

Advanced Data Analysis from an Elementary Point of View

Cosma Rohilla Shalizi

Spring 2013
Last L^AT_EX'd Thursday 25th April, 2013

Contents

Introduction	13
To the Reader	13
0.0.0.1 Updates	14
Concepts You Should Know	14
I Regression and Its Generalizations	16
1 Regression Basics	17
1.1 Statistics, Data Analysis, Regression	17
1.2 Guessing the Value of a Random Variable	18
1.2.1 Estimating the Expected Value	19
1.3 The Regression Function	19
1.3.1 Some Disclaimers	20
1.4 Estimating the Regression Function	23
1.4.1 The Bias-Variance Tradeoff	23
1.4.2 The Bias-Variance Trade-Off in Action	25
1.4.3 Ordinary Least Squares Linear Regression as Smoothing	25
1.5 Linear Smoothers	30
1.5.1 k -Nearest-Neighbor Regression	30
1.5.2 Kernel Smoothers	32
1.6 Exercises	35
2 The Truth about Linear Regression	36
2.1 Optimal Linear Prediction: Multiple Variables	36
2.1.1 Collinearity	38
2.1.2 The Prediction and Its Error	38
2.1.3 Estimating the Optimal Linear Predictor	39
2.1.3.1 Unbiasedness and Variance of Ordinary Least Squares Estimates	40
2.2 Shifting Distributions, Omitted Variables, and Transformations	41
2.2.1 Changing Slopes	41
2.2.1.1 R^2 : Distraction or Nuisance?	41
2.2.2 Omitted Variables and Shifting Distributions	41

2.2.3	Errors in Variables	43
2.2.4	Transformation	47
2.3	Adding Probabilistic Assumptions	49
2.3.1	Examine the Residuals	51
2.3.2	On Significant Coefficients	52
2.4	Linear Regression Is Not the Philosopher's Stone	53
2.5	Exercises	55
3	Model Evaluation	56
3.1	What Are Statistical Models For?	56
3.2	Errors, In and Out of Sample	57
3.3	Over-Fitting and Model Selection	61
3.4	Cross-Validation	66
3.4.1	Data-set Splitting	67
3.4.2	k -Fold Cross-Validation (CV)	67
3.4.3	Leave-one-out Cross-Validation	70
3.5	Warnings	70
3.5.1	Parameter Interpretation	71
3.6	Exercises	72
4	Smoothing in Regression	73
4.1	How Much Should We Smooth?	73
4.2	Adapting to Unknown Roughness	74
4.2.1	Bandwidth Selection by Cross-Validation	84
4.2.2	Convergence of Kernel Smoothing and Bandwidth Scaling	85
4.2.3	Summary on Kernel Smoothing	90
4.3	Kernel Regression with Multiple Inputs	90
4.4	Interpreting Smoothers: Plots	91
4.5	Average Predictive Comparisons	95
4.6	Exercises	98
5	Simulation	99
5.1	What Do We Mean by "Simulation"?	99
5.2	How Do We Simulate Stochastic Models?	100
5.2.1	Chaining Together Random Variables	100
5.2.2	Random Variable Generation	100
5.2.2.1	Built-in Random Number Generators	100
5.2.2.2	Transformations	101
5.2.2.3	Quantile Method	101
5.2.2.4	Rejection Method	102
5.2.2.5	The Metropolis Algorithm and Markov Chain Monte Carlo	105
5.2.2.6	Generating Uniform Random Numbers	106
5.2.3	Sampling	107
5.2.3.1	Sampling Rows from Data Frames	110
5.2.3.2	Multinomials and Multinoullis	110

5.2.3.3	Probabilities of Observation	110
5.2.4	Repeating Simulations	111
5.3	Why Simulate?	111
5.3.1	Understanding the Model; Monte Carlo	111
5.3.2	Checking the Model	112
5.3.3	Sensitivity Analysis	115
5.4	The Method of Simulated Moments	117
5.4.1	The Method of Moments	117
5.4.2	Adding in the Simulation	118
5.4.3	An Example: Moving Average Models and the Stock Market .	118
5.5	Exercises	125
5.6	Appendix: Some Design Notes on the Method of Moments Code .	127
6	The Bootstrap	129
6.1	Stochastic Models, Uncertainty, Sampling Distributions	129
6.2	The Bootstrap Principle	131
6.2.1	Variances and Standard Errors	133
6.2.2	Bias Correction	133
6.2.3	Confidence Intervals	134
6.2.3.1	Other Bootstrap Confidence Intervals	135
6.2.4	Hypothesis Testing	136
6.2.4.1	Double bootstrap hypothesis testing	137
6.2.5	Parametric Bootstrapping Example: Pareto's Law of Wealth Inequality	137
6.3	Non-parametric Bootstrapping	141
6.3.1	Parametric vs. Nonparametric Bootstrapping	144
6.4	Bootstrapping Regression Models	144
6.4.1	Re-sampling Points: Parametric Example	145
6.4.2	Re-sampling Points: Non-parametric Example	147
6.4.3	Re-sampling Residuals: Example	150
6.5	Bootstrap with Dependent Data	152
6.6	Things Bootstrapping Does Poorly	152
6.7	Further Reading	153
6.8	Exercises	153
7	Weighting and Variance	154
7.1	Weighted Least Squares	154
7.2	Heteroskedasticity	155
7.2.1	Weighted Least Squares as a Solution to Heteroskedasticity .	156
7.2.2	Some Explanations for Weighted Least Squares	158
7.2.3	Finding the Variance and Weights	162
7.3	Conditional Variance Function Estimation	163
7.3.1	Iterative Refinement of Mean and Variance: An Example . . .	164
7.3.2	Real Data Example: Old Heteroskedastic	168
7.4	Re-sampling Residuals with Heteroskedasticity	171
7.5	Local Linear Regression	171

7.5.1	Advantages and Disadvantages of Locally Linear Regression	173
7.5.2	Lowess	174
7.6	Exercises	176
8	Splines	177
8.1	Smoothing by Directly Penalizing Curve Flexibility	177
8.1.1	The Meaning of the Splines	178
8.2	Computational Example: Splines for Stock Returns	180
8.2.1	Confidence Bands for Splines	183
8.3	Basis Functions and Degrees of Freedom	186
8.3.1	Basis Functions	186
8.3.2	Degrees of Freedom	188
8.4	Splines in Multiple Dimensions	190
8.5	Smoothing Splines versus Kernel Regression	190
8.6	Further Reading	190
8.7	Exercises	191
9	Additive Models	193
9.1	Partial Residuals and Back-fitting for Linear Models	193
9.2	Additive Models	194
9.3	The Curse of Dimensionality	197
9.4	Example: California House Prices Revisited	199
9.5	Closing Modeling Advice	211
9.6	Further Reading	211
10	Testing Regression Specifications	213
10.1	Testing Functional Forms	213
10.1.1	Examples of Testing a Parametric Model	215
10.1.2	Remarks	224
10.1.2.0.1	Other Nonparametric Regressions	224
10.1.2.0.2	Additive Alternatives	224
10.1.2.0.3	Testing $E[\hat{\epsilon} X] = 0$	224
10.1.2.0.4	Stabilizing the Sampling Distribution of the Test Statistic	224
10.2	Why Use Parametric Models At All?	225
10.3	Why We Sometimes Want Mis-Specified Parametric Models	226
11	More about Hypothesis Testing	230
12	Logistic Regression	231
12.1	Modeling Conditional Probabilities	231
12.2	Logistic Regression	232
12.2.1	Likelihood Function for Logistic Regression	235
12.2.2	Logistic Regression with More Than Two Classes	236
12.3	Newton's Method for Numerical Optimization	237
12.3.1	Newton's Method in More than One Dimension	239

12.3.2 Iteratively Re-Weighted Least Squares	239
12.4 Generalized Linear Models and Generalized Additive Models	240
12.4.1 Generalized Additive Models	241
12.4.2 An Example (Including Model Checking)	241
12.5 Exercises	245
13 GLMs and GAMs	246
13.1 Generalized Linear Models and Iterative Least Squares	246
13.1.1 GLMs in General	248
13.1.2 Examples of GLMs	248
13.1.2.1 Vanilla Linear Models	248
13.1.2.2 Binomial Regression	249
13.1.2.3 Poisson Regression	249
13.1.3 Uncertainty	250
13.2 Generalized Additive Models	250
13.3 Weather Forecasting in Snoqualmie Falls	251
13.4 Exercises	264

II Multivariate Data, Distributions, and Latent Structure 266

14 Multivariate Distributions	267
14.1 Review of Definitions	267
14.2 Multivariate Gaussians	268
14.2.1 Linear Algebra and the Covariance Matrix	270
14.2.2 Conditional Distributions and Least Squares	271
14.2.3 Projections of Multivariate Gaussians	271
14.2.4 Computing with Multivariate Gaussians	271
14.3 Inference with Multivariate Distributions	272
14.3.1 Estimation	272
14.3.2 Model Comparison	273
14.3.3 Goodness-of-Fit	275
14.4 Exercises	276
15 Density Estimation	277
15.1 Histograms Revisited	277
15.2 “The Fundamental Theorem of Statistics”	278
15.3 Error for Density Estimates	279
15.3.1 Error Analysis for Histogram Density Estimates	280
15.4 Kernel Density Estimates	282
15.4.1 Analysis of Kernel Density Estimates	282
15.4.2 Joint Density Estimates	284
15.4.3 Categorical and Ordered Variables	285
15.4.4 Practicalities	286
15.4.5 Kernel Density Estimation in R: An Economic Example	286
15.5 Conditional Density Estimation	288

15.5.1	Practicalities and a Second Example	289
15.6	More on the Expected Log-Likelihood Ratio	292
15.7	Simulating from Density Estimates	293
15.7.1	Simulating from Kernel Density Estimates	293
15.7.1.1	Sampling from a Kernel Joint Density Estimate	294
15.7.1.2	Sampling from Kernel Conditional Density Estimates	294
15.7.2	Sampling from Histogram Estimates	295
15.7.3	Examples of Simulating from Kernel Density Estimates	295
15.8	Exercises	300
16	Relative Distributions and Smooth Tests	301
16.1	Smooth Tests of Goodness of Fit	301
16.1.1	From Continuous CDFs to Uniform Distributions	301
16.1.2	Testing Uniformity	302
16.1.3	Neyman's Smooth Test	302
16.1.3.1	Choice of Function Basis	304
16.1.3.2	Choice of Number of Basis Functions	305
16.1.3.3	Application: Combining p -Values	305
16.1.3.4	Density Estimation by Series Expansion	308
16.1.4	Smooth Tests of Non-Uniform Parametric Families	308
16.1.4.1	Estimated Parameters	310
16.1.5	Implementation in R	311
16.1.5.1	Some Examples	311
16.1.6	Conditional Distributions and Calibration	314
16.2	Relative Distributions	315
16.2.1	Estimating the Relative Distribution	317
16.2.2	R Implementation and Examples	317
16.2.2.1	Example: Conservative versus Liberal Brains	317
16.2.2.2	Example: Economic Growth Rates	321
16.2.3	Adjusting for Covariates	322
16.2.3.1	Example: Adjusting Growth Rates	324
16.3	Further Reading	327
16.4	Exercises	327
17	Principal Components Analysis	328
17.1	Mathematics of Principal Components	328
17.1.1	Minimizing Projection Residuals	329
17.1.2	Maximizing Variance	330
17.1.3	More Geometry; Back to the Residuals	331
17.1.4	Statistical Inference, or Not	332
17.2	Example: Cars	333
17.3	Latent Semantic Analysis	336
17.3.1	Principal Components of the New York <i>Times</i>	337
17.4	PCA for Visualization	339
17.5	PCA Cautions	341
17.6	Exercises	342

