them is normalized, and $x - y$ cannot be less than subnormal, and hence cannot be zero.

Note that even with gradual underflow, multiplication can produce surprising results: even if x and y are both > 0, $x*y$ can be "zero." By default, this will signal an underflow arithmetic fault to the program.

Most numerical programs work fine with no consideration of underflow; most programs never create or use such small numbers. However, when they do, gradual underflow relieves the application designer of a lot of work and error analysis related to the old-fashioned Store 0 approach.

References

[Knu_vol2] Donald E. Knuth, *The Art of Computer Programming*, *Seminumerical Algorithms*, Third Edition (Reading, Massachusetts: Addison-Wesley, 1997), ISBN 0-201-89684-2.

16 Scientific Programming: Discovering Efficiency

Programming for science, especially research,
is substantially *different* than for commercial software.

We focus here on scientific programming, which is neglected in most references.

I was a commercial software engineer for about 15 years, and have been in research science for about 20 more (as of 2023). I have been engineering software for 51 years, for both stand-alone computers and embedded systems. I have coded extensively in C, C++, Fortran, Pascal, and various Assemblers. I've used Python a fair bit, Basic, Emacs MLisp, some specialized application-specific array languages (e.g., CALMA GPL), and experimented with oddball ones like Lisp, Trac, and machine micro-code. I've written Fortran compilers, assemblers, link-editors, CPU and physics simulators, done kernel-level OS modifications, and low-level device drivers. That said, I've been out of mainstream software engineering for 24 years, so I'm not fully current on all the newest trends.

I've seen both research code and commercial software in detail, and seen the *difference*: the tradeoffs are different between scientific programming and commercial programming. Much more of scientific programming is one-off: never to be used again. Scientific programming is often (but not always) done with a smaller group of programmers, often just one. Commercial software typically involves a handful to hundreds of engineers working on the same build, or suite. Scientific programs are often much smaller than commercial programs. That doesn't mean research code should be sloppy; it shouldn't.

One reason commercial programming dogma fails in scientific programming is simple: overkill.

While much commercial dogma is actually bad even for commercial software, it is more-bad for research software. Far too many researchers are barraged with extremism: "never use global variables!", "never use 'goto'!", "never use 'break' or 'continue'!", "never return from the middle of a function!", etc.

That said, much of commercial software development advice is *good* for scientific programming, as well. This chapter presents our views of good and bad advice.

There are two very different kinds of "efficiency":

(1) "development efficiency:" the time it takes you to get your code working (design, code, debug); and

(2) the run-time efficiency of the code itself.

Usually, your development time is by far more important, and the run-time efficiency is not important. However, my own dissertation required attention to run-time efficiency. Furthermore, supercomputer centers *demand* that your code meet specific run-time efficiency standards. Therefore, we address both kinds of efficiency.

So sit back, take an alprazolam, and give it a skim.

Software Development Efficiency

Some Do's and Don't's

We expound later on some of the following recommendations:

- Do design your code up front: before coding, think about where you're going, and might go.
- Do "modularize" your code: separate it into hierarchical chunks.
- Do get your code working first; optimize it later only where needed.
- Do include lots of comments in-line: seconds now save days or weeks later.
- Don't be a zealot; avoid the word "never."
- Do be an engineer and focus on efficiency, not dogma.

- Do use coding guidelines. If you don't have any, use mine: https://elmichelsen.physics.ucsd.edu/ .
- Do use unwieldy language features sparingly: global variables, 'goto', embedded 'return', etc, and include comments explaining why you chose them.
- Don't complicate simple code, unless there is a clear payoff exceeding the bloat cost.
- If it ain't broke, be sure it's properly maintained so it doesn't break in the future.

Considerations on Development Efficiency and Languages

Programming languages are hammers. Not all programming jobs are the same; use the right hammer for the job. The following advice derives from my decades of engineering, programming, and science experience described earlier.

I've seen the productivity benefits of ego-free programming, and I strongly discourage zealotry about hammers, or languages. It's bad for science; it's bad for your career.

I emphasize that C++ is useful *without* a paradigm shift.

Arguably, C++ *allows* a paradigm shift to object-orientation, but it certainly does not *require* it. You can reap lots of productivity gains with just a few simple classes, and *no* paradigm shift. In fact, scientific data analysis often gains nothing from shifting paradigms, which would just waste a lot of time. If you're used to C, or any language, then ease into C++ (or object orientation) gradually. It's OK. The alleged paradigm shift is more useful for complex data structures, which scientific data usually are *not*.

Most people program Python without writing their own classes, again indicating that end-user-defined object-orientation is often not very useful. Especially so in scientific software.

Sophistication Follows Function

It is just as important for a computer program to communicate to humans as to machines.

When coding anything, the sophistication of your code should scale up with the demands of its function. Simple functions need only simple code; don't make complexity for no reason. Especially don't make complexity just to conform with zealous dogma. There's a huge difference between a 1000 line concept experiment, and a 100,000 line solar-system simulator meant to be used for decades. As a concrete example, consider the following two snippets:

```
for(vector<datapoint>::iterator index = data.begin();  index != data.end();
++index)
      do_something(*index);
```

vs.

```
for(int i = 0;  i < ndata;  ++i)
      do_something(data[i]);
```

These two snippets do essentially the same thing. Which would you rather read? For most scientific programming, the latter is far preferable than the bloated former. In science, a simple list of data points is a common and well-understood data structure. Don't bloat it unless there is a clear payoff that exceeds the cost of the bloat. Libraries like the Standard Template Library are designed to be extremely general; such generality inevitably carries a burden of complexity.

In fact, the awkwardness of the first snippet led to a new language feature in C++11, the range-based 'for' loop. This exemplifies a serious problem with the ever-evolving C++ language: the solution to its clumsiness is (too-often) ever-growing complexity. Rarely does the language evolve in a way to eliminate the *need* for ever-more remedies. (Another example is "move constructors" and "move assignment", which only partially solve the target problem, add more complexity to class methods, makes the language self-contradictory, and could have been mostly avoided by simply allowing coders to explicitly specify the

destination object separately from the two operands. This would have made compilers simpler, too. More later.)

Engineering vs. Programming

Software Engineering is much more than computer programming: it is the art and science of designing and implementing programs efficiently, over the long term, across multiple developers. Software engineering maximizes productivity and fun, and minimizes annoyance and roadblocks.

Engineers first design, then implement, systems that are useful, fun, and efficient.

Hackers just write code. Software engineering includes:

- Documentation: lots of it in the code as comments. ("Commercial Fortran codes often contain about 50% comments." [web.stanford.edu/class/me200c/tutorial_77/03_basics.html retrieved 2000-06-03].)
- Documentation: design documents that give an overview and conceptual view that is infeasible to achieve in source code comments.
- Coding guidelines: for consistency among developers. Efficiency can only be achieved by cooperation among the developers, including a consistent coding style that allows others to quickly understand the code. E.g., https://elmichelsen.physics.ucsd.edu/Coding_Guidelines.pdf .
- Clean code: easy to read and follow.
- Maintainable code: it functions in a straightforward and comprehensible way, so that it can be changed easily and still work.

Notice that all of the above are subjective assessments. That's the nature of all engineering:

Engineering is lots of tradeoffs, with subjective approximations of the costs and benefits.

Don't get me wrong: sometimes I hack out code. The judgment comes in knowing when to hack and when to design.

Fun quotes:

"Whenever possible, ignore the coding standards currently in use by thousands of developers in your project's target language and environment."
- Roedy Green, *How To Write Unmaintainable Code*, www.strauss.za.com/sla/code_std.html

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by [implication], not smart enough to debug it." **- Brian W. Kernighan**

Coding guidelines make everyone's life easier, even yours. **- Eric L. Michelsen**

Object Oriented Programming

This is a much used and abused term, with no definitive definition. The goal of Object Oriented Programming (OOP) is to allow reusable code that is clean and maintainable. The best definition I've seen of OOP is that it uses a language and approach with these properties:

- **User defined data types**, called **classes**, which (1) allow a single object (data entity) to have multiple data items, and (2) provide user-defined **methods** (functions and operators) for manipulating objects of that class.
- **Information hiding:** a class can define a public interface which hides the implementation details from the (client) code which uses the class.
- **Overloading:** the same named function or operator can be invoked on multiple data types, including both built-in and user-defined types. The language chooses which of the same-named functions to invoke based on the data types of its arguments.

- **Inheritance:** new data types can be created by extending existing data types. The **derived class** inherits all the data and methods of the **base class**, but can add data, and override or overload any methods it chooses with its own, more specialized versions.
- **Polymorphism:** this is more than just overloading [ASU 1986 p344]. Polymorphism allows derived-class objects to be handled by (often older) code which only knows about the base class (i.e., which does not even know of the existence of the derived class.) Even though the application code knows nothing of the derived class, the data object itself provides proper specialized methods for itself.

In C++, polymorphism is implemented with virtual functions. [There are arguments about the distinctions between "information hiding", "encapsulation", and "abstraction" that I don't think are worth having.]

OOP does not have to be a new "paradigm." It is usually more effective to make OOP an improvement on the good software engineering practices you already use.

Take it one step at a time. Don't redesign the world for no reason.

Further Musings on Coding and OOP

I have a moderated view of coding. I think the difference between "procedural" and "object-oriented" code is much less than many people do. *All* code is procedures for manipulating data. And breaking up processing into hierarchical chunks isn't OO, it's just modularizing. Good code has always done that, long before OO was thought of.

Object oriented code can still be terrible.

There is a much bigger difference between commercial code and scientific code than there is between so-called "procedural" code and OO code. A lot of scientific code has no big future. All the work I did for my dissertation is dead, and will never live again. I'm completely comfortable having used C++ as a simple tool to get my dissertation job done, using only a few small classes, and using all the global variables that I did. That was the nature of the hammer I needed in order to graduate. Anything else would have been a complete waste of my time.

That said, my code is heavily commented, and anyone who might inherit it will be much better off than most physicists.

The key to productivity is making conscious, informed trade-offs, and documenting them in the comments.

I'm not anti-OO, it's just that I know that OO isn't new. The modern tools have more benefits in commercial code than scientific code. Nonetheless, it *would* improve efficiency if more scientists use OO when appropriate, but OO zealotry actually discourages that. The "all or nothing" screed means most coders must choose "nothing," because they don't have time for "all." My credo is "Do what makes sense," which usually means starting with a few small classes. As need demands, do more. Baby steps.

Coding is an art. That's why I moderate the absolutist views of zealots.

Beautiful Irony: An Object Lesson in Orientation

Here is a beautifully ironic example of the failure of modern coding dogma (adapted from https://cplusplus.com/forum/general/21215/ retrieved 9/2022). Their attempt to be "modern" leads to a bug in their code. The web page suggests two implementations of the (nonstandard) strupr() function, which converts all letters in the given string to uppercase. The first is written in C, and the second in C++:

C:
```
#include <ctype.h>

char* strupr( char* s )
{
  char* p = s;
  while( (*p = toupper(*p)) ) ++p;
  return s;
}
```

C++:
```
#include <algorithm>
#include <cctype>
#include <functional>

std::string& stoupper( const std::string& s )
{
  std::string result( s );
  std::transform
    ( s.begin(),
      s.end(),
      result.begin(),
      std::ptr_fun <int, int> ( std::toupper )
    );
  return result;
}
```

Which one is better? Unequivocally, the C version is better, because the C++ version *doesn't work* (it doesn't even compile for me). The complexity of the C++ version obscures a beginner's mistake: you can't return a reference to a local variable (because it does not exist beyond the life of the function).

These implementations clearly illustrate the problem with modern programming "style": even if the second implementation worked, it is vastly harder to read, and requires vastly more background knowledge to understand. And for what? Some sort of "generality"? In a function that has no reasonable application to any data type other than letters? Recall our discussion of overkill. If this is what passes for "good" code today, then I'm glad that I no longer make my living as a software engineer.

There's more to say on this, but I won't. However, for fun, here's a better C++ code, which acts in-place, and therefore is simple, reentrant, and works:

```
#include <string>
#include <cctype>
using std::string;

   string& strupr( string& s )  // makes 's' uppercase in-place
{
  int len = s.length()
  for(int i = 0;  i < len;  ++i)  s[i] = toupper(s[i])
  return s;
}
```

Libraries to Modules to Classes: the Growth of Program Structure

When learning programming (which is much more than learning to code), it is helpful to understand some organization concepts. To understand these organization concepts, I relate *my perception of* the genesis and evolution of program organization, from single-purpose code, to libraries, to modules, and then to classes. There are no precise, universally-accepted definitions of any of these things, so I give reasonable descriptions based on my experience. (Sometimes these concepts are confused in the modern rush to make everything a class.) I've been programming for 51 years, in a wide variety of contexts, and that affords some insight into the reasons for (and failures of) many programming "recommendations." The first sections assume only some familiarity with coding.

Libraries: Bunches of Functions

In the beginning, there was a computational job to do: an application. People created code, and it was formless [`void` came much later]. [Genesis 1:1-3]

Then there was another job to do, and it had some similarities to the first job. So people copied and adapted the code. And it worked. Their code was naked, and the people felt no shame. [Op. cit. 1:25; further references omitted for brevity.]

Then there was a slew of jobs, and people got tired of constantly copying and adapting code. So programmers said, "Let the functions under the sky be gathered to one place, and given clean interfaces." Then future programs didn't have to mess with, or even understand, how they worked (Figure 16.1a). This "library" was a set of simple, general purpose functions like sqrt(), sin(), cos(), etc. Each was "pure": it operated only on its arguments, and had no side effects. And programmers saw that it was good.

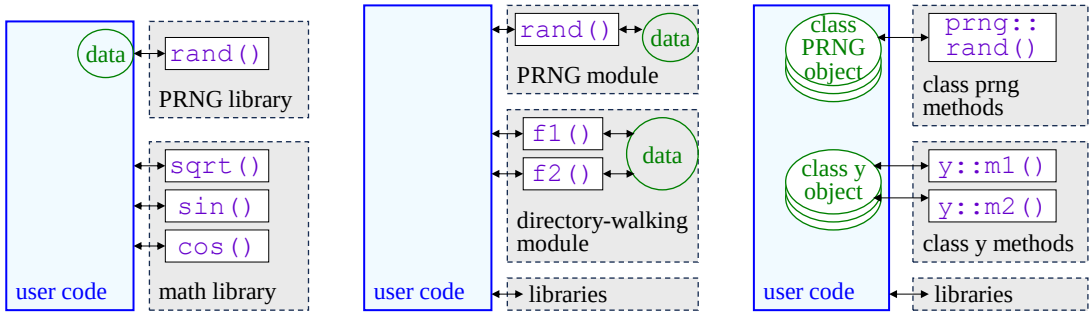


Figure 16.1 (a) Libraries. (b) Modules. (c) Classes.

More needs arose. As an example, consider a pseudo-random number generator (PRNG). It has state (memory): the new value of the PRNG is computed from the previous value. Someone has to remember the previous value when calling for a new one. At first, to keep the prng() function pure, the burden of maintaining state fell to the user: each time you call for a new PRN, you must pass in the previous value. This worked (sort of), and it was pure.

But for a complex program, the burden of carrying around that state becomes significant. Your program may call for random numbers in many diverse places, throughout the tree of calls. How does code in one part of the program know what the previous value of the PRNG was? This is hard. Wouldn't it be nice if the PRNG could keep track on its own? Why should *my* code have to do that? It gets me involved with things I may not understand, and don't want to. I just want random numbers. Isn't that what a library is for?

Yes. Yes it is. So programmers said, "Let us separate the library from the client": the PRNG library maintains its own state. The library now requires persistent memory, so it is no longer pure: the returned function value now depends not only on the arguments you give it (often none at all), but also on its past history.

Modules: Libraries + Data

But the customers teem with applications, each with needs according to their kinds. And programmers said, "Be fruitful and increase in number, for we profit from it." And "libraries" became more complicated, according to their kinds. And they required more state (private local data, Figure 16.1b). For example, I wrote a set of directory walking functions in 1990, which have continuously evolved (like the living creatures), and which I continue to use every day, 33 years later.

But now, a "library" may be complicated, with lots of state (private local data), and its functions are tightly interdependent. And it has a fairly specific purpose, rather than a bunch of general purpose functions. It's really different than a library: it's a "module." Of course, we still have simple, old-fashioned libraries, as well.

One feature of a module is that it has clearly-defined "interfaces:" the set of functions (essentially services) that the module provides. The module also has an implementation, which most users can ignore. And in fact, all users should rely only on the interfaces, and not the internals, because future maintenance of the module can change its internals, but not its interfaces. That's the whole point of separating interfaces from implementations.

And programmers saw all that they had made, and it was very good.

Classes: Like Modules, Only More

There is often a need to make PRN sequences that are repeatable: when performing simulations, we'd like to be able to run them again, and get comparable results. So a PRNG has a way to "initialize" it, from whence it always produces the same sequence. This quickly leads to a need to have two separate PRN sequences running concurrently, each repeatable, but independent from each other. With a "library" or "module" that maintains its own private state, this is not possible. Originally, the module had only one set of state data. But the customers said, "We want multiples instances of the module." (Similarly, customers might want to walk multiple directory trees independently and concurrently, but the module only supports one at a time.)

So the programmers said, "It is not good for the module to be alone. We will make a helper suitable for it." They collected all the module's private data, and put it into a data structure (a "struct" of "data members"). They told the customers, "Use this struct, but do not use the knowledge of its data members, for when you use it you will certainly die." In other words, the customer is again responsible for providing the memory to hold the module's state, but the customer doesn't need to know anything about that state (module private data). Now the customer can define many "instances" of the module state, and call on the module functions to separately process those instances. These instances of the module state are now called "objects," and the module functions are now called "methods." The caller must pass the struct to each method on every call, so the methods can update the module state as needed by the module. For example:

```
struct prng { ... provided by the module implementation ...};
prng seq1; // user creates struct variable (object) to hold module data
prng seq2; // user creates another instance for a 2nd set of module data
  ...
   prng_doit(&seq1);     // user calls method, passes state in & out
     ...
   prng_doit(&seq2);     // same function, but does not affect seq1 in any way
```

The caller (ideally) knows nothing about the module state, or how the module functions work. She knows only what the methods do, and what public information they take in, and put out. In other words, she knows only the module "interface."

The objects (structs) and methods make possible multiple instances of the module. But it is still tedious to have to explicitly pass the structs in and out of the methods.

Object-oriented languages allow object data passing to happen automatically, thus slightly simplifying the user burden. A "class" combines the module data structure with a list of all the methods that can operate on it. With a class, the above code looks like this:

```
class prng { ... provided by the class implementation ...};
prng seq1; // user creates class object to hold class data
prng seq2; // user creates another instance object for a 2nd set of data
  ...
seq1.doit();       // user calls method, object data is automatically passed
  ...
seq2.doit();       // same function, does not affect seq1 in any way
```

So far, this may not seem like much of an improvement. This class code isn't noticeably simpler than the "module" version. But classes provide more benefits that truly improve over multi-instance modules. For one, a class is usually coded within clearly-defined boundaries of the source code. This allows you to be explicit about what code is in the class, and what is not. And it encourages you to clearly define your interfaces.

Full-featured classes provide ways to make some data members public (if you want), so the user is allowed to look at them, and maybe even modify them (without certain death). They also provide for information hiding (e.g., private methods), inheritance, overloading, polymorphism, and constructors and destructors, all of which are described in the section on "Object Oriented Programming."

We digress briefly to consider the PRNG: we noted above that with a PRNG class, the user is again responsible for providing the object (aka memory) in which the methods maintain state. If we need many calls to PRNG, all throughout the call tree, then we have to pass the PRNG object all over the code. (Heavens! This seems to call for a global variable! Be damned!)

Note that "classes" pretty much replace "modules": a class does everything a module can, and just as simply. However, we will always also have simple, old-fashioned libraries.

Declassified: Other Uses of Classes

So (my experience of) the history of classes says that they began as multiple-instance modules. But now that we *have* classes, we can use them for other things (including some that God never intended). Here are some examples of other uses.

Classes as Single-Instance Modules

First, let us return to our historic single-instance module. Not all languages support such a concept. C does not explicitly support it, but (I believe) stumbled onto it by accident. In C, you can make a module by putting its code and data in a separate file. You can then declare each data member to be either public (global) or private (confined to the module), and similarly, the methods can be either public or private. You usually don't want to put more than one module in a source file, because then they can sneak peeks into each other's internals.

In an object-oriented language (i.e., one that supports classes), you can use a class as a single-instance module: just declare one instance of the class object. You can put multiple classes into a single source file, because classes prevent outside code from peeking at their internals. And you get all the other benefits of classes (noted above). (Multiple classes in a single file might seem "bad", but are often useful. I prefer to have fewer large files than zillions of small files, because it's easier to navigate among them.)

Classes As Namespaces

Namespaces are a way to avoid collisions between names from different libraries (or from code, in general). Suppose a user defines a function, say "dmod()". At the time, it was not the name of a standard Fortran function. But then a new standard came out, and added some new standard functions, including one named "dmod". Now there's a collision: when the new compiler sees "dmod", it thinks that's the standard function, but the user code intends it to be the user function.

Collisions can be avoided by using a namespace for either the user identifiers, or the language system, or both. In C++, the standard functions are put in a standard namespace named 'std'. Then you can explicitly tell the compiler you're calling a standard function like this:

```
x = std::some_function(); // std:: means the language standard
```

Of course, doing this with all your functions really clutters up your code, making it hard to read. (Despite this, some people recommend putting "std::" all over everywhere. I'll never do that.)

Back to classes: you *can* put all your variables in a class (called, say, '`user`'), and define no methods. Then declare one instance of that class (call it '`ud`'). Then refer to all your variables as `ud.var1`, `ud.var2`, etc. Again, horribly clutters your code (and I'll never do it). But some people recommend this. Alternatively, you *can* put all your *functions* in a class, and define no data. Then refer to all your functions as `user::fun1`, `user::fun2`, etc. I guess this is OK in languages that have no explicit namespace, but in C++, there *is* an explicit namespace. So if you want a namespace, use a namespace, and not a class. It tells the reader exactly what you are really doing.

Classes As Things That I Don't Like: the State-Pattern

This subsection requires understanding subclasses and polymorphism, described elsewhere in *Funky Mathematical Physics Concepts*.

Books have been written about recurring patterns in programming, and how to code for them with classes. Often, this is helpful. But not always. For example, state-machines. We don't have room here to give a full treatment of state machines, so we hope a simple example will be enough to make our points. We show here that state-machines built from classes/subclasses are perhaps an interesting novelty, but not a desirable design pattern.

A (simplified) phone is a device that can be in one of a few **states**: idle, alerting (ringing), talking, dialing. It changes (transits) from one state to another based on **events** that happen: incoming call, user requests "answer", remote disconnect (hang-up), local disconnect. We can make a "state table" showing, for each state the phone may be in, how the phone changes to a new state as a result of each event (Figure 16.2). The phone is a state-machine. (We've simplified it, and yes, we know there's no way shown to enter "dialing" state.)

As is often the case, the state-machine ignores invalid/impossible events.

[Usually, each entry in the state table also includes some action the phone takes before changing to the new state, but for simplicity, we omit that here.]

event ↓ \ state→	idle	alerting	talking	dialing
incoming call	alerting	invalid	invalid	invalid
answer	idle	talking	invalid	invalid
remote disconnect	invalid	idle	idle	invalid
local disconnect	idle	idle	idle	idle

Figure 16.2 Given the current state, and an event, the table gives the new state of the phone.

Real examples of state-machines: (1) for my dissertation research, I designed and coded a real-time hardware control system with about 12 states, and each state had about 3 substates, totaling ~36 unique states. (2) For years, I maintained a data communication protocol state machine with about 15 states and 12 events.

Back to the phone example: one simple way to code this is with a direct 2D table. [The table can include pointers to action functions, as needed.]

Another approach is a switch/case statement. I think this is the most conceptually direct approach (which isn't necessarily the "best"). Here's a partial example: the state is held in an enumerator 'state', and the event to process is given by an enumerator 'event':

```
switch(state)
{ case idle:
            if(event == INCOMING)  state = alerting;
            break;
  case alerting:
            if(event == ANSWER)  state = talking;
            break;
  case talking: ... etc.
  case dialing: ... etc.
}
```

Note that we simply don't code for invalid events; they are effortless.

Instead of this simple, compact, easily maintainable, efficient code, some people recommend using the "state-pattern." First, we define an abstract base class (never instantiated) that defines a method for each of the possible events:


```
class phone
{ virtual void Incoming(void) {};      // default does nothing
  virtual void Answer(void) {}; // all these can be overridden ...
  virtual void Remote_disc(void) {};// in the subclasses
  virtual void Local_disc(void) {};
};
```

This sets our interface to the state machine, and provides do-nothing default methods (which often allow enormous simplification). Next, for each of the 4 states, we define a subclass, using polymorphism for each to define its own set of 4 methods for processing the 4 possible events. For example, for the 'idle' state:

```
class phone_idle: phone  // subclass derived from class 'phone'
{ void Incoming(void)
  { ... code ...
  };
  // use default (base-class) void Answer(void)
  // use default (base-class) void Remote_disc(void);
  // use default (base-class) void Local_disc(void);
};
```

The 3 other subclasses (for the 3 other states) and other method bodies (definitions) for this are too long to write down here (this should make us nervous). However, do-nothing events can be omitted (as shown), relying on the base-class do-nothing method definitions. From Figure 16.2, we must define only 9 non-default methods

Next, we declare 4 objects, one for each state/subclass. Our state variable is a pointer (in C++) to one of these 4 objects. Note that these objects *have no data*. They're really just tables of functions. (We could have coded them directly as such, and avoided all the subclass overhead). And finally, we *still* need a switch statement with a 'case' for each possible event.

That's a lot of methods/functions, with a *lot* of coding overhead. For my dissertation research machine, the "state-pattern" approach requires $(12 + 1)(3 + 1) = 52$ class definitions, before writing a single line of functioning code. If each state/substate responded to just 2 events, that would be an *additional* $12 \cdot 3 \cdot 2 = 72$ method definitions. For the data communication system, it's up to $15 \cdot 12 = 180$ method definitions, but with default event ignoring, more likely 40 or so. That's a lot of methods.

Code Maintenance: Suppose we need to add a new event to the machine (say, user presses 'send' after dialing). With the switch statement, all our code is in one simple place, and we just scan down the cases, adding code for 'send' only where we need to, which is *only* in the 'dialing' state. Pretty simple.

With the state-pattern approach, we must add a new method declaration/definition to the base class. Then we scan all 4 subclasses, adding a new method declaration/definition to any state subclass that needs it. The code for the state-machine is scattered across the subclasses, with lots of intervening declarations (noise) between the actual processing code.

If instead of a new event, we need to add a new state, the overheads for each approach are similar to the above: easy for a switch statement, and more tedious for the state-pattern.

Let's consider some statements we've seen about the state-pattern:

Concession: "the state-pattern (polymorphic subclasses) requires a lot of subclasses." True. Sometimes an impractical number (as in my dissertation research code, or the communication protocol).

Claim: "polymorphic subclasses can be more efficient than other methods." Flatly false. A simple implementation requires 3 lookups for a subclass virtual function call, and only 2 for a state table.

Claim: "the state-pattern collects state-specific behavior in one place, and separates the state code." True. So does a switch statement, and it also keeps all the different states near each other.

Claim: "adding a new state or event is easier with state-pattern than with a switch statement." False, as shown above.

Claim: "having separate subclasses is better than having long conditionals." False. At least with long conditionals, I know where all the code is, unlike with subclasses (which start to resemble object-oriented spaghetti code).

Claim: "using a pointer to a polymorphic object defines the state more explicitly than a state variable." False, and bizarrely so. The polymorphic state variable is a *pointer*, which is much *less* explicit than an enumerator (while debugging, try figuring out what state you're in from a pointer!).

Subclaim: "if an enumerated state is maintained as a combination of enumerators, then a bug might create inconsistencies of invalid combinations; state-pattern states/subclasses are a single list of all valid combinations, so can't be inconsistent." Nonsense. If it were feasible to combine all state-variables into a single enumerator, the state-variable implementor could (and probably would) do it.

There are many more false claims, too many to continue refuting them.

Conclusion: The class-based state-machine is not a desirable design pattern; instead, state-table and case-statement methods are more simple, efficient, and maintainable. I am startled at the breadth and depth of misinformation about the state-pattern. Sometimes, design patterns are akin to, "I love my hammer, so I'm going to use it for everything, even swatting flies. I love classes, so I'll find a way to use them for everything, even if it doesn't really make sense." Is there ever a case where subclasses and polymorphism are simple and efficient ways to implement a state-machine? Maybe. But I've never seen one.

The Best of Times, the Worst of Times: Run-time Efficiency

We give here some ways to speed up *general* computations, using matrices as *examples*. The principles apply to almost any computation performed over a large amount of data. This is *not* about programming styles or paradigms; it's about how to use language features and machine architecture to speed typical scientific code.

> For the vast majority of programs, run time is so short that it doesn't matter how efficient it is; clarity and simplicity are more important than speed.

In rare cases, time is a concern. For example, in my dissertation I did numerical experiments on both simulated and real data. I often ran an hour or two of computations per day, with me examining intermediate results along the way. I could not have afforded even a 3x slow down; I never would have graduated. Another example: supercomputer programmers are required to use computer time efficiently, and usually must include hardware metrics in the proposal for supercomputer time. Programs that fail efficiency standards are denied time. Supercomputer programmers must understand the concepts in this section.

For some simple examples, we show how to *easily* cut your execution times to 1/3 of original. We also show that execution times change in surprising ways, such as *adding* operations to *reduce* run time. This section assumes knowledge of computer programming with simple classes (the beginning of object oriented programming), and we use C++ as the example language, including some distinctions between C++98 and C++11.

Run time optimization is a huge topic, so we can only touch on some basics. The main point here is:

> For large-data computations, memory management is the key to fast performance. Sometimes, easy changes can yield huge benefits.

We proceed along these lines:

- A simple C++ class for matrix addition. We give run times for this implementation (the worst of times).
- A simple improvement greatly improves execution times (the best of times).
- We try another expected improvement, but things are not as expected.
- We describe the general operation of "memory cache" (pronounced "cash").

- Moving on to matrix multiplication, we find that our previous tricks don't work.
- However, due to the cache, adding *more* operations greatly improves the execution times.

We give actual code listings for the examples, but you can ignore them, and still understand the concepts.

Example Using Matrix Addition

One concept in speeding up large-object methods is to avoid C++'s hidden copy operations. However:

> Computer memory access is tricky, so things aren't always what you'd expect. Nonetheless, we can be efficient, even without details of the computer hardware.

The tricks are due to computer hardware called RAM "cache," whose general principles we describe later, but whose details are beyond our scope.

First, here is a simple C++ class for matrix creation, destruction, and addition. (For simplicity, our sample code has no error checking; real code, of course, does. In this case, we literally don't want reality to interfere with science.) The class data for a matrix are the number of rows, the number of columns, and a pointer to the matrix elements (data block).

```
typedef     double T;           // matrix elements are double precision

class ILmatrix     // 2D matrix
{ public:
    int     nr, nc;                     // # rows & columns
    T       *db;                        // pointer to data

    ILmatrix(int r, int c);             // create matrix of given size
    ILmatrix(const ILmatrix &b);        // copy constructor
    ~ILmatrix();                        // destructor

    // The following allows 2-index subscripting as: a[i][j]
    T * operator [](int r) const {return db + r*nc;};      // subscripting

    ILmatrix & operator =(const ILmatrix& b);       // assignment
    ILmatrix operator +(const ILmatrix& b) const;   // matrix add
};
```

The matrix elements are indexed starting from 0, i.e. the top-left corner of matrix 'a' is referenced as 'a[0][0]'. Following the data are the minimum set of methods (procedures) for matrix addition. Internally, the pointer 'db' points to the matrix elements (data block). The subscripting operator finds a linear array element as (row)(#columns) + column. Here is the code to create, copy, and destroy matrices:

```
// create matrix of given size (constructor)
ILmatrix::ILmatrix(int r, int c) : nr(r), nc(c)     // set nr & nc here
{
    db = new T[nr*nc];                  // allocate data block
} // ILmatrix(r, c)

// copy a matrix (copy constructor)
ILmatrix::ILmatrix(const ILmatrix & b)
{   int     r,c;

    nr = b.nr, nc = b.nc;               // matrix dimensions
    if(b.db)
    {   db = new T[nr*nc];              // allocate data block
        for(r = 0;  r < nr;  ++r)       // copy the data
          for(c = 0;  c < nc;  ++c)
             (*this)[r][c] = b[r][c];
    }
} // copy constructor
```



```
// destructor
ILmatrix::~ILmatrix()
{   if(db)  {delete[] db;}      // free existing data
    nr = nc = 0, db = 0;        // mark it empty
}

// assignment operator
ILmatrix & ILmatrix::operator =(const ILmatrix& b)
{   int     r, c;

     for(r = 0;  r < nr;  ++r)  // copy the data
         for(c = 0;  c < nc;  ++c)
            (*this)[r][c] = b[r][c];
    return *this;
} // operator =()
```

The good stuff: With the tedious preliminaries done, we now implement the simplest matrix addition method. It adds two matrices element by element, and returns the result as a new matrix:

```
// matrix addition to temporary
ILmatrix ILmatrix::operator +(const ILmatrix& b) const
{
    int             r, c;
    ILmatrix        result(nr, nc);     // create temporary for result

    for (r=0;  r < nr;  ++r)
      for (c=0;  c < nc;  ++c)
         result[r][c] = (*this)[r][c] + b[r][c];
    return result;        // invokes copy constructor!
} // operator +()
```

How long does this simple code take? To test it, we standardize on 300 × 300 and 400 x 400 matrix sizes, each on two different computers: computer-1 is a circa 2000 Compaq Workstation W6000 with a 1.7 GHz Xeon. Computer-2 is a circa 2003 Gateway Solo 200 ARC laptop with a 2.4 GHz CPU. We time 100 matrix additions, e.g.:

```
    int             n = 300;            // matrix dimension
    ILmatrix        a(n,n), b(n,n), d(n,n);

// Addition test
    d = a + b;                  // prime memory caches
    cpustamp("start matrix addition\n");
    for(i = 0;  i < 100;  ++i)
       d = a + b;
    cpustamp("end matrix addition\n");
```

> With modern operating systems, you may have to run your code several times before the execution times stabilize.

This may be due to internal operations of allocating memory, and flushing data to disk.

We find that, on computer-1, it takes ~1.36 ± 0.10 s to execute 100 simple matrix additions (see table at end of this section). Wow, that seems like a long time. Each addition is 90,000 floating point adds; 100 additions is 9 million operations. Our 2.4 GHz machine should execute 2.4 additions per ns, which totals only about 4 ms. Where's all the time going? C++ has a major flaw. Though it was pretty easy to create our matrix class:

> C++ copies your data *twice* in a simple class operation on two values.

So besides our actual matrix addition, C++ copies the result *twice* before it reaches the matrix 'd'. The first copy happens at the 'return result' statement in our matrix addition function. Since the variable 'result' will be destroyed (go out of scope) when the function returns, C++ must copy it to a temporary variable in the main program. Notice that the C++ language has no way to tell the addition function that the result is headed for the matrix 'd'. So the addition function has no choice but to copy it into a temporary matrix, created by

the compiler and *hidden* from programmers. The second copy is when the temporary matrix is assigned to the matrix 'd'. Each copy operation copies 90,000 8-byte double-precision numbers, ~720k bytes. That's a lot of copying.

Optimization 1: What can we do about this? The simplest improvement is to make our copies more efficient. Instead of writing our own loops to copy data, we can call the library function memcpy( ), which is specifically optimized for copying blocks of data. Our copy constructor is now both simpler *and* faster:

```
ILmatrix::ILmatrix(const ILmatrix & b)
{   int     r,c;

    nr = b.nr, nc = b.nc;                    // matrix dimensions
    if(b.db)
    {   db = new T[nr*nc];                   // allocate data block
        memcpy(db, b.db, sizeof(T)*nr*nc);   // copy the data
    }
} // copy constructor
```

Similarly for the assignment operator. This new code takes 0.98 ± 0.10 s, 28 % better than the original code.

> Use built-in and professional libraries whenever possible.
> They're already optimized and debugged.

Not bad for such a simple change, but still bad: we still have two needless copies going on.

Optimization 2: For the next improvement, we note that C++ can pass *two* class operands to a class operator function, but not three. Therefore, if we do one copy ourselves, we can then perform the addition "in place," and avoid the second copy. For example:

```
// Faster code to implement d = a + b:
d = a;              // the one and only copy operation
d += b;             // '+=' adds 'b' to the current value of 'd'
```

We can simplify this main code to a single line as:

```
(d = a) += b;
```

To implement this code, we added a "+=" operator function to our class, and it must return by *reference*, not by copy:

```
// matrix addition in-place
ILmatrix &  ILmatrix::operator +=(const ILmatrix & b)
{
    int             r, c;

    for (r = 0;  r < nr;  ++r)
      for (c = 0;  c < b.nc;  ++c)
          (*this)[r][c] += b[r][c];

    return *this;         // returns by reference, NO copy!
}
```

This code runs in 0.45 ± 0.02 s, or 1/3 the original time! This C++98 price, though, is somewhat uglier user code.