18 Factor Analysis	345
18.1 From PCA to Factor Analysis	345
18.1.1 Preserving correlations	347
18.2 The Graphical Model	347
18.2.1 Observables Are Correlated Through the Factors	349
18.2.2 Geometry: Approximation by Hyper-planes	350
18.3 Roots of Factor Analysis in Causal Discovery	350
18.4 Estimation	351
18.4.1 Degrees of Freedom	352
18.4.1.0.1 More unknowns (free parameters) than equations (constraints)	353
18.4.1.0.2 More equations (constraints) than unknowns (free parameters)	353
18.4.2 A Clue from Spearman’s One-Factor Model	354
18.4.3 Estimating Factor Loadings and Specific Variances	355
18.5 Maximum Likelihood Estimation	355
18.5.1 Alternative Approaches	356
18.5.2 Estimating Factor Scores	357
18.6 The Rotation Problem	357
18.7 Factor Analysis as a Predictive Model	358
18.7.1 How Many Factors?	359
18.7.1.1 R^2 and Goodness of Fit	360
18.8 Reification, and Alternatives to Factor Models	361
18.8.1 The Rotation Problem Again	361
18.8.2 Factors or Mixtures?	361
18.8.3 The Thomson Sampling Model	363
19 Mixture Models	368
19.1 Two Routes to Mixture Models	368
19.1.1 From Factor Analysis to Mixture Models	368
19.1.2 From Kernel Density Estimates to Mixture Models	368
19.1.3 Mixture Models	369
19.1.4 Geometry	370
19.1.5 Identifiability	370
19.1.6 Probabilistic Clustering	371
19.1.7 Simulation	372
19.2 Estimating Parametric Mixture Models	372
19.2.1 More about the EM Algorithm	374
19.2.2 Further Reading on and Applications of EM	377
19.2.3 Topic Models and Probabilistic LSA	377
19.3 Non-parametric Mixture Modeling	377
19.4 Worked Computing Example	378
19.4.1 Mixture Models in R	378
19.4.2 Fitting a Mixture of Gaussians to Real Data	378
19.4.3 Calibration-checking for the Mixture	382
19.4.4 Selecting the Number of Components by Cross-Validation . .	384

19.4.5 Interpreting the Mixture Components, or Not	389
19.4.6 Hypothesis Testing for Mixture-Model Selection	394
19.5 Exercises	397
20 Graphical Models	398
20.1 Conditional Independence and Factor Models	398
20.2 Directed Acyclic Graph (DAG) Models	399
20.2.1 Conditional Independence and the Markov Property	400
20.3 Examples of DAG Models and Their Uses	401
20.3.1 Missing Variables	404
20.4 Non-DAG Graphical Models	405
20.4.1 Undirected Graphs	405
20.4.1.0.1 Further reading	406
20.4.2 Directed but Cyclic Graphs	406
20.5 Further Reading	407
III Causal Inference	409
21 Graphical Causal Models	410
21.1 Causation and Counterfactuals	410
21.2 Causal Graphical Models	411
21.2.1 Calculating the “effects of causes”	412
21.2.2 Back to Teeth	413
21.3 Conditional Independence and <i>d</i> -Separation	416
21.3.1 D-Separation Illustrated	418
21.3.2 Linear Graphical Models and Path Coefficients	420
21.3.3 Positive and Negative Associations	421
21.4 Independence and Information	422
21.5 Further Reading	423
21.6 Exercises	424
22 Identifying Causal Effects	425
22.1 Causal Effects, Interventions and Experiments	425
22.1.1 The Special Role of Experiment	426
22.2 Identification and Confounding	427
22.3 Identification Strategies	430
22.3.1 The Back-Door Criterion: Identification by Conditioning	432
22.3.1.1 The Entner Rules	433
22.3.2 The Front-Door Criterion: Identification by Mechanisms	435
22.3.2.1 The Front-Door Criterion and Mechanistic Explanation	436
22.3.3 Instrumental Variables	438
22.3.3.1 Some Invalid Instruments	440
22.3.3.2 Critique of Instrumental Variables	442
22.3.4 Failures of Identification	444

CONTENTS	10
22.4 Summary	446
22.4.1 Further Reading	446
22.5 Exercises	446
23 Estimating Causal Effects	448
23.1 Estimators in the Back- and Front- Door Criteria	448
23.1.1 Estimating Average Causal Effects	449
23.1.2 Avoiding Estimating Marginal Distributions	449
23.1.3 Propensity Scores	450
23.1.4 Matching and Propensity Scores	452
23.2 Instrumental-Variables Estimates	454
23.3 Uncertainty and Inference	455
23.4 Recommendations	455
23.5 Further Reading	456
23.6 Exercises	457
24 Discovering Causal Structure	458
24.1 Testing DAGs	459
24.2 Testing Conditional Independence	460
24.3 Faithfulness and Equivalence	461
24.3.1 Partial Identification of Effects	462
24.4 Causal Discovery with Known Variables	462
24.4.1 The PC Algorithm	465
24.4.2 Causal Discovery with Hidden Variables	466
24.4.2.0.1 Partial identification of effects	466
24.4.3 On Conditional Independence Tests	466
24.5 Software and Examples	467
24.6 Limitations on Consistency of Causal Discovery	472
24.7 Further Reading	473
24.8 Exercises	473
IV Dependent Data	474
25 Time Series	475
25.1 Time Series, What They Are	475
25.1.0.0.2 Other kinds of time series	475
25.1.0.0.3 Notation	477
25.2 Stationarity	477
25.2.1 Autocorrelation	477
25.2.2 The Ergodic Theorem	481
25.2.2.1 The World's Simplest Ergodic Theorem	481
25.2.2.2 Rate of Convergence	482
25.2.2.3 Why Ergodicity Matters	483
25.3 Markov Models	484
25.3.1 Meaning of the Markov Property	485

25.4	Autoregressive Models	486
25.4.1	Autoregressions with Covariates	487
25.4.2	Additive Autoregressions	487
25.4.2.0.1	Example: The lynx	487
25.4.3	Linear Autoregression	487
25.4.3.1	“Unit Roots” and Stationary Solutions	492
25.4.4	Conditional Variance	494
25.4.4.0.1	Example: lynx	494
25.4.5	Regression with Correlated Noise; Generalized Least Squares	494
25.5	Bootstrapping Time Series	497
25.5.1	Parametric or Model-Based Bootstrap	497
25.5.2	Block Bootstraps	497
25.5.3	Sieve Bootstrap	498
25.6	Trends and De-Trending	500
25.6.1	Forecasting Trends	502
25.6.2	Seasonal Components	507
25.6.3	Detrending by Differencing	507
25.7	Further Reading	508
25.8	Exercises	510
26	Time Series with Latent Variables	511
27	Longitudinal, Spatial and Network Data	512
Appendices		514
A	Programming	515
A.1	Functions	515
A.2	First Example: Pareto Quantiles	516
A.3	Functions Which Call Functions	517
A.3.1	Sanity-Checking Arguments	519
A.4	Layering Functions and Debugging	519
A.4.1	More on Debugging	522
A.5	Automating Repetition and Passing Arguments	522
A.6	Avoiding Iteration: Manipulating Objects	533
A.6.1	<code>ifelse</code> and <code>which</code>	534
A.6.2	<code>apply</code> and Its Variants	535
A.7	More Complicated Return Values	537
A.8	Re-Writing Your Code: An Extended Example	538
A.9	General Advice on Programming	544
A.9.1	Comment your code	544
A.9.2	Use meaningful names	545
A.9.3	Check whether your program works	545
A.9.4	Avoid writing the same thing twice	546
A.9.5	Start from the beginning and break it down	546

CONTENTS	12
A.9.6 Break your code into many short, meaningful functions	546
A.10 Further Reading	547
B Big O and Little \circ Notation	548
C χ^2 and the Likelihood Ratio Test	550
D Proof of the Gauss-Markov Theorem	553
E Constrained and Penalized Optimization	555
E.1 Constrained Optimization	555
E.2 Lagrange Multipliers	556
E.3 Penalized Optimization	557
E.4 Mini-Example: Constrained Linear Regression	557
E.4.1 Statistical Remark: “Ridge Regression” and “The Lasso”	559
F Rudimentary Graph Theory	566
G Pseudo-code for the SGS Algorithm	569
G.1 Pseudo-code for the SGS Algorithm	569
G.2 Pseudo-code for the PC Algorithm	570

Introduction

To the Reader

These are the notes for 36-402, Advanced Data Analysis, at Carnegie Mellon. If you are not enrolled in the class, you should know that it's the methodological capstone of the core statistics sequence taken by our undergraduate majors (usually in their third year), and by students from a range of other departments. By this point, they have taken classes in introductory statistics and data analysis, probability theory, mathematical statistics, and modern linear regression ("401"). This class does not presume that you have learned but forgotten the material from the pre-requisites; it presumes that you *know* that material and can go beyond it. The class also presumes a firm grasp on linear algebra and multivariable calculus, and that you can read and write simple functions in R. If you are lacking in any of these areas, now would be an excellent time to leave.

36-402 is a class in *statistical methodology*: its aim is to get students to understand something of the range of modern¹ methods of data analysis, and of the considerations which go into choosing the right method for the job at hand (rather than distorting the problem to fit the methods the student happens to know). Statistical theory is kept to a minimum, and largely introduced as needed.

Since 36-402 is also a class in *data analysis*, there are assignments in which, nearly every week, a new, often large, data set is analyzed with new methods. (I reserve the right to re-use data sets, and even to fake data, but will do so sparingly.) Assignments and data will be on the class web-page.

There is no way to cover every important topic for data analysis in just a semester. Much of what's not here — sampling, experimental design, advanced multivariate methods, hierarchical models, the intricacies of categorical data, graphics, data mining — gets covered by our other undergraduate classes. Other important areas, like dependent data, inverse problems, model selection or robust estimation, have to wait for graduate school.

The mathematical level of these notes is deliberately low; nothing should be beyond a competent second-year student. But every subject covered here can be profitably studied using vastly more sophisticated techniques; that's why this is advanced data analysis from an *elementary* point of view. If reading these pages inspires any-

¹Just as an undergraduate "modern physics" course aims to bring the student up to about 1930 (more specifically, to 1926), this class aims to bring the student up to about 1990.

one to study the same material from an *advanced* point of view, I will consider my troubles to have been amply repaid.

A final word. At this stage in your statistical education, you have gained two kinds of knowledge — a few general statistical principles, and many more specific procedures, tests, recipes, etc. If you are a typical ADA student, you are much more comfortable with the specifics than the generalities. But the truth is that while none of your recipes are *wrong*, they are tied to assumptions which hardly ever hold. Learning more flexible and powerful methods, which have a much better hope of being reliable, will demand a lot of hard thinking and hard work. Those of you who succeed, however, will have done something you can be proud of.

0.0.0.0.1 Updates The page for this book is <http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/>. The latest version will live there. It will eventually be published by Cambridge University Press, at which point there will still be a free next-to-final draft at that URL, and errata. I plan to incorporate the data-analysis problem sets into the text; in the meanwhile, they can be found at the page for the class, <http://www.stat.cmu.edu/~cshalizi/uADA/>.

Concepts You Should Know

If more than a handful of these are unfamiliar, it is very unlikely that you are ready for this course.

Random variable; population, sample. Cumulative distribution function, probability mass function, probability density function. Specific distributions: Bernoulli, binomial, Poisson, geometric, Gaussian, exponential, t , Gamma. Expectation value. Variance, standard deviation. Sample mean, sample variance. Median, mode. Quartile, percentile, quantile. Inter-quartile range. Histograms.

Joint distribution functions. Conditional distributions; conditional expectations and variances. Statistical independence and dependence. Covariance and correlation; why dependence is not the same thing as correlation. Rules for arithmetic with expectations, variances and covariances. Laws of total probability, total expectation, total variation. Contingency tables; odds ratio, log odds ratio.

Sequences of random variables. Stochastic process. Law of large numbers. Central limit theorem.

Parameters; estimator functions and point estimates. Sampling distribution. Bias of an estimator. Standard error of an estimate; standard error of the mean; how and why the standard error of the mean differs from the standard deviation. Confidence intervals and interval estimates.

Hypothesis tests. Tests for differences in means and in proportions; Z and t tests; degrees of freedom. Size, significance, power. Relation between hypothesis tests and confidence intervals. χ^2 test of independence for contingency tables; degrees of freedom. KS test for goodness-of-fit to distributions.

Linear regression. Meaning of the linear regression function. Fitted values and residuals of a regression. Interpretation of regression coefficients. Least-squares estimate of coefficients. Matrix formula for estimating the coefficients; the hat matrix.

R^2 ; why adding more predictor variables never reduces R^2 . The t -test for the significance of individual coefficients given other coefficients. The F -test and partial F -test for the significance of regression models. Degrees of freedom for residuals. Examination of residuals. Confidence intervals for parameters. Confidence intervals for fitted values. Prediction intervals.

Likelihood. Likelihood functions. Maximum likelihood estimates. Relation between maximum likelihood, least squares, and Gaussian distributions. Relation between confidence intervals and the likelihood function. Likelihood ratio test.

14:41 Thursday 25th April, 2013

Part I

Regression and Its Generalizations

Chapter 1

Regression: Predicting and Relating Quantitative Features

1.1 Statistics, Data Analysis, Regression

Statistics is the branch of mathematical engineering which designs and analyses methods for drawing reliable inferences from imperfect data.

The subject of most sciences is some aspect of the world around us, or within us. Psychology studies minds; geology studies the Earth's composition and form; economics studies production, distribution and exchange; mycology studies mushrooms. Statistics does not study the world, but some of the ways we try to understand the world — some of the intellectual tools of the other sciences. Its utility comes indirectly, through helping those other sciences.

This utility is very great, because all the sciences have to deal with imperfect data. Data may be imperfect because we can only observe and record a small fraction of what is relevant; or because we can only observe indirect signs of what is truly relevant; or because, no matter how carefully we try, our data always contain an element of noise. Over the last two centuries, statistics has come to handle all such imperfections by modeling them as random processes, and probability has become so central to statistics that we introduce random events deliberately (as in sample surveys).¹

Statistics, then, uses probability to model inference from data. We try to mathematically understand the properties of different procedures for drawing inferences: Under what conditions are they reliable? What sorts of errors do they make, and how often? What can they tell us when they work? What are signs that something has gone wrong? Like other branches of engineering, statistics aims not just at understanding but also at improvement: we want to analyze data *better*, more reliably, with fewer and smaller errors, under broader conditions, faster, and with less mental

¹Two excellent, but very different, histories of how statistics came to this understanding are Hacking (1990) and Porter (1986).

effort. Sometimes some of these goals conflict — a fast, simple method might be very error-prone, or only reliable under a narrow range of circumstances.

One of the things that people most often want to know about the world is how different variables are related to each other, and one of the central tools statistics has for learning about relationships is regression.² In your linear regression class, you learned about how it could be used in data analysis, and learned about its properties. In this class, we will build on that foundation, extending beyond basic linear regression in many directions, to answer many questions about how variables are related to each other.

This is intimately related to prediction. Being able to make predictions isn't the *only* reason we want to understand relations between variables, but prediction *tests* our knowledge of relations. (If we misunderstand, we might still be able to predict, but it's hard to see how we could understand and *not* be able to predict.) So before we go beyond linear regression, we will first look at prediction, and how to predict one variable from nothing at all. Then we will look at predictive relationships between variables, and see how linear regression is just one member of a big family of smoothing methods, all of which are available to us.

1.2 Guessing the Value of a Random Variable

We have a quantitative, numerical variable, which we'll imaginatively call Y . We'll suppose that it's a random variable, and try to predict it by guessing a single value for it. (Other kinds of predictions are possible — we might guess whether Y will fall within certain limits, or the probability that it does so, or even the whole probability distribution of Y . But some lessons we'll learn here will apply to these other kinds of predictions as well.) What is the best value to guess? More formally, what is the **optimal point forecast** for Y ?

To answer this question, we need to pick a function to be optimized, which should measure how good our guesses are — or equivalently how bad they are, how big an error we're making. A reasonable start point is the **mean squared error**:

$$\text{MSE}(a) \equiv \mathbf{E} [(Y - a)^2] \quad (1.1)$$

²The origin of the name is instructive. It comes from 19th century investigations into the relationship between the attributes of parents and their children. People who are taller (heavier, faster, ...) than average tend to have children who are also taller than average, but not *quite* as tall. Likewise, the children of unusually short parents also tend to be closer to the average, and similarly for other traits. This came to be called “regression towards the mean”, or even “regression towards mediocrity”; hence the line relating the average height (or whatever) of children to that of their parents was “the regression line”, and the word stuck.

So we'd like to find the value r where $\text{MSE}(a)$ is smallest.

$$\text{MSE}(a) = \mathbb{E}[(Y - a)^2] \quad (1.2)$$

$$= (\mathbb{E}[Y - a])^2 + \text{Var}[Y - a] \quad (1.3)$$

$$= (\mathbb{E}[Y - a])^2 + \text{Var}[Y] \quad (1.4)$$

$$= (\mathbb{E}[Y] - a)^2 + \text{Var}[Y] \quad (1.5)$$

$$\frac{d\text{MSE}}{da} = -2(\mathbb{E}[Y] - a) + 0 \quad (1.6)$$

$$2(\mathbb{E}[Y] - r) = 0 \quad (1.7)$$

$$r = \mathbb{E}[Y] \quad (1.8)$$

So, if we gauge the quality of our prediction by mean-squared error, the best prediction to make is the expected value.

1.2.1 Estimating the Expected Value

Of course, to make the prediction $\mathbb{E}[Y]$ we would have to know the expected value of Y . Typically, we do not. However, if we have sampled values, y_1, y_2, \dots, y_n , we can estimate the expectation from the sample mean:

$$\hat{r} \equiv \frac{1}{n} \sum_{i=1}^n y_i \quad (1.9)$$

If the samples are independent and identically distributed (IID), then the law of large numbers tells us that

$$\hat{r} \rightarrow \mathbb{E}[Y] = r \quad (1.10)$$

and the central limit theorem tells us something about how fast the convergence is (namely the squared error will typically be about $\text{Var}[Y]/n$).

Of course the assumption that the y_i come from IID samples is a strong one, but we can assert pretty much the same thing if they're just uncorrelated with a common expected value. Even if they are correlated, but the correlations decay fast enough, all that changes is the rate of convergence. So "sit, wait, and average" is a pretty reliable way of estimating the expectation value.

1.3 The Regression Function

Of course, it's not very useful to predict *just one number* for a variable. Typically, we have lots of variables in our data, and we believe they are related *somewhat*. For example, suppose that we have data on two variables, X and Y , which might look like Figure 1.1. The feature Y is what we are trying to predict, a.k.a. the **dependent variable** or **output** or **response**, and X is the **predictor** or **independent variable** or **covariate** or **input**. Y might be something like the profitability of a customer and X their credit rating, or, if you want a less mercenary example, Y could be some measure of improvement in blood cholesterol and X the dose taken of a drug.

Typically we won't have just one input feature X but rather many of them, but that gets harder to draw and doesn't change the points of principle.

Figure 1.2 shows the same data as Figure 1.1, only with the sample mean added on. This clearly tells us something about the data, but also it seems like we should be able to do better — to reduce the average error — by using X , rather than by ignoring it.

Let's say that we want our prediction to be a *function* of X , namely $f(X)$. What should that function be, if we still use mean squared error? We can work this out by using the law of total expectation, i.e., the fact that $\mathbb{E}[U] = \mathbb{E}[\mathbb{E}[U|V]]$ for any random variables U and V .

$$\text{MSE}(f(X)) = \mathbb{E}[(Y - f(X))^2] \quad (1.11)$$

$$= \mathbb{E}[\mathbb{E}[(Y - f(X))^2|X]] \quad (1.12)$$

$$= \mathbb{E}[\text{Var}[Y|X] + (\mathbb{E}[Y - f(X)|X])^2] \quad (1.13)$$

When we want to minimize this, the first term inside the expectation doesn't depend on our prediction, and the second term looks just like our previous optimization only with all expectations conditional on X , so for our optimal function $r(x)$ we get

$$r(x) = \mathbb{E}[Y|X = x] \quad (1.14)$$

In other words, the (mean-squared) optimal *conditional* prediction is just the conditional expected value. The function $r(x)$ is called the **regression function**. This is what we would like to know when we want to predict Y .

1.3.1 Some Disclaimers

It's important to be clear on what is and is not being assumed here. Talking about X as the “independent variable” and Y as the “dependent” one suggests a causal model, which we might write

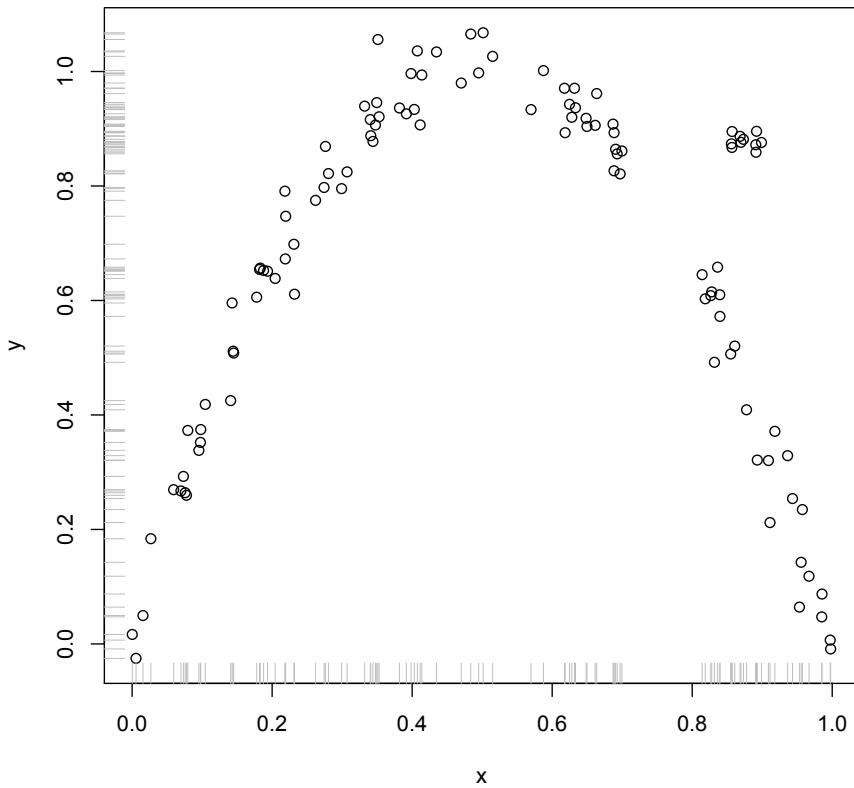
$$Y \leftarrow r(X) + \epsilon \quad (1.15)$$

where the direction of the arrow, \leftarrow , indicates the flow from causes to effects, and ϵ is some noise variable. If the gods of inference are very, very kind, then ϵ would have a fixed distribution, independent of X , and we could without loss of generality take it to have mean zero. (“Without loss of generality” because if it has a non-zero mean, we can incorporate that into $r(X)$ as an additive constant.) However, *no* such assumption is required to get Eq. 1.14. It works when predicting effects from causes, or the other way around when predicting (or “retrodicting”) causes from effects, or indeed when there is no causal relationship whatsoever between X and Y ³. It is always true that