Move semantics: C++11 defines a new feature called "move semantics," that is designed to reduce the burden of the unnecessary copies, and therefore can avoid the code ugliness above. Note that it still retains the unnecessary copies, but it *sometimes* allows them to be less burdensome. Essentially, "move semantics" are only helpful if the class includes dynamically allocated memory. If you simply have large objects with fixed memory, "move semantics" do nothing, and you're stuck with the ugly method above.

"Move semantics" requires us to define two more "class special" methods: a "move constructor," and a "move assignment" method. The key element of move semantics is that the two "move" methods are called only when the right hand operand is about to be destructed, so there is no need to preserve its data. You can simply "hand-off" the data block from the right-operand to the destination object. The right-operand can be

left empty, because it's about to be destructed anyway. We leave the details to C++11 tutorials. (We propose adding a C++ keyword "that", that would very simply avoid all the copies all the time, without requiring any "move semantics," and without any new "class special" functions.)

Attempted optimization 3: Perhaps we can do even better. Instead of using operator functions, which are limited to only two matrix arguments, we can write our own addition function, *with any arguments we want*. We can then eliminate *all* the needless copies. The main code is now:

```
mat_add(d, a, b);          // add a + b, putting result in 'd'
```

Requiring the new function "mat_add( )":

```
// matrix addition to new matrix: d = a + b
ILmatrix & mat_add(ILmatrix & d, const ILmatrix & a, const ILmatrix & b)
{
    int            r, c;

    for (r = 0;  r < d.nr;  ++r)
      for (c = 0;  c < d.nc;  ++c)
          d[r][c] = a[r][c] + b[r][c];

    return d;                    // returned by reference, NO copy constructor
} // mat_add()
```

This runs in 0.49 ± 0.02 s, slightly *worse* than the one-copy version. It's also even uglier for users than the previous version. How can this be?

> Memory access, including data copying, is dominated by the effects of a complex piece of hardware called "memory cache."

There are hundreds of different variations of cache designs, and even if you know the exact design, you can rarely predict its exact effect on real code. We will describe cache shortly, but there is no feasible way to know exactly why the zero-copy code is slower than one-copy. This result also held true for the 400 × 400 matrix on computer-1, and the 300 × 300 matrix on computer-2, but not the 400 × 400 matrix on computer-2. In the end, all we can do is try a few likely optimizations, and keep the ones that tend to perform well. More on this later.

Beware Leaving out a *single character* from your code can produce code that works, but runs over 2 times slower than it should. For example, in the function definition of mat_add(), if we leave out the "&" before argument 'a':

```
ILmatrix & mat_add(ILmatrix & d, const ILmatrix  a, const ILmatrix & b)
```

then the compiler passes 'a' to the function by copying it! This completely defeats our goal of zero copy. [Guess how I found this out.]

Also notice that the 'memcpy( )' optimization doesn't apply to this last method, since it has no copies at all.

Below is a summary of our matrix addition optimizations. The best performance was usually a single copy, with in-place addition. It is medium ugly. While there was a small performance discrepancy on computer-2 at 400 × 400, the zero-copy code is not worth the required additional ugliness.

	Computer-1 times (ms, ± ~ 100 ms)		Computer-2 times (ms, ± ~ 100 ms)	
Algorithm	**300 × 300**	**400 × 400**	**300 × 300**	**400 × 400**
d = a + b, loop copy	1360 ≡ 100 %	5900 ≡ 100 %	1130 ≡ 100 %	2180 ≡ 100 %
d = a + b, memcpy( )	985 = 72 %	4960 = 84 %	950 = 84 %	1793 = 82 %
(d = a) += b	**445 = 33 %**	**3850 = 65 %**	**330 = 29 %**	791 = 36 %
mat_add(d, a, b)	490 = 36 %	4400 = 75 %	371 = 33 %	**721 = 33 %**

Figure 16.3 Run times for matrix addition with various algorithms. Best performance is highlighted.

Memory Consumption vs. Run Time: Cache as Cache Can

In the old days, the claim was clear (but not the reality): the less memory you use, the slower your algorithm, and speeding your algorithm requires more memory.

The fallacy there is that most code is not well written. When you clean up code, you often create implementations that are more efficient in *both* memory and time. I have personally done this many times, even when revising my own code.

However, given reasonably efficient implementations, then with no memory cache (described below), one can usually speed a computation by using an algorithm that requires more memory. Conversely, an algorithm that uses less memory is usually slower.

But finally, since all modern computers are heavily affected by cache, we must take it into account. If you "exceed the cache", i.e. your algorithm repeatedly works through more memory than the cache can hold, you will suffer a dramatic slow-down in speed. In such a case, an algorithm that uses less memory may be faster: the algorithmic performance loss may be offset by the cache performance increase, possibly many times over.

Cache Value

Before about 1990, computations were slower than memory accesses. Therefore, we optimized by increasing memory use, and decreasing computations. Today, things are exactly reversed: a 2.5 GHz processor has a compute cycle time of 0.4 ns, but a main memory access takes around 40 ns.

> Modern CPUs (c. 2018) can compute about 100 times faster than they can access main memory. Therefore, the biggest factor in overall speed is efficient use of memory.

To help reduce the speed degradation of slow main memory, computers use **memory caches**: small memories that are very fast (Figure 16.4). A cache system is usually 3 levels of progressively bigger and slower caches. A typical main memory is 16 GB, while a typical cache (system) is 1-10 MB, or more than 1000x smaller. The CPU can often access level 1 (L1) cache memory as fast as it can compute, so L1 cache can be ~100x faster than main memory. There is huge variety in computer memory designs, so the description below is general. Behavior varies from machine to machine, sometimes greatly. Our data below demonstrate this.

The cache is invisible to program *function*, but is critical to program *speed*. The programmer usually does not have access to details about the cache, but she can use general cache knowledge to greatly reduce run time. (As noted earlier, supercomputer programmers are usually required to use special diagnostics to insure programs use cache efficiently.)

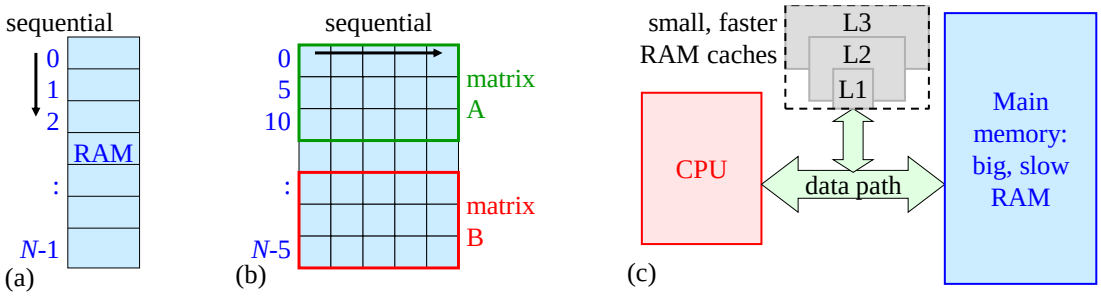


Figure 16.4 (a) Computer memory (RAM) is a linear array of bytes. (b) For convenience, we draw RAM as a 2D array, of arbitrary width. We show sample matrix storage. (c) A fast memory cache system keeps a copy of recently used memory locations, so they can be quickly used again.

The cache does two things (Figure 16.4):

1. Cache remembers recently used memory values, so that if the CPU requests any of them again, the cache provides the value quickly, and the slow main memory access does not happen.
2. Cache "looks ahead" to fetch memory values immediately following the one just used, before the CPU might request it. If the CPU in fact later requests the next sequential memory location, the cache provides the value quickly, having already fetched it from slow main memory.

The cache is small, and eventually fills up. Then, when the CPU requests new data, the cache must discard old data, and replace it with the new. Therefore, if the program jumps around memory a lot, the benefits of the cache are reduced. If a program works repeatedly over a small region of memory (say, a few hundred-k bytes), the benefits of cache increase. Typically, cache can follow at least four separate regions of memory concurrently. This means you can interleave accesses to four different regions of memory, and still retain the benefits of cache. Therefore, we have three simple rules for efficient memory use:

> For efficient memory use: (1) access memory sequentially, or in small steps,
> (2) reuse values as much as possible in the shortest time, and
> (3) access few memory regions concurrently, preferably no more than four.

Cache Benefits

We can now understand some of our timing data given above. We see that the one-copy algorithm unexpectedly takes less time than the zero-copy algorithm. The one-copy algorithm accesses only two memory regions at a time: first matrix 'a' and 'd' for the copy, then matrix 'b' and 'd' for the add. The zero-copy algorithm accesses *three* regions at a time: 'a', 'b', and 'd'. This is probably reducing cache efficiency. Recall that the CPU is also fetching instructions (the program) concurrently with the data, which can be a fourth region. Exact program layout in memory is virtually impossible to know. Also, not all caches support 4-region concurrent access. The newer machine, computer-2, probably has a better cache, and the one- and zero-copy algorithms perform very similarly.

Order matters: Here's a new question for matrix addition: the code given earlier loops over rows in the outer loop, and over columns in the inner loop. What if we reversed them, and looped over columns on the outside, and rows on the inside? The result is 65% *longer* run time, on both machines. Here's why: the matrices are stored by rows, i.e. each row is consecutive memory locations (Figure 16.4b). Looping over columns on the inside accesses memory sequentially, taking advantage of cache look-ahead. When reversed, the program jumps from row to row on the inside, giving up any benefit from look-ahead. The cost is quite substantial. Sequential memory access speeds code on almost every machine. The code behavior of making mostly nearby memory references is called **reference locality** (aka "data locality").

Caution Fortran stores arrays in the opposite order from C and C++. In Fortran, the first index is cycled most rapidly, so you should code with the outer loop on the second index, and the inner loop on the first index. E.g.,

```
DO C = 1, N
  DO R = 1, N
```



```
    A(R, C) = blah blah ...
  ENDDO
ENDDO
```

Scaling behavior: Matrix addition is an $O(N^2)$ operation, so increasing from 300 × 300 to 400 × 400 increase the computations by a factor of 1.8. On the older computer-1, the runtime penalty is much larger, between 4.5x and 9x slower. On the newer computer-2, the difference is much closer to the expected: between 1.8x and 2.2x slower. This discrepancy likely due to cache size. A 300 × 300 double precision matrix takes 720k bytes, or under a MB. A 400 × 400 matrix takes 1280k bytes, just *over* one MB. It could be that on computer-1, with the smaller matrix, a whole matrix or two fits in cache, but with the large matrix, cache is overflowed, and more (slow) main memory accesses are needed. The newer computer probably has bigger caches, and may fit both sized matrices fully in cache.

Cache Withdrawal: Making the Most of Reference Locality

We now show that the above tricks don't work well for, say, large-matrix multiplication, but a different trick cuts multiplication run time dramatically. This example uses matrix multiplication to illustrate the importance of reference locality, but you can apply this principle to any code that accesses lots of memory. (In general, you should *not* write your own matrix multiplication code; there are lots of free libraries to do it, that are optimized with the following method, as well as others. On the other hand, some "free" code isn't optimized at all, and will perform terribly.)

To start the example, we use a simple matrix multiply in the main code:

```
d = a * b;
```

The straightforward matrix multiply operator is this:

```
// Simple matrix multiply to temporary
ILmatrix ILmatrix::operator *(const ILmatrix & b) const
{
    int             r, c, k;
    ILmatrix        result(nr, b.nc);          // temporary for result
    T               sum;

    for(r = 0;  r < nr;  ++r)
    {  for(c = 0;  c < b.nc;  ++c)
       {   sum = 0.;
           for(k = 0;  k < nc;  ++k)  sum += (*this)[r][k] * b[k][c];
           result[r][c] = sum;
       }
    }
    return result;        // invokes copy constructor!
} // operator *()
```

While matrix addition is an $O(N^2)$ operation, matrix multiplication is $O(N^3)$. Multiplying two 300 x 300 matrices is about 54,000,000 floating point operations, which is *much* slower than addition. Timing the simple multiply routine, similarly to timing matrix addition but with only 5 multiplies, we find it takes 7.8 ± 0.1 s on computer-1.

First we try the tricks we already know to improve and avoid data copies: we started already with memcpy( ). For matrix multiply, there is no one-copy option: you can't multiply "in place" because you would overwrite the input with results when you still needed the input. In other words, there is no '*=' operator for matrix multiply. So we compare the two-copy and zero-copy algorithms as with addition, but this time the 4 trials show no measurable difference. Matrix multiply is so slow that the copy times are insignificant. Therefore, we choose the two-copy algorithm, which is easiest on the user. We certainly drop the ugly 3-argument mat_mult( ) function, which gives no benefit.

Now we'll improve our matrix multiply greatly, by *adding more work* to be done. The extra work will result in more efficient memory use, that pays off handsomely in reduced runtime. Notice that in matrix multiplication, for each element of the result, we access a row of the first matrix *a*, and a column of the second matrix *b*. But we learned from matrix addition that accessing a column is much slower than accessing

a row. And in matrix multiplication, we have to access the same column *N* times. Extra bad. If only we could access *both* matrices by rows!

Well, we can. We first make a temporary copy of matrix *b*, and transpose it. Now the columns of *b* become the *rows* of b^T. We perform the multiply as rows of *a* with *rows* of b^T. We've already seen that copy time is insignificant for multiplication, so the cost of one copy and one transpose (similar to a copy) is negligible. But the benefit of cache look-ahead is large. The transpose method reduces runtime by 30% to 50%.

Further thought reveals that we only need *one* column of *b* at a time. We can use it *N* times, and discard it. Then move on to the next column of *b*. This reduces memory usage, because we only need extra storage for one column of *b*, not for the whole transpose of *b*. It costs us nothing in operations, and reduces memory. That will probably help our cache performance. In fact, on computer-1, it cuts runtime by almost another factor of two, to about one third of the original runtime. It has little effect on computer-2. (It does require us to loop over columns of *b* on the outer loop, and rows of *a* on the inner loop, but that's no burden.)

Note that optimizations that at first were insignificant, say reducing runtime by 10%, may become significant after the runtime is cut by a factor of 3. That original 10% is now 30%, and may be worth doing.

	Computer-1 times (ms, ± ~ 100 ms)		Computer-2 times (ms, ± ~ 100 ms)	
Algorithm	**300 × 300**	**400 × 400**	**300 × 300**	**400 × 400**
d = a * b	7760 ≡ 100 %	18,260 ≡ 100 %	5348 ≡ 100 %	16,300 ≡ 100 %
mat_mult(d, a, b)	7720 = 99 %	18,170 = 100 %	5227 = 98 %	16,200 = 99 %
d = a * b, transpose 'b'	4580 = 59 %	12,700 = 70 %	**2900 = 54 %**	**7800 = 48%**
d = a * b, copy 'b' column	**2710 = 35 %**	**7875 = 43 %**	3100 = 58 %	8000 = 49 %

Figure 16.5 Run times for matrix multiplication with various algorithms. Best performing algorithms are highlighted

Data Structures for Efficient Cache Use

Here is a simple example of organizing data structures to improve cache efficiency. Consider a set of data points, each comprising 4 numbers, A, B, C, and D. In the old days, programmers often used "parallel arrays" to store the data:

```
double      a[10000], b[10000], c[10000], d[10000];
```

Suppose the processing requires looping through all the points, processing A, B, C, and D for each point. The parallel arrays above spread a single data point far across memory, the worst thing you can do for cache efficiency. Much better is:

```
struct dpoint
{   double  a;
    double  b;
    double  c;
    double  d;
}
dpoint data[10000];                                                    (16.1)
```

Though this takes many more lines of code, it is conceptually clearer, and keeps data from each point physically adjacent in memory. When looping through the data points, cache prefetch works well to reduce memory delays. However, it does clutter your code because a simple `a[i]` becomes the clumsy `data[i].a`.

Algorithms for Efficient Cache Use

Many algorithms require multiple passes over a list of data (e.g., mergesort for sorting a list of data in ascending order). Rather than simply iterate over the whole list multiple times, it is faster to stay within a cache line (probably 16 - 128 bytes) for multiple iterations, and then move on. The simple, slow way might be:

```
for(int pass = 1;  pass < NPASSES;  ++pass)
      for(int dx = 0;  dx < NDATA;  ++dx)
            (...process data[dx]);
```

A faster way might be:

```
for(int start = 0;  start < NDATA;  start += LINESIZE)
      for(int pass = 1;  pass < NPASSES;  ++pass)
            for(int dx = start;  dx < start + LINESIZE;  ++dx)
                  (... process data[dx]);
```

This might make you cringe, because you've replaced two nested loops with three. However, the cache locality of the 3-loop code is much better, and may well overcome the small increase in loop initialization. It helps if you know the cache line size of the target machine, and beware that running code on a machine with a smaller line size than you coded for will probably be worse than the simple 2-loop code.

Cache Optimization Summary

In the end, exact performance is nearly impossible to predict. However, general knowledge of cache, and following the three rules for efficient cache use (given above), will greatly improve your runtimes.

> Conflicts in memory among pieces of data, and within layout of instructions, cannot be precisely controlled. Sometimes even tiny changes in code will cross a threshold of cache, and cause huge changes in performance.

Remember the 90/10 rule: 90% of your CPU time is spent in only 10% of your code (often its closer to 95/5). Focus on the small fraction of code that matters; don't optimize code that doesn't run enough to matter.

Virtual Memory and Page Locality

Besides RAM cache, there is another kind of reference locality that affects computing speed: Virtual Memory pages. All modern computers divide main memory (RAM) into **pages**: contiguous blocks of memory, of typically 4 KB (Figure 16.6). Your application's memory is also divided up into such pages, though you don't (usually) need to know it. However, as with RAM cache, knowledge is power. To some extent, you can control,your page layout to improve speed.

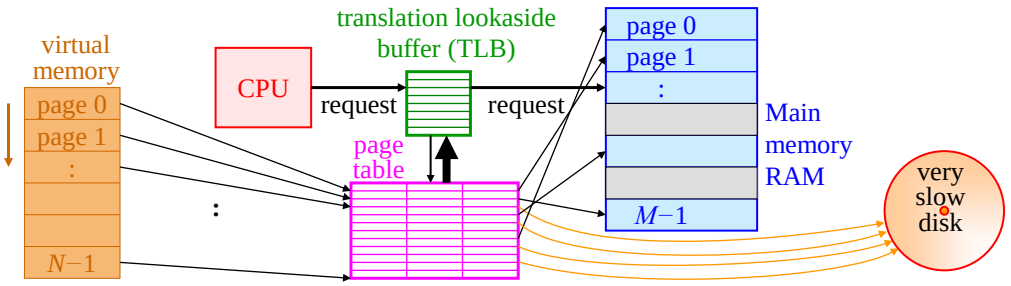


Figure 16.6 Virtual memory is the model of memory seen by an application. The actual storage for virtual memory is both physical RAM, and disk. The gray boxes in RAM are not part of the application's VM, but may be used by the OS or other applications. Cache is omitted for simplicity.

The basic idea behind improved page performance is the same as with RAM cache: keeping your data together on fewer pages allows faster execution. Specifically, there are two reasons page locality improves speed: improved Translation Lookaside Buffer hit rate, and improved page hit rate. Before describing these two aspects, we must have an overview of Virtual Memory (VM).

Virtual Memory: Virtual Memory exists to allow operating systems to efficiently allocate memory to applications, and to give the applications access to more memory than the system has physical RAM. An application on a modern system "sees" a model of memory where a contiguous region of memory occupies contiguous addresses. For example, an application may see 4,096,000 bytes (4 MB) of memory, with **virtual addresses** from 0 to 4,095,999. This is **virtual memory**: the memory model seen by the application. The actual storage for an application's VM is often scattered all over physical RAM. *Every time* an application requests access (read or write) to a virtual address, the CPU must lookup that address in a page table that tells the CPU where the data is *actually* stored. If the data is in physical RAM, access is very fast. The CPU uses the page table to "translate" the virtual address to its physical RAM address, and fulfills the request.

To make management feasible, VM is divided into **pages**, or contiguous block of memory, typically about 4 KB long (as of 2018). The **page table** then has one entry for each *page* of virtual memory. This keeps the page table a feasible size. The page table itself (usually) resides in physical RAM.

All of an application's (virtual) memory may not fit in RAM simultaneously, so the OS may keep some of it out on disk. The OS maintains the page table to either point to a virtual page's physical RAM address, or to note that the page is currently on disk. CPU access to the page table is a main RAM access, and can be slow, as long as 30 - 100 cycles (~10 - 30 ns). If this happened on every memory access, it would be crippling.

The saving grace is a Translation Lookaside Buffer (TLB), which is just a *cache* of page table translations. When the CPU needs to translate a virtual address to physical, it checks the TLB. If found, the CPU gets the physical address instantly, so the CPU runs at full speed. If the virtual page is *not* in the TLB, the CPU must consult the page table in RAM, and suffer the speed penalty. A TLB typically has 32 - 128 entries (as of 2018). A virtual address that is found in the TLB is called a TLB "hit;" a virtual address not found in the TLB is a TLB "miss" (similar to "cache hit", and "cache miss".) Typical TLB miss rates are 0.1% to 20% or more (for badly-written or uncooperative code).

Of course, after a TLB miss, the CPU puts the translation from the page table into the TLB, so the TLB is ready for a future access. To update the TLB, the CPU must overwrite an old entry with the new one.

Fault lines: There's a chance that when the CPU consults the page table, it will find that the data is not in physical RAM, but out on disk. This is called a **page-fault**. It requires a ~30 ms disk access, which is a *million* times slower than a TLB miss. This is perfectly normal, but slow. The ability to put virtual pages on disk allows an application to operate with more virtual memory than the system has physical RAM, because all the virtual memory is not simultaneously held in physical RAM. VM is fetched as needed.

On a page fault, the CPU takes an interrupt to notify the OS of the fault. The OS identifies a virtual page in physical RAM to "flush;" the OS writes (flushes) the data from the current page to disk, and reads the new page from disk into RAM. The OS updates the page table for both virtual pages, and the application can continue.

Be careful to distinguish a "page-fault" from a "segmentation fault" or other access violation. A page-fault is normal; a "seg-fault" is a fatal application bug where it tries to access non-existent virtual memory, usually forcing the OS to terminate the application.

Your mission, should you decide to accept it: The first way to minimize page misses is to organize data into related structures, as in code (16.1). Fortunately, this is the same advice as for cache locality, so if you care about speed, you've already done this.

Similarly, if you must access many different pages, it can be a huge benefit to access *within* just a few pages as much as possible, before moving on to the next set. This maximizes TLB and page-table hit rates (thus minimizing page faults). For example, if using a merge sort algorithm, sort all the values within each page first, and only then move on to merging from the pages.

To code specifically for such specific page locality, you must know some details about your system, especially the page size. This is readily available in documentation, and in some system calls, e.g., Unix sysconf(_SC_PAGESIZE). As of 2018, the page size on almost all machines is 4 KB.

Finally, using a simple sorted data array (or STL vector) for a lookup table can sometimes be much faster than an "optimized" associative container, because the array might be much smaller, and is certainly more localized in virtual address space. See [Mey 2001, *Item 23*, p100-3].

Considerations on Run-Time Efficiency and Languages

In our example in C++98, it is possible to implement matrices to pass pointers to data blocks, and do garbage collections, etc. This would make all operations zero-copy, but is *much* more work. Again, C++11 "move semantics" make this easier than before, but still add complexity, don't always solve the problem, and often only partially solve it.

Also, a minor extension to the C++ language would eliminate one of the copies. For operator methods, C++ could allow trinary functions: operand_1, operand_2, destination. (This would simply be a new allowed prototype for operator methods. I'm surprised no one has proposed this as a new C++ language feature.) I have read that some compilers are smart enough to optimize 'c = a + b' into passing 'c' as the destination object directly to the '+' method function, thus avoiding the assignment copy. This is only possible, though, if the compiler can guarantee that 'c' itself is not involved in the method function, which is not always possible. Also, there are nontrivial construction/destruction issues involved in such an optimization, and the compiler must insure the integrity of all this (e.g., copy construction is different than assignment). C++17 mandates some of these optimizations, called "copy elision" (that actually makes the language self-contradictory, because constructors are *sometimes* bypassed even if they have side-effects).

Note that Fortran by default has no copy of a function return object, and so avoids this C++ burden. Coders who *want* a copy (it happens) can easily add it themselves.

Some scientists believe Fortran (no longer all-caps) is still faster than C++, though I found little evidence to support that, in general (as of 2018). Fortran's array slicing is often easier to code and faster to run than C++ or Python. Fortran also has strong support for parallel processing.

Python is an interpreted byte-code language, like Java. Your source is compiled into a byte code, which is interpreted as it runs. It is inherently about 20-50x slower than languages compiled into direct machine instructions, like C, C++, and Fortran. Python is feasible for large data computations only when the computations are mostly standard functions like matrix multiply, inversion, etc. Standard functions exist as fast Python libraries that you can call (they are usually written in C). However, not all computations fit into standard mathematical routines. For example, in my dissertation, I wrote C++ data reduction code for a physics experiment, and independently, someone else wrote Python code to do the same. The C++ code ran 20x faster than Python. I could not have completed my dissertation, nor earned a PhD, in Python.

Use the right hammer for the job. Don't be a zealot.

17 Algorithms

Why Algorithms?

Every day, new problems arise, and some are solved. New algorithms solve existing problems. Understanding existing algorithms is often useful for devising new ones.

The Fast Fourier Transform (FFT)

The FFT is an efficient algorithm for evaluating a Discrete Fourier Transform (DFT). There are many sources that provide the FFT *algorithm*, but I've seen none that comprehensibly explain how or why it works. The FFT algorithm rests on two things: (1) divide and conquer, and (2) aliasing.

This section assumes you've seen a DFT (p188), and understand aliasing (p185).

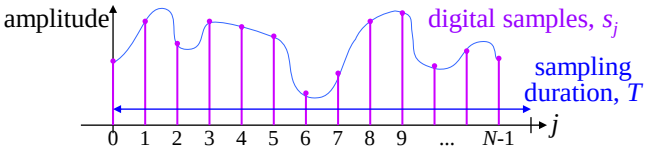


Figure 17.1 Uniformly spaced samples s_j of a signal. j is the time index.

Consider a data sequence s_j, $j = 0, ..., N–1$ (Figure 17.1). It contains N frequency components (eq. (12.1)). For simplicity, we consider a DFT-like sum without the $1/N$ scaling (cf (12.2)); the k^{th} frequency component is:

$$S_k \equiv \sum_{j=0}^{N-1} e^{-ikwj} s_j = s_0 e^{-i0kw} + s_1 e^{-i1kw} + s_2 e^{-i2kw} + ... + s_{N-1} e^{-i(N-1)kw}$$

$$\textit{where} \quad w \equiv 2\pi f_1 = 2\pi / N \text{ rad/sample, and } j \text{ is the time index.}$$

For convenience, we choose our frequencies f_0, ... f_{N-1} from 0 to just-under 1 (Figure 12.5a), where $f_0 = 0$, $f_1 = 1/T$ cyc/sample is the fundamental, and $f_k = kf_1$. $w = 2\pi f_1$ is the fundamental *angular* frequency, in radians per sample.

This basic definition of a complete DFT is an $O(N^2)$ operation: $j, k = 0, ..., N–1$. For N samples, there are N frequencies. This is computationally expensive (sometimes N is in the millions). Can we do better? Yes. To illustrate, we give two examples: an 8-point FFT, and then a 9-point FFT. We then use a 7-point DFT to show why prime N cannot be "fast."

Consider an 8-point a data sequence s_0, ... s_7. It's DFT has 8 frequencies, f_0, ... f_7. For frequency index k, and therefore angular frequency kw (rad/sample), the (complex) DFT component S_k is:

$$S_k = s_0 e^{-i0kw} + s_1 e^{-i1kw} + s_2 e^{-i2kw} + s_3 e^{-i3kw} + s_4 e^{-i4kw} + s_5 e^{-i5kw} + s_6 e^{-i6kw} + s_7 e^{-i7kw} . \qquad (17.1)$$

We notice the RHS is a sequence with uniformly spaced exponents. We can rearrange this into the form of two, smaller sequences of uniformly space exponents:

$$S_k = \left(s_0 e^{-i0kw} + s_2 e^{-i2kw} + s_4 e^{-i4kw} + s_6 e^{-i6kw} \right) + \left(s_1 e^{-i1kw} + s_3 e^{-i3kw} + s_5 e^{-i5kw} + s_7 e^{-i7kw} \right)$$

$$= \underbrace{\left(s_0 e^{-i0kw} + s_2 e^{-i2kw} + s_4 e^{-i4kw} + s_6 e^{-i6kw} \right)}_{\text{subsequence } a} + e^{-ikw} \underbrace{\left(s_1 e^{-i0kw} + s_3 e^{-i2kw} + s_5 e^{-i4kw} + s_7 e^{-i6kw} \right)}_{\text{subsequence } b}$$

Then each of the smaller sequences itself has the form of a DFT! Each subsequence has half as many points, and twice the frequency ($2kw$), just as required for a DFT. Thus, we can evaluate S_k by evaluating the two subsequences as smaller DFTs, and combining them as above. The prefactor for subsequence b is called the **twiddle factor** (really).

This decomposition is true for any k; in our example k = 0, ..., 7, corresponding to frequencies of 0, 1/8, 2/8, 3/8, 4/8, 5/8, 6/8, and 7/8 cyc/sample. So far, we haven't saved much (though we have avoided the need to calculate e^{-i3kw}, e^{-i5kw}, and e^{-i7kw}, which are expensive because they involve trig functions). But the real savings will come from aliasing. We must compute the first 4 frequencies (k = 0, 1, 2, 3) by brute force, but when k = 4, the exponents on the subsequences become:

$$k = 4: \quad -i0w, -i8w, -i16w, -i24w .$$

All of these equal $e^{-i0} = 1$: $w = 2\pi/N = 2\pi/8$. So $-i8w = -i2\pi$, $-16iw = -i4\pi$, etc. (This kind of equivalence is called "aliasing".) When k = 5, the exponents become:

$$k = 5: \quad -i0w, -i10w, -i20w, -i30w \quad \overset{\text{alias}}{\rightarrow} \quad -i0w, -i2w, -i4w, -i6w .$$

For example, $e^{-i10w} = \cancel{e^{-i8w}} e^{-i2w} = e^{-i2w}$. In each subsequence, these exponents are the same as k = 1, and so each subsequence has the same sum as k = 1. The k = 1 results we already computed can be reused for free. Similarly, k = 6, 7 alias to k = 2, 3. So all the frequencies for k = 4, 5, 6, 7 require no more exponentiations, and just 1 multiply each. The only factor that changes for higher k is the twiddle factor.

Recall that the DFT is $O(N^2)$. By splitting the computation into two, smaller DFTs, each ½ the size of the original, we have a net savings in computation by about a factor of 2 (the twiddle factor multiply is cheap compared to large N). Of course, if our subsequences are themselves divisible, we can split each of them in two, and so on. Each such splitting is called a "stage", and there are $\log_2 N$ of them. Each stage takes $O(N)$ operations, and there are $\log_2 N$ stages, so the entire DFT is done in $O(N \log_2 N)$. This is the essence of the Fast Fourier Transform (FFT) algorithm.

Furthermore, instead of 2 subsequences, we can have any number. Consider a 9-point DFT:

$$S_k = s_0 e^{-i0kw} + s_1 e^{-i1kw} + s_2 e^{-i2kw} + s_3 e^{-i3kw} + s_4 e^{-i4kw} + s_5 e^{-i5kw} + s_6 e^{-i6kw} + s_7 e^{-i7kw} + s_8 e^{-i8kw} .$$

Rearrange into 3 subsequences of uniformly space exponents:

$$S_k = \left(s_0 e^{-i0kw} + s_3 e^{-i3kw} + s_6 e^{-i6kw}\right) + \left(s_1 e^{-i1kw} + s_4 e^{-i4kw} + s_7 e^{-i7kw}\right) + \left(s_2 e^{-i2kw} + s_5 e^{-i5kw} + s_8 e^{-i8kw}\right)$$

$$= \underbrace{\left(s_0 e^{-i0kw} + s_3 e^{-i3kw} + s_6 e^{-i6kw}\right)}_{\text{subsequence } a} + e^{-ikw} \underbrace{\left(s_1 e^{-i0kw} + s_4 e^{-i3kw} + s_7 e^{-i6kw}\right)}_{\text{subsequence } b}$$

$$+ e^{-i2kw} \underbrace{\left(s_2 e^{-i0kw} + s_5 e^{-i3kw} + s_8 e^{-i6kw}\right)}_{\text{subsequence } c}$$

Now we have two twiddle factors. We perform the 3 sub-DFTs separately, and combine with the twiddle factors. For k = 0, 1, 2 this is direct computation. When k = 3 or 6, each subsequence aliases back to k = 0 (while the twiddle factors remain unique for each k). Similarly, k = 4, 5 alias into k = 1, 2, as do k = 7, 8.

In general, if N has a factor g, we can (a) split the sum (17.1) into g subsequences of length N/g, (b) perform DFTs on each subsequence, (c) for each frequency k = 0, ... N–1, combine the subsequences with twiddle factors.

It is often said that an FFT requires N be a power of 2; as we've shown, this is not true.

Modern FFT algorithms accept any number of data points. If yours doesn't, get a better one. Remember, don't pad your data with zeros; it screws up the spectrum (see p195).

An inverse DFT has the same form as a forward DFT, but with a plus-sign in the exponent. Aliasing works the same, so this same algorithm works for both forward and inverse transforms.

Counterexample: The FFT fails for prime N, because the aliasing is not helpful. Consider a 7-point sequence s_0, ... s_6, and its DFT. For frequency f_k, the (complex) DFT component S_k is:

$$S_k = s_0 e^{-i0kw} + s_1 e^{-i1kw} + s_2 e^{-i2kw} + s_3 e^{-i3kw} + s_4 e^{-i4kw} + s_5 e^{-i5kw} + s_6 e^{-i6kw} .$$

We rearrange this into the form of two, smaller, sequences of uniformly space exponents:

$$S_k = \left(s_0 e^{-i0kw} + s_2 e^{-i2kw} + s_4 e^{-i4kw} + s_6 e^{-i6kw}\right) + \left(s_1 e^{-i1kw} + s_3 e^{-i3kw} + s_5 e^{-i5kw}\right)$$

$$= \underbrace{\left(s_0 e^{-i0kw} + s_2 e^{-i2kw} + s_4 e^{-i4kw} + s_6 e^{-i6kw}\right)}_{\text{subsequence } a} + e^{-ikw} \underbrace{\left(s_1 e^{-i0kw} + s_3 e^{-i2kw} + s_5 e^{-i4kw}\right)}_{\text{subsequence } b}$$

Each of the smaller sequences itself has the form of a DFT. We can evaluate S_k by evaluating the two subsequences, and combining them with a twiddle factor, as above.

The full DFT requires computing with 7 different k values. For k = 0, 1, 2, 3, we use direct computation (we can evaluate subsequence b at these four k values, even though it has only 3 terms).

But at k = 4, the aliasing on the subsequences does not help us:

$$k = 4: \quad -i0w, -i8w, -i16w, -i24w \quad \overset{\text{alias}}{\rightarrow} \quad -i0w, -i1w, -i2w, -i3w\,.$$

This is not a sequence of exponents that we've computed already. So we must evaluate the subsequences from scratch. No savings. Higher k are similar, so no FFT. Life is harsh.

(Deprecated) Lomb-Scargle: The Meaning Behind the Math

Understanding exactly what Lomb-Scargle does, and how it works, puts you in a powerful position to understand why it is deprecated. Also, if you ever want to develop a novel algorithm, or wonder how others develop them, Lomb-Scargle provides an interesting and informative example of the process. (However, our derivation here is very different from Lomb's original [Lom], who first derived this result using the standard "normal equations" for least-squares fitting [Lom]).

The Lomb-Scargle formula may look daunting, but we can understand and derive it in just a few high-level steps:

1. Given our basis of cosine and sine, find a way to make them orthogonal.
2. Use standard orthogonal decomposition of our data into our two basis functions.
3. Normalize our coefficients, being careful to distinguish power-estimate from detection parameter.
4. Prove that the correlation amplitude of the previous steps is equivalent to the least-squares fit.

We complete these steps below, in full detail.

1. Make Cosine and Sine Orthogonal

When making a LS periodogram, we are *not* performing a basis decomposition. We are separately finding correlations with each periodogram frequency, without regard to any other frequencies. For real-valued data (i.e., not complex), there are two basis functions at any frequency: cosine and sine. We need both to find the detection level (and also the "power") at that frequency.

At any given frequency, ω, we have two basis functions, $\cos(\omega t)$ and $\sin(\omega t)$, which we write as a sum: $A\cos(\omega t) + B\sin(\omega t)$. Recall how a uniformly sampled DFT works: ω is a multiple of the fundamental frequency, *and* our sample times are *uniformly* spaced. Then cosine and sine are naturally orthogonal:

$$\sum_{j=0}^{n-1} \cos\left(\omega j\Delta t\right)\sin\left(\omega j\Delta t\right) = 0, \qquad \textit{where} \quad \begin{cases} \omega \equiv \text{a multiple of fundamental frequency} \\ j \equiv \text{sample number} \\ \Delta t \equiv \text{sampling period} = 1/\,f_{samp}\,. \end{cases}$$

Using this orthogonality, we find our coefficients A and B separately, using inner products:

$$A = \langle s | \cos \rangle = \frac{2}{n} \sum_{j=0}^{n-1} s_j \cos(\omega j \Delta t), \qquad B = \langle s | \sin \rangle = \frac{2}{n} \sum_{j=0}^{n-1} s_j \sin(\omega j \Delta t).$$

The power at frequency ω is then $A^2 + B^2$.