$$Y|X = r(X) + \eta(X) \quad (1.16)$$

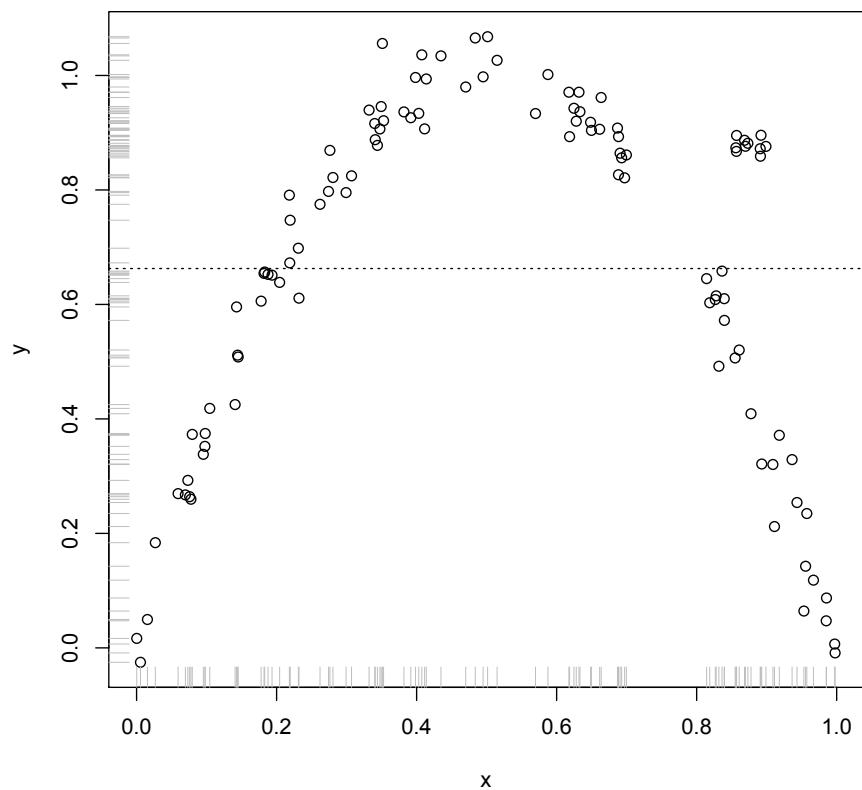
where $\eta(X)$ is a noise variable with mean zero, but as the notation indicates the distribution of the noise generally depends on X .

³We will cover causal inference in considerable detail in Part III.



```
plot(all.x,all.y,xlab="x",ylab="y")
rug(all.x,side=1,col="grey")
rug(all.y,side=2,col="grey")
```

Figure 1.1: Scatterplot of the example data. (These are made up.) The `rug` commands add horizontal and vertical ticks to the axes to mark the location of the data (in grey so they're less strong than the main tick-marks). This isn't necessary but is often helpful. The data are in the `example.dat` file.



```
abline(h=mean(all.y),lty=3)
```

Figure 1.2: Data from Figure 1.1, with a horizontal line showing the sample mean of Y .

It's also important to be clear that when we find the regression function is a constant, $r(x) = r_0$ for all x , that this does not mean that X and Y are statistically independent. If they are independent, then the regression function is a constant, but turning this around is the logical fallacy of "affirming the consequent"⁴.

1.4 Estimating the Regression Function

We want to find the regression function $r(x) = \mathbb{E}[Y|X=x]$, and what we've got is a big set of training examples, of pairs $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. How should we proceed?

If X takes on only a finite set of values, then a simple strategy is to use the conditional sample means:

$$\hat{r}(x) = \frac{1}{\#\{i : x_i = x\}} \sum_{i:x_i=x} y_i \quad (1.17)$$

By the same kind of law-of-large-numbers reasoning as before, we can be confident that $\hat{r}(x) \rightarrow \mathbb{E}[Y|X=x]$.

Unfortunately, this *only* works if X has only a finite set of values. If X is continuous, then in general the probability of our getting a sample at any *particular* value is zero, is the probability of getting *multiple* samples at *exactly* the same value of x . This is a basic issue with estimating any kind of function from data — the function will always be **undersampled**, and we need to fill in between the values we see. We also need to somehow take into account the fact that each y_i is a *sample* from the conditional distribution of $Y|X=x_i$, and so is not generally equal to $\mathbb{E}[Y|X=x_i]$. So any kind of function estimation is going to involve interpolation, extrapolation, and smoothing.

Different methods of estimating the regression function — different regression methods, for short — involve different choices about how we interpolate, extrapolate and smooth. This involves our making a choice about how to approximate $r(x)$ by a limited class of functions which we know (or at least hope) we can estimate. There is no guarantee that our choice leads to a *good* approximation in the case at hand, though it is sometimes possible to say that the approximation error will shrink as we get more and more data. This is an extremely important topic and deserves an extended discussion, coming next.

1.4.1 The Bias-Variance Tradeoff

Suppose that the true regression function is $r(x)$, but we use the function \hat{r} to make our predictions. Let's look at the mean squared error at $X = x$ in a slightly different way than before, which will make it clearer what happens when we can't use r to

⁴As in combining the fact that all human beings are featherless bipeds, and the observation that a cooked turkey is a featherless biped, to conclude that cooked turkeys are human beings. An econometrician stops there; an econometrician who wants to be famous writes a best-selling book about how this proves that Thanksgiving is really about cannibalism.

make predictions. We'll begin by expanding $(Y - \hat{r}(x))^2$, since the MSE at x is just the expectation of this.

$$(Y - \hat{r}(x))^2 \quad (1.18)$$

$$\begin{aligned} &= (Y - r(x) + r(x) - \hat{r}(x))^2 \\ &= (Y - r(x))^2 + 2(Y - r(x))(r(x) - \hat{r}(x)) + (r(x) - \hat{r}(x))^2 \end{aligned} \quad (1.19)$$

We saw above (Eq. 1.16) that $Y - r(x) = \eta$, a random variable which has expectation zero (and is uncorrelated with x). When we take the expectation of Eq. 1.19, nothing happens to the last term (since it doesn't involve any random quantities); the middle term goes to zero (because $E[Y - r(x)] = E[\eta] = 0$), and the first term becomes the variance of η . This depends on x , in general, so let's call it σ_x^2 . We have

$$\text{MSE}(\hat{r}(x)) = \sigma_x^2 + ((r(x) - \hat{r}(x))^2) \quad (1.20)$$

The σ_x^2 term doesn't depend on our prediction function, just on how hard it is, intrinsically, to predict Y at $X = x$. The second term, though, is the extra error we get from not knowing r . (Unsurprisingly, ignorance of r cannot *improve* our predictions.) This is our first **bias-variance decomposition**: the total MSE at x is decomposed into a (squared) bias $r(x) - \hat{r}(x)$, the amount by which our predictions are *systematically* off, and a variance σ_x^2 , the unpredictable, "statistical" fluctuation around even the best prediction.

All of the above assumes that \hat{r} is a single fixed function. In practice, of course, \hat{r} is something we estimate from earlier data. But if those data are random, the exact regression function we get is random too; let's call this random function \widehat{R}_n , where the subscript reminds us of the finite amount of data we used to estimate it. What we have analyzed is really $\text{MSE}(\widehat{R}_n(x)|\widehat{R}_n = \hat{r})$, the mean squared error *conditional on* a particular estimated regression function. What can we say about the prediction error of the *method*, averaging over all the possible training data sets?

$$\text{MSE}(\widehat{R}_n(x)) = E[(Y - \widehat{R}_n(X))^2|X = x] \quad (1.21)$$

$$= E[E[(Y - \widehat{R}_n(X))^2|X = x, \widehat{R}_n = \hat{r}]|X = x] \quad (1.22)$$

$$= E[\sigma_x^2 + (r(x) - \widehat{R}_n(x))^2|X = x] \quad (1.23)$$

$$= \sigma_x^2 + E[(r(x) - \widehat{R}_n(x))^2|X = x] \quad (1.24)$$

$$= \sigma_x^2 + E[(r(x) - E[\widehat{R}_n(x)]) + E[\widehat{R}_n(x)] - \widehat{R}_n(x))^2] \quad (1.25)$$

$$= \sigma_x^2 + (r(x) - E[\widehat{R}_n(x)])^2 + \text{Var}[\widehat{R}_n(x)] \quad (1.26)$$

This is our second bias-variance decomposition — I pulled the same trick as before, adding and subtract a mean inside the square. The first term is just the variance of the process; we've seen that before and isn't, for the moment, of any concern. The second term is the bias in using \widehat{R}_n to estimate r — the **approximation bias** or

approximation error. The third term, though, is the variance in our *estimate* of the regression function. Even if we have an unbiased *method* ($r(x) = E[\widehat{R}_n(x)]$), if there is a lot of variance in our estimates, we can expect to make large errors.

The approximation bias has to depend on the true regression function. For example, if $E[\widehat{R}_n(x)] = 42 + 37x$, the error of approximation will be zero if $r(x) = 42 + 37x$, but it will be larger and x -dependent if $r(x) = 0$. However, there are flexible methods of estimation which will have small approximation biases for *all* r in a broad range of regression functions. The catch is that, at least past a certain point, decreasing the approximation bias can only come through increasing the estimation variance. This is the **bias-variance trade-off**. However, nothing says that the trade-off has to be one-for-one. Sometimes we can lower the total error by *introducing* some bias, since it gets rid of more variance than it adds approximation error. The next section gives an example.

In general, both the approximation bias and the estimation variance depend on n . A method is **consistent**⁵ when both of these go to zero as $n \rightarrow \infty$ — that is, if we recover the true regression function as we get more and more data.⁶ Again, consistency depends on how well the method matches the actual data-generating process, not just on the method, and again, there is a bias-variance trade-off. There can be multiple consistent methods for the same problem, and their biases and variances don't have to go to zero at the same *rates*.

1.4.2 The Bias-Variance Trade-Off in Action

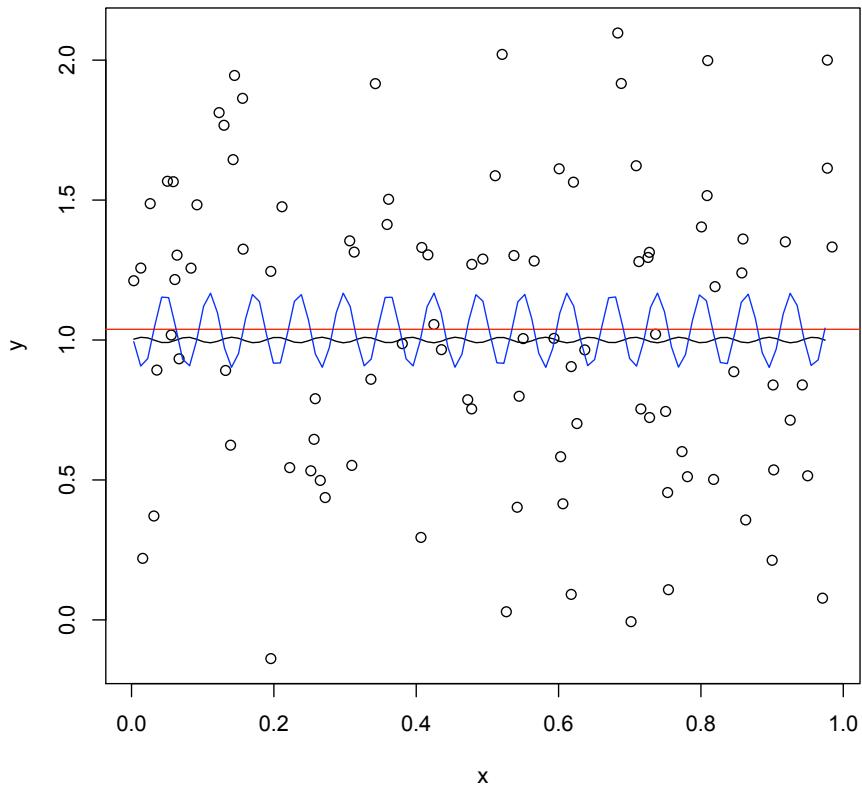
Let's take an extreme example: we could decide to approximate $r(x)$ by a constant r_0 . The implicit smoothing here is very strong, but sometimes appropriate. For instance, it's appropriate when $r(x)$ really is a constant! Then trying to estimate any additional structure in the regression function is just so much wasted effort. Alternately, if $r(x)$ is *nearly* constant, we may still be better off approximating it as one. For instance, suppose the true $r(x) = r_0 + a \sin(\nu x)$, where $a \ll 1$ and $\nu \gg 1$ (Figure 1.3 shows an example). With limited data, we can actually get better predictions by estimating a constant regression function than one with the correct functional form.