In contrast, for arbitrary sample times t_j (as in much observational data), *or* for arbitrary ω, cos(·) and sin(·) are not orthogonal (i.e., they are "correlated"):

$$C \equiv \sum_{j=0}^{n-1} \cos(\omega t_j) \sin(\omega t_j) \neq 0, \qquad \textit{where} \quad \begin{cases} \omega \equiv \text{arbitrary frequency} \\ j \equiv \text{sample number} \\ t_j \equiv \text{arbitrary sample times}. \end{cases}$$

Being correlated, we cannot use simple inner-products to find *A* and *B* separately. Furthermore, the presence of other components prevents us from simply simultaneously solving for the amplitudes *A* and *B*.

Despite being correlated, cosines and sines are usually still a convenient basis, because they are the eigenfunctions of linear, time-invariant systems, and appear frequently in physical systems. So we ask: Is there a way to "orthogonalize" the cosines and sines over the *given* set of arbitrary sample times? Yes, there is, as we now show.

Consider the basis-function parameters we have to play with: amplitude, frequency, and phase. We are given the frequency, and are seeking the amplitudes. The only parameter left to adjust is phase (or equivalently, a shift in time). So we could write the correlation amplitude *C* above as a function of some phase shift ϕ:

$$C(\phi) \equiv \sum_{j=0}^{n-1} \cos(\omega t_j - \phi) \sin(\omega t_j - \phi).$$

Can we find a phase shift ϕ_0 such that $C(\phi_0) = 0$, thus constructing a pair of orthogonal cosine and sine? The simplest shift I can think of is π: $\cos(\omega t_j + \pi) = -\cos(\omega t_j)$, and similarly for sin(·). Thus a phase shift of π negates both cosine and sine, and the correlation is not affected: $C(\phi + \pi) = C(\phi)$. The next simplest shift is $\pi/2$. This converts cos(·) → sin(·), and sin(·) → –cos(·), so $C(\phi + \pi/2) = -C(\phi)$. This is great: $C(\phi)$ is a continuous function of ϕ, and it changes sign in every interval of $\pi/2$. This means that somewhere between $-\pi/4$ and $+\pi/4$, $C(\phi) = 0$, i.e. the cosine and sine are orthogonal.

The existence of a phase-shift ϕ_0 which makes cosine and sine orthogonal is important, because we can always find the required ϕ_0 numerically. Even better, it turns out that we can find a closed-form expression for ϕ_0. We notice that the correlation $C(\phi)$ can be rewritten, using a simple identity:

$$\sin 2\theta = 2 \cos\theta \sin\theta \quad \Rightarrow \quad C(\phi_0) = 0 = \sum_{j=0}^{n-1} \frac{1}{2} \sin(2\omega t_j - 2\phi_0), \quad \text{or} \quad \sum_{j=0}^{n-1} \sin(2\omega t_j - 2\phi_0) = 0.$$

Given the sample times t_j, how do we find ϕ_0? We can use geometry: let's set $\phi = 0$ for now, and plot the sum of the vectors corresponding to ($x = \cos(2\omega t_j)$, $y = \sin(2\omega t_j)$), for some hypothetical sample times, t_j. Each vector is unit length (see Figure 17.2).

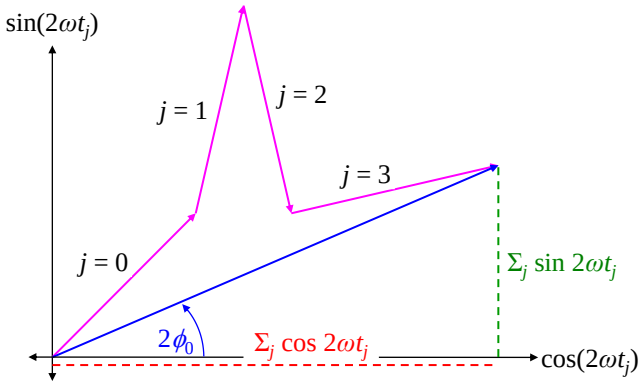


Figure 17.2 Sum (blue) of $n = 4$ vectors corresponding to ($x = \cos(2\omega t_j)$, $y = \sin(2\omega t_j)$).

(This is equivalent to plotting the complex numbers $\exp(i2\omega t_j)$ in the complex plane.) If we rotate all the vectors clockwise by $2\phi_0$, then the sum of the sine components will be zero, as required. The components of the vector sum are the sums of the components, so:

$$\tan(2\phi_0) = \frac{\sum_{j=0}^{n-1} \sin 2\omega t_j}{\sum_{j=0}^{n-1} \cos 2\omega t_j} \quad \Rightarrow \quad C(\phi_0) \equiv \sum_{j=0}^{n-1} \cos\left(\omega t_j - \phi_0\right) \sin\left(\omega t_j - \phi_0\right) = 0 \,.$$

In other words, we rotate each component (in the 2ω set) by $-2\phi_0$, which corresponds to rotating each component of our original (1ω) set by $-\phi_0$. This gives the condition we need for orthogonality. Since the interval $-\pi/4$ to $+\pi/4$ must contain a ϕ_0, we can use a simple 2-quadrant arctangent, and divide the result by 2.

Any phase shift, at a given frequency, can be written as a time shift. By convention, Lomb-Scargle uses a subtracted time shift, so:

$$2\omega\tau = 2\phi_0 \quad \Rightarrow \quad C(\phi_0) = \sum_{j=0}^{n-1} \cos\Big(\omega\big(t_j - \tau\big)\Big) \sin\Big(\omega\big(t_j - \tau\big)\Big) = 0 \,.$$

This time shift is fully equivalent to the ϕ_0 phase shift, and the cosines and sines are orthogonal over the given sample times. Be careful to distinguish ϕ_0, the orthogonalizing phase shift, from a fitted-sinusoid phase, usually called ϕ.

2. Use orthogonal decomposition of our data into our basis functions

Now that we have orthogonal basis functions (though not yet normalized), we can find our cosine and sine coefficients with simple correlations (aka inner-products):

$$A' = \langle h|\cos\rangle = \sum_{j=0}^{n-1} h_j \cos\omega\big(t_j - \tau\big), \qquad B' = \langle h|\sin\rangle = \sum_{j=0}^{n-1} h_j \sin\omega\big(t_j - \tau\big) \qquad \text{(unnormalized)} \,,$$

where the primes indicate unnormalized coefficients. Note that, because the offset cosine and sine functions are orthogonal, *A*' and *B*' fit for both components *simultaneously*. That is, orthogonality implies that the *individual* best fits for cosine and sine are also the *simultaneous* best fit.

3. Normalize Our Coefficients

With all orthonormal basis decompositions, we require *normalized* basis functions to get properly scaled components. We normalize our coefficients by dividing them by the squared-norm of the (unnormalized) basis functions:

$$A = \frac{A'}{\langle \cos | \cos \rangle} = \frac{\sum_{j=0}^{n-1} h_j \cos\omega\left(t_j - \tau\right)}{\sum_{j=0}^{n-1} \cos^2 \omega\left(t_j - \tau\right)}, \qquad B = \frac{B'}{\langle \sin | \sin \rangle} = \frac{\sum_{j=0}^{n-1} h_j \sin\omega\left(t_j - \tau\right)}{\sum_{j=0}^{n-1} \sin^2 \omega\left(t_j - \tau\right)} \qquad \text{(normalized)}.$$

These formulas are similar to the two terms in the Lomb-Scargle formula. The normalized coefficients, A and B, yield a true power estimate for a best-fit sinusoid:

$$P_{true}(\omega) = A^2 + B^2 \qquad \text{from} \qquad S_{fit}(\omega) = A\cos\omega\left(t_j - \tau\right) + B\sin\omega\left(t_j - \tau\right).$$

To arrive at the Lomb-Scargle detection parameter, we must consider not the true *power* estimate, $P_{true}(\omega)$, but the *contribution* to the total sample set "energy" (sum of squares) from our fitted sinusoid, $S_{fit}(\omega)$, *at the given sample times*. For example, for a frequency component with a given true power, if the sinusoid happens to be small at the sample times, then that component contributes a small amount to the sample-set "energy." On the other hand, if the sinusoid happens to be large at the sample times, then that component contributes a large amount to the sample-set "energy."

The *significance* of a frequency component at ω is a function of the ratio of the component's energy to that expected from pure noise. Given a component with cosine and sine amplitudes A and B, its energy in the sample set is the sums of the squares of its samples, at the given sample times:

$$\begin{aligned} E(\omega) &= \sum_{j=0}^{n-1}\left[A\cos\omega\left(t_j - \tau\right)\right]^2 + \sum_{j=0}^{n-1}\left[B\sin\omega\left(t_j - \tau\right)\right]^2 \\ &= A^2\sum_{j=0}^{n-1}\cos^2\omega\left(t_j - \tau\right) + B^2\sum_{j=0}^{n-1}\sin^2\omega\left(t_j - \tau\right) \\ &= \frac{\left(\sum_{j=0}^{n-1} h_j \cos\omega\left(t_j - \tau\right)\right)^2}{\sum_{j=0}^{n-1}\cos^2\omega\left(t_j - \tau\right)} + \frac{\left(\sum_{j=0}^{n-1} h_j \sin\omega\left(t_j - \tau\right)\right)^2}{\sum_{j=0}^{n-1}\sin^2\omega\left(t_j - \tau\right)}. \end{aligned}$$

This is almost the Lomb-Scargle detection parameter.

For white noise (not necessarily gaussian), with no signal, the samples h_j are independent identically distributed (IID), with variance equal to the noise power, σ^2. Across many sets of noise, then, the numerators above have variance:

$$\sigma_A^2 = \sigma^2\sum_{j=0}^{n-1}\cos^2\omega\left(t_j - \tau\right), \qquad \sigma_B^2 = \sigma^2\sum_{j=0}^{n-1}\sin^2\omega\left(t_j - \tau\right).$$

This means each term in $E(\omega)$ has variance = σ^2.

For *gaussian* noise, A and B are gaussian, and $E(\omega)$ is the sum of their squares, scaled to the *estimated* variance = σ^2. Therefore, $E(\omega)/\sigma^2$ (always ≤ 1) is distributed with CDF an incomplete beta function [A. Schwarzenberg-Czerny, 1997]. (For decades, it was thought that $E(\omega)/\sigma^2$ was $\chi^2_{\nu=2}$ distributed, but it is easy to show that it is not: χ^2 has no upper bound, but $E(\omega)/\sigma^2 \le 1$.) Nonetheless, assuming the incorrect $\chi^2_{\nu=2}$ distribution, Lomb-Scargle divides by 2 to get a more-convenient exponential distribution with $\mu = 1$:

$$D(\omega) = \frac{1}{2} \cdot \frac{E(\omega)}{\sigma^2} = \frac{1}{2\sigma^2} \left[\frac{\left(\sum_{j=0}^{n-1} h_j \cos \omega \left(t_j - \tau \right) \right)^2}{\sum_{j=0}^{n-1} \cos^2 \omega \left(t_j - \tau \right)} + \frac{\left(\sum_{j=0}^{n-1} h_j \sin \omega \left(t_j - \tau \right) \right)^2}{\sum_{j=0}^{n-1} \sin^2 \omega \left(t_j - \tau \right)} \right].$$

This is the standard (though flawed??) formula for LS detection. Note again that we don't know the true σ^2; we must estimate it from the samples.

4. Proof that the correlation amplitude of the previous steps is equivalent to the least-squares fit

This is a general theorem: any correlation amplitude for a component of a sequence s_j is equivalent to a least-squares fit. We prove it by contradiction. Given any single basis function, b_k, we can construct a complete, orthonormal basis set which includes it. In that case, the component of b_k is found by correlation, as usual. Call it A_k.

The least-squares residue is simply the "energy" (sum of squares) of the sequence after subtracting off the b_k component. Since the basis set is orthonormal, Parseval's theorem holds. Thus, the residual energy after subtracting the b_k component from s_j is the sum of the squares of all the *other* component amplitudes. If there existed some *other* value of A_k which had less residual energy, then that would imply a *different* decomposition into the *other* basis functions. But the decomposition into a given orthogonal basis is unique. Therefore, no A_k other than the one given by correlation can have a smaller residual.

The basis coefficient given by correlation is a least-squares-residual fit.

This proof holds equally well for discrete sequences s_j, and for continuous functions $s(t)$.

18 Tensors, Without the Tension

Approach

We'll present tensors as follows:

1. Two physical examples: magnetic susceptibility, and deformable solids
2. A non-example: when is a matrix not a tensor?
3. Forward looking definitions (don't get stuck on these)
4. Review of vector spaces and notation (don't get stuck on this, either)
5. A short, but at first unhelpful, definition (really, really don't get stuck on this)
6. A discussion which clarifies the above definition
7. Examples, including dot products and cross-products as tensors
8. Higher rank tensors
9. Change of basis
10. Non-orthonormal systems: contravariance and covariance
11. Indefinite metrics of Special and General Relativity
12. Mixed basis linear functions (transformation matrices, the Pauli vector)

Tensors are all about vectors. They let you do things with vectors you never thought possible. We define tensors in terms of what they do (their linearity properties), and then show that linearity *implies* the transformation properties. This gets most directly to the true importance of tensors. [Most references define tensors in terms of transformations, but then fail to point out the all-important linearity properties.]

We also take a geometric approach, treating vectors and tensors as geometric objects that exist independently of their representation in any basis. Inevitably, though, there is a fair amount of unavoidable algebra.

Later, we introduce contravariance and covariance in terms of non-orthonormal coordinates, but first with a familiar positive-definite metric from classical mechanics. This makes for a more intuitive understanding of contra- and co-variance, before applying the concept to the more bizarre indefinite metrics of special and general relativity.

There is deliberate repetition of several points, because it usually takes me more than once to grok something. So I repeat:

> If you don't understand something, read it again once, then keep reading.
> Don't get stuck on one thing. Often, the following discussion will clarify an ambiguity.

Two Physical Examples

We start with two physical examples: magnetic susceptibility, and deformation of a solid. We start with matrix notation, because we assume it is familiar to you. Later we will see that matrix notation is not ideal for tensor algebra.

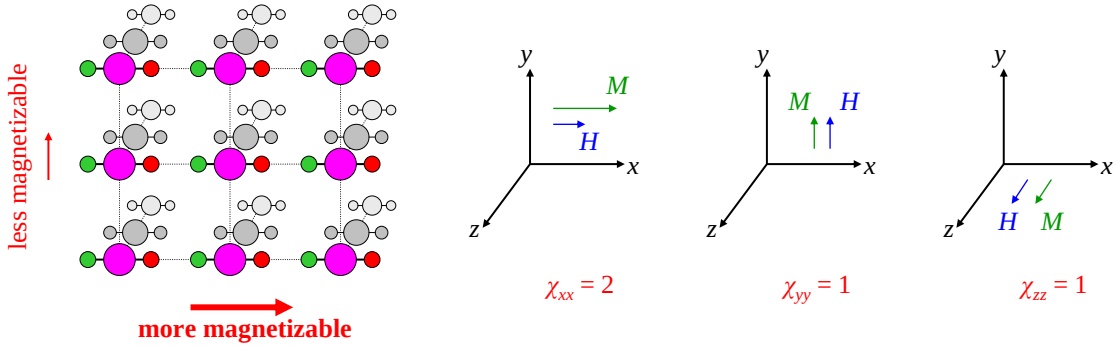
Magnetic Susceptibility

We assume you are familiar with **susceptibility** of magnetic materials: when placed in an H-field, magnetizable (susceptible) materials acquire a magnetization, which adds to the resulting B-field. In simple cases, the susceptibility χ is a scalar, and

$\mathbf{M} = \chi\mathbf{H}$ *where* $\mathbf{M}$ is the magnetization,
χ is the susceptibility, and
$\mathbf{H}$ is the applied magnetic field

The susceptibility in this simple case is the same in any direction; i.e., the material is isotropic.

However, there exist materials which are more magnetizable in some directions than others. E.g., imagine a cubic lattice of axially-symmetric molecules which are more magnetizable along the molecular axis than perpendicular to it:

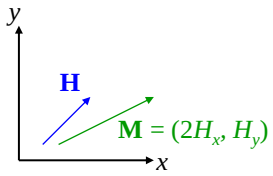


Magnetization, M, as a function of external field, H, for a material with a tensor-valued susceptibility, χ.

In each direction, the magnetization is proportional to the applied field, but χ is larger in the x-direction than y or z. In this example, for an arbitrary H-field, we have

$$\mathbf{M} = (M_x, M_y, M_z) = (2H_x, H_y, H_z) \qquad \text{or} \qquad \mathbf{M} = \boldsymbol{\chi}\mathbf{H} = \underbrace{\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{\chi} \equiv \chi_{ij}} \mathbf{H} .$$

Note that in general, **M** is not parallel to **H** (below, dropping the z axis for now):



M need not be parallel to **H** for a material with a tensor-valued χ.

But **M** is a linear function of **H**, which means: $\mathbf{M}(k\mathbf{H}_1 + \mathbf{H}_2) = k\mathbf{M}(\mathbf{H}_1) + \mathbf{M}(\mathbf{H}_2)$.

This linearity is reflected in the fact that matrix multiplication is linear:

$$\mathbf{M}(k\mathbf{H}_1 + \mathbf{H}_2) = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} (k\mathbf{H}_1 + \mathbf{H}_2) = k \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{H}_1 + \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{H}_2 = k\mathbf{M}(\mathbf{H}_1) + \mathbf{M}(\mathbf{H}_2) .$$

The matrix notation might seem like overkill, since χ is diagonal, but it is *only* diagonal in this basis of x, y, and z. We'll see in a moment what happens when we change basis. First, let us understand what the

matrix χ_{ij} really means. Recall the visualization of pre-multiplying a vector by a matrix: a matrix $\boldsymbol{\chi}$ times a column vector $\mathbf{H}$, is a weighted sum of the columns of $\boldsymbol{\chi}$:

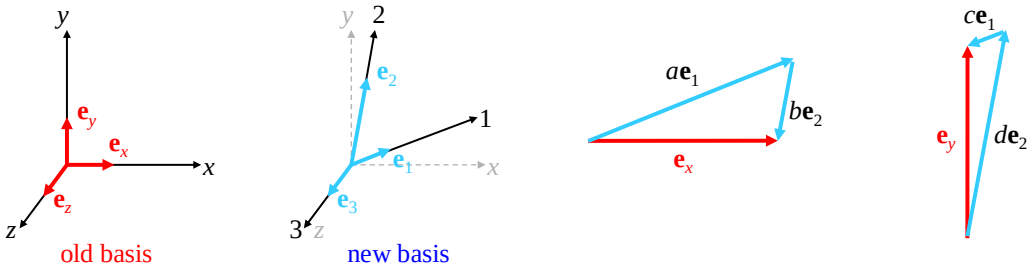
$$\boldsymbol{\chi}\mathbf{H} = \begin{bmatrix} \chi_{xx} & \chi_{xy} & \chi_{xz} \\ \chi_{yx} & \chi_{yy} & \chi_{yz} \\ \chi_{zx} & \chi_{zy} & \chi_{zz} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} \equiv H_x \begin{bmatrix} \chi_{xx} \\ \chi_{yx} \\ \chi_{zx} \end{bmatrix} + H_y \begin{bmatrix} \chi_{xy} \\ \chi_{yy} \\ \chi_{zy} \end{bmatrix} + H_z \begin{bmatrix} \chi_{xz} \\ \chi_{yz} \\ \chi_{zz} \end{bmatrix}$$

We can think of the matrix $\boldsymbol{\chi}$ as a set of 3 column vectors: the first is the magnetization vector for $\mathbf{H} = \mathbf{e}_x$; the 2nd column is $\mathbf{M}$ for $\mathbf{H} = \mathbf{e}_y$; the 3rd column is $\mathbf{M}$ for $\mathbf{H} = \mathbf{e}_z$. Since magnetization is linear in $\mathbf{H}$, the magnetization for any $\mathbf{H}$ can be written as the weighted sum of the magnetizations for each of the basis vectors:

$$\mathbf{M}(\mathbf{H}) = H_x\mathbf{M}(\mathbf{e}_x) + H_y\mathbf{M}(\mathbf{e}_y) + H_z\mathbf{M}(\mathbf{e}_z) \qquad \textit{where} \quad \mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z \text{ are the unit vectors in } x, y, z\,.$$

This is just the matrix multiplication above: $\mathbf{M} = \boldsymbol{\chi}\mathbf{H}$. (We're writing all indexes as subscripts for now; later on we'll see that $\mathbf{M}$, $\boldsymbol{\chi}$, and $\mathbf{H}$ should be indexed as M^i, $\chi^i{}_j$, and H^i.)

Now let's change bases from $\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$, to some $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$, defined below. We use a simple transformation, but the 1-2-3 basis is *not* orthonormal:



Transformation to a non-orthogonal, non-normal basis. $\mathbf{e}_1$ and $\mathbf{e}_2$ are in the *x*-*y* plane, but are neither orthogonal nor normal. For simplicity, we choose $\mathbf{e}_3 = \mathbf{e}_z$. Here, *b* and *c* are negative.

To find the transformation equations to the new basis, we first write the old basis vectors in the new basis. We've chosen for simplicity a transformation in the *x*-*y* plane, with the *z*-axis unchanged:

$$\mathbf{e}_x = a\mathbf{e}_1 + b\mathbf{e}_2 \qquad \mathbf{e}_y = c\mathbf{e}_1 + d\mathbf{e}_2 \qquad \mathbf{e}_z = \mathbf{e}_3\,.$$

Now write a vector, $\mathbf{v}$, in the old basis, and substitute out the old basis vectors for the new basis. We see that the new components are a linear combination of the old components:

$$\mathbf{v} = v_x\mathbf{e}_x + v_y\mathbf{e}_y + v_z\mathbf{e}_z = v_x\underbrace{\left(a\mathbf{e}_1 + b\mathbf{e}_2\right)}_{\mathbf{e}_x} + v_y\underbrace{\left(c\mathbf{e}_1 + d\mathbf{e}_2\right)}_{\mathbf{e}_y} + v_z\mathbf{e}_3$$

$$= \left(av_x + cv_y\right)\mathbf{e}_1 + \left(bv_x + dv_y\right)\mathbf{e}_2 + v_z\mathbf{e}_3 = v_1\mathbf{e}_1 + v_2\mathbf{e}_2 + v_3\mathbf{e}_3$$

$$\Rightarrow \qquad v_1 = av_x + cv_y, \qquad v_2 = bv_x + dv_y, \qquad v_3 = v_z$$

Recall that matrix multiplication is defined to be the operation of linear transformation, so we can write this basis transformation in matrix form:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{pmatrix} a & c & 0 \\ b & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = v_x \underbrace{\begin{bmatrix} a \\ b \\ 0 \end{bmatrix}}_{\mathbf{e}_x} + v_y \underbrace{\begin{bmatrix} c \\ d \\ 0 \end{bmatrix}}_{\mathbf{e}_y} + v_z \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{\mathbf{e}_z}.$$

The columns of the transformation matrix are the old basis vectors written in the new basis.

This is illustrated explicitly on the right hand side, which is just $v_x\mathbf{e}_x + v_y\mathbf{e}_y + v_z\mathbf{e}_z$.

Finally, we look at how the susceptibility matrix χ_{ij} transforms to the new basis. We saw above that the columns of $\boldsymbol{\chi}$ are the **M** vectors for **H** = each of the basis vectors. So right away, we must transform each column of $\boldsymbol{\chi}$ with the transformation matrix above, to convert it to the new basis. Since matrix multiplication **A**·**B** is distributive across the columns of **B**, we can write the transformation of all 3 columns in a single expression by pre-multiplying with the above transformation matrix:

$$\text{Step 1 of } \boldsymbol{\chi}^{new} \equiv \boldsymbol{\chi} \text{ in new basis} = \begin{pmatrix} a & c & 0 \\ b & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \boldsymbol{\chi} = \begin{pmatrix} a & c & 0 \\ b & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2a & c & 0 \\ 2b & d & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

But we're not done. This first step expressed the column vectors in the new basis, but the columns of the RHS (right hand side) are still the **M**'s for basis vectors $\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$. Instead, we need the columns of $\boldsymbol{\chi}^{new}$ to be the **M** vectors for $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$. Please don't get bogged down yet in the details, but we do this transformation similarly to how we transformed the column vectors. We transform the contributions to **M** due to $\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$ to that due to $\mathbf{e}_1$ by writing $\mathbf{e}_1$ in terms of $\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$:

$$\mathbf{e}_1 = e\mathbf{e}_x + f\mathbf{e}_y \qquad \Rightarrow \qquad \mathbf{M}\left(\mathbf{H} = \mathbf{e}_1\right) = e\mathbf{M}\left(\mathbf{H} = \mathbf{e}_x\right) + f\mathbf{M}\left(\mathbf{H} = \mathbf{e}_y\right).$$

Similarly,

$$\mathbf{e}_2 = g\mathbf{e}_x + h\mathbf{e}_y \qquad \Rightarrow \qquad \mathbf{M}\left(\mathbf{H} = \mathbf{e}_2\right) = g\mathbf{M}\left(\mathbf{H} = \mathbf{e}_x\right) + h\mathbf{M}\left(\mathbf{H} = \mathbf{e}_y\right)$$

$$\mathbf{e}_3 = \mathbf{e}_z \qquad \Rightarrow \qquad \mathbf{M}\left(\mathbf{H} = \mathbf{e}_3\right) = \mathbf{M}\left(\mathbf{H} = \mathbf{e}_z\right)$$

Essentially, we need to transform among the columns, i.e. transform the rows of $\boldsymbol{\chi}$. These two transformations (once of the columns, and once of the rows) is the essence of a rank-2 tensor:

A tensor matrix (rank-2 tensor) has columns that are vectors, and simultaneously, its rows are also vectors. Therefore, transforming to a new basis requires two transformations: once for the rows, and once for the columns (in either order).

[Aside: The details (which you can skip at first): We just showed that we transform using the *inverse* of our previous transformation. The reason for the inverse is related to the up/down indexes mentioned earlier; please be patient. In matrix notation, we write the row transformation as post-multiplying by the transpose of the needed transformation:

$$\text{Final} \qquad \boldsymbol{\chi}^{new} = \begin{pmatrix} a & c & 0 \\ b & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} e & f & 0 \\ g & h & 0 \\ 0 & 0 & 1 \end{pmatrix}^T = \begin{pmatrix} a & c & 0 \\ b & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} e & g & 0 \\ f & h & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

]

[Another aside: A direction-dependent susceptibility requires χ to be promoted from a scalar to a rank-2 tensor (skipping any rank-1 tensor). This is necessary because a rank-0 tensor (a scalar) and a rank-2 tensor can both act on a vector (**H**) to produce a vector (**M**). There is no sense to a rank-1 (vector) susceptibility, because there is no simple way a rank-1 tensor (a vector) can act on another vector **H** to produce an output vector **M**. More on this later.]

Mechanical Strain

A second example of a tensor is the mechanical strain tensor. When I push on a deformable material, it deforms. A simple model is just a spring, with Hooke's law:

$$\Delta x = +\frac{1}{k} F_{applied} \, .$$

We write the formula with a plus sign, because (unlike freshman physics spring questions) we are interested in how a body deforms when *we* apply a force *to* it. For an isotropic material, we can push in any direction, and the deformation is parallel to the force. This makes the above equation a vector equation:

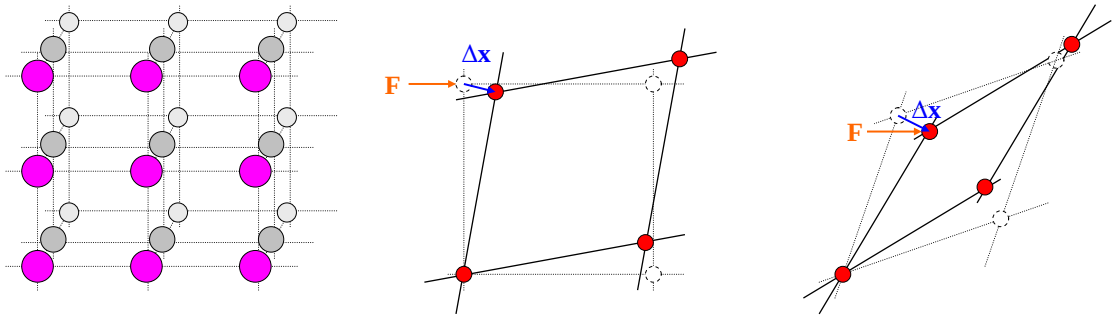
$$\Delta \mathbf{x} = s\mathbf{F} \qquad \textit{where} \quad s = \frac{1}{k} \equiv \text{the strain constant} \, .$$

Strain is defined as the displacement of a given point under force. [**Stress** is the force per unit area applied to a body. Stress produces strain.] In an isotropic material, the stress constant is a simple scalar. Note that if we transform to another basis for our vectors, the stress constant is unchanged. That's the definition of a scalar:

A scalar is a number that is the same in any coordinate system. A scalar is a rank-0 tensor.

The scalar is unchanged even in a non-ortho-normal coordinate system.

But what if our material is a bunch of microscopic blobs connected by stiff rods, like atoms in a crystal?



(Left) A constrained deformation crystal structure. (Middle) The deformation vector, Δx, is not parallel to the force. (Right) More extreme geometries lead to a larger angle between the force and displacement.

The diagram shows a 2D example: pushing in the x-direction results in both x and y displacements. The same principle could result in a 3D $\Delta\mathbf{x}$, with some component into the page. For small deformations, the deformation is *linear* with the force: pushing twice as hard results in twice the displacement. Pushing with the sum of two (not necessarily parallel) forces results in the sum of the individual displacements. But the displacement is *not proportional* to the force (because the displacement is not parallel to it). In fact, each component of force results in a deformation *vector*. Mathematically:

$$\Delta\mathbf{x} = F_x \begin{bmatrix} s_{xx} \\ s_{yx} \\ s_{zx} \end{bmatrix} + F_y \begin{bmatrix} s_{xy} \\ s_{yy} \\ s_{zy} \end{bmatrix} + F_z \begin{bmatrix} s_{xz} \\ s_{yz} \\ s_{zz} \end{bmatrix} = \underbrace{\begin{pmatrix} s_{xx} & s_{xy} & s_{xz} \\ s_{yx} & s_{yy} & s_{yz} \\ s_{zx} & s_{zy} & s_{zz} \end{pmatrix}}_{\mathbf{s}} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \mathbf{sF} \, .$$

Much like the anisotropy of the magnetization in the previous example, the anisotropy of the strain requires us to use a rank-2 tensor to describe it. The *linearity* of the strain with force allows us to write the strain tensor as a matrix. Linearity also guarantees that we can change to another basis using a method similar to that shown above for the susceptibility tensor. Specifically, we must transform both the columns and the

rows of the strain tensor **s**. Furthermore, the linearity of deformation with force also insures that we can use non-orthonormal bases, just as well as orthonormal ones.

When Is a Matrix Not a Tensor?

I would say that most matrices are *not* tensors. A matrix *is* a tensor when its rows and columns are both vectors. This implies that there is a vector space, basis vectors, and the possibility of changing basis. As a counter example, consider the following graduate physics problem:

Two pencils, an eraser, and a ruler cost $2.20. Four pencils, two erasers, and a ruler cost $3.45. Four pencils, an eraser, and two rulers cost $3.85. How much does each item cost?

We can write this as simultaneous equations, and as shorthand in matrix notation:

$$\begin{aligned} 2p + e + r &= 220 \\ 4p + 2e + r &= 345 \\ 4p + e + 2r &= 385 \end{aligned} \qquad or \qquad \begin{pmatrix} 2 & 1 & 1 \\ 4 & 2 & 1 \\ 4 & 1 & 1 \end{pmatrix} \begin{bmatrix} p \\ e \\ r \end{bmatrix} = \begin{bmatrix} 220 \\ 345 \\ 385 \end{bmatrix}.$$

It is possible to use a matrix for this problem because the problem takes *linear combinations* of the costs of 3 items. Matrix multiplication is defined as the process of linear combinations, which is the same process as linear transformations. However, the above matrix is *not* a tensor, because there are no vectors of school supplies, no bases, and no linear combinations of (say) part eraser and part pencil. Therefore, the matrix has no well-defined transformation properties. Hence, it is a lowly matrix, but no tensor.

However, later (in "We Don't Need No Stinking Metric") we'll see that under the right conditions, we *can* form a vector space out of seemingly unrelated quantities.

Heading In the Right Direction

An ordinary vector associates a *number* with each direction of space:

$$\mathbf{v} = v_x \hat{\mathbf{x}} + v_y \hat{\mathbf{y}} + v_z \hat{\mathbf{z}} .$$

The vector **v** associates the number v_x with the *x*-direction; it associates the number v_y with the *y*-direction, and the number v_z with the *z*-direction.

The above tensor examples illustrate the basic nature of a rank-2 tensor:

A rank-2 tensor associates a *vector* with each direction of space:

$$\mathbf{T} = \begin{bmatrix} T_{xx} \\ T_{yx} \\ T_{zx} \end{bmatrix} \hat{\mathbf{x}} + \begin{bmatrix} T_{xy} \\ T_{yy} \\ T_{zy} \end{bmatrix} \hat{\mathbf{y}} + \begin{bmatrix} T_{xz} \\ T_{yz} \\ T_{zz} \end{bmatrix} \hat{\mathbf{z}} .$$

Some Definitions and Review

These definitions will make more sense as we go along. Don't get stuck on these:

"ordinary" vector = contravariant vector = contravector = $\binom{1}{0}$ tensor

1-form = covariant vector = covector = $\binom{0}{1}$ tensor. (Yes, there are 4 different ways to say the same thing.)

covariant	the same. E.g., General Relativity says that the mathematical form of the laws of physics are covariant (i.e., the same) with respect to arbitrary coordinate transformations. This is a *completely different meaning* of "covariant" than the one above.
rank	The number of indexes of a tensor; T^{ij} is a rank-2 tensor; $R^{i}{}_{jkl}$ is a rank-4 tensor. Rank is unrelated to the dimension of the vector space in which the tensor operates.
MVE	mathematical vector element. Think of it as a vector for now.

Caution: a rank $\binom{0}{1}$ tensor is a 1-form, but a rank $\binom{0}{2}$ tensor is not always a 2-form. [Don't worry about it, but just for completeness, a 2-form (or any *n*-form) has to be fully anti-symmetric in all pairs of vector arguments.]

Notation:

(a, b, c) is a row vector; $(a, b, c)^T$ is a column vector (the transpose of a row vector).

To satisfy our pathetic word processor, we write $\binom{m}{n}$, even though the 'm' is supposed to be directly above the 'n'.

$\mathbf{T}$	is a tensor, without reference to any basis or representation.
T^{ij}	is the matrix of components of **T**, contravariant in both indexes, with an understood basis.
$\mathbf{T}(\mathbf{v}, \mathbf{w})$	is the result of **T** acting on **v** and **w**.
$\mathbf{v}$ *or* $\vec{v}$	are two equivalent ways to denote a vector, without reference to any basis or representation. Note that a vector is a rank-1 tensor.
$\tilde{\mathbf{a}}$ *or* $a\sim$	are two equivalent ways to denote a covariant vector (aka 1-form), without reference to any basis or representation
a_i	the components of the covecter (1-form) **a**, in an understood basis.

Vector Space Summary

Briefly, a **vector space** comprises a **field** of scalars, a **group** of vectors, and the operation of scalar multiplication of vectors (details below). Quantum mechanical vector spaces have two additional characteristics: they define a dot product between two vectors, and they define linear operators which act on vectors to produce other vectors.

Before understanding tensors, it is very helpful, if not downright necessary, to understand vector spaces. *Quirky Quantum Concepts* has a more complete description of vector spaces. Here is a *very* brief summary: a **vector space** comprises a **field** of scalars, a **group** of vectors, and the operation of scalar multiplication of vectors. The scalars can be any mathematical "field," but are usually the real numbers, or the complex numbers (e.g., quantum mechanics). For a given vector space, the vectors are a class of things, which can be one of many possibilities (physical vectors, matrices, kets, bras, tensors, ...). In particular, the vectors are *not* necessarily lists of scalars, nor need they have anything to do with physical space. Vector spaces have the following properties, which allow solving simultaneous linear equations both for unknown scalars, and unknown vectors:

Scalars	Mathematical Vectors
Scalars form a commutative group (closure, unique identity, inverses) under operation +.	Vectors form a commutative group (closure, unique identity, inverses) under operation +.
Scalars, excluding 0, form a commutative group under operation (·).	
Distributive property of (·) over +.	
Scalar multiplication of vector produces another vector.	
Distributive property of scalar multiplication over both scalar + and vector +.	