1.4.3 Ordinary Least Squares Linear Regression as Smoothing

Let's revisit ordinary least-squares linear regression from this point of view. Let's assume that the independent variable X is one-dimensional, and that both X and Y

⁵To be precise, consistent for r , or consistent for conditional expectations. More generally, an estimator of any property of the data, or of the whole distribution, is consistent if it converges on the truth.

⁶You might worry about this claim, especially if you've taken more probability theory — aren't we just saying something about average performance of the \widehat{R} , rather than any *particular* estimated regression function? But notice that if the estimation variance goes to zero, then by Chebyshev's inequality, $\Pr(|X - E[X]| \geq \eta) \leq \text{Var}[X]/\eta^2$, each $\widehat{R}_n(x)$ comes arbitrarily close to $E[\widehat{R}_n(x)]$ with arbitrarily high probability. If the approximation bias goes to zero, therefore, the estimated regression functions converge *in probability* on the true regression function, not just *in mean*.



```

ugly.func = function(x) {1 + 0.01*sin(100*x)}
r = runif(100); y = ugly.func(r) + rnorm(length(r),0,0.5)
plot(r,y,xlab="x",ylab="y"); curve(ugly.func,add=TRUE)
abline(h=mean(y),col="red")
sine.fit = lm(y ~ 1+ sin(100*x))
curve(sine.fit$coefficients[1]+sine.fit$coefficients[2]*sin(100*x),
      col="blue",add=TRUE)

```

Figure 1.3: A rapidly-varying but nearly-constant regression function; $Y = 1 + 0.01 \sin 100x + \epsilon$, with $\epsilon \sim \mathcal{N}(0, 0.1)$. (The x values are uniformly distributed between 0 and 1.) Red: constant line at the sample mean. Blue: estimated function of the same form as the true regression function, i.e., $r_0 + \alpha \sin 100x$. If the data set is small enough, the constant actually generalizes better — the bias of using the wrong functional form is smaller than the additional variance from the extra degrees of freedom. Here, the root-mean-square (RMS) error of the constant on new data is 0.50, while that of the estimated sine function is 0.51 — using the right function actually hurts us!

are centered (i.e. have mean zero) — neither of these assumptions is really necessary, but they reduce the book-keeping.

We choose to approximate $r(x)$ by $\alpha + \beta x$, and ask for the best values a, b of those constants. These will be the ones which minimize the mean-squared error.

$$MSE(\alpha, \beta) = E[(Y - \alpha - \beta X)^2] \quad (1.27)$$

$$= E[(Y - \alpha - \beta X)^2 | X] \quad (1.28)$$

$$= E[\text{Var}[Y|X] + (E[Y - \alpha - \beta X|X])^2] \quad (1.29)$$

$$= E[\text{Var}[Y|X]] + E[(E[Y - \alpha - \beta X|X])^2] \quad (1.30)$$

The first term doesn't depend on α or β , so we can drop it for purposes of optimization. Taking derivatives, and then bringing them inside the expectations,

$$\frac{\partial MSE}{\partial \alpha} = E[2(Y - \alpha - \beta X)(-1)] \quad (1.31)$$

$$E[Y - \alpha - \beta X] = 0 \quad (1.32)$$

$$\alpha = E[Y] - bE[X] = 0 \quad (1.33)$$

using the fact that X and Y are centered; and,

$$\frac{\partial MSE}{\partial \beta} = E[2(Y - \alpha - \beta X)(-X)] \quad (1.34)$$

$$E[XY] - bE[X^2] = 0 \quad (1.35)$$

$$b = \frac{\text{Cov}[X, Y]}{\text{Var}[X]} \quad (1.36)$$

again using the centering of X and Y . That is, the mean-squared optimal linear prediction is

$$r(x) = x \frac{\text{Cov}[X, Y]}{\text{Var}[X]} \quad (1.37)$$

Now, if we try to estimate this from data, there are (at least) two approaches. One is to replace the true population values of the covariance and the variance with their sample values, respectively

$$\frac{1}{n} \sum_i y_i x_i \quad (1.38)$$

and

$$\frac{1}{n} \sum_i x_i^2 \quad (1.39)$$

(again, assuming centering). The other is to minimize the residual sum of squares,

$$RSS(\alpha, \beta) \equiv \sum_i (y_i - \alpha - \beta x_i)^2 \quad (1.40)$$

You may or may not find it surprising that both approaches lead to the same answer:

$$\hat{a} = 0 \tag{1.41}$$

$$\hat{b} = \frac{\sum_i y_i x_i}{\sum_i x_i^2} \tag{1.42}$$

Provided that $\text{Var}[X] > 0$, this will converge with IID samples, so we have a consistent estimator.⁷

We are now in a position to see how the least-squares linear regression model is really a smoothing of the data. Let's write the estimated regression function explicitly in terms of the training data points.

$$\hat{r}(x) = \hat{b}x \tag{1.43}$$

$$= x \frac{\sum_i y_i x_i}{\sum_i x_i^2} \tag{1.44}$$

$$= \sum_i y_i \frac{x_i}{\sum_j x_j^2} x \tag{1.45}$$

$$= \sum_i y_i \frac{x_i}{n s_X^2} x \tag{1.46}$$

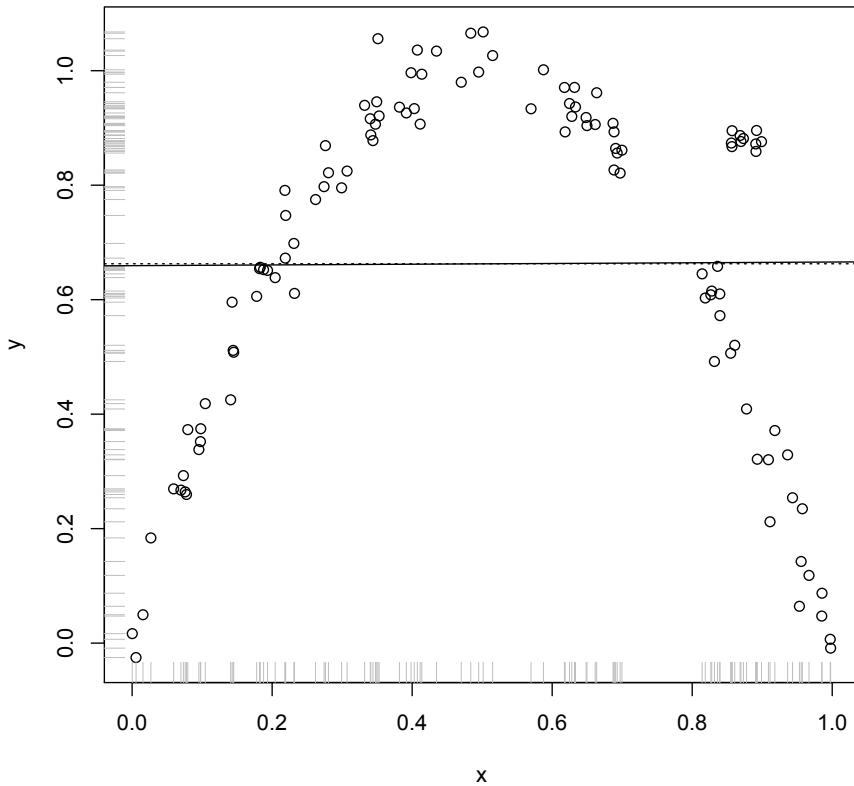
where s_X^2 is the sample variance of X . In words, our prediction is a weighted average of the observed values y_i of the dependent variable, where the weights are proportional to how far x_i is from the center (relative to the variance), and proportional to the magnitude of x . If x_i is on the same side of the center as x , it gets a positive weight, and if it's on the opposite side it gets a negative weight.

Figure 1.4 shows the data from Figure 1.1 with the least-squares regression line added. It will not escape your notice that this is very, very slightly different from the constant regression function; the coefficient on X is 6.3×10^{-3} . Visually, the problem is that there should be a positive slope in the left-hand half of the data, and a negative slope in the right, but the slopes are the densities are balanced so that the best *single* slope is zero.⁸

Mathematically, the problem arises from the peculiar way in which least-squares linear regression smoothes the data. As I said, the weight of a data point depends on how far it is from the *center* of the data, not how far it is from the *point at which we are trying to predict*. This works when $r(x)$ really is a straight line, but otherwise — e.g., here — it's a recipe for trouble. However, it does suggest that if we could somehow just tweak the way we smooth the data, we could do better than linear regression.

⁷Eq. 1.41 may look funny, but remember that we're assuming X and Y have been centered. Centering doesn't change the slope of the least-squares line but does change the intercept; if we go back to the uncentered variables the intercept becomes $\bar{Y} - \hat{b}\bar{X}$, where the bar denotes the sample mean.

⁸The standard test of whether this coefficient is zero is about as far from rejecting the null hypothesis as you will ever see, $p = 0.95$. Remember this the next time you look at regression output.



```
fit.all = lm(all.y~all.x)
abline(fit.all)
```

Figure 1.4: Data from Figure 1.1, with a horizontal line at the mean (dotted) and the ordinary least squares regression line (solid). If you zoom in online you will see that there really are two lines there. (The `abline` adds a line to the current plot with intercept `a` and slope `b`; it's set up to take the appropriate coefficients from the output of `lm`.)

1.5 Linear Smoothers

The sample mean and the linear regression line are both special cases of **linear smoothers**, which are estimates of the regression function with the following form:

$$\hat{r}(x) = \sum_i y_i \hat{w}(x_i, x) \quad (1.47)$$

The sample mean is the special case where $\hat{w}(x_i, x) = 1/n$, regardless of what x_i and x are.

Ordinary linear regression is the special case where $\hat{w}(x_i, x) = (x_i / ns_X^2)x$.

Both of these, as remarked, ignore how far x_i is from x .