With just the scalars, you can solve ordinary scalar linear equations such as:

$$\left.\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \ldots a_{1n}x_n = c_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots a_{2n}x_n = c_2 \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots a_{nn}x_n = c_n \end{array}\right\} \qquad \textit{written in matrix form as} \qquad \mathbf{ax} = \mathbf{c}\,.$$

All the usual methods of linear algebra work to solve the above equations: Cramer's rule, Gaussian elimination, etc. With the whole vector space, you can solve simultaneous linear *vector* equations for unknown vectors, such as

$$\left.\begin{array}{l} a_{11}\mathbf{v}_1 + a_{12}\mathbf{v}_2 + \ldots a_{1n}\mathbf{v}_n = \mathbf{w}_1 \\ a_{21}\mathbf{v}_1 + a_{22}\mathbf{v}_2 + \ldots a_{2n}\mathbf{v}_n = \mathbf{w}_2 \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \\ a_{n1}\mathbf{v}_1 + a_{n2}\mathbf{v}_2 + \ldots a_{nn}\mathbf{v}_n = \mathbf{w}_n \end{array}\right\} \qquad \textit{written in matrix form as} \qquad \mathbf{av} = \mathbf{w}\,,$$

where **a** is again a matrix of scalars. The same methods of linear algebra work just as well to solve vector equations as scalar equations.

Vector spaces may also have these properties:

Dot product produces a scalar from two vectors.
Linear operators act on vectors to produce other vectors.

The key points of mathematical vectors are (1) we can form linear combinations of them to make other vectors, and (2) any vector can be written as a linear combination of basis vectors:

$\mathbf{v} = (v^1, v^2, v^3) = v^1\mathbf{e}_1 + v^2\mathbf{e}_2 + v^3\mathbf{e}_3$
where $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ are basis vectors, and
v^1, v^2, v^3 are the *components* of **v** in the $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ basis.

Note that v^1, v^2, v^3 are *numbers*, while $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ are *vectors*. There is a (kind of bogus) reason why basis vectors are written with subscripts, and vector components with superscripts, but we'll get to that later.

The **dimension** of a vector space, *N*, is the number of basis vectors needed to construct every vector in the space.

Do not confuse the dimension of physical space (typically 1D, 2D, 3D, or (in relativity) 4D), with the dimension of the mathematical objects used to work a problem.

For example, a 3×3 matrix is an element of the vector space of 3×3 matrices. This is a 9-D vector space, because there are 9 basis matrices needed to construct an arbitrary matrix.

Given a basis, components are equivalent to the vector. Components alone (without a basis) are insufficient to be a vector.

[Aside: Note that for position vectors defined by $\mathbf{r} = (r, \theta, \phi)$, r, θ, and ϕ are *not* the components of a vector. The tip off is that with two vectors, you can *always* add their components to get another vector. Clearly, $\mathbf{r}_1 + \mathbf{r}_2 \neq (r_1 + r_2, \theta_1 + \theta_2, \phi_1 + \phi_2)$, so (r, θ, ϕ) *cannot be* the components of a vector. This failure to add is due to **r** being a displacement vector from the origin, where there is no consistent basis: e.g., what is $\mathbf{e}_r$ at the origin? At points *off* the origin, there *is* a consistent basis: $\mathbf{e}_r$, $\mathbf{e}_\theta$, and $\mathbf{e}_\phi$ are well-defined.]

When Vectors Collide

There now arises a collision of terminology: to a physicist, "vector" usually means a physical vector in 3- or 4-space, but to a mathematician, "vector" means an element of a mathematical vector-space. These are two different meanings, but they share a common aspect: linearity (i.e., we can form linear combinations of vectors to make other vectors, and any vector can be written as a linear combination of basis vectors). Because of that linearity, we can have general rank-*n* tensors whose components are arbitrary elements of a

mathematical vector-space. To make the terminology confusion worse, an $\binom{m}{n}$ tensor whose components are simple numbers is itself a "vector-element" of the vector-space of $\binom{m}{n}$ tensors.

Mathematical vector-elements of a vector space are much more general than physical vectors (e.g. force, or velocity), though physical vectors and tensors are elements of mathematical vector spaces. To be clear, we'll use **MVE** to refer to a mathematical vector-element of a vector space, and "vector" to mean a normal physics vector (3-vector or 4-vector). Recall that MVEs are usually written as a set of components in some basis, just like vectors are. In the beginning, we choose all the input MVEs to be vectors.

If you're unclear about what an MVE is, just think of it as a physical vector for now, like "force."

"Tensors" vs. "Symbols"

There are lots of tensors: metric tensors, electromagnetic tensors, Riemann tensors, etc. There are also "symbols:" Levi-Civita symbols, Christoffel symbols, etc. What's the difference? "Symbols" aren't tensors. Symbols look like tensors, in that they have components indexed by multiple indices, they are referred to basis vectors, and are summed with tensors. But they are defined to have *specific* components, which may depend on the basis, and therefore symbols don't change basis (transform) the way tensors do. Hence, symbols are *not* geometric entities, with a meaning in a manifold, independent of coordinates. For example, the Levi-Civita symbol is defined to have specific constant components in *all* bases. It doesn't follow the usual change-of-basis rules. Therefore, it cannot be a tensor.

Notational Nightmare

If you come from a differential geometry background, you may wonder about some insanely confusing notation. It is a fact that "$d\mathbf{x}$" and "$\mathbf{d}x$" are two different things:

$d\mathbf{x} = (dx, dy, dz)$ is a **vector**, *but*

$\mathbf{d}x = \nabla x(\mathbf{r})$ is a **1-form**

We don't use the second notation (or exterior derivatives) in this chapter, but we might in the Differential Geometry chapter.

Tensors? What Good Are They?

A Short, Complicated Definition

It is very difficult to give a short definition of a tensor that is useful to anyone who doesn't already know what a tensor is. Nonetheless, you've got to start somewhere, so we'll give a short definition, to point in the right direction, but it may not make complete sense at first (don't get hung up on this, skip if needed):

A **tensor** is an operator on one or more **mathematical vector elements** (MVEs), *linear in each operand*, which produces another mathematical vector element.

The key point is this (which we describe in more detail in a moment):

Linearity in all the operands is the essence of a tensor.

I should add that the basis vectors for all the MVEs must be the same (or tensor products of the same) for an operator to qualify as a tensor. But that's too much to put in a "short" definition. We clarify this point later.

Note that a **scalar** (i.e., a coordinate-system-invariant number, but for now, just a number) satisfies the definition of a "mathematical vector element."

Many definitions of tensors dwell on the transformation properties of tensors. This is mathematically valid, but such definitions give no insight into the use of tensors, or why we like them. Note that to satisfy the transformation properties, all the input vectors and output tensors must be expressed in the same basis (or tensor products of that basis with itself).

Some coordinate systems require distinguishing between **contravariant** and **covariant** components of tensors; superscripts denote contravariant components; subscripts denote covariant components. However, orthonormal positive definite systems, such as the familiar Cartesian, spherical, and cylindrical systems, do not require such a distinction. So for now, let's ignore the distinction, even though the following notation properly represents both contravariant and covariant components. Thus, in the following text, contravariant components are written with superscripts, and covariant components are written with subscripts, but we don't care right now. Just think of them all as components in an arbitrary coordinate system.

Building a Tensor

Oversimplified, a tensor operates on vectors to produce a scalar or a vector. Let's construct a tensor which accepts (operates on) two 3-vectors to produce a scalar. (We'll see later that this is a rank-2 tensor.) Let the tensor **T** act on vectors **a** and **b** to produce a scalar, s; in other words, this tensor is a scalar function of two vectors:

$$s = \mathbf{T}(\mathbf{a}, \mathbf{b}) .$$

Call the first vector $\mathbf{a} = (a^1, a^2, a^3)$ in some basis, and the second vector $\mathbf{b} = (b^1, b^2, b^3)$ (in the same basis). A tensor, by definition, must be linear in both **a** and **b**; if we double **a**, we double the result, if we triple **b**, we triple the result, etc. Also,

$$\mathbf{T}(\mathbf{a} + \mathbf{c}, \mathbf{b}) = \mathbf{T}(\mathbf{a}, \mathbf{b}) + \mathbf{T}(\mathbf{c}, \mathbf{b}), \quad \text{and} \quad \mathbf{T}(\mathbf{a}, \mathbf{b} + \mathbf{d}) = \mathbf{T}(\mathbf{a}, \mathbf{b}) + \mathbf{T}(\mathbf{a}, \mathbf{d}) .$$

So the result *must* involve at least the product of a component of **a** with a component of **b**. Let's say the tensor takes a^2b^1 as that product, and additionally multiplies it by a constant, T_{21}. Then we have built a tensor acting on **a** and **b**, and it is linear in both:

$$\mathbf{T}(\mathbf{a},\mathbf{b}) = T_{21}a^2b^1. \qquad \text{Example:} \quad \mathbf{T}(\mathbf{a},\mathbf{b}) = 7a^2b^1 .$$

But, if we add to this some *other* weighted product of some *other* pair of components, the result is still a tensor: it is still linear in both **a** and **b**:

$$\mathbf{T}(\mathbf{a},\mathbf{b}) = T_{13}a^1b^3 + T_{21}a^2b^1. \qquad \text{Example:} \quad \mathbf{T}(\mathbf{a},\mathbf{b}) = 4a^1b^3 + 7a^2b^1 .$$

In fact, we can extend this to the weighted sum of *all* combinations of components, one each from **a** and **b**. Such a sum is still linear in both **a** and **b**:

$$\mathbf{T}(\mathbf{a},\mathbf{b}) = \sum_{i=1}^{3}\sum_{j=1}^{3} T_{ij}a^ib^j \qquad \text{Example:} \quad T_{ij} = \begin{bmatrix} -2 & 6 & 4 \\ 7 & 5 & -1 \\ -6 & 0 & 8 \end{bmatrix}.$$

Further, nothing else can be added to this that is linear in **a** and **b**.

> A tensor is the most general linear function of **a** and **b** that exists,
> i.e. any linear function of **a** and **b** can be written as a 3×3 matrix.

(We'll see that the **rank** of a tensor is equal to the number of its indices; **T** is a rank-2 tensor.) The T_{ij} are the **components** of the tensor (in the basis of the vectors **a** and **b**.) At this point, we consider the components of **T**, **a**, and **b** all as just numbers.

Why does a tensor have a separate weight for each *combination* of components, one from each input mathematical vector element (MVE)? Couldn't we just weight each input MVE as a whole? No, because that would restrict tensors to only *some* linear functions of the inputs.

> Any linear function of the input vectors can be represented as a tensor.

Note that tensors, just like vectors, can be written as components in some basis. And just like vectors, we can transform the components from one basis to another. Such a transformation does *not* change the tensor itself (nor does it change a vector); it simply changes how we represent the tensor (or vector). More on transformations later.

Tensors don't have to produce scalar results!

Some tensors accept one or more vectors, and produce a vector for a result. Or they produce some rank-*r* tensor for a result. In general, a rank-*n* tensor accepts '*m*' vectors as inputs, and produces a rank '*n*–*m*' tensor as a result. Since any tensor is an element of a mathematical vector space, tensors can be written as linear combinations of other (same rank & type) tensors. So even when a tensor produces another (lower rank) tensor as an output, the tensor is still a linear function of all its input vectors. It's just a tensor-valued function, instead of a scalar-valued function. For example, the force on a charge: a B-field operates on a vector, q**v**, to produce a vector, **f**. Thus, we can think of the B-field as a rank-2 tensor which acts on a vector to produce a vector; it's a vector-valued function of one vector.

Also, in general, tensors aren't limited to taking just vectors as inputs. Some tensors take rank-2 tensors as inputs. For example, the quadrupole moment tensor operates on the 2nd derivative matrix of the potential (the rank-2 "Hessian" tensor) to produce the (scalar) work stored in the quadrupole of charges. And a density matrix in quantum mechanics is a rank-2 tensor that acts on an operator matrix (rank-2 tensor) to produce the ensemble average of that operator.

Tensors in Action

Let's consider how rank-0, rank-1, and rank-2 tensors operate on a single vector. Recall that in "tensor-talk," a scalar is an invariant number, i.e. it is the same number in any coordinate system.

Rank-0: A rank-0 tensor is a scalar, i.e. a coordinate-system-independent number. Multiplying a vector by a rank-0 tensor (a scalar), produces a new vector. Each component of the vector contributes to the corresponding component of the result, and each component is weighted *equally* by the scalar, *a*:

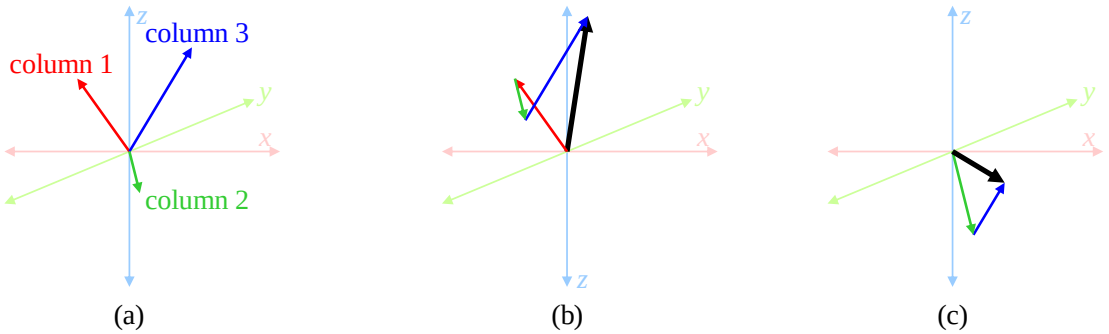
$$\vec{v} = v^x\mathbf{i} + v^y\mathbf{j} + v^z\mathbf{k} \qquad \Rightarrow \qquad a\vec{v} = av^x\mathbf{i} + av^y\mathbf{j} + av^z\mathbf{k}\ .$$

Rank-1: A rank-1 tensor **a** operates on (contracts with) a vector to produce a scalar. Each component of the input vector contributes a *number* to the result, but each component is weighted *separately* by the corresponding component of the tensor **a**:

$$\tilde{\mathbf{a}}(\mathbf{v}) = a_x v^x + a_y v^y + a_z v^z = \sum_{i=1}^{3} a_i v^i\ .$$

Note that a vector is itself a rank-1 tensor. Above, instead of considering **a** acting on **v**, we can equivalently consider that **v** acts on **a**: **a**(**v**) = **v**(**a**). Both **a** and **v** are of equal standing.

Rank-2: Filling one slot of a rank-2 tensor with a vector produces a new vector. Each component of the input vector contributes a *vector* to the result, and each input vector component weights a *different* vector.



(a) (b) (c)

(a) A hypothetical rank-2 tensor with an *x*-vector (red), a *y*-vector (green), and a *z*-vector (blue). (b) The tensor acting on the vector (1, 1, 1) producing a vector (heavy black). Each component (column) vector of the tensor is weighted by 1, and summed. (c) The tensor acting on the vector (0, 2, 0.5),

producing a vector (heavy black). The *x*-vector is weighted by 0, and so does not contribute; the *y*-vector is weighted by 2, so contributes double; the *z*-vector is weighted by 0.5, so contributes half.

$$\mathbf{B}(\_,\mathbf{v}) = B^i{}_j v^j = \begin{bmatrix} B^x{}_x & B^x{}_y & B^x{}_z \\ B^y{}_x & B^y{}_y & B^y{}_z \\ B^z{}_x & B^z{}_y & B^z{}_z \end{bmatrix} \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} = v^x \begin{bmatrix} B^x{}_x \\ B^y{}_x \\ B^z{}_x \end{bmatrix} + v^y \begin{bmatrix} B^x{}_y \\ B^y{}_y \\ B^z{}_y \end{bmatrix} + v^z \begin{bmatrix} B^x{}_z \\ B^y{}_z \\ B^z{}_z \end{bmatrix}$$

$$= \mathbf{B}_x v^x + \mathbf{B}_y v^y + \mathbf{B}_z v^z = \left(\sum_{j=1}^{3} B^x{}_j v^j \right) \mathbf{i} + \left(\sum_{j=1}^{3} B^y{}_j v^j \right) \mathbf{j} + \left(\sum_{j=1}^{3} B^z{}_j v^j \right) \mathbf{k}$$

The columns of **B** are the vectors which are weighted by each of the input vector components, v^j; or equivalently, the columns of B are the vector weights for each of the input vector components

Example of a simple rank-2 tensor: the moment-of-inertia tensor, I_{ij}. Every blob of matter has one. We know from mechanics that if you rotate an arbitrary blob around an arbitrary axis, the angular momentum vector of the blob does *not* in general line up with the axis of rotation. So what is the angular momentum vector of the blob? It is a vector-valued linear function of the angular velocity vector, i.e. given the angular velocity vector, you can operate on it with the moment-of-inertia tensor, to get the angular momentum vector. Therefore, by the definition of a tensor as a linear operation on a vector, the relationship between angular momentum vector and angular velocity vector can be given as a tensor; it is the moment-of-inertia tensor. It takes as an input the angular velocity vector, and produces as output the angular momentum vector, therefore it is a rank-2 tensor:

$$\mathbf{I}(\boldsymbol{\omega},\_) = \mathbf{L}, \qquad \mathbf{I}(\boldsymbol{\omega},\boldsymbol{\omega}) = \mathbf{L}\cdot\boldsymbol{\omega} = 2KE\,.$$

[Since **I** is constant in the blob frame, it rotates in the lab frame. Thus, in the lab frame, the above equations are valid only at a single instant in time. In effect, **I** is a function of time, **I**(*t*).]

[?? This may be a bad example, since **I** is only a **Cartesian tensor** [L&L3, p ??], which is not a real tensor. Real tensors can't have finite displacements on a curved manifold, but blobs of matter have finite size. If you want to get the kinetic energy, you have to use the metric to compute **L**·**ω**. Is there a simple example of a real rank-2 tensor??]

Note that some rank-2 tensors operate on *two* vectors to produce a scalar, and some (like **I**) can either act on one vector to produce a vector, or act on two vectors to produce a scalar (twice the kinetic energy). More of that, and higher rank tensors, later.

Tensor Fields

A vector is a single mathematical object, but it is quite common to define a *field* of vectors. A **field** in this sense is a function of space. A **vector field** defines a vector for each point in a space. For example, the electric field is a vector-valued function of space: at each point in space, there is an electric field vector.

Similarly, a tensor is a single mathematical object, but it is quite common to define a *field* of tensors. At each point in space, there is a tensor. The metric tensor field is a tensor-valued function of space: at each point, there is a metric tensor. Almost universally, the word "field" is omitted when calling out tensor fields: when you say "metric tensor," everyone is expected to know it is a tensor *field*. When you say "moment of inertia tensor," everyone is expected to know it is a single tensor (not a field).

Dot Products and Cross Products as Tensors

Symmetric tensors are associated with elementary dot products, and anti-symmetric tensors are associated with elementary cross-products.

A dot product is a linear operation on two vectors: $\mathbf{A}\cdot\mathbf{B} = \mathbf{B}\cdot\mathbf{A}$, which produces a scalar. Because the dot product is a linear function of two vectors, it can be written as a tensor. (Recall that *any* linear function of vectors can be written as a tensor.) Since it takes two rank-1 tensors, and produces a rank-0 tensor, the dot product is a rank-2 tensor. Therefore, we can achieve the same result as a dot product with a rank-2 symmetric tensor that accepts two vectors and produces a scalar; call this tensor **g**:

$$\mathbf{g}(\mathbf{A}, \mathbf{B}) = \mathbf{g}(\mathbf{B}, \mathbf{A}) .$$

'**g**' is called the **metric tensor**: it produces the dot product (aka scalar product) of two vectors. Quite often, the metric tensor varies with position (i.e., it is a function of the generalized coordinates of the system); then it is a metric tensor *field*. It happens that the dot product is symmetric: $\mathbf{A}\cdot\mathbf{B} = \mathbf{B}\cdot\mathbf{A}$.; therefore, **g** is symmetric. If we write the components of **g** as a matrix, the matrix will be symmetric, i.e. it will equal its own transpose. (Do I need to expand on this??)

On the other hand, a cross product is an anti-symmetric linear operation on two vectors, which produces another vector: $\mathbf{A}\times\mathbf{B} = -\mathbf{B}\times\mathbf{A}$. Therefore, we can associate one vector, say **B**, with a rank-2 anti-symmetric tensor, that accepts one vector and produces another vector:

$$\mathbf{B}(\_, \mathbf{A}) = -\mathbf{B}(\mathbf{A}, \_) .$$

For example, the Lorentz force law: $\mathbf{F} = \mathbf{v}\times\mathbf{B}$. We can write **B** as a $\binom{1}{1}$ tensor:

$$\mathbf{F} = \mathbf{v}\times\mathbf{B} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ v^x & v^y & v^z \\ B_x & B_y & B_z \end{vmatrix} = \mathbf{B}(\_, \mathbf{v}) = B^i{}_j v^j = \begin{bmatrix} 0 & B_z & -B_y \\ -B_z & 0 & B_x \\ B_y & -B_x & 0 \end{bmatrix} \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} = \begin{bmatrix} B_z v^y - B_y v^z \\ -B_z v^x + B_x v^z \\ B_y v^x - B_x v^y \end{bmatrix} .$$

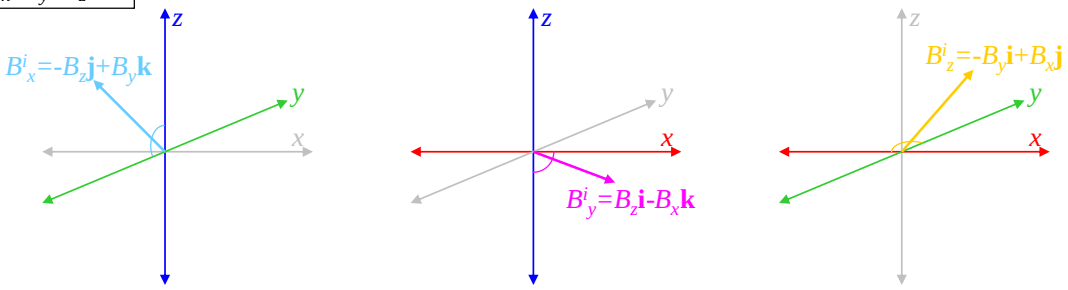
We see again how a rank-2 tensor, **B**, contributes a *vector* for each *component* of **v**:

$B^i{}_x\,\mathbf{e}_i = -B_z\mathbf{j} + B_y\mathbf{k}$ (the first column of B) is weighted by v^x.

$B^i{}_y\,\mathbf{e}_i = B_z\mathbf{i} - B_x\mathbf{k}$ (the 2nd column of B) is weighted by v^y.

$B^i{}_z\,\mathbf{e}_i = -B_y\mathbf{i} + B_x\mathbf{j}$ (the 3rd column of B) is weighted by v^z.

$B_x, B_y, B_z > 0$



A rank-2 tensor acting on a vector to produce their cross-product.

TBS: We can also think of the cross product as a fully anti-symmetric rank-3 tensor, which acts on 2 vectors to produce a vector (their cross product). This is the **anti-symmetric symbol** ε_{ijk} (not a tensor).

Note that both the dot product and cross-product are linear on both of their operands. For example:

$$(\alpha\mathbf{A} + \gamma\mathbf{C})\cdot\mathbf{B} = \alpha(\mathbf{A}\cdot\mathbf{B}) + \gamma(\mathbf{C}\cdot\mathbf{B})$$

$$\mathbf{A}\cdot(\beta\mathbf{B} + \eta\mathbf{D}) = \beta(\mathbf{A}\cdot\mathbf{B}) + \eta(\mathbf{A}\cdot\mathbf{D})$$

Linearity in all the operands is the essence of a tensor.

Note also that a "rank" of a tensor **contracts** with (is summed over) a "rank" of one of its operands to eliminate both of them: one rank of the B-field tensor contracts with one input vector, leaving one surviving rank of the B-field tensor, which is the vector result. Similarly, one rank of the metric tensor, **g**, contracts with the first operand vector; another rank of **g** contracts with the second operand vector, leaving a rank-0 (scalar) result.

The Danger of Matrices

There are some dangers to thinking of tensors as matrices: (1) it doesn't work for rank 3 or higher tensors, and (2) non-commutation of matrix multiplication is harder to follow than the more-explicit summation convention. Nonetheless, the matrix conventions are these:

- contravariant components and basis covectors ("up" indexes) → column vector. E.g.,

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \qquad \text{basis 1-forms:} \quad \begin{bmatrix} \mathbf{e}^1 \\ \mathbf{e}^2 \\ \mathbf{e}^3 \end{bmatrix}.$$

- covariant components and basis contravectors ("down" indexes) → row vector

$$\mathbf{w} = \begin{pmatrix} w_1, w_2, w_3 \end{pmatrix}, \qquad \text{basis vectors: } \begin{pmatrix} \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \end{pmatrix}.$$

Matrix rows and columns are indicated by spacing of the indexes, and are independent of their "upness" or "downness." The first matrix index is always the row; the second, the column:

$$T^r{}_c \qquad T_r{}^c \qquad T_{rc} \qquad T^{rc} \qquad \textit{where} \quad r = \text{row index}, c = \text{column index}.$$

Reading Tensor Component Equations

Tensor equations can be written as equations with tensors as operators (written in bold):

KE = ½ **I**(**ω**, **ω**).

Or, they can be written in component form:

(1) $\text{KE} = \tfrac{1}{2} I_{ij}\, \omega^i\, \omega^j$.

We'll be using lots of tensor equations written in component form, so it is important to know how to read them. Note that some standard notations almost *require* component form: In GR, the Ricci tensor is $R^{\mu\nu}$, and the Ricci scalar is R:

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} R g_{\mu\nu}.$$

In component equations, tensor indexes are written explicitly. There are two kinds of tensor indexes: dummy (aka summation) indexes, and free indexes. Dummy indexes appear exactly twice in any term. Free indexes appear only once in each term, and the same free indexes must appear in each term (except for scalar terms). In the above equation, both μ and ν are free indexes, and there are no dummy indexes. In eq. (1) above, i and j are both dummy indexes and there are no free indexes.

Dummy indexes appear exactly twice in any term, and are used for implied summation, e.g.:

$$KE = \frac{1}{2} I_{ij} \omega^i \omega^j \quad \equiv \quad KE = \frac{1}{2} \sum_{i=1}^{3} \sum_{j=1}^{3} I_{ij} \omega^i \omega^j.$$

Free indexes are a shorthand for writing several equations at once. Each free index takes on all possible values for it. Thus,

$$C^i = A^i + B^i \quad \equiv \quad C^x = A^x + B^x, \quad C^y = A^y + B^y, \quad C^z = A^z + B^z \quad (3 \text{ equations}),$$

and

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} R g_{\mu\nu} \quad \equiv$$

$$G_{00} = R_{00} - \frac{1}{2} R g_{00}, \quad G_{01} = R_{01} - \frac{1}{2} R g_{01}, \quad G_{02} = R_{02} - \frac{1}{2} R g_{02}, \quad G_{03} = R_{03} - \frac{1}{2} R g_{03}$$

$$G_{10} = R_{10} - \frac{1}{2} R g_{10}, \quad G_{11} = R_{11} - \frac{1}{2} R g_{11}, \quad G_{12} = R_{12} - \frac{1}{2} R g_{12}, \quad G_{13} = R_{13} - \frac{1}{2} R g_{13}$$

$$G_{20} = R_{20} - \frac{1}{2} R g_{20}, \quad G_{21} = R_{21} - \frac{1}{2} R g_{21}, \quad G_{22} = R_{22} - \frac{1}{2} R g_{22}, \quad G_{23} = R_{23} - \frac{1}{2} R g_{23}$$

$$G_{30} = R_{30} - \frac{1}{2} R g_{30}, \quad G_{31} = R_{31} - \frac{1}{2} R g_{31}, \quad G_{32} = R_{32} - \frac{1}{2} R g_{32}, \quad G_{33} = R_{33} - \frac{1}{2} R g_{33}$$

(16 equations).

It is common to have both dummy and free indexes in the same equation. Thus the GR statement of conservation of energy and momentum uses μ as a dummy index, and ν as a free index:

$$\nabla_\mu T^{\mu\nu} = 0 \quad \equiv \quad \sum_{\mu=0}^{3} \nabla_\mu T^{\mu 0} = 0, \quad \sum_{\mu=0}^{3} \nabla_\mu T^{\mu 1} = 0, \quad \sum_{\mu=0}^{3} \nabla_\mu T^{\mu 2} = 0, \quad \sum_{\mu=0}^{3} \nabla_\mu T^{\mu 3} = 0 .$$

(4 equations). Notice that scalars apply to all values of free indexes, and don't need indexes of their own. However, any free indexes must match on all tensor terms. It is nonsense to write something like:

$$A^{ij} = B^i + C^j \qquad \text{(nonsense)} .$$

However, it is reasonable to have

$$A^{ij} = B^i C^j \qquad \text{E.g., angular momentum: } M^{ij} = r^i p^j - r^j p^i .$$

Adding, Subtracting, Differentiating Tensors

Since tensors are linear operations, you can add or subtract any two tensors that take the same type arguments and produce the same type result. Just add the tensor components individually.

$$\mathbf{S} = \mathbf{T} + \mathbf{U} \qquad E.g. \qquad S^{ij} = T^{ij} + U^{ij}, \quad i, j = 1,...N .$$

You can also scalar multiply a tensor. Since these properties of tensors are the defining requirements for a vector space, all the tensors of given rank and index types compose a vector space, and every tensor is an MVE in its space. This implies that:

A tensor field can be differentiated (or integrated), and in particular, it has a gradient.

Higher Rank Tensors

When considering higher rank tensors, it may be helpful to recall that multi-dimensional matrices can be thought of as lower-dimensional matrices with each element itself a vector or matrix. For example, a 3 x 3 matrix can be thought of as a "column vector" of 3 row-vectors. Matrix multiplication works out the same whether you consider the 3 x 3 matrix as a 2-D matrix of numbers, or a 1-D column vector of row vectors:

$$\begin{pmatrix} x & y & z \end{pmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{pmatrix} ax + dy + gz & bx + ey + hz & cx + fy + iz \end{pmatrix}$$

or

$$\begin{pmatrix} x & y & z \end{pmatrix} \begin{bmatrix} (a,b,c) \\ (d,e,f) \\ (g,h,i) \end{bmatrix} = x(a,b,c) + y(d,e,f) + z(g,h,i) = \begin{pmatrix} ax + dy + gz & bx + ey + hz & cx + fy + iz \end{pmatrix}$$

Using this same idea, we can compare the gradient of a scalar field, which is a $({}^0{}_1)$ tensor field (a 1-form), with the gradient of a rank-2 (say $({}^0{}_2)$) tensor field, which is a $({}^0{}_3)$ tensor field. First, the gradient of a scalar field is a $({}^0{}_1)$ tensor field with 3 components, where each component is a number-valued function:

$$\nabla f = \mathbf{D} = \frac{\partial f}{\partial x}\boldsymbol{\omega}^1 + \frac{\partial f}{\partial y}\boldsymbol{\omega}^2 + \frac{\partial f}{\partial z}\boldsymbol{\omega}^3, \qquad \boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3 \text{ are basis (co)vectors}$$

$$\mathbf{D} \text{ can be written as } (D_1, D_2, D_3), \text{ where} \qquad D_1 = \frac{\partial f}{\partial x}, \quad D_2 = \frac{\partial f}{\partial y}, \quad D_3 = \frac{\partial f}{\partial z}.$$

The gradient operates on an infinitesimal displacement vector to produce the change in the function when you move through the given displacement: $df = \mathbf{D}(d\mathbf{r}) = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial z}dz$.

Now let $\mathbf{R}$ be a $({}^0{}_2)$ tensor field, and $\mathbf{T}$ be its gradient. $\mathbf{T}$ is a $({}^0{}_3)$ tensor field, but can be thought of as a $({}^0{}_1)$ tensor field where each component is itself a $({}^0{}_2)$ tensor.

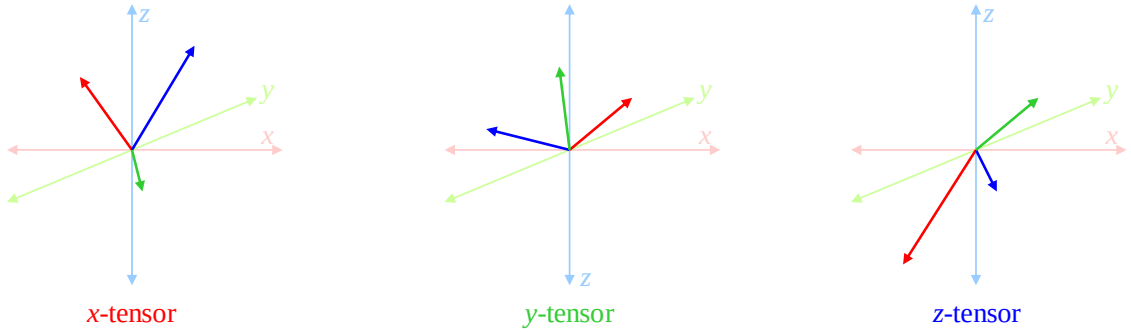
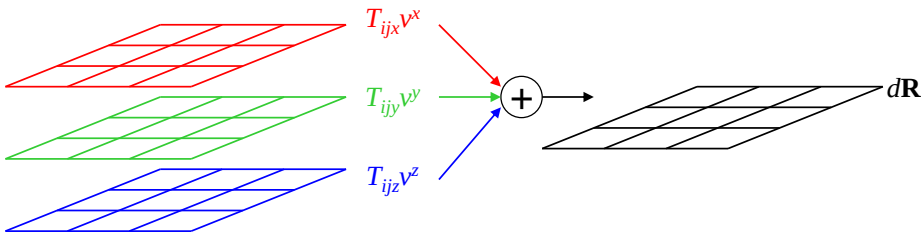


Figure 18.1 A rank-3 tensor considered as a set of 3 rank-2 tensors: an *x*-tensor, a *y*-tensor, and a *z*-tensor.

The gradient operates on an infinitesimal displacement vector to produce the change in the $({}^0{}_2)$ tensor field when you move through the given displacement.

$$\mathbf{T} = \nabla\mathbf{R} = \frac{\partial \mathbf{R}}{\partial x}\boldsymbol{\omega}^1 + \frac{\partial \mathbf{R}}{\partial y}\boldsymbol{\omega}^2 + \frac{\partial \mathbf{R}}{\partial z}\boldsymbol{\omega}^3$$

$$= \left(\begin{bmatrix} T_{11x} & T_{12x} & T_{13x} \\ T_{21x} & T_{22x} & T_{23x} \\ T_{31x} & T_{32x} & T_{33x} \end{bmatrix} \begin{bmatrix} T_{11y} & T_{12y} & T_{13y} \\ T_{21y} & T_{22y} & T_{23y} \\ T_{31y} & T_{32y} & T_{33y} \end{bmatrix} \begin{bmatrix} T_{11z} & T_{12z} & T_{13z} \\ T_{21z} & T_{22z} & T_{23z} \\ T_{31z} & T_{32z} & T_{33z} \end{bmatrix} \right).$$



$$d\mathbf{R} = \mathbf{T}(\mathbf{v}) = \sum_{k=x,y,z} T_{ijk}v^k \qquad \leftrightarrow \qquad (dR)_{ij} = T_{ijk}v^k \ .$$

Note that if **R** had been a $\binom{2}{0}$ (fully *contravariant*) tensor, then its gradient would be a $\binom{2}{1}$ mixed tensor. Taking the gradient of any field simply adds a covariant index, which can then be contracted with a displacement vector to find the change in the tensor field when moving through the given displacement.

The contraction considerations of the previous section still apply: a rank of an tensor operator contracts with a rank of one of its inputs to eliminate both. In other words, each rank of input tensors eliminates one rank of the tensor operator. The rank of the result is the number of surviving ranks from the tensor operator:

$$\text{rank}(\textit{tensor}) = \left(\sum \text{rank}(\textit{inputs})\right) + \text{rank}(\textit{result})$$

or $$\text{rank}(\textit{result}) = \text{rank}(\textit{tensor}) - \left(\sum \text{rank}(\textit{inputs})\right).$$

Tensors of Mathematical Vector Elements: The operation of a tensor on vectors involves multiplying components (one from the tensor, and one from each input vector), and then summing. E.g.,

$$\mathbf{T}(\mathbf{a},\mathbf{b}) = T_{11}a^1b^1 + ... + T_{ij}a^ib^j + ... \ .$$

Similar to the above example, the T_{ij} *components* could themselves be a vector of a mathematical vector space (i.e., could be MVEs), while the a^i and b^j components are scalars of that vector space. In the example above, we could say that each of the $T_{ij;x}$, $T_{ij;y}$, and $T_{ij;z}$ is a rank-2 tensor (an MVE in the space of rank-2 tensors), and the components of **v** are scalars in that space (in this case, real numbers).

Tensors In General

In complete generality then, a tensor **T** is a linear operation on one or more MVEs:
T(**a**, **b**, ...).

Linearity implies that **T** can be written as a numeric weight for each *combination of* components, one component from each input MVE. Thus, the "linear operation" performed by **T** is equivalent to a weighted sum of all combinations of components of the input MVEs. (Since **T** and the **a**, **b**, ... are simple objects, not functions, there is no concept of derivative or integral operations. Derivatives and integrals are linear operations *on functions*, but not linear functions of MVEs.)

Given the components of the inputs **a**, **b**, ..., and the components of **T**, we can contract **T** with (operate with **T** on) the inputs to produce a MVE result. Note that all input MVEs have to have the same basis. Also, **T** may have units, so the output units are arbitrary. Note that in generalized coordinates, different components of a tensor may have different units (much like the vector parameters r and θ have different units).