1.5.1 k -Nearest-Neighbor Regression

At the other extreme, we could do **nearest-neighbor regression**:

$$\hat{w}(x_i, x) = \begin{cases} 1 & x_i \text{ nearest neighbor of } x \\ 0 & \text{otherwise} \end{cases} \quad (1.48)$$

This is very sensitive to the distance between x_i and x . If $r(x)$ does not change too rapidly, and X is pretty thoroughly sampled, then the nearest neighbor of x among the x_i is probably close to x , so that $r(x_i)$ is probably close to $r(x)$. However, $y_i = r(x_i) + \text{noise}$, so nearest-neighbor regression will include the noise into its prediction. We might instead do k -nearest neighbor regression,

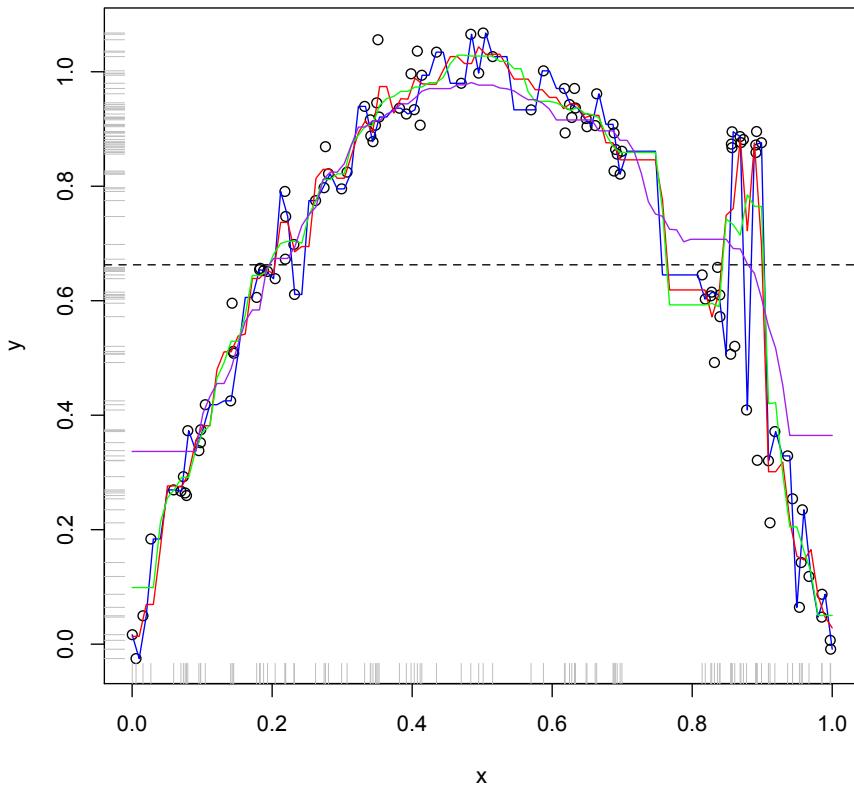
$$\hat{w}(x_i, x) = \begin{cases} 1/k & x_i \text{ one of the } k \text{ nearest neighbors of } x \\ 0 & \text{otherwise} \end{cases} \quad (1.49)$$

Again, with enough samples all the k nearest neighbors of x are probably close to x , so their regression functions there are going to be close to the regression function at x . But because we average their values of y_i , the noise terms should tend to cancel each other out. As we increase k , we get smoother functions — in the limit $k = n$ and we just get back the constant. Figure 1.5 illustrates this for our running example data.⁹

To use k -nearest-neighbors regression, we need to pick k somehow. This means we need to decide *how much* smoothing to do, and this is not trivial. We will return to this point.

Because k -nearest-neighbors averages over only a fixed number of neighbors, each of which is a noisy sample, it always has some noise in its prediction, and is generally not consistent. This may not matter very much with moderately-large data (especially once we have a good way of picking k). However, it is sometimes useful to let k systematically grow with n , but not too fast, so as to avoid just doing a global average; say $k \propto \sqrt{n}$. Such schemes *can* be consistent.

⁹The code uses the k -nearest neighbor function provided by the package `knnflex` (available from CRAN). This requires one to pre-compute a matrix of the distances between all the points of interest, i.e., training data and testing data (using `knn.dist`); the `knn.predict` function then needs to be told which rows of that matrix come from training data and which from testing data. See `help(knnflex.predict)` for more, including examples.



```

library(knnflex)
all.dist = knn.dist(c(all.x,seq(from=0,to=1,length.out=100)))
all.nn1.predict = knn.predict(1:110,111:210,all.y,all.dist,k=1)
abline(h=mean(all.y),lty=2)
lines(seq(from=0,to=1,length.out=100),all.nn1.predict,col="blue")
all.nn3.predict = knn.predict(1:110,111:210,all.y,all.dist,k=3)
lines(seq(from=0,to=1,length.out=100),all.nn3.predict,col="red")
all.nn5.predict = knn.predict(1:110,111:210,all.y,all.dist,k=5)
lines(seq(from=0,to=1,length.out=100),all.nn5.predict,col="green")
all.nn20.predict = knn.predict(1:110,111:210,all.y,all.dist,k=20)
lines(seq(from=0,to=1,length.out=100),all.nn20.predict,col="purple")

```

Figure 1.5: Data points from Figure 1.1 with horizontal dashed line at the mean and the k -nearest-neighbor regression curves for $k = 1$ (blue), $k = 3$ (red), $k = 5$ (green) and $k = 20$ (purple). Note how increasing k smoothes out the regression line, and pulls it back towards the mean. ($k = 100$ would give us back the dashed horizontal line.)

1.5.2 Kernel Smoothers

Changing k in a k -nearest-neighbors regression lets us change how much smoothing we're doing on our data, but it's a bit awkward to express this in terms of a number of data points. It feels like it would be more natural to talk about a range in the independent variable over which we smooth or average. Another problem with k -NN regression is that each testing point is predicted using information from only a few of the training data points, unlike linear regression or the sample mean, which always uses all the training data. If we could somehow use all the training data, but in a location-sensitive way, that would be nice.

There are several ways to do this, as we'll see, but a particularly useful one is to use a **kernel smoother**, a.k.a. **kernel regression** or **Nadaraya-Watson regression**. To begin with, we need to pick a **kernel function**¹⁰ $K(x_i, x)$ which satisfies the following properties:

1. $K(x_i, x) \geq 0$
2. $K(x_i, x)$ depends only on the distance $x_i - x$, not the individual arguments
3. $\int x K(0, x) dx = 0$
4. $0 < \int x^2 K(0, x) dx < \infty$

These conditions together (especially the last one) imply that $K(x_i, x) \rightarrow 0$ as $|x_i - x| \rightarrow \infty$. Two examples of such functions are the density of the $\text{Unif}(-h/2, h/2)$ distribution, and the density of the standard Gaussian $\mathcal{N}(0, \sqrt{h})$ distribution. Here h can be any positive number, and is called the **bandwidth**.

The Nadaraya-Watson estimate of the regression function is

$$\hat{r}(x) = \sum_i y_i \frac{K(x_i, x)}{\sum_j K(x_j, x)} \quad (1.50)$$

i.e., in terms of Eq. 1.47,

$$\hat{w}(x_i, x) = \frac{K(x_i, x)}{\sum_j K(x_j, x)} \quad (1.51)$$

(Notice that here, as in k -NN regression, the sum of the weights is always 1. Why?)¹¹

What does this achieve? Well, $K(x_i, x)$ is large if x_i is close to x , so this will place a lot of weight on the training data points close to the point where we are trying to predict. More distant training points will have smaller weights, falling off towards zero. If we try to predict at a point x which is very far from any of the training data points, the value of $K(x_i, x)$ will be small for all x_i , but it will typically be *much*,

¹⁰There are many other mathematical objects which are *also* called "kernels". Some of these meanings are related, but not all of them. (Cf. "normal".)

¹¹What do we do if $K(x_i, x)$ is zero for some x_i ? Nothing; they just get zero weight in the average. What do we do if *all* the $K(x_i, x)$ are zero? Different people adopt different conventions; popular ones are to return the global, unweighted mean of the y_i , to do some sort of interpolation from regions where the weights are defined, and to throw up our hands and refuse to make any predictions (computationally, return NA).

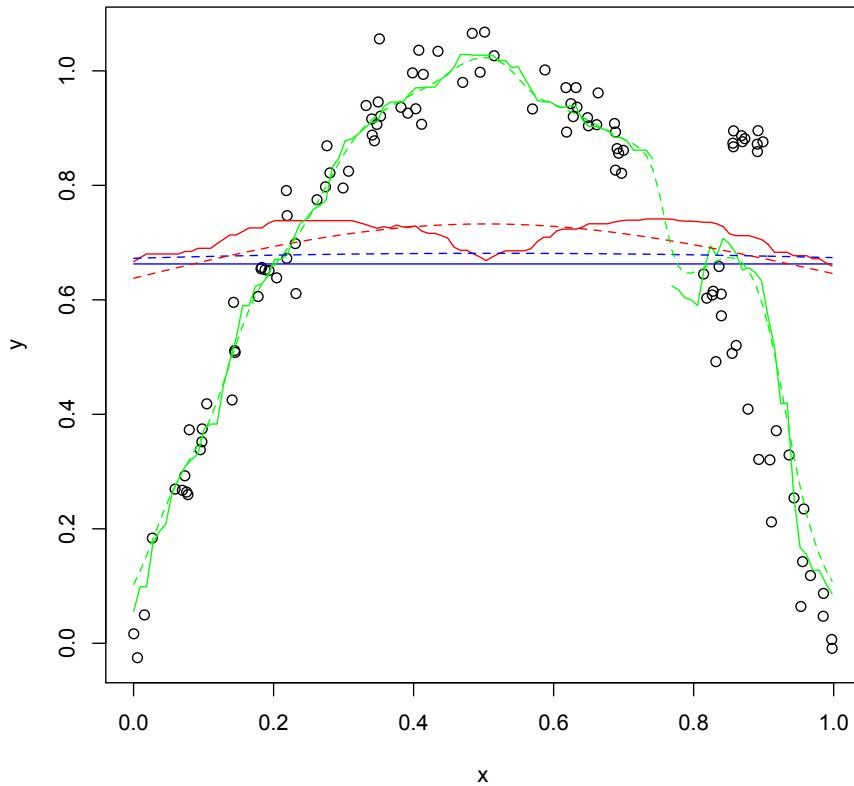
much smaller for all the x_i which are not the nearest neighbor of x , so $\hat{w}(x_i, x) \approx 1$ for the nearest neighbor and ≈ 0 for all the others.¹² That is, far from the training data, our predictions will tend towards nearest neighbors, rather than going off to $\pm\infty$, as linear regression's predictions do. Whether this is good or bad of course depends on the true $r(x)$ — and how often we have to predict what will happen very far from the training data.

Figure 1.6 shows our running example data, together with kernel regression estimates formed by combining the uniform-density, or **box**, and Gaussian kernels with different bandwidths. The box kernel simply takes a region of width h around the point x and averages the training data points it finds there. The Gaussian kernel gives reasonably large weights to points within h of x , smaller ones to points within $2h$, tiny ones to points within $3h$, and so on, shrinking like $e^{-(x-x_i)^2/2h^2}$. As promised, the bandwidth h controls the degree of smoothing. As $h \rightarrow \infty$, we revert to taking the global mean. As $h \rightarrow 0$, we tend to get spikier functions — with the Gaussian kernel at least it tends towards the nearest-neighbor regression.

If we want to use kernel regression, we need to choose both which kernel to use, and the bandwidth to use with it. Experience, like Figure 1.6, suggests that the bandwidth usually matters a lot more than the kernel. This puts us back to roughly where we were with k -NN regression, needing to control the degree of smoothing, without knowing how smooth $r(x)$ really is. Similarly again, with a fixed bandwidth h , kernel regression is generally not consistent. However, if $h \rightarrow 0$ as $n \rightarrow \infty$, but doesn't shrink *too* fast, then we can get consistency.

In Chapter 2, we'll look more at the limits of linear regression and some extensions; Chapter 3 will cover some key aspects of evaluating statistical models, including regression models; and then Chapter 4 will come back to kernel regression.

¹²Take a Gaussian kernel in one dimension, for instance, so $K(x_i, x) \propto e^{-(x_i-x)^2/2h^2}$. Say x_i is the nearest neighbor, and $|x_i - x| = L$, with $L \gg h$. So $K(x_i, x) \propto e^{-L^2/2h^2}$, a small number. But now for any other x_j , $K(x_i, x) \propto e^{-L^2/2h^2} e^{-(x_j-x_i)L/2h^2} e^{-(x_j-x_i)^2/2h^2} \ll e^{-L^2/2h^2}$. — This assumes that we're using a kernel like the Gaussian, which never quite goes to zero, unlike the box kernel.



```

plot(all.x,all.y,xlab="x",ylab="y")
lines(ksmooth(all.x, all.y, "box", bandwidth=2),col="blue")
lines(ksmooth(all.x, all.y, "box", bandwidth=1),col="red")
lines(ksmooth(all.x, all.y, "box", bandwidth=0.1),col="green")
lines(ksmooth(all.x, all.y, "normal", bandwidth=2),col="blue",lty=2)
lines(ksmooth(all.x, all.y, "normal", bandwidth=1),col="red",lty=2)
lines(ksmooth(all.x, all.y, "normal", bandwidth=0.1),col="green",lty=2)

```

Figure 1.6: Data from Figure 1.1 together with kernel regression lines. Solid colored lines are box-kernel estimates, dashed colored lines Gaussian-kernel estimates. Blue, $b = 2$; red, $b = 1$; green, $b = 0.1$ (per the definition of bandwidth in the `ksmooth` function). Note the abrupt jump around $x = 0.75$ in the box-kernel/ $b = 0.1$ (solid green) line — with a small bandwidth the box kernel is unable to interpolate smoothly across the break in the training data, while the Gaussian kernel can.