Change of Basis: Transformations

Since tensors are linear operations on MVEs, we can represent a tensor by components. If we know a tensor's operations on all combinations of basis vectors, we have fully defined the tensor. Consider a rank-2 tensor **T** acting on two vectors, **a** and **b**. We expand **T**, **a**, and **b** into components, using the linearity of the tensor:

$$\begin{aligned}
\mathbf{T}(\mathbf{a},\mathbf{b}) &= \mathbf{T}(a^1\mathbf{i} + a^2\mathbf{j} + a^3\mathbf{k}, b^1\mathbf{i} + b^2\mathbf{j} + b^3\mathbf{k}) \\
&= a^1b^1\mathbf{T}(\mathbf{i},\mathbf{i}) + a^2b^1\mathbf{T}(\mathbf{j},\mathbf{i}) + a^3b^1\mathbf{T}(\mathbf{k},\mathbf{i}) \\
&\quad + a^1b^2\mathbf{T}(\mathbf{i},\mathbf{j}) + a^2b^2\mathbf{T}(\mathbf{j},\mathbf{j}) + a^3b^2\mathbf{T}(\mathbf{k},\mathbf{j}) \\
&\quad + a^1b^3\mathbf{T}(\mathbf{i},\mathbf{k}) + a^2b^3\mathbf{T}(\mathbf{j},\mathbf{k}) + a^3b^3\mathbf{T}(\mathbf{k},\mathbf{k})
\end{aligned}$$

Define $T_{ij} = \mathbf{T}(\mathbf{e}_i, \mathbf{e}_j)$, *where* $\mathbf{e}_1 = \mathbf{i},\ \mathbf{e}_2 = \mathbf{j},\ \mathbf{e}_3 = \mathbf{k}$

then $$\mathbf{T}(\mathbf{a},\mathbf{b}) = \sum_{i=1}^{3}\sum_{j=1}^{3} a^ib^j\mathbf{T}(\mathbf{e}_i,\mathbf{e}_j) = \sum_{i=1}^{3}\sum_{j=1}^{3} T_{ij}a^ib^j$$

The tensor's values on all combinations of input basis vectors are the **components** of the tensor (in the basis of the input vectors.)

Now let's transform **T** to another basis. To change from one basis to another, we need to know how to find the new basis vectors from the old ones, or equivalently, how to transform components in the old basis to components in the new basis. We write the new basis with primes, and the old basis without primes.

Because vector spaces demand linearity, any change of basis can be written as a linear transformation of the basis vectors or components, so we can write (eq. #s from Talman):

$$\mathbf{e}'_i = \sum_{k=1}^{N} \Lambda^k{}_i \mathbf{e}_k = \Lambda^k{}_i \mathbf{e}_k \qquad \text{[Tal 2.4.5]}$$

$$v'^i = \sum_{k=1}^{N} \left(\Lambda^{-1}\right)^i{}_k \, v^k = \left(\Lambda^{-1}\right)^i{}_k \, v^k \qquad \text{[Tal 2.4.8]}$$

where the last form uses the summation convention. There is a very important difference between equations 2.4.5 and 2.4.8. The first is a set of 3 *vector* equations, expressing each of the *new* basis vectors in the *old* basis

Aside: Let's look more closely at the difference between equations 2.4.5 and 2.4.8. The first is a set of 3 *vector* equations, expressing each of the *new* basis vectors in the *old* basis. Basis vectors are vectors, and hence can themselves be expressed in any basis:

$$\left.\begin{aligned} \mathbf{e}'_1 &= \Lambda^1{}_1 \mathbf{e}_1 + \Lambda^2{}_1 \mathbf{e}_2 + \Lambda^3{}_1 \mathbf{e}_3 \\ \mathbf{e}'_2 &= \Lambda^1{}_2 \mathbf{e}_1 + \Lambda^2{}_2 \mathbf{e}_2 + \Lambda^3{}_2 \mathbf{e}_3 \\ \mathbf{e}'_3 &= \Lambda^1{}_3 \mathbf{e}_1 + \Lambda^2{}_3 \mathbf{e}_2 + \Lambda^3{}_3 \mathbf{e}_3 \end{aligned}\right\} \qquad \textit{or more simply} \qquad \left\{\begin{aligned} \mathbf{e}'_1 &= a^1 \mathbf{e}_1 + a^2 \mathbf{e}_2 + a^3 \mathbf{e}_3 \\ \mathbf{e}'_2 &= b^1 \mathbf{e}_1 + b^2 \mathbf{e}_2 + b^3 \mathbf{e}_3 \\ \mathbf{e}'_3 &= c^1 \mathbf{e}_1 + c^2 \mathbf{e}_2 + c^3 \mathbf{e}_3 \end{aligned}\right.$$

where the *a*'s are the components of $\mathbf{e}'_1$ in the old basis, the *b*'s are the components of $\mathbf{e}'_2$ in the old basis, and the *c*'s are the components of $\mathbf{e}'_3$ in the old basis.

In contrast, equation 2.4.8 is a set of 3 *number* equations, relating the components of a *single vector*, taking its *old* components into the *new* basis. In other words, in the first equation, we are taking new basis vectors and expressing them in the old basis (new → old). In the second equation, we are taking old components and converting them to the new basis (old → new). The two equations go in opposite directions: the first takes new to old, the second takes old to new. So it is natural that the two equations use inverse matrices to achieve those conversions. However, because of the inverse matrices in these equations, vector *components* are said to transform "contrary" (oppositely) to basis *vectors*, so they are called contravariant vectors.

I think it is misleading to say that contravariant vectors transform "oppositely" to basis vectors. In fact, that is impossible. Basis vectors are contravectors, and transform like any other contravector. A vector of (1, 0, 0) (in some basis) *is* a basis vector. It may also happen to be the value of some physical vector. In both cases, the expression of the vector (1, 0, 0) (old basis) in the new-basis is the same.

Now we can use 2.4.5 to evaluate the components of **T** in the primed basis:

$$T'_{ij} = T(\mathbf{e}'_i, \mathbf{e}'_j) = T(\Lambda^k{}_i \mathbf{e}_k, \Lambda^l{}_j \mathbf{e}_l) = \sum_{k=1}^{N}\sum_{l=1}^{N} \Lambda^k{}_i \Lambda^l{}_j T(\mathbf{e}_k, \mathbf{e}_l) = \sum_{k=1}^{N}\sum_{l=1}^{N} \Lambda^k{}_i \Lambda^l{}_j T_{kl} \ .$$

Notice that there is one use of the transformation matrix Λ for each index of **T** to be transformed.

Matrix View of Basis Transformation

The concept of tensors seems clumsy at first, but it's a very fundamental concept. Once you get used to it, tensors are essentially simple things (though it took me 3 years to understand how "simple" they are). The rules for transformations are pretty direct. Transforming a rank-*n* tensor requires using the transformation matrix *n* times. A vector is rank-1, and transforms by a simple matrix multiply, or in tensor terms, by a summation over indices. Here, since we must distinguish row basis from column basis, we put the primes on the indices, to indicate which index is in the new basis, and which is in the old basis.

$$\mathbf{a'} = \mathbf{\Lambda a} \qquad \leftrightarrow \qquad \begin{bmatrix} a^{0'} \\ a^{1'} \\ a^{2'} \\ a^{3'} \end{bmatrix} = \begin{pmatrix} \Lambda^{0'0} & \Lambda^{0'1} & \Lambda^{0'2} & \Lambda^{0'3} \\ \Lambda^{1'0} & \Lambda^{1'1} & \Lambda^{1'2} & \Lambda^{1'3} \\ \Lambda^{2'0} & \Lambda^{2'1} & \Lambda^{2'2} & \Lambda^{2'3} \\ \Lambda^{3'0} & \Lambda^{3'1} & \Lambda^{3'2} & \Lambda^{3'3} \end{pmatrix} \begin{bmatrix} a^0 \\ a^1 \\ a^2 \\ a^3 \end{bmatrix} \qquad \leftrightarrow \qquad a^{\mu'} = \Lambda^{\mu'}{}_{\nu} a^{\nu} .$$

Notice that when you sum over (contract over) two indices, they disappear, and you're left with the unsummed index. So above when we sum over old-basis indices, we're left with a new-basis vector.

Rank-2 example: The electromagnetic field tensor **F** is rank-2, and transforms using the transformation matrix twice, by two summations over indices, transforming both stress-energy indices. This is clumsy to write in matrix terms, because you have to use the transpose of the transformation matrix to transform the rows; this transposition has no physical significance. In the rank-2 (or higher) case, the tensor notation is both simpler, and more physically meaningful:

$$\mathbf{F'} = \mathbf{\Lambda F \Lambda}^T \qquad \leftrightarrow$$

$$\begin{bmatrix} F^{0'0'} & F^{0'1'} & F^{0'2'} & F^{0'3'} \\ F^{1'0'} & F^{1'1'} & F^{1'2'} & F^{1'3'} \\ F^{2'0'} & F^{2'1'} & F^{2'2'} & F^{2'3'} \\ F^{3'0'} & F^{3'1'} & F^{3'2'} & F^{3'3'} \end{bmatrix} = \begin{pmatrix} \Lambda^{0'0} & \Lambda^{0'1} & \Lambda^{0'2} & \Lambda^{0'3} \\ \Lambda^{1'0} & \Lambda^{1'1} & \Lambda^{1'2} & \Lambda^{1'3} \\ \Lambda^{2'0} & \Lambda^{2'1} & \Lambda^{2'2} & \Lambda^{2'3} \\ \Lambda^{3'0} & \Lambda^{3'1} & \Lambda^{3'2} & \Lambda^{3'3} \end{pmatrix} \begin{bmatrix} F^{00} & F^{01} & F^{02} & F^{03} \\ F^{10} & F^{11} & F^{12} & F^{13} \\ F^{20} & F^{21} & F^{22} & F^{23} \\ F^{30} & F^{31} & F^{32} & F^{33} \end{bmatrix} \begin{pmatrix} \Lambda^{0'0} & \Lambda^{1'0} & \Lambda^{2'0} & \Lambda^{3'0} \\ \Lambda^{0'1} & \Lambda^{1'1} & \Lambda^{2'1} & \Lambda^{3'1} \\ \Lambda^{0'2} & \Lambda^{1'2} & \Lambda^{2'2} & \Lambda^{3'2} \\ \Lambda^{0'3} & \Lambda^{1'3} & \Lambda^{2'3} & \Lambda^{3'3} \end{pmatrix}$$

$$\leftrightarrow \qquad F^{\mu'\nu'} = \Lambda^{\mu'}{}_{\nu} \Lambda^{\nu'}{}_{\nu} F^{\mu\nu}$$

In general, you have to transform every index of a tensor, each index requiring one use of the transformation matrix.

Geometric (Coordinate-Free) Dot and Cross Products

Coordinate-free methods use definitions in physical or geometric terms, without reference to any particular coordinates. For example, in physics, the product of the *parallel components* of two vectors has direct physical meaning and application. Similarly, the familiar vector cross product also has direct physical meaning. We here introduce some rudiments of coordinate-free methods, and how they *lead to* the familiar formulas for dot and cross products. From the geometric definitions of cross and dot products, we show this crucial property of both:

The dot product and cross product are linear.

The problem with coordinate-based definitions is they give no *insight*.

Dot Product: Parallel Components

We define the product of parallel components of two vectors as the **dot product** (Figure 18.2a):

dot product The dot product of two vectors is the product of their parallel components, which is a scalar.

All properties of the dot product follow from this geometric definition, including the well-known formula for ortho-normal rectangular coordinates:

$$\mathbf{a} \bullet \mathbf{c} = a_x c_x + a_y c_y + a_z c_z .$$

This formula is *not* a definition; it is a *consequence* of the coordinate-free definition.

Figure 18.2b shows graphically that the dot product is distributive in two dimensions (2D). Because this proof is coordinate-free, it is true even in oblique (non-orthogonal) non-normal coordinates. The full proof of bilinearity includes showing that the dot product is also commutative, and commutes with scalar multiplication:

$$\mathbf{a} \bullet \mathbf{c} = \mathbf{c} \bullet \mathbf{a}, \qquad (k\mathbf{a}) \bullet \mathbf{c} = k(\mathbf{a} \bullet \mathbf{c}) .$$

We leave that as an exercise. (We examine cross products below, but there can be no cross-product in only two dimensions.)

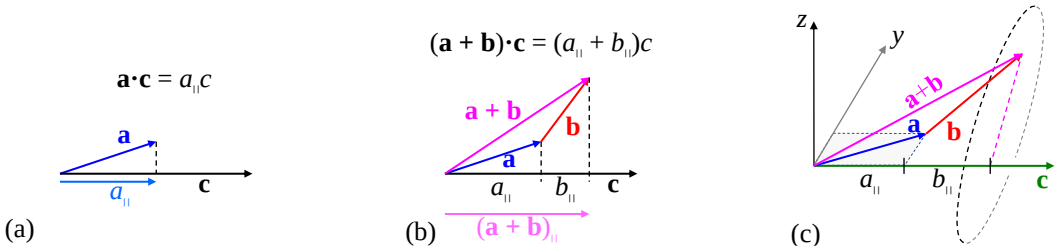


Figure 18.2 (a) Dot product is the product of parallel components. (b) Coordinate-free proof that the dot product is distributive in 2D. (c) Proof that the dot product is distributive in 3D.

It's a little harder to visualize, but it is crucially important that we can show linearity in 3D as well, using coordinate free methods. This linearity is what *justifies* the (coordinate *dependent*) algebraic formulas for dot product (and cross product); we cannot *start* with those formulas, and use them to prove linearity.

For the dot product in 3D, consider $(\mathbf{a}+\mathbf{b})\cdot\mathbf{c}$, shown in Figure 18.3c. We can always choose a *y*-axis such that the *c*-*y* plane contains both **a** and **c**. **b**, however, points partly upward (say) above the *c*-*y* plane. To construct the component of **a** (or **b**) parallel to **c**, construct a plane perpendicular to **c**, and containing the tip of **a** (or **b**). Thus the sum of the parallel components equals the parallel component of the sum. Therefore, the dot product is linear (in its first factor). Since dot product is commutative, it must be linear in its second factor, as well. Therefore, the dot product is bilinear.

Cross Product: Perpendicular Components

Similarly, though a little more complicated, we can define the **cross product** in coordinate free terms (Figure 18.3a):

cross product — The cross product of two vectors, $\mathbf{c}\times\mathbf{a}$, is a vector whose magnitude is the product of **c**'s and **a**'s perpendicular components, and whose direction is perpendicular to both **c** and **a**, and oriented with a right-hand screw sense from **c** to **a**.

Figure 18.3 (a) Construction of a cross product. (b) The cross product of a sum: The projection of **b** slides its tip to the *y*-*z* plane.

For the cross-product, first consider the geometric construction of a simple product $\mathbf{c}\times\mathbf{a}$ (Figure 18.3,a). We project **a** into the plane perpendicular to **c** and containing **c**'s tail (the *y*-*z* plane). This yields $\mathbf{a}_\perp$, which is a *vector*. We then rotate $\mathbf{a}_\perp$ a quarter turn (90 degrees) around **c**, in a direction given by the right-hand-rule. This vector points in the direction of $\mathbf{c}\times\mathbf{a}$. Multiply its length by the magnitude of **c** to get $\mathbf{c}\times\mathbf{a}$.

The right-hand rule implies that the cross product is anti-commutative: $\mathbf{c}\times\mathbf{a} = -\mathbf{a}\times\mathbf{c}$.

Now repeat this process for the product $\mathbf{c}\times(\mathbf{a}+\mathbf{b})$ (Figure 18.3b). We start by projecting **a** and **b** onto the *y*-*z* plane. As shown, the (vector) sum of the projections equals the projection of the sum. Now we must rotate the projections about **c** by 90 degrees. Rotation is a linear operator, so the sum of the rotations equals

the rotation of the sum. Hence, the cross product is linear (in the second factor). Since cross product is anti-commutative, it must be linear in the first factor, as well. Cross product is bilinear.

Non-Orthonormal Systems: Contravariance and Covariance

Many systems *cannot* be represented with orthonormal coordinates, e.g. the (surface of a) sphere. Dealing with non-orthonormality *requires* a more sophisticated view of tensors, and introduces the concepts of contravariance and covariance. We start with an introduction to coordinate-free methods.

Dot Products in Oblique Coordinates

Oblique coordinates (non-orthogonal axes) appear in many areas of physics and engineering, such as generalized coordinates in classical mechanics, and in the differential geometry of relativity. Understanding how to compute dot products in oblique coordinates is the foundation for many physically meaningful computations, and for the mathematics of contravariant and covariant components of a vector. The "usual" components of a vector are the ones called the "contravariant" components.

We here give several views of dot products and metric tensors. Then, we define the "covariant" components of a vector, and show why they are useful (and unavoidable). Finally, we show that a gradient is "naturally" covariant. To illustrate, we use a two-dimensional manifold, which is the archetype of all higher-dimensional generalizations. This section uses Einstein summation, and makes reference to tensors, but you need not understand tensors to follow it. In fact, this is a step on the road to understanding tensors.

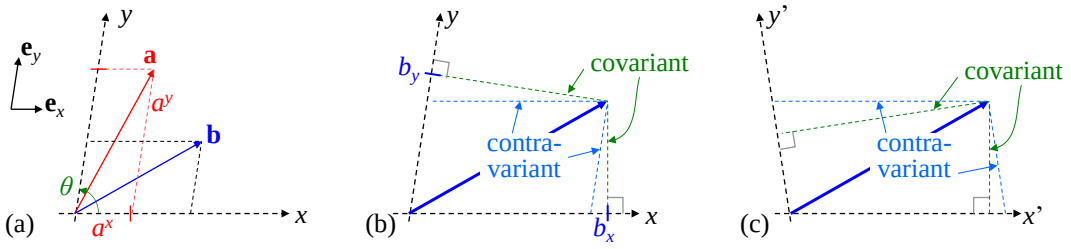


Figure 18.4 (a) Two vectors in oblique coordinates. (b) Geometric meaning of contravariant and covariant components of a vector. (c) Example of contravariant and covariant components in a different basis.

In oblique coordinates, we still write a vector as a sum of components (Figure 18.4a):

$$\mathbf{a} = a^x \mathbf{e}_x + a^y \mathbf{e}_y \quad \textit{where} \quad \mathbf{e}_x, \mathbf{e}_y \equiv \text{basis vectors}.$$

Note that when a distinction is made between contravariant and covariant components, the "usual" ones are called contravariant, and are written with a superscript. That is, we construct the vector **a** by walking a^x units in the x-direction, and then a^y units in the y-direction, even though x and y are not perpendicular.

The dot product is defined geometrically, without reference to coordinates, as the product of the parallel components of two vectors. We showed earlier, also on purely geometric grounds without reference to any coordinates, that the dot product is bilinear (the distributive property holds for both vectors). Therefore, we can say:

$$\mathbf{a}\bullet\mathbf{b} = \left(a^x \mathbf{e}_x + a^y \mathbf{e}_y\right)\bullet\left(b^x \mathbf{e}_x + b^y \mathbf{e}_y\right) = a^x b^x \mathbf{e}_x\bullet\mathbf{e}_x + a^x b^y \mathbf{e}_x\bullet\mathbf{e}_y + a^y b^x \mathbf{e}_y\bullet\mathbf{e}_x + a^y b^y \mathbf{e}_y\bullet\mathbf{e}_y \,. \tag{18.1}$$

In Figure 18.4, the angle between the x and y axes is θ. If $\mathbf{e}_x$ and $\mathbf{e}_y$ are unit vectors, then we have:

$$\mathbf{a}\bullet\mathbf{b} = a^x b^x + a^x b^y \cos\theta + a^y b^x \cos\theta + a^y b^y \,. \tag{18.2}$$

In orthonormal coordinates, this reduces to the familiar formula for dot product:

$$\mathbf{e}_x\bullet\mathbf{e}_y = \cos\theta = 0 \qquad \Rightarrow \qquad \mathbf{a}\bullet\mathbf{b} = a^x b^x + a^y b^y \,.$$

In general, the basis vectors need not be unit magnitude.

For brevity, it is standard to collect all the dot products of the unit vectors in (18.1) into a matrix, $g_{\mu\nu}$. This makes it easier to write our dot product:

$$g_{\mu\nu} \equiv \begin{bmatrix} \mathbf{e}_x \bullet \mathbf{e}_x & \mathbf{e}_x \bullet \mathbf{e}_y \\ \mathbf{e}_y \bullet \mathbf{e}_x & \mathbf{e}_y \bullet \mathbf{e}_y \end{bmatrix} \quad \Rightarrow \quad \mathbf{a}\bullet\mathbf{b} = g_{\mu\nu} a^\mu b^\nu \qquad \text{(Einstein summation)} .$$

In our example of unit vectors $\mathbf{e}_x$ and $\mathbf{e}_y$, and axis angle θ:

$$g_{\mu\nu} \equiv \begin{bmatrix} 1 & \cos\theta \\ \cos\theta & 1 \end{bmatrix} .$$

For orthonormal coordinates, $\theta = \pi/2$, and $g_{\mu\nu} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, yielding $\mathbf{a}\bullet\mathbf{b} = a^x b^x + a^y b^y$, as usual.

Because the dot product is commutative, $g_{\mu\nu}$ is always symmetric. It can be readily shown that $g_{\mu\nu}$ is a tensor; it is called the **metric tensor**. Usually, the dot products of unit vectors that compose $g_{\mu\nu}$ are functions of the coordinates x and y (or, say, r and θ). This means there is a different metric tensor at each point of the manifold:

$$g_{\mu\nu} = g_{\mu\nu}(x, y) .$$

Any function of a manifold is called a **field**, so $g_{\mu\nu}(x, y)$ is the **metric tensor field**. Often when people refer to the "metric tensor," they mean the metric tensor *field*. The metric tensor field is a crucial property of any curved manifold.

Covariant Components of a Vector

Consider the dot product $\mathbf{a}\bullet\mathbf{b}$. It is often helpful to consider separately the contributions to the dot product from a^x and a^y. From the linearity of dot products, we can write:

$$\mathbf{a}\bullet\mathbf{b} = \left(a^x \mathbf{e}_x + a^y \mathbf{e}_y\right)\bullet\mathbf{b} = a^x \left(\mathbf{e}_x \bullet \mathbf{b}\right) + a^y \left(\mathbf{e}_y \bullet \mathbf{b}\right) .$$

As shown in Figure 18.4b, the quantities in parentheses are just the component of **b** parallel to the x-axis and the y-axis. We define these quantities as the **covariant** components of **b**, written with subscripts:

$$b_x \equiv \mathbf{e}_x \bullet \mathbf{b}, \qquad b_y \equiv \mathbf{e}_y \bullet \mathbf{b} \qquad \Rightarrow \qquad \mathbf{a}\bullet\mathbf{b} = a^\mu b_\mu . \tag{18.3}$$

We have not changed the vector **b**; we have simply projected it onto the axes in a different way. In comparison: the "usual" contravariant component b^x is the projection onto the x-axis taken parallel to all *other* axes (in this case the y axis). The covariant component b_x is the component of **b** parallel to the x-axis. Note:

To find $\mathbf{a}\bullet\mathbf{b}$ from a^μ and b^μ, we need the metric tensor;
to find it from a^μ and b_μ (or from a_μ and b^μ) we don't need a metric or anything else.

Raising and Lowering Indexes: Is there an algebraic way to find b_x from b^x? Of course there is. We can evaluate the dot products in the definitions of (18.3) with the metric tensor:

$$b_x \equiv \mathbf{e}_x \bullet \mathbf{b} = g_{\mu\nu} \left(\mathbf{e}_x\right)^\mu b^\nu = g_{\mu\nu} \underbrace{(1,0)}_{\mathbf{e}_x}{}^\mu b^\nu = g_{x\nu} b^\nu , \qquad \text{and similarly,} \qquad b_y = g_{y\nu} b^\nu .$$

We can write both b_x and b_y in a single formula by using a free index, say μ:

$$b_\mu = g_{\mu\nu} b^\nu , \qquad \mu = x, y .$$

We could have derived this directly from the metric form of a dot product, though it wouldn't have illuminated the geometric meaning of "covariant":

$$\mathbf{a}\bullet\mathbf{b} = g_{\mu\nu}a^\mu b^\nu = a^\mu \underbrace{\left(g_{\mu\nu}b^\nu\right)}_{\equiv b_\mu} \equiv a^\mu b_\mu \,.$$

What is a gradient? The familiar form of a gradient holds, even in curved manifolds:

$$\nabla f(x,y) = \frac{\partial}{\partial x} f\left(\mathbf{e}x?\right) + \frac{\partial}{\partial y} f\left(\mathbf{e}y?\right).$$

The components of the vector gradient are $\partial f/\partial x$ and $\partial f/\partial y$. But are $\partial f/\partial x$ and $\partial f/\partial y$ the *contra*variant or *co*variant components of the gradient vector? We answer that by considering how we use the gradient in a formula. Consider how the function f changes in an infinitesimal step from (x, y) to $(x + dx, y + dy)$:

$$df = \left(\nabla f\right)\bullet(dx\,\mathbf{e}_x + dy\,\mathbf{e}_y) = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy \,.$$

We did *not* need to use the metric to evaluate this dot product. Since the displacement vector (dx, dy) is contravariant, it must be that the gradient $(\partial f/\partial x, \partial f/\partial y)$ is *co*variant. Therefore, we write it with a subscript:

$$\nabla f \equiv \partial_\mu f\,, \qquad b^\mu \equiv (dx, dy) \qquad \Rightarrow \qquad df = \left(\partial_\mu f\right)b^\mu \qquad \text{(Einstein summation)}\,.$$

In general, derivative operators (gradient, covariant derivative, exterior derivative, Lie derivative) produce covariant vector components (or a covariant index on a higher rank tensor). If our function f has units of "things", and our displacement b^μ is in meters, then our covariant gradient $\partial_\mu f$ is in "things per meter." Thus, covariant components can sometimes be thought of as "rates."

As a physical example, canonical momentum, a derivative of the lagrangian, is a covariant vector:

$$p_i = \frac{\partial L}{\partial \dot{q}^i} \qquad \textit{where} \quad q^i \text{ are the (contravariant) generalized coordinates}\,.$$

This allows us to calculate the classical action of mechanics, a physical invariant that is independent of coordinates, without using a metric:

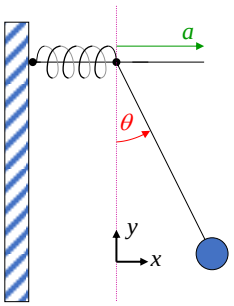
$$I = \int_{q_{i,\,initial}}^{q_{i,\,final}} p_i(q^i)\,dq^i \qquad \text{(Einstein summation)}\,.$$

This is crucial because *phase space has no metric*! Therefore, there is no such thing as a contravariant momentum p^i. Furthermore, viewing the canonical momenta as covariant components helps clarify the meaning of Noether's theorem (See *Funky Mechanics Concepts*).

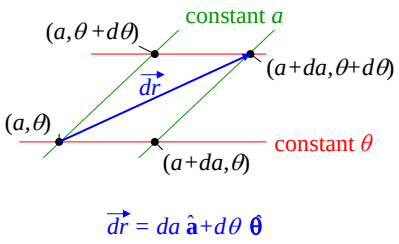
Elsewhere, we describe an alternative geometric description of covariant vector components: the 1-form.

Example: Classical Mechanics with Oblique Generalized Coordinates

Consider the following problem from classical mechanics: a pendulum is suspended from a pivot point which slides horizontally on a spring. The generalized coordinates are (a, θ).



(a)



(b)

Figure 18.5 (a) Classical mechanical system. (b) Differential area of configuration space.

To compute kinetic energy, we need to compute $|\mathbf{v}|^2$, conveniently done in some orthogonal coordinates, say x and y. We start by converting the generalized coordinates to the orthonormal x-y coordinates, to compute the length of a physical displacement from the changes in generalized coordinates:

$$x = a + l\sin\theta, \qquad dx = da + l\cos\theta\, d\theta$$

$$y = l\cos\theta, \qquad dy = -l\sin\theta\, d\theta \qquad \Rightarrow$$

$$ds^2 = dx^2 + dy^2 = da^2 + 2l\left(\cos\theta\right) da\, d\theta + l^2\cos^2\theta\, d\theta^2 + l^2\sin^2\theta\, d\theta^2$$

$$= da^2 + 2l\left(\cos\theta\right) da\, d\theta + l^2 d\theta^2 .$$

We have just computed the metric tensor field, which is a function of position in the (a, θ) configuration space. We can write the metric tensor field components by inspection:

Let $\quad x^1 = a, \;\; x^2 = \theta$

$$ds^2 = \sum_{i=1}^{2}\sum_{j=1}^{2} g_{ij} dx^i dx^j = da^2 + 2l\cos\theta\, da\, d\theta + l^2 d\theta^2 \qquad \Rightarrow \qquad g_{ij} = \begin{pmatrix} 1 & l\cos\theta \\ l\cos\theta & l^2 \end{pmatrix}$$

Then for velocities:

$$|\mathbf{v}|^2 = \dot{x}^2 + \dot{y}^2 = \left[\dot{a} + l\left(\cos\theta\right)\dot{\theta}\right]^2 + \left[-l\left(\sin\theta\right)\dot{\theta}\right]^2$$

$$= \dot{a}^2 + 2l\left(\cos\theta\right)\dot{a}\dot{\theta} + l^2\left(\cos^2\theta\right)\dot{\theta}^2 + l^2\left(\sin^2\theta\right)\dot{\theta}^2$$

$$= \dot{a}^2 + 2l\left(\cos\theta\right)\dot{a}\dot{\theta} + l^2\dot{\theta}^2$$

$$= \underbrace{\begin{pmatrix} 1 & l\cos\theta \\ l\cos\theta & l^2 \end{pmatrix}}_{g_{ij}} \begin{bmatrix} \dot{a} \\ \dot{\theta} \end{bmatrix} = g_{ij}\dot{x}^i\dot{x}^j .$$

A key point here is that the *same* metric tensor computes a physical displacement from generalized coordinate displacements, or a physical velocity from generalized coordinate velocities, or a physical acceleration from generalized coordinate accelerations, etc., because time is the same for any generalized coordinate system (no Relativity here!). Note that we symmetrize the cross-terms of the metric, $g_{ij} = g_{ji}$, which is necessary to insure that $\mathbf{g}(\mathbf{v}, \mathbf{w}) = \mathbf{g}(\mathbf{w}, \mathbf{v})$.

Now consider the scalar product of two vectors. The same metric tensor (field) helps compute the scalar product (dot product) of any two (infinitesimal) vectors, from their generalized coordinates:

$$d\mathbf{v} \cdot d\mathbf{w} = \mathbf{g}(d\mathbf{v}, d\mathbf{w}) = g_{ij} dv^i dw^j .$$

Since the metric tensor takes two input vectors, is linear in both, and produces a scalar result, it is a rank-2 tensor. Also, since $\mathbf{g}(\mathbf{v}, \mathbf{w}) = \mathbf{g}(\mathbf{w}, \mathbf{v})$, $\mathbf{g}$ is a **symmetric** tensor.

Now, let's define a scalar field as a function of the generalized coordinates; say, the potential energy:

$$U = \frac{k}{2}a^2 - mg\cos\theta .$$

It is quite useful to know the gradient of the potential energy:

$$\mathbf{D} = \nabla U = \frac{\partial U}{\partial a}\boldsymbol{\omega}^a + \frac{\partial U}{\partial \theta}\boldsymbol{\omega}^\theta \qquad \Rightarrow \qquad dU = \mathbf{D}(d\mathbf{r}) = \frac{\partial U}{\partial a} da + \frac{\partial U}{\partial \theta} d\theta .$$

The gradient takes an infinitesimal displacement vector $d\mathbf{r} = (da, d\theta)$, and produces a differential in the value of potential energy, dU (a scalar). Further, dU is a linear function of the displacement vector. Hence, by definition, the gradient at each point in a-θ space is a rank-1 tensor, i.e. the gradient is a tensor field.

Do we need to use the metric (computed earlier) to make the gradient operate on $d\mathbf{r}$? No! The gradient operates directly on $d\mathbf{r}$, without the need for any "assistance" by a metric. So the gradient is a rank-1 tensor that can directly contract with a vector to produce a scalar. This is markedly different from the dot product case above, where the first vector (a rank-1 tensor) could *not* contract directly with an input vector to produce a scalar. So clearly,

> There are two kinds of rank-1 tensors: those (like the gradient) that can contract directly with an input vector, and those that need the metric to "help" them operate on an input vector.

Those tensors that can operate directly on a vector are called **covariant tensors**, and those that need help are called **contravariant**, for reasons we will show soon. To indicate that **D** is covariant, we write its components with subscripts, instead of superscripts. Its basis vectors are covariant vectors, related to $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$:

$$\mathbf{D} = D_i\boldsymbol{\omega}^i = D_a\boldsymbol{\omega}^a + D_\theta\boldsymbol{\omega}^\theta \qquad \textit{where} \quad \boldsymbol{\omega}^r, \boldsymbol{\omega}^\theta \text{ are covariant basis vectors .}$$

In general, covariant tensors result from differentiation operators on other (scalar or) tensor fields: gradient, covariant derivative, exterior derivative, Lie derivative, etc.

Note that just as we can say that **D** acts on $d\mathbf{r}$, we can say that $d\mathbf{r}$ is a rank-1 tensor that acts on **D** to produce dU:

$$\mathbf{D}(d\mathbf{r}) = d\mathbf{r}(\mathbf{D}) = \sum_i \frac{\partial U}{\partial x^i} dx^i = \frac{\partial U}{\partial a} da + \frac{\partial U}{\partial \theta} d\theta \,.$$

The contractions are the same with either acting on the other, so the definitions are symmetric.

Interestingly, when we compute small oscillations of a system of particles, we need both the potential matrix, which is the gradient of the gradient of the potential field, and the "mass" matrix, which really gives us kinetic energy (rather than mass). The potential matrix is fully covariant, and we need no metric to compute it. The kinetic energy matrix requires us to compute absolute magnitudes of $|\mathbf{v}|^2$, and so requires us to compute the metric.

We know that a vector, which is a rank-1 tensor, can be visualized as an arrow. How do we visualize this covariant tensor, in a way that reveals how it operates on a vector (an arrow)? We use a set of equally spaced parallel planes (Figure 18.6).

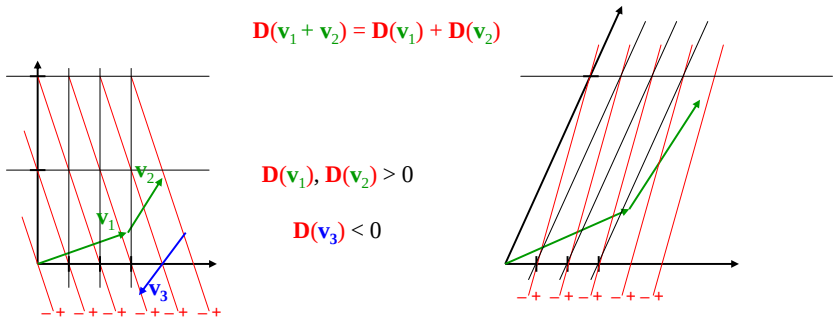


Figure 18.6 Visualization of a covariant vector (1-form) as oriented parallel planes. The 1-form is a linear operator on vectors (see text).

Let **D** be a covariant tensor (aka **1-form**). The value of **D** on a vector, **D**(**v**), is the number of planes "pierced" by the vector when laid on the parallel planes. Clearly, **D**(**v**) depends on the magnitude and direction of **v**.

It is also a linear function of **v**: the sum of planes pierced by two different vectors equals the number of planes pierced by their vector sum, and scales with the vectors: $\mathbf{D}(a\mathbf{v}+b\mathbf{w}) = a\mathbf{D}(\mathbf{v}) + b\mathbf{D}(\mathbf{w})$.

There is an orientation to the planes. One side is negative, and the other positive. Vectors crossing in the negative to the positive direction "pierce" a positive number of planes. Vectors crossing in the positive to negative direction "pierce" a negative number of planes.

Note also we could redraw the two axes arbitrarily oblique (non-orthogonal), and rescale the axes arbitrarily, but keeping the intercept values of the planes with the axes unchanged (thus stretching the arrows and planes). The number of planes pierced would be the same, so the two diagrams above are equivalent. Hence, this geometric construction of the operation of a covector on a contravector is completely general, and even applies to vector spaces which have no metric (aka "non-metric" spaces). All you need for the construction is a set of arbitrary basis vectors (not necessarily orthonormal), and the values $\mathbf{D}(\mathbf{e}_i)$ on each, and you can draw the parallel planes that illustrate the covector.

The "direction" of **D**, analogous to the direction of a vector, is *normal to* (perpendicular to) the planes used to graphically represent **D**.

What Goes Up Can Go Down: Duality of Contravariant and Covariant Vectors

Recall the dot product is given by:

$$d\mathbf{v} \cdot d\mathbf{w} = \mathbf{g}(d\mathbf{v}, d\mathbf{w}) = g_{ij} dv^i dw^j .$$

If I fill only one slot of **g** with **v**, and leave the 2nd slot empty, then **g**(**v**, _) is a linear function of one vector, and can be directly contracted with that vector; hence **g**(**v**, _) is a rank-1 covariant vector. For any given contravariant vector v^i, I can define this "dual" covariant vector, **g**(**v**, _), which has N components I'll call v_i.

$$v_i = \mathbf{g}(\mathbf{v}, \_) = g_{ik} v^k .$$

So long as I have a metric, the contravariant and covariant forms of **v** contain equivalent information, and are thus two ways of expressing the same vector (geometric object).

The covariant representation can contract directly with a contravariant vector, and the contravariant representation can contract directly with a covariant vector, to produce the dot product of the two vectors. Therefore, we can use the metric tensor to "lower" the components of a contravariant vector into their covariant equivalents.

Note that the metric tensor itself has been written with two covariant (lower) indexes, because it contracts directly with two contravariant vectors to produce their scalar-product.

Why do I need two forms of the same vector? Consider the vector "force:"

$$\mathbf{F} = m\mathbf{a} \qquad or \qquad F^i = ma^i \qquad \text{(naturally contravariant).}$$

Since position x^i is naturally contravariant, so is its derivative v^i, and 2nd derivative, a^i. Therefore, force is "naturally" contravariant. But force is also the gradient of potential energy:

$$\mathbf{F} = -\nabla U \qquad \text{or} \qquad F_i = -\frac{\partial}{\partial x^i} U \qquad \text{(naturally covariant).}$$

Oops! Now "force" is naturally *covariant*! But physically, it's the same force as above. So which is more natural for "force?" Neither. Use whichever one you need. Nurture supersedes nature.

The inverse of the metric tensor matrix is the contravariant metric tensor, g^{ij}. It contracts directly with two covariant vectors to produce their scalar product. Hence, we can use g^{ij} to "raise" the index of a covariant vector to get its contravariant components:

$$v^i = \mathbf{g}(\mathbf{v}, \_) = g^{ik} v_k \qquad\qquad g^{ik} g_{kj} = g^i{}_j \,.$$

Notice that raising and lowering works on the metric tensor itself. Note that in general, even for symmetric tensors, $T_i{}^j \neq T_j{}^i$, and $T_i{}^j \neq T^i{}_j$.

For rank-2 or higher tensors, each index is separately of the contravariant or covariant type. Each index may be raised or lowered separately from the others. Each lowering requires a contraction with the fully covariant metric tensor; each raising requires a contraction with the fully contravariant metric tensor.

In Euclidean space with orthonormal coordinates, the metric tensor is the identity matrix. Hence, the covariant and contravariant components of any vector are identical. This is why there is no distinction made in elementary treatments of vector mathematics; displacements, gradients, everything, are simply called "vectors."

The space of covectors is a vector space, i.e. it satisfies the properties of a vector space. However, it is called "dual" to the vector space of contravectors, because covectors operate on contravectors to produce scalar invariants. A thing is **dual** to another thing if the dual can act on the original thing to produce a scalar, and vice versa. E.g., in QM, bras are dual to kets. "Vectors in the dual space" are covectors.

Just like basis contravectors, basis covectors always have components (in their own basis):

$$\boldsymbol{\omega}^1 = (1,0,0...), \quad \boldsymbol{\omega}^2 = (0,1,0...), \quad \boldsymbol{\omega}^3 = (0,0,1...), \qquad \textit{etc.}$$

and we can write an arbitrary covector as $\tilde{\mathbf{f}} = f_1\boldsymbol{\omega}^1 + f_2\boldsymbol{\omega}^2 + f_3\boldsymbol{\omega}^3 + \ldots$.

TBS: construction and units of a dual covector from its contravector.

The *Real* Summation Convention

The summation convention says repeated indexes in an arithmetic expression are implicitly summed (contracted). We now understand that only a contravariant/covariant pair can be meaningfully summed. Two covariant or two contravariant indexes require contracting with the metric tensor to be meaningful. Hence, the *real* Einstein summation convention is that any two matching indexes, one "up" (contravariant) and one "down" (covariant), are implicitly summed (contracted). Two matching contravariant or covariant indexes are meaningless, and not allowed.

Now we can see why basis contravectors are written $\mathbf{e}_1$, $\mathbf{e}_2$, ... (with subscripts), and basis covectors are written $\boldsymbol{\omega}^1$, $\boldsymbol{\omega}^2$, ... (with superscripts). It is purely a trick to comply with the *real* summation convention that requires summations be performed over one "up" index and one "down" index. Then we can write a vector as a linear combination of the basis vectors, using the summation convention:

$$\mathbf{v} = v^1\mathbf{e}_1 + v^2\mathbf{e}_2 + v^3\mathbf{e}_3 = v^i\mathbf{e}_i \qquad\qquad \tilde{\mathbf{a}} = a_1\boldsymbol{\omega}^1 + a_2\boldsymbol{\omega}^2 + a_3\boldsymbol{\omega}^3 = a_i\boldsymbol{\omega}^i \,.$$

Note well: there is nothing "covariant" about $\mathbf{e}_i$, even though it has a subscript; there is nothing "contravariant" about $\boldsymbol{\omega}^i$, even though it has a superscript. It's just a notational trick.

Transformation of Covariant Indexes

It turns out that the components of a covariant vector transform with the same matrix as used to express the new (primed) basis vectors in the old basis:

$$f'_k = f_j \Lambda^j{}_k \qquad\qquad [\text{Tal } 2.4.11] \,.$$

Again, somewhat bogusly, eq. 2.4.11 is said to "transform covariantly with" (the same as) the basis vectors, so 'f_j' is called a **covariant** vector.

For a rank-2 tensor such as T_{ij}, each index of T_{ij} transforms "like" the basis vectors (i.e., covariantly with the basis vectors). Hence, each index of T_{ij} is said to be a "covariant" index. Since both indexes are covariant, T_{ij} is sometimes called "fully covariant."

Indefinite Metrics: Relativity

In short, a covariant index of a tensor is one which can be contracted with (summed over) a contravariant index of an input MVE to produce a meaningful resultant MVE.

In relativity, the metric tensor has some negative signs. The scalar-product is a frame-invariant "interval." No problem. All the math, raising, and lowering, works just the same. In special relativity, the metric ends up simply putting minus signs where you need them to get SR intervals. The covariant form of a vector has the minus signs "pre-loaded," so it contracts directly with a contravariant vector to produce a scalar.

Let's use the sign convention where $\eta_{\mu\nu}$ = diag(–1, 1, 1, 1). When considering the dual 1-forms for Minkowski space, the only unusual aspect is that the 1-form for time increases in the *opposite* direction as the vector for time. For the space components, the dual 1-forms increase in the *same* direction as the vectors. This means that:

$$\boldsymbol{\omega}^t \mathbf{e}_t = -1, \qquad \boldsymbol{\omega}^x \mathbf{e}_x = 1, \qquad \boldsymbol{\omega}^y \mathbf{e}_y = 1, \qquad \boldsymbol{\omega}^z \mathbf{e}_z = 1,$$

as it should for the Minkowski metric.

Is a Transformation Matrix a Tensor?

Sort of. When applied to a vector, it converts components from the "old" basis to the "new" basis. It is clearly a linear function of its argument. However, a tensor usually has all its inputs and outputs in the *same* basis (or tensor products of that basis). But a transformation matrix is specifically constructed to take inputs in one basis, and produce outputs in a *different* basis. Essentially, the columns are indexed by the old basis, and the rows are indexed by the new basis. It basically works like a tensor, but the transformation rule is that to transform the columns, you use a transformation matrix for the old basis; to transform the rows, you use the transformation matrix for the new basis.

Consider a vector:

$$\mathbf{v} = v^1 \mathbf{e}_1 + v^2 \mathbf{e}_2 + v^3 \mathbf{e}_3 .$$

This is a vector equation, and despite its appearance, it is true in *any* basis, not just the ($\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$) basis. If we write $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ as vectors in some new ($\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$) basis, the vector equation above still holds:

$$\mathbf{e}_1 = (\mathbf{e}_1)^x \mathbf{e}_x + (\mathbf{e}_1)^y \mathbf{e}_y + (\mathbf{e}_1)^z \mathbf{e}_z$$

$$\mathbf{e}_2 = (\mathbf{e}_2)^x \mathbf{e}_x + (\mathbf{e}_2)^y \mathbf{e}_y + (\mathbf{e}_2)^z \mathbf{e}_z$$

$$\mathbf{e}_3 = (\mathbf{e}_3)^x \mathbf{e}_x + (\mathbf{e}_3)^y \mathbf{e}_y + (\mathbf{e}_3)^z \mathbf{e}_z$$

$$\mathbf{v} = v^1 \mathbf{e}_1 + v^2 \mathbf{e}_2 + v^3 \mathbf{e}_3$$

$$= v^1 \underbrace{\left[(\mathbf{e}_1)^x \mathbf{e}_x + (\mathbf{e}_1)^y \mathbf{e}_y + (\mathbf{e}_1)^z \mathbf{e}_z \right]}_{\mathbf{e}_1} + v^2 \underbrace{\left[(\mathbf{e}_2)^x \mathbf{e}_x + (\mathbf{e}_2)^y \mathbf{e}_y + (\mathbf{e}_2)^z \mathbf{e}_z \right]}_{\mathbf{e}_2} + v^3 \underbrace{\left[(\mathbf{e}_3)^x \mathbf{e}_x + (\mathbf{e}_3)^y \mathbf{e}_y + (\mathbf{e}_3)^z \mathbf{e}_z \right]}_{\mathbf{e}_3}$$

The vector $\mathbf{v}$ is just a weighted sum of basis vectors, and therefore the columns of the transformation matrix are the old basis vectors expressed in the new basis. E.g., to transform the components of a vector from the ($\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$) to the ($\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$) basis, the transformation matrix is:

$$\begin{bmatrix} (\mathbf{e}_1)^x & (\mathbf{e}_2)^x & (\mathbf{e}_3)^x \\ (\mathbf{e}_1)^y & (\mathbf{e}_2)^y & (\mathbf{e}_3)^y \\ (\mathbf{e}_1)^z & (\mathbf{e}_2)^z & (\mathbf{e}_3)^z \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 \cdot \mathbf{e}_x & \mathbf{e}_2 \cdot \mathbf{e}_x & \mathbf{e}_3 \cdot \mathbf{e}_x \\ \mathbf{e}_1 \cdot \mathbf{e}_y & \mathbf{e}_2 \cdot \mathbf{e}_y & \mathbf{e}_3 \cdot \mathbf{e}_y \\ \mathbf{e}_1 \cdot \mathbf{e}_z & \mathbf{e}_2 \cdot \mathbf{e}_z & \mathbf{e}_3 \cdot \mathbf{e}_z \end{bmatrix}.$$

You can see directly that the first column is $\mathbf{e}_1$ written in the *x*-*y*-*z* basis; the 2nd column is $\mathbf{e}_2$ in the *x*-*y*-*z* basis; and the 3rd column is $\mathbf{e}_3$ in the *x*-*y*-*z* basis.

How About the Pauli Vector?

In quantum mechanics, the Pauli vector is a vector of three 2x2 matrices: the Pauli matrices. Each 2x2 complex-valued matrix corresponds to a spin-1/2 operator in some *x*, *y*, or *z* direction. It is a 3rd rank object in the tensor product space of $\mathbf{R}^3 \otimes \mathbf{C}^2 \otimes \mathbf{C}^2$, i.e. *xyz* $\otimes$ spinor $\otimes$ spinor. The *xyz* rank is clearly in a different basis than the complex spinor ranks, since *xyz* is a completely different vector space than spin-1/2 spinor space. However, it is a linear operator on various objects, so each rank transforms according to the transformation matrix for its basis.

$$\begin{matrix} & x & & y & & z \end{matrix}$$
$$\vec{\sigma} = \left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \right)$$

It's interesting to note that the term **tensor product** produces, in general, an object of mixed bases, and often, mixed vector spaces. Nonetheless, the term "tensor" seems to be used most often for mathematical objects whose ranks are all in the same basis.

Cartesian Tensors

Cartesian tensors aren't quite tensors, because they don't transform into non-Cartesian coordinates properly. (Note that despite their name, Cartesian tensors are *not* a special kind of tensor; they aren't really tensors. They're tensor wanna-be's.) Cartesian tensors have two failings that prevent them from being true tensors: they don't distinguish between contravariant and covariant components, and they treat finite displacements in space as vectors. In non-orthogonal coordinates, you must distinguish contravariant and covariant components. In non-Cartesian coordinates, only infinitesimal displacements are vectors. Details:

Recall that in Cartesian coordinates, there is no distinction between contravariant and covariant components of a tensor. This allows a certain sloppiness that one can only get away with if one sticks to Cartesian coordinates. This means that Cartesian "tensors" only transform reliably by rotations from one set of Cartesian coordinates to a new, rotated set of Cartesian coordinates. Since both the new and old bases are Cartesian, there is no need to distinguish contravariant and covariant components in either basis, and the transformation (to a rotated coordinate system) "works."

For example, the moment of inertia "tensor" is a Cartesian tensor. There is no problem in its first use, to compute the angular momentum of a blob of mass given its angular velocity:

$$\mathbf{I}(\boldsymbol{\omega}, \_) = \mathbf{L} \qquad \Rightarrow \qquad L^i = I^i_j \omega^j \qquad \Rightarrow$$
$$\begin{bmatrix} L^x \\ L^y \\ L^z \end{bmatrix} = \begin{bmatrix} I^x{}_x & I^x{}_y & I^x{}_z \\ I^y{}_x & I^y{}_y & I^y{}_z \\ I^z{}_x & I^z{}_y & I^z{}_z \end{bmatrix} \begin{bmatrix} \omega^x \\ \omega^y \\ \omega^z \end{bmatrix} = \omega^x \begin{bmatrix} I^x{}_x \\ I^y{}_x \\ I^z{}_x \end{bmatrix} + \omega^y \begin{bmatrix} I^x{}_y \\ I^y{}_y \\ I^z{}_y \end{bmatrix} + \omega^z \begin{bmatrix} I^x{}_z \\ I^y{}_z \\ I^z{}_z \end{bmatrix}$$

But notice that if **I** accepts a contravariant vector, then **I**'s components for that input vector must be covariant. However, **I** produces a contravariant output, so its output components are contravariant. So far, so good.

But now we want to find the kinetic energy. Well, $\frac{1}{2} I \omega^2 = \frac{1}{2} \mathbf{L} \cdot \boldsymbol{\omega} = \left(\frac{1}{2} \mathbf{I}(\boldsymbol{\omega}, \_) \right) \cdot \boldsymbol{\omega}$. The dot product is a dot product of two contravariant vectors. To evaluate that dot product, in a general coordinate system, we have to use the metric:

$$KE = \frac{1}{2} I^i_j \omega^j \omega_i = \frac{1}{2} I^i_j \omega^j g_{ik} \omega^k \qquad \neq \frac{1}{2} I^i_j \omega^j \omega^i \,.$$

However, in Cartesian coordinates, the metric matrix is the identity matrix, the contravariant components equal the covariant components, and the final "not-equals" above becomes an "equals." Hence, we neglect the distinction between contravariant components and covariant components, and "incorrectly" sum the components of **I** on the components of ω, even though both are contravariant in the 2nd sum.

In general coordinates, the direct sum for the dot product doesn't work, and you must use the metric tensor for the final dot product.

Example of failure of finite displacements: TBS: The electric quadrupole tensor acts on two copies of the finite displacement vector to produce the electric potential at that displacement. Even in something as simple as polar coordinates, this method fails.

The Real Reason Why the Kronecker Delta Is Symmetric

TBS: It is a mixed tensor, $\delta^{\alpha}{}_{\beta}$, but symmetry can only be assessed by comparing interchange of two indices of the same "up-" or "down-ness" (contravariance or covariance). We can lower, say α, in $\delta^{\alpha}{}_{\beta}$ with the metric:

$$\delta_{\alpha\beta} = g_{\alpha\gamma}\delta^{\gamma}{}_{\beta} = g_{\alpha\beta}\,.$$

The result is the metric $g_{\alpha\beta}$, which is always symmetric. Hence, $\delta^{\alpha}{}_{\beta}$ is a symmetric tensor, but *not* because its matrix looks symmetric. In general, a mixed rank-2 symmetric tensor does *not* have a symmetric matrix representation. Only when both indices are up or both down is its matrix symmetric.

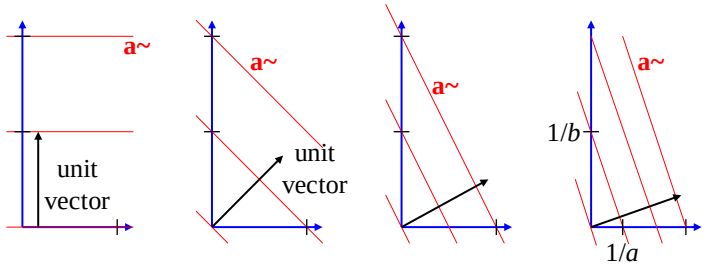
The Kronecker delta is a special case that does not generalize.

Things are not always what they seem.

Tensor Appendices

Pythagorean Relation for 1-forms

Demonstration that 1-forms satisfy the Pythagorean relation for magnitude:



0 **d***x* + 1 **d***y* | 1 **d***x* + 1 **d***y* | 2 **d***x* + 1 **d***y* | *a* **d***x* + *b* **d***y*
$|\mathbf{a}\sim| = 1$ | $|\mathbf{a}\sim| = \sqrt{2}$ | $|\mathbf{a}\sim| = \sqrt{5}$ | $|\mathbf{a}\sim| = \sqrt{(a^2+b^2)}$

Examples of three 1-forms, and a generic 1-form. Here, **d***x* is the *x* basis 1-form, and **d***y* is the *y* basis 1-form.

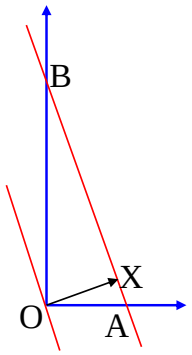
From the diagram above, a max-crossing vector (perpendicular to the planes of a~) has (*x*, *y*) components (1/*b*, 1/*a*). Dividing by its magnitude, we get a unit vector:

$$\text{max crossing unit vector } \mathbf{u} = \frac{\frac{1}{b}\hat{\mathbf{x}} + \frac{1}{a}\hat{\mathbf{y}}}{\sqrt{\frac{1}{b^2}+\frac{1}{a^2}}}. \qquad \textit{Note that} \qquad \mathbf{d}x(\hat{\mathbf{x}}) = 1, \textit{ and } \mathbf{d}y(\hat{\mathbf{y}}) = 1\,.$$

The magnitude of a 1-form is the scalar resulting from the 1-form's action on a max-crossing unit vector:

$$|\tilde{\boldsymbol{a}}| = \tilde{\boldsymbol{a}}(\boldsymbol{u}) = \frac{(a\,\mathrm{d}x + b\,\mathrm{d}y)\left(\frac{1}{b}\hat{\mathbf{x}} + \frac{1}{a}\hat{\mathbf{y}}\right)}{\sqrt{\frac{1}{b^2}+\frac{1}{a^2}}} = \frac{\left(\frac{a}{b}+\frac{b}{a}\right)}{\sqrt{\frac{1}{b^2}+\frac{1}{a^2}}} = \frac{\left(a^2+b^2\right)}{ab\sqrt{\frac{1}{b^2}+\frac{1}{a^2}}} = \frac{\left(a^2+b^2\right)}{\sqrt{a^2+b^2}} = \sqrt{a^2+b^2}\ .$$

Here's another demonstration that 1-forms satisfy the Pythagorean relation for magnitude. The magnitude of a 1-form is the inverse of the plane spacing:



$$\Delta OXA \sim \Delta BOA \quad \Rightarrow \quad \frac{OX}{OA} = \frac{BO}{BA}$$

$$\Rightarrow \quad OX = \frac{(BO)(OA)}{BA} = \frac{\frac{1}{b}\cdot\frac{1}{a}}{\sqrt{\frac{1}{b^2}+\frac{1}{a^2}}}$$

$$|\tilde{\boldsymbol{a}}| = \frac{1}{OX} = \frac{\sqrt{\frac{1}{b^2}+\frac{1}{a^2}}}{\frac{1}{b}\cdot\frac{1}{a}} = ab\sqrt{\frac{1}{b^2}+\frac{1}{a^2}} = \sqrt{a^2+b^2}\,\phi$$

Figure 18.7 Demonstration that 1-forms satisfy the Pythagorean relation for magnitude.

Geometric Construction Of The Sum Of Two 1-Forms:

Example of a~ + b~

a~(**x**) = 2
b~(**x**) = 1
(a~ + b~)(**x**) = 3

Construction of a~ + b~

a~($\mathbf{v}_a$) = 1	a~($\mathbf{v}_b$) = 0
b~($\mathbf{v}_a$) = 0	b~($\mathbf{v}_b$) = 1
(a~ + b~)($\mathbf{v}_a$) = 1	(a~ + b~)($\mathbf{v}_b$) = 1

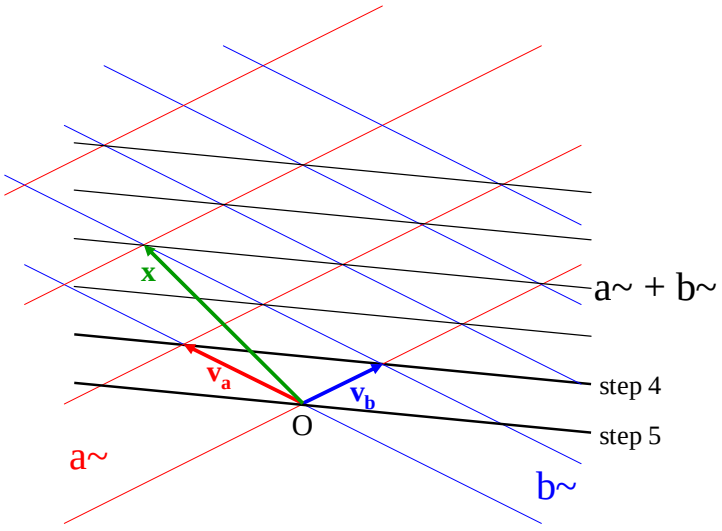


Figure 18.8 Geometric construction of the sum of two 1-forms.

To construct the sum of two 1-forms, **a**~ + **b**~:

1. Choose an origin at the intersection of a plane of a~ and a plane of b~.

2. Draw vector $\mathbf{v}_a$ from the origin along the planes of b~, so b~($\mathbf{v}_a$) = 0, and of length such that a~($\mathbf{v}_a$) = 1. [This is the dual vector of a~.]
3. Similarly, draw $\mathbf{v}_b$ from the origin along the planes of a~, so a~($\mathbf{v}_b$) = 0, and b~($\mathbf{v}_b$) = 1. [This is the dual vector of b~.]
4. Draw a plane through the heads of $\mathbf{v}_a$ and $\mathbf{v}_b$ (black above). This defines the orientation of (a~ + b~).
5. Draw a parallel plane through the common point (the origin). This defines the spacing of planes of (a~ + b~).
6. Draw all other planes parallel, and with the same spacing. This is the geometric representation of (a~ + b~).

Now we can easily draw the test vector **x**, such that **a~**(**x**) = 2, and **b~**(**x**) = 1.

"Fully Anti-symmetric" Symbols Expanded

Everyone hears about them, but few ever see them. They are quite sparse: the 3-D fully anti-symmetric symbol has 6 nonzero values out of 27; the 4-D one has 24 nonzero values out of 256.

3-D, from the 6 permutations, *ijk*: 123+, 132-, 312+, 321-, 231+, 213-

$$\varepsilon_{ijk} = \underset{k=1}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}}, \underset{k=2}{\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}, \underset{k=3}{\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}$$

4-D, from the 24 permutations, $\alpha\beta\gamma\delta$:

0123+	0132-	0312+	0321-	0231+	0213-
1023-	1032+	1302-	1320+	1230-	1203+
2013+	2031-	2301+	2310-	2130+	2103-
3012-	3021+	3201-	3210+	3120-	3102+

$$\varepsilon_{\alpha\beta\gamma\delta} = \begin{array}{c} \\ \alpha=0 \\ \alpha=1 \\ \alpha=2 \\ \alpha=3 \end{array} \begin{array}{c} \beta=0 \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}, \begin{array}{c} \beta=1 \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}, \begin{array}{c} \beta=2 \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}, \begin{array}{c} \beta=3 \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

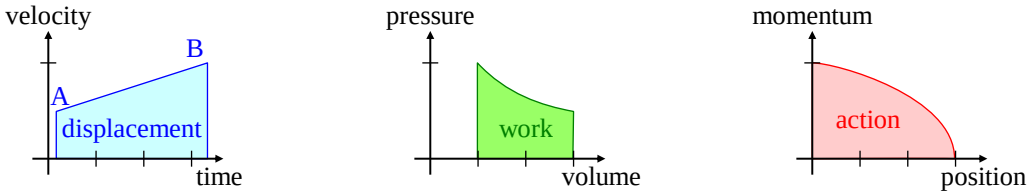
Metric? We Don't Need No Stinking Metric!

Examples of Useful, Non-metric Spaces

Non-metric spaces are everywhere. A non-metric space has no concept of "distance" between arbitrary points, or even between arbitrary "nearby" points (points with infinitesimal coordinate differences). However:

> Non-metric spaces have no concept of "distance,"
> but many still have a well-defined concept of "area," in the sense of an integral.

For example, consider a plot of velocity (of a particle in 1D) vs. time (below, left).



Some useful non-metric spaces: (left) velocity vs. time; (middle) pressure vs. volume; (right) momentum vs. position. In each case, there is no distance, but there *is* area.

The area under the velocity curve is the total displacement covered. The area under the *P*-*V* curve is the work done by an expanding fluid. The area under the momentum-position curve (*p*-*q*) is the action of the motion in classical mechanics. Though the points in each of these plots exist on 2D manifolds, the two coordinates are incomparable (they have different units). It is meaningless to ask what is the distance between two arbitrary points on the plane. For example, points A and B on the *v*-*t* curve differ in both velocity and time, so how could we define a distance between them (how can we add m/s and seconds)?

In the above cases, we have one coordinate value as a function of the other, e.g. velocity as a function of time. We now consider another case: rather than consider the function as one of the coordinates in a manifold, we consider the manifold as comprising only the independent variables. Then, the function is defined *on* that manifold. As usual, keeping track of the units of all the quantities will help in understanding both the physical and mathematical principles.

For example, the speed of light in air is a function of 3 independent variables: temperature, pressure, and humidity. At 633 nm, the effects amount to speed changes of about +1 ppm per kelvin, –0.4 ppm per mm-Hg pressure, and +0.01 ppm per 1% change in relative humidity (RH) (see http://patapsco.nist.gov/mel/div821/Wavelength/Documentation.asp#CommentsRegardingInputstotheEquations):

$$s(T, P, H) = s_0 + aT - bP + cH\ .$$

where $a \approx 300$ (m/s)/k, $b \approx 120$ (m/s)/mm-Hg, and $c \approx 3$ (m/s)/% are positive constants, and the function s is the speed of light at the given conditions, in m/s. Our manifold is the set of TPH triples, and s is a function on that manifold. We can consider the TPH triple as a (contravariant, column) vector: $(T, P, H)^{\mathrm{T}}$. These vectors constitute a 3D vector space over the field of reals. $s(\cdot)$ is a real function on that vector space.

Note that the 3 components of a vector each have different units: the temperature is measured in kelvins (K), the pressure in mm-Hg, and the relative humidity in %. Note also that there is no metric on (T, P, H) space (which is bigger, 1 K or 1 mm-Hg?). However, the gradient of s is still well defined:

$$\nabla s = \frac{\partial s}{\partial T}\tilde{\mathbf{d}}\mathbf{T} + \frac{\partial s}{\partial P}\tilde{\mathbf{d}}\mathbf{P} + \frac{\partial s}{\partial H}\tilde{\mathbf{d}}\mathbf{H} = a\,\tilde{\mathbf{d}}\mathbf{T} - b\,\tilde{\mathbf{d}}\mathbf{P} + c\,\tilde{\mathbf{d}}\mathbf{H}\ .$$

What are the units of the gradient? As with the vectors, each component has different units: the first is in (m/s) per kelvin; the second in (m/s) per mm-Hg; the third in (m/s) per %. The gradient has different units than the vectors, and is not a part of the original vector space. The gradient, ∇s, operates on a vector $(T, P, H)^{\mathrm{T}}$ to give the change in speed from one set of conditions, say (T_0, P_0, H_0) to conditions incremented by the vector $(T_0 + T, P_0 + P, H_0 + H)$.

One often thinks of the gradient as having a second property: it specifies the "direction" of steepest increase of the function, *s*. But:

Without a metric, "steepest" is not defined.

Which is steeper, moving one unit in the temperature direction, or one unit in the humidity direction? In desperation, we might ignore our units of measure, and choose the Euclidean metric (thus equating one unit of temperature with one unit of pressure and one unit of humidity); then the gradient produces a "direction" of steepest increase. However, with no justification for such a choice of metric, the result is probably meaningless.

What about basis vectors? The obvious choice is, including units, (1 K, 0 mm-Hg, 0 %)^T, (0 K, 1 mm-Hg, 0 %)^T, and (0 K, 0 mm-Hg, 1 %)^T, or omitting units: (1, 0, 0), (0, 1, 0), and (0, 0, 1). Note that these are *not* unit vectors, because there is no such thing as a "unit" vector, because there is no metric by which to measure one "unit." Also, if I ascribe units to the basis vectors, then the *components* of an arbitrary vector in that basis are dimensionless.

Now let's change the basis: suppose now I measure temperature in some unit equal to ½ K (almost the Rankine scale). Now all my temperature measurements "double", i.e. $T_{new} = 2\,T_{old}$. In other words, (½ K, 0, 0)^T is a different basis than (1 K, 0, 0)^T. As expected for a covariant component, the temperature component of the gradient $(\nabla s)_T$ is cut in half if the basis vector "halves." So when the half-size gradient component operates on the double-size temperature vector component, the product remains invariant, i.e., the speed of light is a function of temperature, not of the units in which you measure temperature.

The above basis change was a simple change of scale of one component in isolation. The other common basis change is a "rotation" of the axes, "mixing" the old basis vectors.

Can we rotate axes when the units are different for each component? Surprisingly, we can.

We simply define new basis vectors as linear combinations of old ones, which is all that a rotation does. For example, suppose we measured the speed of light on 3 different days, and the environmental conditions were different on those 3 days. We choose those measurements as our basis, say $\mathbf{e}_1$ = (300 K, 750 mm-Hg, 20%), $\mathbf{e}_2$ = (290 K, 760 mm-Hg, 30 %), and $\mathbf{e}_3$ = (290 K, 770 mm-Hg, 10 %). These basis vectors are not orthogonal, but are (of course) linearly independent. Suppose I want to know the speed of light at (296 K, 752 mm-Hg, 18 %). I decompose this into my new basis and get (0.6, 0.6, –0.2). I compute the speed of light function in the new basis, and then compute its gradient, to get $d_1\tilde{\mathbf{e}}^1 + d_2\tilde{\mathbf{e}}^2 + d_3\tilde{\mathbf{e}}^3$. I then operate on the vector with the gradient to find the change in speed: $\Delta s = \nabla s(0.6, 0.6, -0.2) = 0.6\,d_1 + 0.6\,d_2 - 0.2\,d_3$.

We could extend this to a more complex function, and then the gradient is not constant. For example, a more accurate equation for the speed of light is:

$$s(T,P,H) = c_0 - f\frac{P}{T} + gH\left((T-273)^2 + 160\right).$$

where $f \approx 7.86 \times 10^{-4}$ and $g \approx 1.5 \times 10^{-11}$ are constants. Now the gradient is a function of position (in TPH space), and there is still no metric.

Comment on the metric: In desperation, you might define a metric, i.e. the length of a vector, to be Δs, the change in the speed of light due to the environmental changes defined by that vector. However, such a metric is in general non-Euclidean (not a Pythagorean relationship), indefinite (non-zero vectors can have zero or negative "lengths"), and still doesn't define a meaningful dot product. Our more-accurate equation for the speed of light provides examples of these failures.

References:

[Knu] Knuth, Donald, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd Ed., p. 117.

[Mic] Eric L. Michelsen, *Quirky Quantum Concepts*, Springer, 2014. ISBN-13: 978-1461493044.

[Sch] Schutz, Bernard, *A First Course in General Relativity*, Cambridge University Press, 1990.

[Sch2] Schutz, Bernard, *Geometrical Methods of Mathematical Physics*, Cambridge University Press, 1980.

[Tal] Talman, Richard, *Geometric Mechanics*, John Wiley and Sons, 2000.

19 Differential Geometry

Manifolds

A **manifold** is a "space": a set of points with coordinate labels. We are free to choose coordinates many ways, but a manifold must be able to have coordinates that are real numbers. We are familiar with "metric manifolds", where there is a concept of distance. However, there are many useful manifolds which have no metric, e.g. phase space (see "We Don't Need No Stinking Metric" above).

Even when a space is non-metric, it still has concepts of "locality" and "continuity."

Such locality and continuity are defined in terms of the coordinates, which are real numbers. It may also have a "volume", e.g. the oft-mentioned "phase-space volume." It may seem odd that there's no definition of "distance," but there is one of "volume." **Volume** in this case is simply defined in terms of the coordinates, $dV = dx_1\, dx_2\, dx_3$ ..., and has no absolute meaning.

Coordinate Bases

Coordinate bases are basis vectors derived from coordinates on the manifold. They are extremely useful, and built directly on basic multivariate calculus. Coordinate bases can be defined a few different ways. Perhaps the simplest comes from considering a small displacement vector on a manifold. We use 2D polar coordinates in (r, θ) as our example. A **coordinate basis** can be defined as the basis in which the components of an infinitesimal displacement vector are just the differentials of the coordinates:

(Left) Coordinate bases: the components of the displacement vector are the differentials of the coordinates. (Right) Coordinate basis vectors around the manifold.

Note that $\mathbf{e}_\theta$ (the θ basis vector) far from the origin must be bigger than near, because a small change in angle, $d\theta$, causes a bigger displacement vector far from the origin than near. The advantage of a coordinate basis is that it makes dot products, such as a gradient dotted into a displacement, appear in the simplest possible form:

$$\textit{Given} \quad f(r,\theta), \qquad df = \nabla f(r,\theta)\cdot d\mathbf{p} = \left(\frac{\partial f}{\partial r} + \frac{\partial f}{\partial \theta}\right)\cdot(dr, d\theta) = \frac{\partial f}{\partial r}dr + \frac{\partial f}{\partial \theta}d\theta\,.$$

The last equality is assured from elementary multivariate calculus.

The basis vectors are defined by differentials, but are themselves finite vectors. Any physical vector, finite or infinitesimal, can be expressed in the coordinate basis, e.g., velocity, which is finite.

"Vectors" as derivatives: There is a huge confusion about writing basis "vectors" as derivatives. From our study of tensors (earlier), we know that a vector can be considered an operators on a 1-form, which produces a scalar. We now describe how vector fields can be considered operators on scalar functions, which produce scalar fields. I don't like this view, since it is fairly arbitrary, confuses the much more consistent tensor view, and is easily replaced with tensor notation.

We will see that in fact, the derivative "basis vectors" are operators which create 1-forms (dual-basis components), not traditional basis vectors. The vector basis is then *implicitly* defined as the dual of the dual-basis, which is always the coordinate basis. In detail:

We know from the "Tensors" chapter that the gradient of a scalar field is a 1-form with partial derivatives as its components. For example:

$$\nabla f(x,y,z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right) = \frac{\partial f}{\partial x}\boldsymbol{\omega}^1 + \frac{\partial f}{\partial y}\boldsymbol{\omega}^2 + \frac{\partial f}{\partial z}\boldsymbol{\omega}^3, \qquad \textit{where} \quad \boldsymbol{\omega}^1, \boldsymbol{\omega}^2, \boldsymbol{\omega}^3 \text{ are basis 1-forms}$$

Many texts *define* vectors in terms of their action on scalar functions (aka scalar fields), e.g. [Wald p15]. Given a point (*x*, *y*, *z*), and a function *f*(*x*, *y*, *z*), the definition of a vector **v** amounts to

$$\mathbf{v} \equiv \left(v^x, v^y, v^z\right) \quad \textit{such that} \qquad \mathbf{v}\left[\, f\left(x,y,z\right)\right] \equiv \mathbf{v}\cdot\nabla f = v^x\frac{\partial f}{\partial x} + v^y\frac{\partial f}{\partial y} + v^z\frac{\partial f}{\partial z} \qquad \text{(a scalar field)}$$

.

Roughly, the action of **v** on *f* produces a scaled directional derivative of *f*: Given some small displacement *dt*, as a fraction of |**v**| and in the direction of **v**, **v** tells you how much *f* will change when moving from (*x*, *y*, *z*) to ($x + v^x dt$, $y + v^y dt$, $z + v^z dt$):

$$df = \mathbf{v}\left[\, f\,\right] dt \qquad \textit{or} \qquad \frac{df}{dt} = \mathbf{v}\left[\, f\,\right].$$

If *t* is time, and **v** is a velocity, then **v**[*f*] is the time rate of change of *f*. While this notation is compact, I'd rather write it simply as the dot product of **v** and ∇*f*, which is more explicit, and consistent with tensors:

$$df = \mathbf{v}\cdot\nabla f\ dt \qquad \textit{or} \qquad \frac{df}{dt} = \mathbf{v}\cdot\nabla f\ .$$

The definition of **v** above requires an auxiliary function *f*, which is messy. We remove *f* by redefining **v** as an operator:

$$\mathbf{v} \equiv \left(v^x\frac{\partial}{\partial x} + v^y\frac{\partial}{\partial y} + v^z\frac{\partial}{\partial x}\right) \qquad \text{(an operator)}.$$

Given this form, it *looks like* ∂/∂*x*, ∂/∂*y*, and ∂/∂*z* are some kind of "basis vectors." Indeed, standard terminology is to refer to ∂/∂*x*, ∂/∂*y*, and ∂/∂*z* as the "coordinate basis" for vectors, but they are really operators for creating *1-forms!* Then:

$$\mathbf{v}\left[\, f\,\right] = v^x\frac{\partial f}{\partial x} + v^y\frac{\partial f}{\partial y} + v^z\frac{\partial f}{\partial z} = \sum_{i=x,y,z} v^i\left(\nabla f\right)_i \qquad \text{(a scalar field)}.$$

The vector **v** contracts directly with the 1-form ∇*f* (without need of any metric), hence **v** is a vector *implicitly* defined in the basis dual to the 1-form ∇*f*.