1.6 Exercises

Not to hand in.

1. Suppose we use the **mean absolute error** instead of the mean squared error:

$$\text{MAE}(\alpha) = \mathbf{E}[|Y - \alpha|] \quad (1.52)$$

Is this also minimized by taking $\alpha = \mathbf{E}[Y]$? If not, what value $\tilde{\alpha}$ minimizes the MAE? Should we use MSE or MAE to measure error?

2. Derive Eqs. 1.41 and 1.42 by minimizing Eq. 1.40.
3. What does it mean for Gaussian kernel regression to approach nearest-neighbor regression as $h \rightarrow 0$? Why does it do so? Is this true for all kinds of kernel regression?
4. COMPUTING The file `ckm.csv` on the class website¹³ contains data from a famous study on the “diffusion of innovations”, in this case, the adoption of tetracycline, a then-new antibiotic, among doctors in four cities in Illinois in the 1950s. In particular, the column `adoption_date` records how many months after the beginning of the study each doctor surveyed began prescribing tetracycline. Note that some doctors did not do so before the study ended (these have an adoption date of `Inf`, infinity), and this information is not available for others (`NA`).
 - (a) Load the data as a data-frame called `ckm`.
 - (b) What does


```
adopters <- sapply(1:17, function(y) { sum(na.omit(ckm$adoption_date) <= y) } )
```

 do? Why 17?
 - (c) Plot the number of doctors who have already adopted tetracycline at the start of each month against the number of new adopters that month. This should look somewhat like a sparser version of the scatter-plot used as a running example in this chapter. *Hints: diff, plot*.
 - (d) Linearly regress the number of new adopters against the number of adopters. Add the regression line to your scatterplot. It should suggest that increasing the number of doctors who have adopted the drug strictly decreases the number who will adopt.
 - (e) Add Gaussian kernel smoothing lines to your scatterplot, as in Figure 1.6. Do these suggest that the relationship is monotonic?
 - (f) Plot the residuals of the linear regression against the predictor variable. (*Hint: residuals*.) Do the residuals look independent of the predictor? What happens if you kernel smooth the residuals?

¹³Slightly modified from <http://moreno.ss.uci.edu/data.html> to fit R conventions.

Chapter 2

The Truth about Linear Regression

We need to say some more about how linear regression, and especially about how it really works and how it can fail. Linear regression is important because

1. it's a fairly straightforward technique which often works reasonably well for prediction;
2. it's a simple foundation for some more sophisticated techniques;
3. it's a standard method so people use it to communicate; and
4. it's a standard method so people have come to confuse it with prediction and even with causal inference as such.

We need to go over (1)–(3), and provide prophylaxis against (4).

A very good resource on regression is Berk (2004). It omits technical details, but is superb on the high-level picture, and especially on what must be assumed in order to do certain things with regression, and what cannot be done under any assumption.

2.1 Optimal Linear Prediction: Multiple Variables

We have a response variable Y and a p -dimensional vector of predictor variables or features \vec{X} . To simplify the book-keeping, we'll take these to be centered — we can always un-center them later. We would like to predict Y using \vec{X} . We saw last time that the best predictor we could use, at least in a mean-squared sense, is the conditional expectation,

$$r(\vec{x}) = \mathbb{E} [Y | \vec{X} = \vec{x}] \quad (2.1)$$

Instead of using the optimal predictor $r(\vec{x})$, let's try to predict as well as possible while using only a linear function of \vec{x} , say $\vec{x} \cdot \beta$. This is not an assumption about the

world, but rather a decision on our part; a choice, not a hypothesis. This decision can be good — $\vec{x} \cdot \beta$ could be a close approximation to $r(\vec{x})$ — even if the linear hypothesis is wrong.

One reason to think it's not a crazy decision is that we may hope r is a smooth function. If it is, then we can Taylor expand it about our favorite point, say \vec{u} :

$$r(\vec{x}) = r(\vec{u}) + \sum_{i=1}^p \left(\frac{\partial r}{\partial x_i} \Big|_{\vec{u}} \right) (x_i - u_i) + O(\|\vec{x} - \vec{u}\|^2) \quad (2.2)$$

or, in the more compact vector-calculus notation,

$$r(\vec{x}) = r(\vec{u}) + (\vec{x} - \vec{u}) \cdot \nabla r(\vec{u}) + O(\|\vec{x} - \vec{u}\|^2) \quad (2.3)$$

If we only look at points \vec{x} which are close to \vec{u} , then the remainder terms $O(\|\vec{x} - \vec{u}\|^2)$ are small, and a linear approximation is a good one¹.

Of course there are lots of linear functions so we need to pick one, and we may as well do that by minimizing mean-squared error again:

$$MSE(\beta) = \mathbf{E} \left[(Y - \vec{X} \cdot \beta)^2 \right] \quad (2.4)$$

Going through the optimization is parallel to the one-dimensional case (see last chapter), with the conclusion that the optimal β is

$$\beta = \mathbf{v}^{-1} \text{Cov} [\vec{X}, Y] \quad (2.5)$$

where \mathbf{v} is the covariance matrix of \vec{X} , i.e., $v_{ij} = \text{Cov}[X_i, X_j]$, and $\text{Cov}[\vec{X}, Y]$ is the vector of covariances between the predictor variables and Y , i.e. $\text{Cov}[\vec{X}, Y]_i = \text{Cov}[X_i, Y]$.

Multiple regression would be a lot simpler if we could just do a simple regression for each predictor variable, and add them up; but really, this is what multiple regression *does*, just in a disguised form. If the input variables are uncorrelated, \mathbf{v} is diagonal ($v_{ij} = 0$ unless $i = j$), and so is \mathbf{v}^{-1} . Then doing multiple regression breaks up into a sum of separate simple regressions across each input variable. When the input variables are correlated and \mathbf{v} is not diagonal, we can think of the multiplication by \mathbf{v}^{-1} as **de-correlating** \vec{X} — applying a linear transformation to come up with a new set of inputs which are uncorrelated with each other.²

Notice: β depends on the marginal distribution of \vec{X} (through the covariance matrix \mathbf{v}). If that shifts, the optimal coefficients β will shift, *unless* the real regression function is linear.

¹If you are not familiar with the big-O notation like $O(\|\vec{x} - \vec{u}\|^2)$, now would be a good time to read Appendix B.

²If \vec{Z} is a random vector with covariance matrix I , then $\mathbf{w}^T \vec{Z}$ is a random vector with covariance matrix $\mathbf{w}^T \mathbf{w}$. Conversely, if we start with a random vector \vec{X} with covariance matrix \mathbf{v} , the latter has a “square root” $\mathbf{v}^{1/2}$ (i.e., $\mathbf{v}^{1/2} \mathbf{v}^{1/2} = \mathbf{v}$), and $\mathbf{v}^{-1/2} \vec{X}$ will be a random vector with covariance matrix \mathbf{I} . When we write our predictions as $\vec{X} \mathbf{v}^{-1} \text{Cov}[\vec{X}, Y]$, we should think of this as $(\vec{X} \mathbf{v}^{-1/2}) (\mathbf{v}^{-1/2} \text{Cov}[\vec{X}, Y])$. We use one power of $\mathbf{v}^{-1/2}$ to transform the input features into uncorrelated variables before taking their correlations with the response, and the other power to decorrelate \vec{X} .

2.1.1 Collinearity

The formula $\beta = \mathbf{v}^{-1}\text{Cov}[\vec{X}, Y]$ makes no sense if \mathbf{v} has no inverse. This will happen if, and only if, the predictor variables are linearly dependent on each other — if one of the predictors is really a linear combination of the others. Then (as we learned in linear algebra) the covariance matrix is of less than “full rank” (i.e., “rank deficient”) and it doesn’t have an inverse.

So much for the algebra; what does that mean statistically? Let’s take an easy case where one of the predictors is just a multiple of the others — say you’ve included people’s weight in pounds (X_1) and mass in kilograms (X_2), so $X_1 = 2.2X_2$. Then if we try to predict Y , we’d have

$$\hat{Y} = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \dots + \beta_p X_p \quad (2.6)$$

$$= 0X_1 + (2.2\beta_1 + \beta_2)X_2 + \sum_{i=3}^p \beta_i X_i \quad (2.7)$$

$$= (\beta_1 + \beta_2/2.2)X_1 + 0X_2 + \sum_{i=3}^p \beta_i X_i \quad (2.8)$$

$$= -2200X_1 + (1000 + \beta_1 + \beta_2)X_2 + \sum_{i=3}^p \beta_i X_i \quad (2.9)$$

In other words, because there’s a linear relationship between X_1 and X_2 , we make the coefficient for X_1 whatever we like, provided we make a corresponding adjustment to the coefficient for X_2 , and it has no effect at all on our prediction. So rather than having one optimal linear predictor, we have infinitely many of them.

There are three ways of dealing with collinearity. One is to get a different data set where the predictor variables are no longer collinear. A second is to identify one of the collinear variables (it doesn’t matter which) and drop it from the data set. This can get complicated; principal components analysis (Chapter 17) can help here. Thirdly, since the issue is that there are infinitely many different coefficient vectors which all minimize the MSE, we could appeal to some extra principle, beyond prediction accuracy, to select just one of them, e.g., try to set as many of the coefficients to zero as possible (Appendix E.4.1).

2.1.2 The Prediction and Its Error

Once we have coefficients β , we can use them to make predictions for the expected value of Y at *arbitrary* values of \vec{X} , whether we’ve an observation there before or not. How good are these?

If we have the optimal coefficients, then the prediction error will be uncorrelated with the predictor variables:

$$\text{Cov}[Y - \vec{X} \cdot \beta, \vec{X}] = \text{Cov}[Y, \vec{X}] - \text{Cov}[\vec{X} \cdot (\mathbf{v}^{-1}\text{Cov}[\vec{X}, Y]), \vec{X}] \quad (2.10)$$

$$= \text{Cov}[Y, \vec{X}] - \mathbf{v}\mathbf{v}^{-1}\text{Cov}[Y, \vec{X}] \quad (2.11)$$

$$= 0 \quad (2.12)$$

Moreover, the expected prediction error, averaged over all \vec{X} , will be zero (exercise). In general, however, the conditional expectation of the error is not zero,

$$\mathbb{E} [Y - \vec{X} \cdot \beta | \vec{X} = \vec{x}] \neq 0 \quad (2.13)$$

and the conditional variance is not constant in \vec{x} ,

$$\text{Var} [Y - \vec{X} \cdot \beta | \vec{X} = \vec{x}_1] \neq \text{Var} [Y - \vec{X} \cdot \beta | \vec{X} = \vec{x}_2] \quad (2.14)$$

2.1.3 Estimating the Optimal Linear Predictor

To actually estimate β from data, we need to make some probabilistic assumptions about where the data comes from. A quite weak but sufficient assumption is that observations (\vec{X}_i, Y_i) are independent for different values of i , with unchanging covariances. Then if we look at the sample covariances, they will, by the law of large numbers, converge on the true covariances:

$$\frac{1}{n} \mathbf{x}^T \mathbf{y} \rightarrow \text{Cov} [\vec{X}, Y] \quad (2.15)$$

$$\frac{1}{n} \mathbf{x}^T \mathbf{x} \rightarrow \mathbf{v} \quad (2.16)$$

where as before \mathbf{x} is the data-frame matrix with one row for each data point and one column for each feature, and similarly for \mathbf{y} .

So, by continuity,

$$\hat{\beta} = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y} \rightarrow \beta \quad (2.17)$$

and we have a consistent estimator.