Note that if **v** = **v**(*x*, *y*, *z*) is a vector field, then:

$$\mathbf{v}\left[\, f\left(x,y,z\right)\right] \equiv \mathbf{v}(x,y,z)\cdot\nabla f(x,y,z) \qquad \text{(a scalar field)}.$$

These derivative operators can be drawn as basis vectors in the usual manner, as arrows on the manifold. They are just the coordinate basis vectors shown earlier. For example, consider polar coordinates (*r*, *θ*):

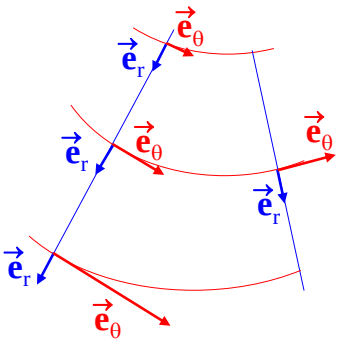


Figure 19.1 Examples of coordinate basis vectors around the manifold. $\mathbf{e}_r$ happens to be unit magnitude everywhere, but $\mathbf{e}_\theta$ is not.

The manifold in this case is simply the flat plane, $\mathbb{R}^2$. The r-coordinate basis vectors are all the same size, but have different directions at different places. The θ coordinate basis vectors get larger with r, and also vary in direction around the manifold.

Covariant Derivatives

Notation: Due to word-processor limitations, the following two notations are equivalent:

$$\vec{h}(\,) \equiv \mathbf{h}(\,), \qquad \vec{r} \equiv \mathbf{r}\,.$$

The following description is similar to one in [Sch].

We start with the familiar concepts of derivatives, and see how that evolves into the covariant derivative. Given a real-valued function of one variable, $f(x)$, we want to know how f varies with x near a value, a. The answer is the derivative of $f(x)$, where

$df = f'(a)\,dx$ and therefore $\quad f(a + dx) \approx f(a) + df = f(a) + f'(a)\,dx\,.$

Extending to two variables, $g(x, y)$, we'd like to know how g varies in the 2-D neighborhood around a point (a, b), given a displacement vector $d\mathbf{r} = (dx, dy)$. We can compute its gradient:

$$\tilde{\nabla} g = \frac{\partial g}{\partial x}\tilde{\mathbf{dx}} + \frac{\partial g}{\partial y}\tilde{\mathbf{dy}} \qquad \text{and therefore} \qquad g\left(a + dx, b + dy\right) \approx g(a,b) + \tilde{\nabla} g(d\vec{r})\,.$$

The gradient is also called a directional derivative, because the rate at which g changes depends on the direction in which you move away from the point (a, b).

The gradient extends to a vector valued function (a vector **field**) $\mathbf{h}(x, y) = h^x(x, y)\mathbf{i} + h^y(x, y)\mathbf{j}$:

$$\nabla\mathbf{h} = \frac{\partial \mathbf{h}}{\partial x}\tilde{\mathbf{dx}} + \frac{\partial \mathbf{h}}{\partial y}\tilde{\mathbf{dy}}, \qquad \frac{\partial \mathbf{h}}{\partial x} = \frac{\partial h^x}{\partial x}\mathbf{e}_x + \frac{\partial h^y}{\partial x}\mathbf{e}_y \qquad \textit{and} \qquad \frac{\partial \mathbf{h}}{\partial y} = \frac{\partial h^x}{\partial y}\mathbf{e}_x + \frac{\partial h^y}{\partial y}\mathbf{e}_y$$

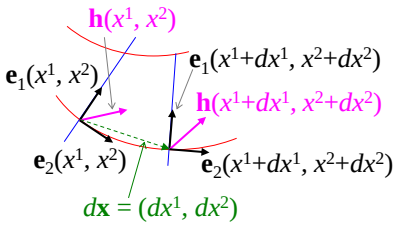
$$d\mathbf{h} = \nabla\mathbf{h}(d\mathbf{r}) = \frac{\partial \mathbf{h}}{\partial x}dx + \frac{\partial \mathbf{h}}{\partial y}dy = \begin{bmatrix} \dfrac{\partial h^x}{\partial x} & \dfrac{\partial h^x}{\partial y} \\ \dfrac{\partial h^y}{\partial x} & \dfrac{\partial h^y}{\partial y} \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} = dx \begin{bmatrix} \dfrac{\partial h^x}{\partial x} \\ \dfrac{\partial h^y}{\partial x} \end{bmatrix} + dy \begin{bmatrix} \dfrac{\partial h^x}{\partial y} \\ \dfrac{\partial h^y}{\partial y} \end{bmatrix} = \frac{\partial \mathbf{h}}{\partial x}dx + \frac{\partial \mathbf{h}}{\partial y}dy$$

We see that the columns of $\nabla\mathbf{h}$ are vectors which are weighted by dx and dy, and then summed to produce a vector result. Therefore, $\nabla\mathbf{h}$ is linear in the displacement vector $d\mathbf{r} = (dx, dy)$. This linearity insures that it transforms like a duck . . . I mean, like a tensor. Thus $\nabla\mathbf{h}$ is a rank-2 $\binom{1}{1}$ tensor: it takes a single vector input, and produces a vector result.

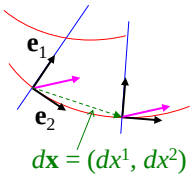
So far, all this has been in rectangular coordinates. Now we must consider what happens in curvilinear coordinates, such as polar. Note that we're still in a simple, flat space. (We'll get to curved spaces later). Our goal is still to find the change in the vector value of **h**( ), given an infinitesimal vector change of position, $d\mathbf{x} = (dx^1, dx^2)$. We use the same approach as above, where a vector valued function comprises two (or *n*) real-valued component functions: $\mathbf{h}(x^1,x^2) = h^1(x^1,x^2)\mathbf{e}_1 + h^2(x^1,x^2)\mathbf{e}_2$. However, in this general case, the basis vectors are themselves functions of position (previously the basis vectors were constant everywhere). So **h**( ) is really:

$$\mathbf{h}(x^1,x^2) = h^1(x^1,x^2)\mathbf{e}_1(x^1,x^2) + h^2(x^1,x^2)\mathbf{e}_2(x^1,x^2)\,.$$

Hence, partial derivatives of the *component functions alone* are no longer sufficient to define the change in the vector value of **h**( ); we must also account for the change in the basis vectors.



(a) Components constant, but vector changes



(b) Vector constant, but components change

Figure 19.2 The distinction between a component of a derivative, and a derivative of a component.

Note that a component of the derivative is distinctly *not* the same as the derivative of the component (see Figure 19.2). Therefore, the *i*th component of the derivative depends on *all* the components of the vector field.

We compute partial derivatives of the vector field $\boldsymbol{h}(x^1, x^2)$ using the product rule:

$$\begin{aligned}\frac{\partial \mathbf{h}}{\partial x^1} &= \frac{\partial h^1}{\partial x^1}\mathbf{e}_1(x^1,x^2) + h^1(x^1,x^2)\frac{\partial \mathbf{e}_1}{\partial x^1} + \frac{\partial h^2}{\partial x^1}\mathbf{e}_2(x^1,x^2) + h^2(x^1,x^2)\frac{\partial \mathbf{e}_2}{\partial x^1} \\ &= \sum_{j=1}^{n}\left(\frac{\partial h^j}{\partial x^1}\mathbf{e}_j(x^1,x^2) + h^j(x^1,x^2)\frac{\partial \mathbf{e}_j}{\partial x^1}\right).\end{aligned}$$

This is a *vector* equation: all terms are vectors, each with components in all *n* basis directions. This is equivalent to *n* numerical component equations. Note that $(\partial \boldsymbol{h}/\partial x_1)$ has components in both (or all *n*) directions. Of course, we can write similar equations for the components of the derivative in any basis direction, $\mathbf{e}_k$:

$$\begin{aligned}\frac{\partial \mathbf{h}}{\partial x^k} &= \frac{\partial h^1}{\partial x^k}\mathbf{e}_1(x^1,x^2) + h^1(x^1,x^2)\frac{\partial \mathbf{e}_1}{\partial x^k} + \frac{\partial h^2}{\partial x^k}\mathbf{e}_2(x^1,x^2) + h^2(x^1,x^2)\frac{\partial \mathbf{e}_2}{\partial x^k} \\ &= \sum_{j=1}^{n}\left(\frac{\partial h^j}{\partial x^k}\mathbf{e}_j(x^1,x^2) + h^j(x^1,x^2)\frac{\partial \mathbf{e}_j}{\partial x^k}\right).\end{aligned}$$

Because we must frequently work with components and component equations, rather than whole vector equations, let us now consider only the *i*th component of the above:

$$\left(\frac{\partial \mathbf{h}}{\partial x^k}\right)^i = \frac{\partial h^i}{\partial x^k} + \sum_{j=1}^{n} h^j(x^1,x^2)\left(\frac{\partial \mathbf{e}_j}{\partial x^k}\right)^i.$$

The first term moves out of the summation because each of the first terms in the summation of eq. (1) are vectors, and each points exactly in the $\mathbf{e}_j$ direction. Only the $j = i$ term contributes to the *i*th component; the purely $\mathbf{e}_j$ directed vector contributes nothing to the *i*th component when $j \neq i$.

Recall that these equations are true for *any arbitrary* coordinate system; we have made no assumptions about unit length or orthogonal basis vectors. Note that

$$\frac{\partial \mathbf{h}}{\partial x_k} = (\nabla \mathbf{h})_k = \text{ the } k\text{th (covariant) component of } \nabla \mathbf{h} \,.$$

Since $\nabla \mathbf{h}$ is a rank-2 tensor, the kth covariant component of $\nabla \mathbf{h}$ is the kth column of $\nabla \mathbf{h}$:

$$\nabla \mathbf{h} = \begin{bmatrix} \left(\frac{\partial \mathbf{h}}{\partial x^1}\right)^1 & \left(\frac{\partial \mathbf{h}}{\partial x^2}\right)^1 \\ \left(\frac{\partial \mathbf{h}}{\partial x^1}\right)^2 & \left(\frac{\partial \mathbf{h}}{\partial x^2}\right)^2 \end{bmatrix}.$$

Since the change in $\mathbf{h}(\,)$ is linear with small changes in position,

$$d\mathbf{h} = \nabla \mathbf{h}(d\mathbf{x}), \qquad \textit{where} \quad d\mathbf{x} = (dx^1, dx^2) \,.$$

Going back to Equations (1) and (2), we can now write the full covariant derivative of $\mathbf{h}(\,)$ in 3 ways: vector, verbose component, and compact component:

$$(\nabla \mathbf{h})_k \equiv \nabla_k \mathbf{h} = \frac{\partial \mathbf{h}}{\partial x^k} + \sum_{j=1}^{n} h^j(x^1, x^2) \frac{\partial \mathbf{e}_j}{\partial x^k} = \frac{\partial \mathbf{h}}{\partial x^k} + \sum_{j=1}^{n} h^j(x^1, x^2) \sum_{i=1}^{n} \Gamma^i_{jk} \mathbf{e}_i$$

$$(\nabla_k \mathbf{h})^i = \frac{\partial h^i}{\partial x^k} + \sum_{j=1}^{n} h^j(x^1, x^2) \left(\frac{\partial \mathbf{e}_j}{\partial x^k}\right)^i$$

$$(\nabla_k \mathbf{h})^i = \frac{\partial h^i}{\partial x^k} + h^j \Gamma^i_{jk}, \qquad \textit{where} \quad \Gamma^i_{jk} \equiv \left(\frac{\partial \mathbf{e}_j}{\partial x^k}\right)^i \quad \Rightarrow \quad \Gamma^i_{jk} \mathbf{e}_i = \frac{\partial \mathbf{e}_j}{\partial x^k}$$

Aside: Some mathematicians complain that you can't define the Christoffel symbols as derivatives of basis vectors, because you can't compare vectors from two different points of a manifold without already having the Christoffel symbols (aka the "connection"). Physicists, including Schutz [Sch], say that physics *defines* how to compare vectors at different points of a manifold, and thus you can *calculate* the Christoffel symbols. In the end, it doesn't really matter. Either way, by physics or by fiat, the Christoffel symbols are, in fact, the derivatives of the basis vectors.

Christoffel Symbols

Christoffel symbols are the covariant derivatives of the basis vector fields. We use ordinary plane polar coordinates (r, θ) as an example.

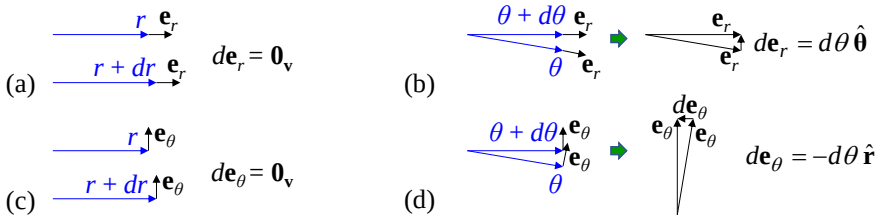


Figure 19.3 (a) Derivative of $\mathbf{e}_r$ in the r direction is the zero vector. (b) Derivative of $\mathbf{e}_r$ in the θ direction is $\hat{\boldsymbol{\theta}}$. (c) Derivative of $\mathbf{e}_\theta$ in the r direction is the zero vector. (d) Derivative of $\mathbf{e}_\theta$ in the θ direction is $-\hat{\mathbf{r}}$.

Figure 19.3 shows the derivatives of the r basis vector in the r direction, and in the θ direction. From this, we can fill in four components of the Christoffel symbols:

$$\frac{\partial \mathbf{e}_r}{\partial r} \equiv \Gamma^{\mu}{}_{rr} = \mathbf{0}_{\mathbf{v}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \frac{\partial \mathbf{e}_r}{\partial \theta} \equiv \Gamma^{\mu}{}_{r\theta} = \hat{\boldsymbol{\theta}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad \text{or} \qquad \Gamma^{\mu}{}_{r\nu} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

Similarly, the derivatives of the θ basis vector are:

$$\frac{\partial \mathbf{e}_\theta}{\partial r} \equiv \Gamma^{\mu}{}_{\theta r} = \mathbf{0}_{\mathbf{v}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \frac{\partial \mathbf{e}_\theta}{\partial \theta} \equiv \Gamma^{\mu}{}_{\theta\theta} = -\hat{\mathbf{r}} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \quad \text{or} \qquad \Gamma^{\mu}{}_{\theta\nu} = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}.$$

These are the 8 components of the Christoffel symbols. In general, in n-dimensional space, each basis vector has a derivative in each direction, with n components, for a total of n^3 components in $\Gamma^{\mu}{}_{\beta\nu}$.

Visualization of n-Forms

TBS: 1-forms as oriented planes

2-forms (in 3 or more space) as oriented parallelograms

3-forms (in 3 or more space) as oriented parallelepipeds

4-forms (in 4-space): how are they oriented??

Review of Wedge Products and Exterior Derivative

This is a quick insert that needs proper work. ?? This section requires understanding outer-products, and anti-symmetrization of matrices.

Wedge Products

We can get an overview of the meaning of a wedge product from a simple example: the wedge product of two vectors in 3D space. We first review two preliminaries: anti-symmetrization of a matrix, and the outer product of two vectors.

Recall that any matrix can by written as a sum of a symmetric and an anti-symmetric matrix (much like any function can be written as a sum of an even and an odd function):

$$\mathbf{B} = \mathbf{B}_S + \mathbf{B}_A \qquad \textit{where} \quad \mathbf{B}_s = \mathbf{B}_S^T,\ \mathbf{B}_A = -\mathbf{B}_A^T. \tag{19.1}$$

For example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 3 & 5 & 7 \\ 5 & 7 & 9 \end{bmatrix} + \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}.$$

We can derive explicit expressions for the symmetric and anti-symmetric parts of a matrix from (19.1):

$$\mathbf{B} + \mathbf{B}^T = \mathbf{B}_S + \mathbf{B}_A + \mathbf{B}_S^T + \mathbf{B}_A^T = 2\mathbf{B}_S, \qquad \mathbf{B}_s = \frac{1}{2}\left(\mathbf{B} + \mathbf{B}^T\right)$$
$$\text{Similarly:} \qquad \mathbf{B}_A = \frac{1}{2}\left(\mathbf{B} - \mathbf{B}^T\right). \tag{19.2}$$

Also recall that the outer product of two vectors is a matrix (in this case, a rank-2 tensor):

$$\mathbf{a} \otimes \mathbf{b} \equiv [\mathbf{a}]\left[\mathbf{b}^T\right] = \begin{bmatrix} a_x b_x & a_x b_y & a_x b_z \\ a_y b_x & a_y b_y & a_y b_z \\ a_z b_x & a_z b_y & a_z b_z \end{bmatrix}. \qquad E.g., \quad \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \otimes \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{bmatrix}.$$

Finally, the **wedge product** of two vectors is the anti-symmetric part of the outer product:

$$\mathbf{a} \wedge \mathbf{b} \equiv \frac{1}{2}\Big[\, \mathbf{a} \otimes \mathbf{b} - (\mathbf{a} \otimes \mathbf{b})^T \,\Big] .$$

To simplify our notation, we can define a linear operator on a matrix which takes the anti-symmetric part. This is the **anti-symmetrization operator**:

$$\hat{A}(\mathbf{B}) \equiv \frac{1}{2}\left(\mathbf{B} - \mathbf{B}^T\right) \qquad \rightarrow \qquad \mathbf{a} \wedge \mathbf{b} \equiv \hat{A}(\mathbf{a} \otimes \mathbf{b}) .$$

Commutation: A crucial property of the wedge product is that it is anti-commutative:

$$\mathbf{a} \wedge \mathbf{b} = -\mathbf{b} \wedge \mathbf{a} .$$

This follows directly from the fact that the outer product is not commutative: $\mathbf{b} \otimes \mathbf{a} = (\mathbf{a} \otimes \mathbf{b})^T$. Then the anti-symmetric part of a transposed matrix is the negative of the anti-symmetric part of the original matrix:

$$\mathbf{b} \wedge \mathbf{a} = \hat{A}(\mathbf{b} \otimes \mathbf{a}) = \hat{A}\Big[(\mathbf{a} \otimes \mathbf{b})^T\Big] = -\hat{A}(\mathbf{a} \otimes \mathbf{b}) = -\mathbf{a} \wedge \mathbf{b} \,.\phi c$$

Tensor Notation

In tensor notation, the symmetric and anti-symmetric parts of a matrix are written:

$$B_S^{\alpha\beta} = \frac{1}{2}\left(B^{\alpha\beta} + B^{\beta\alpha}\right), \qquad B_A^{\alpha\beta} = \frac{1}{2}\left(B^{\alpha\beta} - B^{\beta\alpha}\right).$$

Note that both α and β are free indexes, so (in a 3 dimensional space) each of these is 9 separate equations. They are fully equivalent to the matrix equations (19.2).

1D

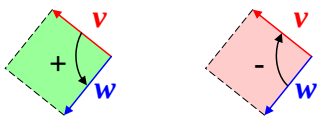
I don't know of any meaning for a wedge-product in 1D, where a "vector" is just a signed number. The "direction" of a vector is either + or –. Also, the 1D exterior derivative is a degenerate case, because the "exterior" of a line segment is just the 2 endpoints, and all functions are scalar functions. In all higher dimensions, the "exterior" or boundary of a region is a *closed* path/ surface/ volume/ hyper-volume/etc. In 1D the boundary of a line segment cannot be closed. So instead of integrating around a closed exterior (aka boundary), we simply take the difference in the function value at the endpoints, divided by a differential displacement. This is simply the ordinary derivative of a function, $f\,'(x)$.

2D

The exterior derivative of a scalar function $f(x, y)$ follows the 1D case, and is similarly degenerate, where the "exterior" is simply the two endpoints of a differential displacement. Since the domain is a 2D space, the displacements are vectors, and there are 2 (partial) derivatives, one for displacements in x, and one for displacements in y. Hence the exterior derivative is just the one-form "gradient" of the function:

$$\tilde{\mathbf{d}}\mathbf{f}(x, y) = \text{"}gradient\text{"} = \frac{\partial f}{\partial x}\tilde{\mathbf{d}}\mathbf{x} + \frac{\partial f}{\partial y}\tilde{\mathbf{d}}\mathbf{y}$$

In 2D, the **wedge product** of the basis 1-forms, $\tilde{\mathbf{d}}\mathbf{x} \wedge \tilde{\mathbf{d}}\mathbf{y}$, is a two-form, which accepts two vectors to produce the signed area of the parallelogram defined by them. A *signed* area can be + or –; a counter-clockwise direction is positive, and clockwise is negative.



$$\tilde{\mathbf{dx}} \wedge \tilde{\mathbf{dy}}(\vec{v}, \vec{w}) = \text{signed area defined by } (\vec{v}, \vec{w}) = -\tilde{\mathbf{dx}} \wedge \tilde{\mathbf{dy}}(\vec{w}, \vec{v})$$
$$= \tilde{\mathbf{dx}}(\vec{v})\tilde{\mathbf{dy}}(\vec{w}) - \tilde{\mathbf{dy}}(\vec{v})\tilde{\mathbf{dx}}(\vec{w})$$
$$= \det \begin{vmatrix} \tilde{\mathbf{dx}}(\vec{v}) & \tilde{\mathbf{dx}}(\vec{w}) \\ \tilde{\mathbf{dy}}(\vec{v}) & \tilde{\mathbf{dy}}(\vec{w}) \end{vmatrix} = \det \begin{vmatrix} v^x & w^x \\ v^y & w^y \end{vmatrix}$$

The exterior derivative of a 1-form is the ratio of the closed path integral of the 1-form to the area of the parallelogram of two vectors, for infinitesimal vectors. This is very similar to the definition of curl, only applied to a 1-form field instead of a vector field.

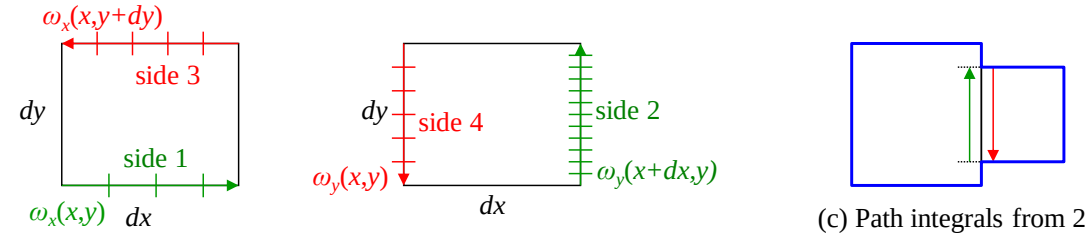


Figure 19.4 (a) 2D closed-path integral: contributions from *x* displacements. (b) Contributions from *y* displacements. (c) Path integrals from adjacent areas of any shape add.

Consider the horizontal and vertical contributions to the path integral separately:

$$\tilde{\boldsymbol{\omega}}(x, y) = \omega_x(x, y)\tilde{\mathbf{dx}} + \omega_{\mathbf{y}}(x, y)\tilde{\mathbf{dy}} \qquad \vec{r} = (x, y) \qquad d\vec{r} = (dx, dy)$$
$$\int_1 \tilde{\boldsymbol{\omega}}(d\vec{r}) + \int_3 \tilde{\boldsymbol{\omega}}(d\vec{r}) = \omega_x(r)dx - \omega_x(r + dy)dx = -\frac{\partial \omega_x}{\partial y} dy\, dx$$
$$\int_2 \tilde{\boldsymbol{\omega}}(d\vec{r}) + \int_4 \tilde{\boldsymbol{\omega}}(d\vec{r}) = \omega_y(r + dx)dy - \omega_y(r)dy = \frac{\partial \omega_y}{\partial x} dx\, dy$$

The horizontal integrals (sides 1 & 3) are linear in *dx*, because that is the length of the path. They are linear in *dy*, because *dy* is proportional to the difference in ω_x. Hence, the contribution is linear in both *dx* and *dy*, and therefore proportional to the area (*dx*)(*dy*).

A similar argument holds for the vertical contributions, sides 2 & 4. Therefore, the path integral varies proportionately to the area enclosed by two orthogonal vectors.

It is easy to show this is true for any two vectors, and any shaped area bounded by an infinitesimal path. For example, when you butt up two rectangles, the path integral around the combined boundary equals the sum of the individual path integrals, because the contributions from the common segment cancel from each rectangle, and hence omitting them does not change the path integral. The area integrals add.

3D

In 3D, the wedge product of the basis 1-forms is a 3-form, that can either:

1. Accept 2 vectors to produce an *oriented* area; it doesn't have a sign, it has a *direction*. Analogous to the cross-product. Or,
2. Accept 3 vectors (*u*, *v*, *w* below) to produce a *signed* volume.

$$\tilde{\mathbf{d}}\mathbf{x} \wedge \tilde{\mathbf{d}}\mathbf{y} \wedge \tilde{\mathbf{d}}\mathbf{z}(\vec{u}, \vec{v}, \vec{w}) = \text{signed volume defined by } (\vec{u}, \vec{v}, \vec{w})$$

$$= \det \begin{vmatrix} \tilde{\mathbf{d}}\mathbf{x}(\vec{u}) & \tilde{\mathbf{d}}\mathbf{x}(\vec{v}) & \tilde{\mathbf{d}}\mathbf{x}(\vec{w}) \\ \tilde{\mathbf{d}}\mathbf{y}(\vec{u}) & \tilde{\mathbf{d}}\mathbf{y}(\vec{v}) & \tilde{\mathbf{d}}\mathbf{y}(\vec{w}) \\ \tilde{\mathbf{d}}\mathbf{z}(\vec{u}) & \tilde{\mathbf{d}}\mathbf{z}(\vec{v}) & \tilde{\mathbf{d}}\mathbf{z}(\vec{w}) \end{vmatrix} = \det \begin{vmatrix} u^x & v^x & w^x \\ u^y & v^y & w^y \\ u^z & v^z & w^z \end{vmatrix}.$$

Being a 3-form (all wedge products are *p*-forms), the wedge-product is anti-symmetric in its arguments:

$$\tilde{\mathbf{d}}\mathbf{x} \wedge \tilde{\mathbf{d}}\mathbf{y} \wedge \tilde{\mathbf{d}}\mathbf{z}(\vec{u}, \vec{v}, \vec{w}) = -\tilde{\mathbf{d}}\mathbf{x} \wedge \tilde{\mathbf{d}}\mathbf{y} \wedge \tilde{\mathbf{d}}\mathbf{z}(\vec{u}, \vec{w}, \vec{v}), \quad \text{etc.}$$

The exterior derivative of a scalar or 1-form field is essentially the same as in the 2D case, except that now the areas defined by vectors are oriented instead of simply signed. In this case, the "exterior" is a closed surface; the "interior" is a volume.

20 Math That's Fun and Interesting

"The first time we use a particular mathematical process, we call it a 'trick'. The second time, it's a 'device'. The third time, it's a 'method'." – Unknown (to me).

Here are some math "methods" that either come up a lot and are worth knowing about, or are just fun and interesting.

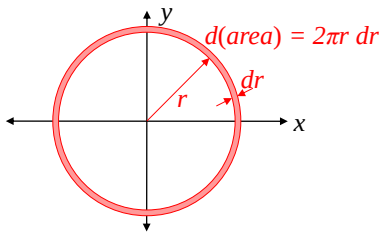
Math Tricks That Come Up A Lot

The Gaussian Integral

You can look this up anywhere, but here goes: we evaluate the basic integral, $\int_{-\infty}^{\infty} e^{-x^2}\,dx$, and throw in an '$a$' with x^2 at the end by a simple change of variable. First, we square the integral, then rewrite the second factor calling the dummy integration variable y instead of x:

$$\left(\int_{-\infty}^{\infty} dx\, e^{-x^2}\right)^2 = \left(\int_{-\infty}^{\infty} dx\, e^{-x^2}\right)\left(\int_{-\infty}^{\infty} dy\, e^{-y^2}\right) = \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy\, e^{-\left(x^2+y^2\right)}$$

This is just a double integral over the entire x-y plane, so we can switch to polar coordinates. Note that the exponential integrand is constant at constant r, so we can replace the differential area $dx\,dy$ with $2\pi r\,dr$:



$$\text{Let}\quad r^2 = x^2 + y^2 \qquad \Rightarrow \qquad \int_{-\infty}^{\infty} dx \int_{-\infty}^{\infty} dy\, e^{-\left(x^2+y^2\right)} = \int_0^{\infty} dr\, 2\pi r\, e^{-r^2} = -\pi\left[e^{-r^2}\right]_0^{\infty} = \pi\,.$$

$$\left(\int_{-\infty}^{\infty} dx\, e^{-x^2}\right)^2 = \pi \qquad \Rightarrow \qquad \int_{-\infty}^{\infty} dx\, e^{-x^2} = \sqrt{\pi}, \qquad \text{and} \qquad \boxed{\int_{-\infty}^{\infty} dx\, e^{-ax^2} = \sqrt{\frac{\pi}{a}}}\,.$$

Math Tricks That Are Fun and Interesting

Continuous Infinite Crossings

The following function has an infinite number of zero crossings near the origin, but is everywhere continuous (even at $x = 0$). That seems bizarre to me. Recall the definition:

$f(x)$ is **continuous** at a iff $\lim_{x\to a} f(x) = f(a)$.

Then let

$$f(x) = \begin{cases} x\sin\left(\dfrac{1}{x}\right), & x \neq 0 \\ 0, & x = 0 \end{cases}$$

$$\lim_{x\to 0} f(x) = 0 = f(0) \qquad\qquad f(0) \text{ is continuous!}$$

Picture

Technique for Integration

$$\int \frac{dx}{\sin x}$$ TBS.

Phasors

Phasors are complex numbers that represent sinusoids. The phasor defines the magnitude and phase of the sinusoid, but not its frequency. See *Funky Electromagnetic Concepts* for a full description.

Miserables Coset

Groups are *the* fundamental structures of symmetries, and are crucial to advanced physics. Groups also underlie discrete algebra, needed for encryption and error-control coding. One of the most astounding results of finite group theory is that the size of any subgroup divides evenly into the size of the parent group. The short proof illustrates important and useful concepts in group theory, is accessible to non-mathematicians, and proceeds roughly as follows:

- Define a "coset"
- Show that cosets are (a) the same size as the subgroup S, (b) don't overlap, and (c) cover the whole parent group G.
- Therefore, the size of the subgroup S divides the size of G.

Though the concept of a coset is straightforward, in my experience discussions of cosets are severely lacking (and sometimes incorrect). We here attempt to close that gap, and prove the above theorem, called **Lagrange's Theorem**.

Brief review of groups: A group is a set of elements, and an operator (here we call it "multiplication"), with 4 defining properties (see our detailed definitions elsewhere):

- Closure: the "product" of any two group elements is a group element.
- An identity element, which we call "e": for every element g, $g{\cdot}e = g$ $(= e{\cdot}g)$.
- Every element g has an inverse such that $gg^{-1} = e$ $(= g^{-1}g)$.
- The group operator is associative: $a(bc) = (ab)c$.

These simple properties produce an enormously rich structure and theory. Note that the group operator need *not* be commutative, and often is not. Nonetheless, even for noncommutative groups, $g{\cdot}e = e{\cdot}g$, and $gg^{-1} = g^{-1}g$. (Commutative groups often write the operator as "+", and the identity as "0".)

An example of a group is the set {1, 2} under multiplication mod 3. The identity is "1", and its group operation table is below left:

	1	2
1	1	2
2	2	1

	Lucy	Fred
Lucy	Lucy	Fred
Fred	Fred	Lucy

A group is fully defined by its operation table.

The group elements are abstract entities, not really numbers, though it is sometimes convenient to represent the elements as numbers. By simple renaming, the two tables above are equivalent, and define the *same* group. (Some groups have elements that can be thought of as *operators*, e.g., rotations or reflections, with the group-operation being the composition of the element operators.)

Subgroups: Sometimes a group is structured so that a subset of the elements, with the same group operator, forms a smaller group. For example, the numbers 0 - 11 form a group under *addition* mod 12, with identity 0. Then the set S = {0, 4, 8} forms a subgroup of the parent group. A subgroup inherits the identity and associativity from the parent. Notice that the parent group has size (aka "order") 12, and the subgroup has order 3. Three divides 12, and this is an example of Lagrange's Theorem.

[Mathematicians call the size of a *group* the **order**, but the size of a *set* is its **cardinality**, thus defining two new and *different* words for the same idea as the existing word "size." In a surprising show of consistency, though, the size (order) of a group G is written $|G|$, and the size (cardinality) of a set C is written $|C|$.

Another nit: strictly speaking, a group is a subgroup of itself, so it is the only subgroup that isn't "smaller" than the group.]

Cosets: Given a group G, a subgroup S, and a group element g, the "left **coset** of S with respect to g" is:

$$gS \equiv \{g, gs_1, gs_2, ...\} \qquad \textit{where} \quad s_i \equiv \text{the non-identity elements of } S\ .$$

In our subgroup example above, the coset of S with respect to 0 is just {0, 4, 8}. The coset with respect to 1 is {1, 5, 9}; w.r.t 2 is {2, 6, 10}; and w.r.t. 3 is {3, 7, 11}. You might think there are 8 more cosets, but the other elements of G (4 through 11) each reproduce one of these 4 cosets. For example, the coset w.r.t. 5 = {5, 9, 1}, which is the same as the coset w.r.t. 1. Thus a subgroup S of a group G defines a *set of cosets*.

We now prove three properties of the set of cosets of a subgroup. (a) The size of a coset is the size of the subgroup. By definition, a coset consists of one element from each of the elements of S. Furthermore, the coset elements are distinct, because the group operation is invertible: if $gs_1 = gs_2$, then premultiplying by g^{-1} proves that $s_1 = s_2$. Therefore, if $s_1 \neq s_2$, then $gs_1 \neq gs_2$. Hence, the size of the coset is the size of S.

(b) No cosets overlap, i.e. they are all disjoint. We show that by showing that given any element of a coset, we can generate all the other elements of the coset. Consider an element $a = gs_a$ in the coset w.r.t. g. Then for any subgroup element s, as is also in the coset with a:

$$as = (gs_a)s = g(s_a s), \qquad \text{and} \qquad s_a s \in S\ .$$

Now, let s take on all values in S, then $a{\cdot}s$ produces a distinct list of coset members (again from invertibility). In fact, $a{\cdot}s$ produces *all* of the coset members, one for each value of s. Thus, if two cosets contain the same member a, they are identical. So different cosets must be disjoint.

(c) Every element h of the group G appears in some coset. To be in a coset, we must have $h = gs$, for some g in G, and some s in S. To prove that a given h appears in some coset, we pick an arbitrary s in S, and solve for g: $g = hs^{-1}$. Thus every element of G appears in some coset, and since cosets don't overlap, every element of G appears in exactly one coset.

[In fact, we can choose s to be the identity, then $g = h$: every element h appears in the coset with respect to itself.]

Lagrange's Theorem: Putting the 3 properties together, we find that all the elements of G are distributed into equal-sized cosets, so the size of the cosets divides the size of G. And the size of S is the size of the cosets, ergo the size of S divides the size of G. QED.

Public Key Cryptography on a Hand Calculator

Public key cryptography is increasingly important in our digital world. The theory is clever, but understandable with only high school math. We present here the theory, along with simple numerical examples. We describe the RSA algorithm in detail, and give insight into the number theory behind it. (However, we do not prove the well-known number theorems.) Finally, get your name in lights by being the first to decrypt the message at the end, and send me an encrypted reply.

This section assumes some familiarity with modular arithmetic.

Public Key Cryptography Overview

Figure 20.1 shows old-fashioned cryptography, where the receiver must share a secret key with the sender. This requires a secure channel be temporarily established ahead of time to share the secret, which can be inconvenient.

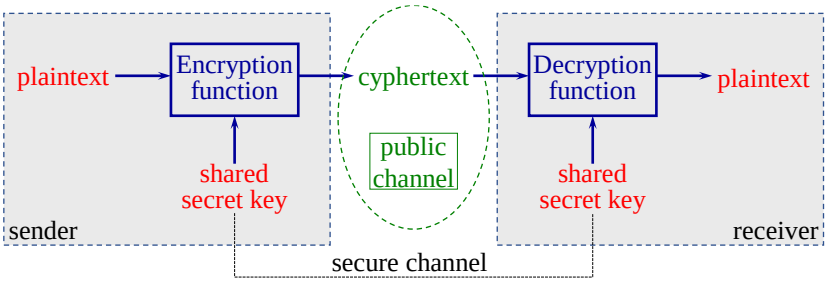


Figure 20.1 Old-fashioned, secret key cryptography.

Figure 20.2 shows public key cryptography, where the encryption and decryption keys are different. The receiver generates a matched-pair of keys: one private and one public. She publishes the public key widely. It allows anyone to encrypt a message to her, but gives no useful information on how to decrypt the message. The receiver keeps the secret key private; it allows decryption. The sender uses the public key to encrypt a message, and sends over public (insecure) channels to the receiver. The receiver decrypts the message with the private key.

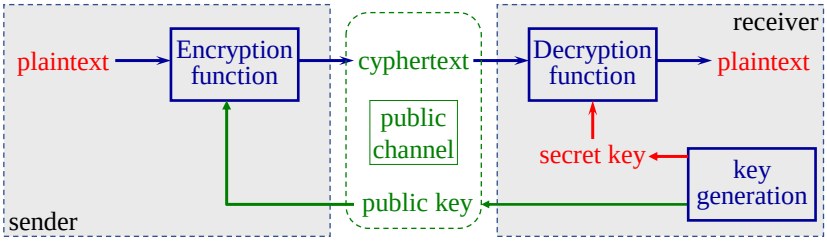


Figure 20.2 Public key cryptography.

The most widely used public key cryptography method is the RSA algorithm (Rivest, Shamir, Adleman). It was patented in 1983, but was released to the public domain in 2000, just before the patent expired [Wik01].

RSA Overview

The RSA public key comprises two integers (e, n): an exponent e, and a modulus n. For a message $m < n$, we encrypt it into cyphertext c as:

$$c = m^e \bmod n \,.$$

The private key is another integer exponent d such that:

$$c^d = \left(m^e\right)^d = m^{ed} = m \qquad (\bmod n)\,. \tag{20.1}$$

Many equations are written with "(mod n)" at the end (as in (20.1)), which denotes that both sides of all equals signs are taken mod n.

RSA is secure because there exist keys (e, n) for which it is computational infeasible to find m given c, e, and n. RSA is practical because it is computationally feasible to find such sets of (e, n, d), and to perform the encryption and decryption operations. In practice, n has around 600 digits (2048 bits), e ~ 5 digits, and d ~ 600 digits.

Examples of Relevant Modular Arithmetic

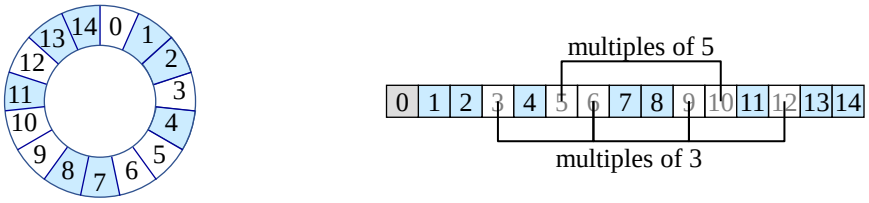
Modular arithmetic is used in lots of cryptographic algorithms. It is easy to compute, and simple to understand. All numbers are integers, and for RSA, positive. Modular arithmetic's key properties (get it?) are:

- $x \bmod n \equiv$ remainder of x / n, *e.g.* $16 \bmod 15 = 1$
- $(ab \bmod n) = (a \bmod n)(b \bmod n) \bmod n$.
- $\left(a^e\right)^d \bmod n = \left(a^e \bmod n\right)^d \bmod n$.

In other words, when doing a big calculation that will be taken "mod *n*" at the end, you can save calculation time by taking intermediate results "mod *n*" along the way.

Note that the notation "mod n" has two subtly different meanings: it is a remainder operator with two operands, as in "*x* mod *n*", and it is also an equation modifier, as in (20.1).

The RSA algorithm is based on the difficulty of (a) modular roots of large numbers, and (b) factoring large numbers.



(a) (b)

Figure 20.3 (a) Arithmetic mod 15. The 8 numbers with no common factor with 15 are highlighted. (b) Striking out numbers with common factors with 15.

Let's look at some simple examples of modular powers: say, some powers of integers mod 15 (n = 15):

$2^0 \bmod 15 = 1$, $2^1 \bmod 15 = 2$, $2^2 \bmod 15 = 4$, $2^3 \bmod 15 = 8$, $2^4 \bmod 15 = 1$.

At this point, the powers of two simply repeat, because $2^4 \bmod 15 = 2^0 = 1$. The period of repetition is T = 4.

Here's another (from now on, the "mod 15" is understood):

$7^0 = 1$, $7^1 = 7$, $7^2 = 4$, $7^3 = 13$, $7^4 = 1$. $T = 4$.

And another:

$11^0 = 1$, $11^1 = 11$, $11^2 = 1$. $T = 2$.

From group theory, it can be shown that for integers m that have no common factor with 15 (i.e., 1, 2, 4, 7, 8, 11, 13, 14), the period is at most 8, but might be 4 or 2 (more later). For n = 15, it turns out that the longest period is 4. This means for all such m, $m^5 = m$. But so does $m^9 = m$, and $m^{k \cdot 4+1} = m$ for any integer $k > 0$.

What about the *m with* a common factor with n = 15? Let's see:

$3^0 = 1$, $3^1 = 3$, $3^2 = 9$, $3^3 = 12$, $3^4 = 6$, $3^5 = 3$. $T = 4$.

The sequence repeats, but never returns to 1. Any power of 3 is a multiple of 3, and any multiple of 15 is a multiple of 3, so 3^e can never have a remainder of 1, i.e., $3^e \bmod 15$ can never be 1. In this case, the period is 4, and $3^5 = 3$, much like the m with no common factor. The same argument applies to any m with a common factor with n. It might seem like a coincidence that $m^5 = m$ even for m with a common factor with n, but it can be proved that this is always true [Wik01]. This is of theoretical interest, but in practice, it never happens. With $n \sim 600$ digits, it has about 10^{300} factors, out of 10^{600} possible messages m. Your chance of hitting a factor with your message is about 1 in 10^{300}, which may explain why this special case of RSA is rarely discussed in the literature. (It *is* mentioned in the original paper [RSA77 p7].)

The big result is this: for any $m < n = 15$, $m^{k \cdot 4+1} = m$, k any integer > 0.

Application to RSA Cryptosystem

RSA applies the results of the previous section to a public key cryptosystem. For a message $m < n$, we encrypt it by choosing a fixed exponent e, and defining the encrypted cyphertext to be:

$$c = m^e \bmod n\ .$$

For example, we might choose $e = 3$, and then $c = m^3 \bmod n$. Continuing our example of $n = 15$, if our message is 7, our encrypted codeword is $c = 7^3 = 13$. To decrypt it, we find the exponent d such that:

$$c^d = \left(m^e\right)^d = m^{ed} = m \qquad \Rightarrow \qquad \text{for } n = 15,\ ed = k \cdot 4 + 1.$$

For $e = 3$, $d = 3$ gives $ed = 9$, so our decryption exponent is $d = 3$ (that $d = e$ is an artifact of our unrealistically small integers, and never happens in practice). The

If our message $m > n$, we break it into smaller chunks, such that each chunk $< n$, and encrypt each chunk separately.

As another example, take $n = 35$. One can show that the largest $T = 12$, and all periods divide 12. Thus $m^{k\cdot 12+1} = m$, for all $m < 35$. We might again choose $e = 3$, and then must find d such that $ed = k\cdot 12 + 1$. But this is impossible, because all multiples of 12 are also multiples of 3, and so cannot be one less than a multiple of 3. We must choose $e < 12$ with no common factor with 12, say $e = 5$. Now find d such that $ed = k\cdot 12 + 1$. Trying multiples of 12 in order, we find that $24+1 = 5\cdot 5$, so $d = 5$. (This is another annoying coincidence that does not happen in practice, but *also* happens with $n = 35$ for the other possible values of e: 7 and 11.)

Another example: for $n = 77$, the maximum period is $T = 30$. Then we might choose an encryption exponent $e = 7$. The decryption exponent is 13: for all $m < 77$, $\left(m^7\right)^{13} = m^{91} = m \pmod{77}$.

In general, a decode exponent d satisfies $ed = k\phi(n) + 1$, or equivalently:

$$ed \bmod \phi(n) = 1, \quad \text{written as} \qquad d = e^{-1} \quad \left(\bmod \phi(n)\right) \qquad \textit{where} \quad e^{-1} \equiv \text{integer}\ .$$

In principle, this all you need to implement the RSA algorithm. The details are now in *feasibly* finding a set of numbers (e, n, d) for which the exponentiation is feasible, and modular roots are *infeasible*. Finding the set (e, n, d) is called **key generation**, and for n ~2048 bits, can be done in a few seconds on an ordinary computer (c. 2020).

Raising to Large Powers

Raising a number to a large power, such as 65537 or 10^{600} seems like a daunting task. But raising to a power mod n is feasible. Arbitrary size arithmetic packages are widely available for computers (e.g., Python includes arbitrary sized integers in the language). Thus 600-digit multiplies and remainder (modulus) operations are readily attainable.

One tool for evaluating large exponents is that after each multiply, we can take the result mod n, which keeps the size of the results within the size of n. Furthermore, squaring a number doubles its exponent:

$$\left(m^x\right)^2 = m^{2x}\ .$$

We can achieve any exponent by intermixing multiplies of m with the squarings. E.g.:

$$\left(m^2 \cdot m\right)^2 = m^6, \quad \left(m^2 \cdot m\right)^2 m = m^7, \qquad m^{65537} = (m)^{\textit{sqaured 16 times}} \cdot m\ .$$

Thus it takes only 17 multiplies to raise a number to the 65537th power. For decoding, $d \sim n$ or ~2,000 binary bits. Therefore, squaring a number 2,000 times raises it to the 2^{2000}th power. Totally feasible.

Binary bit-twiddlers will recognize that a general algorithm to raise m to the x^{th} power starts by writing x in binary, e.g. x = 1011 1101. Then:

```
r = 1.      // running accumulated product
```



```
For each bit of x from left to right:
      r = r*r
      if current bit == 1: r = r*m
```

The computation time is only $O(\log_2 x)$.

Finding the Maximum Period

For decoding, we need to know T, the maximum period of a power sequence mod n. In our trivial examples, we found T by trial and error. With 10^{600} possibilities, that is not feasible. Number theory comes to the rescue. The Euler **totient function** $\phi(n)$ gives either T, or a multiple of T, for any power sequence mod n. Either way, $\phi(n)$ is always a period. $\phi(n)$ is the number of numbers $1 \le r \le n-1$ that have no common factor with n. Figure 20.2b shows this for $n = 15$, where $\phi(15) = 8$. For n which are the product of two primes, the totient function is easy to calculate (Figure 20.2b):

1. We start with all the natural numbers < 15 = 3·5. There are 14 of them.
2. Then cross out all the multiples of 3; there are 15/3 – 1 = 4 of them.
3. Then cross out all the multiples of 5; there are 15/5 – 1 = 2 of them.
4. There are 14 – 4 – 2 = 8 numbers left, which have no common factor with 15.

Generalizing, for $n = pq$, with p, q both prime:

$$\phi(n) = (n-1) - \left(\underbrace{n/p}_{q} - 1 \right) - \left(\underbrace{n/q}_{p} - 1 \right) = n - p - q + 1 \qquad \text{[RSA77 6 p6]}.$$

Then the decode exponent d satisfies $ed = k\phi(n) + 1$, or equivalently:

$$ed \bmod \phi(n) = 1, \quad \text{written as} \quad d = e^{-1} \pmod{\phi(n)} \quad \textit{where} \quad e^{-1} \equiv \text{integer}.$$

For any two integers a, b, the extended Euclid algorithm efficiently finds a^{-1} mod b, and so can find the required $d = e^{-1}$. We do not describe the extended Euclid algorithm here.

In practice, $\phi(n)$ might be larger than needed, since it may be a multiple of T. Then d above is slightly inefficient. Instead, the slightly more complicated Carmichael function $\lambda(n)$ gives the maximum period T exactly [Wik02]. The optimum $d = e^{-1} \pmod{\lambda(n)}$.

Finding Large Primes: It's a Gamble

The previous section shows that, to be able to compute d, we must choose an n which is the product of two large primes, p and q. How do we find such primes? Guess and check: guess a number at random, and test it for being prime. The density of prime numbers decreases approximately like 1/log p, and for 1000-digit numbers, about 1 in 2,000 is prime.

For each guess, number theory is essential to test for "primality". There are efficient algorithms for *statistical* primality tests. To get a feel for how such tests work, recall Fermat's little theorem: if p is prime, then for any integer b, $b^p = b \pmod{p}$. (This is related to the totient function theorem.) For a random guess r, we can choose any base b, and compute b^r mod r. If the result $\neq b$, then r is not prime, and we guess again. If r passes the test, it may or may not be prime, i.e. our test may give a "false positive". Number theory can put upper bounds on the fraction of tests $f < 1$ yielding false positive. If we perform g independent tests, then the probability of a false positive on *all* of them is $< f^g$. Thus we can make our probability of error as small as we wish, by choosing g as large as we wish. It is easily feasible to make our chance of error $< 10^{-12}$.

Thus we cannot *prove* that our "primes" p and q are indeed prime, but we can make the chance of error so small as to be negligible.

RSA By the Numbers

Putting all the above together, an RSA matched key pair can be generated as follows:

1. The receiver finds two large primes, p and q, and defines $n = pq$.

2. e is typically chosen as 65 537, which is prime, and has no common factors with anything.
3. $d = e^{-1} \pmod{\lambda(n)}$.

The receiver publishes (e, n) in a public directory for all to see. She keeps d, p, and q secret. In fact, she can destroy p and q forever, since only d is needed going forward.

Practical Note on Efficiency

RSA is slow to compute: large exponents are costly. In practice, to save compute time, a sender with a large message usually encrypts it with a standard, fast algorithm (e.g., AES-256) using a randomly generated one-time key. The sender then RSA encrypts this key, and sends both the message cyphertext and the publicly-encrypted one-time key to the receiver.

Digital Signatures

Since $\left(m^e\right)^d = \left(m^d\right)^e$, RSA can be used for a digital signature: a number that proves the message came from the claimed sender. Suppose Bob sends Alice a message. Since it is rather personal, he wants Alice to know the message came from him, and no one is impersonating him. So he actually encrypts his message first with his secret *decode* exponent. Then encrypts it with Alice's public encode key. Alice decrypts it first with her private decode key, and then decodes it again with Bob's *public* encode exponent. Only Bob could know the secret d needed to make this last decoding work. Therefore, Bob's message is both private, and cannot be forged.

However, since Bob is rather wordy when expressing his feelings, his missive is too long to be efficiently encoded with RSA. For a more efficient signature, he computes a short cryptographic hash of his message (e.g., MD5 is a standard 128-bit hash). He then encrypts this hash with his secret decode key, and then with Alice's public key. Alice can verify that the signed hash matches the secret message, and therefore it must have come from Bob. Furthermore, Alice is free to publicly reveal Bob's RSA signature of the hash, so she can prove in court that Bob sent the message. Bob can never again deny his feelings.

Some experts recommend using a different key-pair for digital signing than for your received message public-key encryption, to minimize the exposure of encrypted messages to would-be attackers.

Vulnerabilities

Cryptographic security is very intricate and difficult. We can only touch on a few issues here, and we are *not* cryptographic experts. To protect your data, *carefully* follow the advice of cryptographic experts.

The biggest vulnerability is always hubris: people without extensive education and experience who invent their own, or modify, cryptosystems. (See IEEE's WiFi WEP debacle for a particularly egregious example of this.)

A mathematical breakthrough in quickly finding either modular roots of large numbers, or factors of large numbers, would break RSA. RSA has been known since 1977 [RSA77], and many people have tried to break it, but it remains secure as of 2020. Note that another public key method of the same era, the knapsack algorithm, has been broken [ref??].

Replay attack: An intruder can see an encrypted message. They may be able to inject a copy of this message at a later time, and thus fool the receiver. For example, if a genuine message says "Please send me $100. -Bob", an intruder could replay that at a later time, tricking Alice into sending another $100. Replays are usually prevented by having the sender put a sequence number (such as a time-stamp) in each message. The genuine sender always increments the sequence number, so the receiver will recognize an old replayed message as invalid. However, requiring a sequence number introduces "state" in the communication channel: the receiver must keep track of a separate sequence number for *each* sender.

RSA has many vulnerabilities for poorly chosen values of p and q, which are beyond our scope [Wik01].

Secret Message

To get your name in lights, be the first to decrypt this secret message, and send me an encrypted reply (show your work for several decryptions and encryptions). Encrypt each character separately. The "alphabet" is space = 2, A - Z = 3 - 28, (e, n) = (7, 77):

39 10 69 21 11 47 21 51 48 23 42 28 21

References

[Wik01] Wikipedia, "RSA (cryptosystem)", https://en.wikipedia.org/wiki/RSA_(cryptosystem), retrieved 2021-01-17.

[Wik02] Wikipedia, "Carmichael function", https://en.wikipedia.org/wiki/Carmichael_function, retrieved 2021-01-19.

[RSA77] Ronald L. Rivest, Adi Shamir, Len Adleman, MIT/LCS/TM-82 "On Digital Signatures And Public-Key Cryptosystems," April 1977, https://apps.dtic.mil/dtic/tr/fulltext/u2/a039036.pdf .

21 Appendices

References

[A&S] Abramowitz and Stegun, *Handbook of Mathematical Functions*, ??

[ASU 1986] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools*, 1986, Prentice Hall International, Inc., ISBN 0-201-10088-6.

[Chu] Churchill, Ruel V., Brown, James W., and Verhey, Roger F., *Complex Variables and Applications*, 1974, McGraw-Hill. ISBN 0-07-010855-2.

[Det] Dettman, John W., *Applied Complex Variables, 1965*, Dover. ISBN 0-486-64670-X.

[F&W] Fetter, Alexander L. and John Dirk Walecka, *Theoretical Mechanics for Particles and Continua*, McGraw-Hill Companies, February 1, 1980. ISBN-13: 978-0070206588.

[IEEE-754] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*, Std 754-2008, 8/29/2008.

[IEEE-754-1985] IEEE/ANSI, *IEEE Standard for Binary Floating-Point Arithmetic*, Std 754-1985, 7/26/1985.

[Jac 1999] John David Jackson, *Classical electrodynamics*, 3rd ed., John Wiley & Sons, Inc., 1999. ISBN 0-471-30932-X.

[M&T] Marion & Thornton, 4th ed.

[Mey 2001] Scott Meyers, *Effective STL*, Addison-Wesley Professional, June 16, 2001, ISBN-13: 978-0201749625.

[One] O'Neill, Barrett, *Elementary Differential Geometry*, 2nd ed., 1997, Academic Press. ISBN 0-12-526745-2.

[Sch] Schutz, Bernard F., *A First Course in General Relativity*, Cambridge University Press (January 31, 1985), ISBN 0521277035.

[Sch2] Schutz, Bernard F., *Geometrical Methods of Mathematical Physics*, Cambridge University Press ??, ISBN

[Schwa 1998] Schwarzenberg-Czerny, A., "The distribution of empirical periodograms: Lomb–Scargle and PDM spectra," Monthly Notices of the Royal Astronomical Society, vol 301, p831–840 (1998).

[Sea] Sean, *Sean's Applied Math Book*, 1/24/2004. http://www.its.caltech.edu/~sean/book.html.

[Strutz] Strutz, Tilo, *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and Beyond*, Springer, 2011. ISBN-13: 978-3-8348-1022-9 ISBN-10: 3834810223.

[Sun 2003] Sun Microsystems, Inc., *Numerical Computation Guide*, 2003.

[Tal] Talman, Richard, *Geometric Mechanics*, Wiley-Interscience; 1st edition (October 4, 1999), ISBN 0471157384

[Tay] Taylor, Angus E., *General Theory of Functions and Integration*, 1985, Dover. ISBN 0-486-64988-1.

[W&M] Walpole, Ronald E. and Raymond H. Myers, *Probability and Statistics for Engineers and Scientists*, 3rd edition, 1985, Macmillan Publishing Company, ISBN 0-02-424170-9.

[WMMY 2007] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye, *Probability & Statistics for Engineers & Scientists*, 8th edition, 2007, Pearson Prentice Hall, ISBN 0-13-187711-9.

[Wyl] Wyld, H. W., *Mathematical Methods for Physics*, 1999, Perseus Books Publishing, LLC, ISBN 0-7382-0125-1.

[Unk 1999] *Unknown author*, https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html , section "Differences Among IEEE 754 Implementations," retrieved 4/12/2020.

Glossary

Definitions of common mathematical physics terms. "Special" mathematical definitions are noted by "(math)". These are technical mathematical terms that you shouldn't have to know, but will make reading math books a lot easier because they are very common. These definitions try to be conceptual and accurate, but also comprehensible to "normal" people (including physicists, but not mathematicians).

1-1 function — A mapping (function) from a set A to a set B is 1-1 if every value of B under the map has only one value of A that maps to it. In other words, given the value of B under the map, we can uniquely find the value of A which maps to it. Equivalently, $Ma = Mb$ implies $a = b$. However, see "1-1 correspondence." Synonym: "injection." A 1-1 mapping is invertible.

1-1 correspondence — A mapping, between two sets A and B, is a 1-1 correspondence if it uniquely associates each value of A with a value of B, and each value of B with a value of A. Synonym: bijection.

accumulation point — syn. for limit point.

adjoint[1] — The adjoint of an operator produces a bra from a bra in the same way the original operator produces a ket from a ket: $\hat{\mathcal{O}}|\psi\rangle = |\phi\rangle \;\Rightarrow\; \langle\psi|\hat{\mathcal{O}}^\dagger = \langle\phi|, \;\forall\, |\psi\rangle$. The adjoint of an operator is the operator which preserves the inner product of two vectors as $\langle \mathbf{v}|\cdot(\mathrm{O}|\mathbf{w}\rangle) = (\mathrm{O}^\dagger|\mathbf{v}\rangle)^\dagger\cdot|\mathbf{w}\rangle$. The adjoint of an operator matrix is the conjugate-transpose. This has nothing to do with matrix adjoints (below).

adjoint[2] — In matrices, the transpose of the cofactor matrix is called the adjoint of a matrix. This has nothing to do with linear operator adjoints (above).

adjugate — the transpose of the cofactor matrix: $\mathrm{adj}(\mathbf{A})_{ij} = C_{ji} = (-1)^{i+j}M_{ji}$, where M_{ji} is the transpose of the minor matrix: $M_{ij} = \det(\mathbf{A}$ deleting row i and column $j)$.

analytic — A function is analytic in some domain IFF it has continuous derivatives to all orders, i.e. is infinitely differentiable. For complex functions of complex variables, if a function has a continuous first derivative in some region, then it has continuous derivatives to all orders, and is therefore analytic.

analytic geometry — the use of coordinate systems along with algebra and calculus to study geometry. Aka "coordinate geometry"

bijection — Both an "injection" and a "surjection," i.e. 1-1 and "onto." A mapping between sets A and B is a bijection iff it uniquely associates a value of A with every value of B. Synonym: 1-1 correspondence.

BLUE — In statistics, Best Linear Unbiased Estimator.

branch point — A branch point is a point in the domain of a complex function $f(z)$, z also complex, with this property: when z traverses a closed path around the branch point, following continuous values of $f(z)$, $f(z)$ has a different value at the end of the path than at the beginning, even though the beginning and end point are the same point in the domain. Example TBS: square root around the origin.

boundary point — (math) see "limit point."

by definition — in the very nature of the definition itself, without requiring any logical steps. To be distinguished from "by implication."

by implication — By combining the definition with other true statements, a conclusion can be shown by implication.

C or ℂ — the set of complex numbers.

closed — (math) contains any limit points. For finite regions, a closed region includes its boundary. Note that in math talk, a set can be both open and closed! The surface of a sphere is open (every point has a neighborhood in the surface), and closed (no excluded limit points; in fact, no limit points).

cofactor — The *ij*-th **minor** of an *n*×*n* matrix is the determinant of the (*n*–1)×(*n*–1) matrix formed by crossing out the *i*-th row and *j*-th column. A **cofactor** is just a minor with a plus or minus sign affixed, according to whether (*i*, *j*) is an even or odd number of steps away from (1,1):
$C_{ij} = (-1)^{i+j} M_{ij}$

compact — (math) for our purposes, closed and bounded [Tay thm 2-6I p66]. A compact region may comprise multiple (infinite number??) disjoint closed and bounded regions.

congruence — a set of 1D non-intersecting curves that cover every point of a manifold. Equivalently, a foliation of a manifold with 1D curves. Compare to "foliation."

contrapositive — The contrapositive of the statement "If A then B" is "If not B then not A." The contrapositive is equivalent to the statement: if the statement is true (or false), the contrapositive is true (or false). If the contrapositive is true (or false), the statement is true (or false).

convergent — approaches a definite limit. For functions, this usually means "convergent in the mean."

convergent in the mean — a function $f(x)$ is convergent in the mean to a limit function $L(x)$ IFF the mean-squared error approaches zero. Cf "uniformly convergent".

converse — The converse of the statement "If A then B" is "If B then A". In general, if a statement is true, its converse may be either true or false. The converse is the contrapositive of the inverse, and hence the converse and inverse are equivalent statements.

connected — There exists a continuous path between any two points in the set (region). See also: simply connected. [One p178].

coordinate geometry — the use of coordinate systems along with algebra and calculus to study geometry. Aka "analytic geometry".

decreasing — non-increasing: a function is decreasing IFF for all $b > a$, $f(b) \le f(a)$. Note that "monotonically decreasing" is the same as "decreasing". This may be restricted to an interval, e.g. $f(x)$ is decreasing on [0, 1]. Compare "strictly decreasing".

device — a procedure that's only used twice. See "trick" and "method."

diffeomorphism — a C^∞ (1-1) map, with a C^∞ inverse, from one manifold *onto* another. "Onto" implies the mapping covers the whole range manifold. Two diffeomorphic manifolds are topologically identical, but may have different geometries.

divergent — not convergent: a sequence is divergent IFF it is not convergent.

domain — of a function: the set of numbers (usually real or complex) on which the function is defined.

elliptic operator — A second-order differential operator in multiple dimensions, whose second-order coefficient matrix has eigenvalues all of the same algebraic sign (none zero).

entire — A complex function is entire IFF it is analytic over the entire complex plane. An entire function is also called an "integral function."

error — Statistics: residual, the difference between a measurement and the model value.

essential singularity — a "pole" of infinite order, i.e. a singularity around which the function is unbounded, and cannot be made finite by multiplication by any power of $(z - z_0)$ [Det p165].

expected value — a misnomer for "average."

fact — A small piece of information backed by solid evidence (in hard science, usually repeatable evidence). If someone disputes a fact, it is still a fact. "If a thousand people say a foolish thing, it is still a foolish thing." See also "speculation," and "theory."

factor — a number (or more general object) that is *multiplied* with others. E.g., in "$(a + b)(x + y)$", there are two factors: "$(a + b)$", and "$(x + y)$".

finite — a non-zero number. In other words, not zero, and not infinity.

foliation — a set of non-intersecting submanifolds that cover every point of a manifold. E.g., 3D real space can be foliated into 2D "sheets stacked on top of each other," or 1D curves packed around each other. Compare to "congruence."

functionality — a pompous word for "function" or "operation".

holomorphic — syn. for analytic. Other synonyms are regular, and differentiable. Also, a "holomorphic map" is just an analytic function.

homomorphic — something from abstract categories that should not be confused with homeomorphism.

homeomorphism — a continuous (1-1) map, with a continuous inverse, from one manifold *onto* another. "Onto" implies the mapping covers the whole range manifold. A homeomorphism that preserves distance is an **isometry**.

hyperbolic operator — A second-order differential operator in multiple dimensions, whose second-order coefficient matrix has non-zero eigenvalues, with one of opposite algebraic sign to all the others.

identify — to establish a 1-1 and *onto* relationship. If we identify two mathematical things, they are essentially the same thing.

IFF (or iff) — if, and only if.

injection — A mapping from a set A to a set B is an injection if it is 1-1, that is, if a value of B in the mapping uniquely determines the value of A which maps to it. Note that every value of A is included by the definition of "mapping" [CRC 30th], but the mapping does *not* have to cover all the elements of B.

integral function — Syn. for "entire function:" a function that is analytic over the entire complex plane.

inverse — The inverse of the statement "If A then B" is "If not A then not B." In general, if a statement is true, its inverse may be either true or false. The inverse is the contrapositive of the converse, and hence the converse and inverse are equivalent statements.

invertible — A map (or function) from a set A to a set B is invertible iff for every value in B, there exists a unique value in A which maps to it. In other words, a map is invertible iff it is a bijection.

isolated singularity — a singularity at a point, which has a surrounding neighborhood of analyticity [Det p165].

isometry — a homeomorphism that preserves distance, i.e. a continuous, invertible (1-1) map from one manifold *onto* another that preserves distance ("onto" in the mathematical sense).

isomorphic — "same structure." A widely used general term, with no single precise definition.

limit point — of a domain is a boundary of a region of the domain: for example, the open interval (0, 1) on the number line and the closed interval [0, 1] both have limit points of 0 and 1. In this case, the open interval excludes its limit points; the closed interval includes them (definition of "closed"). Some definitions define all points in the domain as also limit

points. Formally, a point p is a limit point of domain D iff every open subset containing p also contains a point in D other than p.

mantissa — the number part of scientific notation: e.g., 3.14×10^7 has a mantissa of 3.14. In binary, 1.01×2^{15} has a mantissa of 1.01.

mapping — syn. "function." A mapping from a set A to a set B defines a value of B for *every* value of A [CRC 30th].

meromorphic — A function is meromorphic on a domain IFF it is analytic except at a set of isolated *poles* of finite order (i.e., non-essential poles). Note that branch points are nonanalytic points, but some are not poles (such as $\sqrt{z}$ at zero), so a function including such a branch point is *not* meromorphic.

method — a procedure that's used 3 or more times. See "trick" and "device."

minor — The ij-th minor of an $n\times n$ matrix is the determinant of the $(n–1)\times(n–1)$ matrix formed by crossing out the i-th row and j-th column, i.e., the minor matrix: M_{ij} = det(**A** deleting row i and column j). See also "cofactor."

monotonic — all the same: a monotonic function is either increasing or decreasing (on a given interval). Note that "monotonically increasing" is the same as "increasing". See also "increasing" and "decreasing".

$\mathbb{N}$ — the set of natural numbers (positive integers).

noise — *unpredictable* variations in a quantity.

oblique — non-orthogonal and not parallel.

one-to-one — see "1-1".

onto — covering every possible value. A mapping from a set A *onto* the set B covers every possible value of B, i.e. the mapping is a surjection.

open — (math) An region is open iff every point in the region has a finite neighborhood of points around it that are also all in the region. In other words, every point is an interior point. Note that open is *not* "not closed;" a region can be both open and closed.

OS — operating system, e.g. Unix, OS X, Windows.

parabolic operator — A second-order differential operator in multiple dimensions, whose second-order coefficient matrix has at least one zero eigenvalue.

PDM — phase dispersion minimization: an algorithm for finding arbitrarily shaped periodic signals in a (t, y), or (x, y), data set.

pole — a singularity near which a function is unbounded, but which becomes finite by multiplication by $(z - z_0)^k$ for some finite k [Det p165]. The value k is called the **order** of the pole.

positive definite — a matrix or operator which is > 0 for all *non-zero* operands. It may be 0 when acting on a "zero" operand, such as the zero vector. This implies that all eigenvalues > 0.

positive semidefinite — a matrix or operator which is ≥ 0 for all *non-zero* operands. It may be 0 when acting on a non-zero operands. This implies that all eigenvalues ≥ 0.

predictor — in regression: a variable put into a model to predict another value, e.g. $y_{mod}(x_1, x_2)$ is a model (function) of the predictors x_1 and x_2.

PT — perturbation theory.

Q or $\mathbb{Q}$ — the set of rational numbers. $\mathbf{Q}^+ \equiv$ the set of positive rationals.

R or $\mathbb{R}$ — the set of real numbers.

refactor — programming: a pompous word for "restructure".

RMS — root-mean-square.

RV — random variable.

removable singularity — an isolated singularity that can be made analytic by simply defining a value for the function at that point. For example, $f(x) = sin(x)/x$ has a singularity at $x = 0$. You can remove it by defining $f(0) = 1$. Then f is everywhere analytic. [Det p165]

residue — The residue of a complex function at a complex point z_0 is the a_{-1} coefficient of the Laurent expansion about the point z_0.

significand — IEEE word for "mantissa."

simply connected — There are no holes in the set (region), not even point holes. I.e., you can shrink any closed curve in the region down to a point, the curve staying always within the region (including at the point).

singularity — of a function: a point on a boundary (i.e. a limit point) of the domain of analyticity, but where the function is not analytic. [Det def 4.5.2 p156]. Note that the function may be defined at the singularity, but it is not analytic there. E.g., $\sqrt{z}$ is continuous at 0, but not differentiable.

smooth — for most references, "smooth" means infinitely differentiable, i.e. C^∞. For some, though, "smooth" means at least one continuous derivative, i.e. C^1, meaning first derivative continuous. This latter definition looks "smooth" to our eye (no kinks, or sharp points).

speculation — A guess, possibly hinted at by evidence, but not well supported. Every scientific fact and theory started as a speculation. See also "fact," and "theory."

STL — Standard Template Library, a set of libraries in the C++ language.

strictly decreasing — a function is strictly decreasing IFF for all $b > a$, $f(b) < f(a)$. This may be restricted to an interval, e.g. $f(x)$ is strictly decreasing on [0, 1]. Compare "decreasing".

strictly increasing — a function is strictly increasing IFF for all $a < b$, $f(a) < f(b)$. This may be restricted to an interval, e.g. $f(x)$ is strictly increasing on [0, 1]. Compare "increasing".

surjection — A mapping from a set A "onto" the set B, i.e. that covers every possible value of B. Note that every value of A is included by the definition of "mapping" [CRC 30th], however multiple values of A may map to the same value of B.

technical debt — software development: the maintenance burden incurred from messy code. It often results from hasty changes made in order to meet a deadline. The code works, but is difficult to maintain, and will likely need to be cleaned up in the future.

term — a number (or more general object) that is *added* to others. E.g., in "$ax + by + cz$", there are three terms: "ax", "by", and "cz".

theory — the highest level of scientific achievement: a quantitative, predictive, testable model which unifies and relates a body of facts. A theory becomes accepted science only after being supported by overwhelming evidence. A theory is not a speculation, e.g. Maxwell's electromagnetic theory. See also "fact," and "speculation."

trace — the trace of a square matrix is the sum of its diagonal elements.

trick — a procedure that's only used once. See "device" and "method."

uniform convergence — a sequence of functions $f_n(z)$ is uniformly convergent in an open (or partly open) region IFF its error ε after the N^{th} function can be made arbitrarily small with a single value of N (dependent *only* on ε) for every point in the region. I.e. given ε, a single N works for all points z in the region [Chu p156].

voila — French contraction of "see there!"

WLOG or WOLOG without loss of generality

$\mathbf{Z}$ or $\mathbb{Z}$ the set of integers. $\mathbf{Z}^+$ or $\mathbb{N}$ ≡ the set of positive integers (natural numbers).

Formulas

completing the square: $ax^2 + bx = a\left(x + \frac{b}{2a}\right)^2 - \frac{b^2}{4a}$ (x-shift = $-b / 2a$)

Definite Integrals

$$\int_{-\infty}^{\infty} dx\, e^{-ax^2} = \sqrt{\frac{\pi}{a}} \qquad \int_{-\infty}^{\infty} dx\, x^2 e^{-ax^2} = \frac{1}{2}\sqrt{\frac{\pi}{a^3}} \qquad \int_0^{\infty} dr\, r^3 e^{-ar^2} = \frac{1}{2a^2}$$

Statistical distributions

χ^2_ν: $avg = \nu$ $\sigma^2 = 2\nu$

exponential: $avg = \tau$ $\sigma^2 = \tau^2$

error function [A&S]: $erf(x) \equiv \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$

gaussian included probability between $-z$ and $+z$:

$$p_{gaussian}(z) \equiv \int_{-z}^{+z} \text{pdf}_{gaussian}(u)\, du = \frac{1}{\sqrt{2\pi}} \int_{-z}^{+z} e^{-u^2/2}\, du \qquad Let \qquad u^2/2 \equiv t^2,\ du = \sqrt{2}dt$$

$$= \frac{2}{\sqrt{2\pi}} \int_0^{+z/\sqrt{2}} e^{-t^2}\, \sqrt{2}dt = erf\left(z/\sqrt{2}\right)$$

Special Functions

$\Gamma(n) = (n-1)!$ $\Gamma(a) \equiv \int_0^{\infty} dx\, x^{a-1} e^{-x}$ $\Gamma(a) = (a-1)\Gamma(a-1)$ $\Gamma(1/2) = \sqrt{\pi}$

The functions below use the Condon-Shortley phase:

$$Y_{lm}(\theta,\phi) \equiv \begin{cases} (-1)^m \sqrt{\frac{(2l+1)}{2}\frac{(l-m)!}{(l+m)!}} P_{lm}(\cos\theta) \frac{e^{im\phi}}{\sqrt{2\pi}}, & m \ge 0, \\ \sqrt{\frac{(2l+1)}{2}\frac{(l-|m|)!}{(l+|m|)!}} P_{l|m|}(\cos\theta) \frac{e^{im\phi}}{\sqrt{2\pi}}, & m < 0, \end{cases}$$

$P_{lm}(x)$ is the associated Legendre function,

$l = 0, 1, 2..., \qquad m = -l, -l+1, ... l-1, l$. [Wyl 3.6.5 p96]

Index

The index is not yet developed, so go to the web page on the front cover, and text-search in this document.