On the other hand, we could start with the residual sum of squares

$$\text{RSS}(\beta) \equiv \sum_{i=1}^n (y_i - \vec{x}_i \cdot \beta)^2 \quad (2.18)$$

and try to minimize it. The minimizer is the same $\hat{\beta}$ we got by plugging in the sample covariances. No probabilistic assumption is needed to minimize the RSS, but it doesn't let us say anything about the *convergence* of $\hat{\beta}$. For that, we do need some assumptions about \vec{X} and Y coming from distributions with unchanging covariances.

(One can also show that the least-squares estimate is the linear prediction with the minimax prediction risk. That is, its worst-case performance, when everything goes wrong and the data are horrible, will be better than any other linear method. This is some comfort, especially if you have a gloomy and pessimistic view of data, but other methods of estimation may work better in less-than-worst-case scenarios.)

2.1.3.1 Unbiasedness and Variance of Ordinary Least Squares Estimates

The very weak assumptions we have made are still strong enough to let us say a little bit more about the properties of the ordinary least squares estimate $\hat{\beta}$. To do so, we need to think

To get at the variance of $\hat{\beta}$, we need to think a little bit about why it fluctuates. For the moment, let's take \mathbf{x} as fixed, but allow \mathbf{Y} to vary randomly.

The key fact is that $\hat{\beta}$ is linear in the observed responses \mathbf{y} . We can use this by writing, as you're used to from your linear regression class,

$$Y = \vec{X} \cdot \beta + \epsilon \quad (2.19)$$

We just have to remember that while $E[\epsilon] = 0$ and $Cov[\epsilon, \vec{X}] = 0$, it is not generally true that $E[\epsilon | \vec{X} = \vec{x}] = 0$ or that $Var[\epsilon | \vec{X} = \vec{x}]$ is constant. Even with these limitations, we can still say that

$$\hat{\beta} = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{Y} \quad (2.20)$$

$$= (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T (\mathbf{x}\beta + \epsilon) \quad (2.21)$$

$$= \beta + (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \epsilon \quad (2.22)$$

This directly tells us that $\hat{\beta}$ is unbiased:

$$E[\hat{\beta} | \mathbf{x}] = \beta + (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T E[\epsilon] \quad (2.23)$$

$$= \beta + 0 = \beta \quad (2.24)$$

We can also get the variance matrix of $\hat{\beta}$:

$$Var[\hat{\beta} | \mathbf{x}] = Var[\beta + (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \epsilon | \mathbf{x}] \quad (2.25)$$

$$= Var[(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \epsilon | \mathbf{x}] \quad (2.26)$$

$$= (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T Var[\epsilon | \mathbf{x}] \mathbf{x} (\mathbf{x}^T \mathbf{x})^{-1} \quad (2.27)$$

Let's write $Var[\epsilon | \mathbf{x}]$ as a single matrix $\Sigma(\mathbf{x})$. If the linear-prediction errors are uncorrelated with each other, then Σ will be diagonal. If they're also of equal variance, then $\Sigma = \sigma^2 \mathbf{I}$, and we have

$$Var[\hat{\beta} | \mathbf{x}] = \sigma^2 (\mathbf{x}^T \mathbf{x})^{-1} \quad (2.28)$$

and (by the law of total variance)

$$Var[\hat{\beta}] = \frac{\sigma^2}{n} \mathbf{v}^{-1} \quad (2.29)$$

Said in words, this means that the variance of our estimates of the linear-regression coefficient will (i) go down with the sample size n , (ii) go up as the linear regression gets worse (σ^2 grows), and (iii) go down as the predictor variables, the components of \vec{X} , have more variance themselves, and are more nearly uncorrelated with each other.

2.2 Shifting Distributions, Omitted Variables, and Transformations

2.2.1 Changing Slopes

I said earlier that the best β in linear regression will depend on the distribution of the predictor variable, unless the conditional mean is exactly linear. Here is an illustration. For simplicity, let's say that $p = 1$, so there's only one predictor variable. I generated data from $Y = \sqrt{X} + \epsilon$, with $\epsilon \sim \mathcal{N}(0, 0.05^2)$ (i.e. the standard deviation of the noise was 0.05).

Figure 2.1 shows the regression lines inferred from samples with three different distributions of X : the black points are $X \sim \text{Unif}(0, 1)$, the blue are $X \sim \mathcal{N}(0.5, 0.01)$ and the red $X \sim \text{Unif}(2, 3)$. The regression lines are shown as colored solid lines; those from the blue and the black data are quite similar — and similarly wrong. The dashed black line is the regression line fitted to the complete data set. Finally, the light grey curve is the true regression function, $r(x) = \sqrt{x}$.

2.2.1.1 R^2 : Distraction or Nuisance?

This little set-up, by the way, illustrates that R^2 is not a stable property of the distribution either. For the black points, $R^2 = 0.92$; for the blue, $R^2 = 0.70$; and for the red, $R^2 = 0.77$; and for the complete data, 0.96. Other sets of x_i values would give other values for R^2 . Note that while the global linear fit isn't even a good approximation anywhere in particular, it has the highest R^2 .

This kind of perversity can happen even in a completely linear set-up. Suppose now that $Y = a\vec{X} + \epsilon$, and we happen to know a exactly. The variance of Y will be $a^2\text{Var}[X] + \text{Var}[\epsilon]$. The amount of variance our regression “explains” — really, the variance of our predictions — will be $a^2\text{Var}[X]$. So $R^2 = \frac{a^2\text{Var}[X]}{a^2\text{Var}[X] + \text{Var}[\epsilon]}$. This goes to zero as $\text{Var}[X] \rightarrow 0$ and it goes to 1 as $\text{Var}[X] \rightarrow \infty$. It thus has little to do with the quality of the fit, and a lot to do with how spread out the independent variable is.

Notice also how easy it is to get a very high R^2 even when the true model is not linear!

2.2.2 Omitted Variables and Shifting Distributions

That the optimal regression coefficients can change with the distribution of the predictor features is annoying, but one could after all *notice* that the distribution has shifted, and so be cautious about relying on the old regression. More subtle is that the regression coefficients can depend on variables which you do not measure, and those can shift without your noticing anything.

Mathematically, the issue is that

$$\mathbb{E}[Y|\vec{X}] = \mathbb{E}[\mathbb{E}[Y|Z, \vec{X}]|\vec{X}] \quad (2.30)$$

Now, if Y is independent of Z given \vec{X} , then the extra conditioning in the inner expectation does nothing and changing Z doesn't alter our predictions. But in general

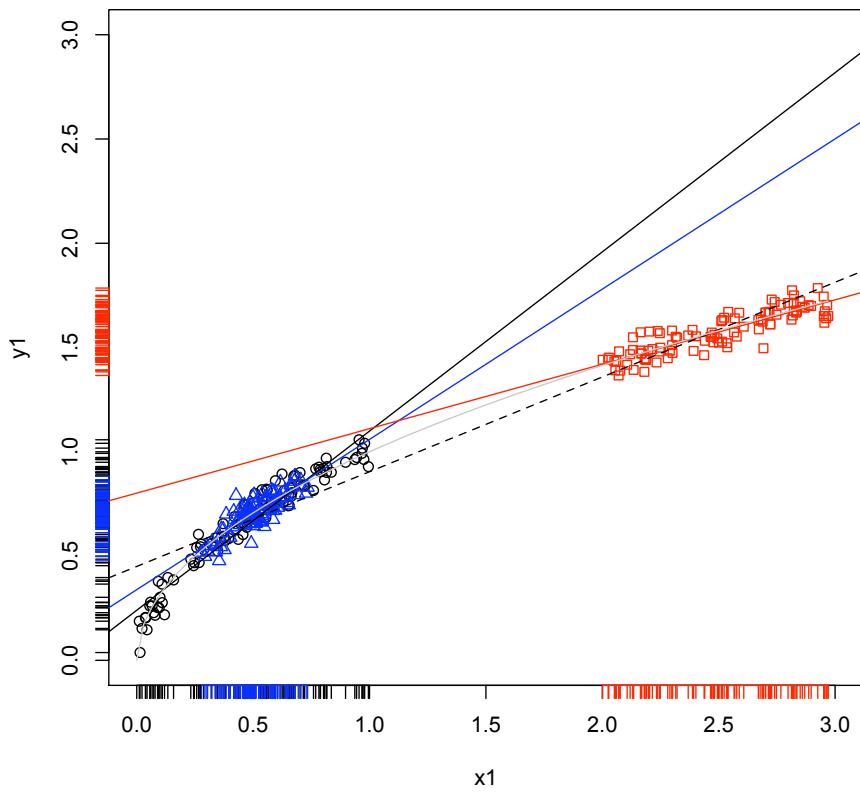


Figure 2.1: Behavior of the conditioning distribution $Y|X \sim \mathcal{N}(\sqrt{X}, 0.05^2)$ with different distributions of X . Black circles: X is uniformly distributed in the unit interval. Blue triangles: Gaussian with mean 0.5 and standard deviation 0.1. Red squares: uniform between 2 and 3. Axis tick-marks show the location of the actual sample points. Solid colored lines show the three regression lines obtained by fitting to the three different data sets; the dashed line is from fitting to all three. The grey curve is the true regression function. (See accompanying R file for commands used to make this figure.)

there will be plenty of variables Z which we don't measure (so they're not included in \vec{X}) but which have some non-redundant information about the response (so that Y depends on Z even conditional on \vec{X}). If the distribution of Z given \vec{X} changes, then the optimal regression of Y on \vec{X} should change too.

Here's an example. X and Z are both $\mathcal{N}(0, 1)$, but with a positive correlation of 0.1. In reality, $Y \sim \mathcal{N}(X + Z, 0.01)$. Figure 2.2 shows a scatterplot of all three variables together ($n = 100$).

Now I change the correlation between X and Z to -0.1 . This leaves both marginal distributions alone, and is barely detectable by eye (Figure 2.3).

Figure 2.4 shows just the X and Y values from the two data sets, in black for the points with a positive correlation between X and Z , and in blue when the correlation is negative. Looking by eye at the points and at the axis tick-marks, one sees that, as promised, there is very little change in the *marginal* distribution of either variable. Furthermore, the correlation between X and Y doesn't change much, going only from 0.75 to 0.74. On the other hand, the regression lines are noticeably different. When $\text{Cov}[X, Z] = 0.1$, the slope of the regression line is 1.2 — high values for X tend to indicate high values for Z , which also increases Y . When $\text{Cov}[X, Z] = -0.1$, the slope of the regression line is 0.80, because now extreme values of X are signs that Z is at the opposite extreme, bringing Y closer back to its mean. But, to repeat, the difference here is due to a change in the correlation between X and Z , not how those variables themselves relate to Y . If I regress Y on X and Z , I get $\hat{\beta} = (0.99, 0.99)$ in the first case and $\hat{\beta} = (0.99, 0.99)$ in the second.

We'll return to this issue of omitted variables when we look at causal inference in Part III.

2.2.3 Errors in Variables

It is often the case that the input features we can actually measure, \vec{X} , are distorted versions of some other variables \vec{U} we wish we could measure, but can't:

$$\vec{X} = \vec{U} + \vec{\eta} \tag{2.31}$$

with $\vec{\eta}$ being some sort of noise. Regressing Y on \vec{X} then gives us what's called an **errors-in-variables** problem.

In one sense, the errors-in-variables problem is huge. We are often much more interested in the connections between *actual* variables in the *real* world, than with our imperfect, noisy measurements of them. Endless ink has been spilled, for instance, on what determines students' examination scores. One thing commonly thrown into the regression — a feature included in \vec{X} — is the income of children's families. But this is typically *not* measured with absolute precision³, so what we are really interested in — the relationship between actual income and school performance — is not what we are estimating in our regression. Typically, adding noise to the input

³One common proxy is to ask *the child* what they think their family income is. (I didn't believe that either when I first heard about it.)

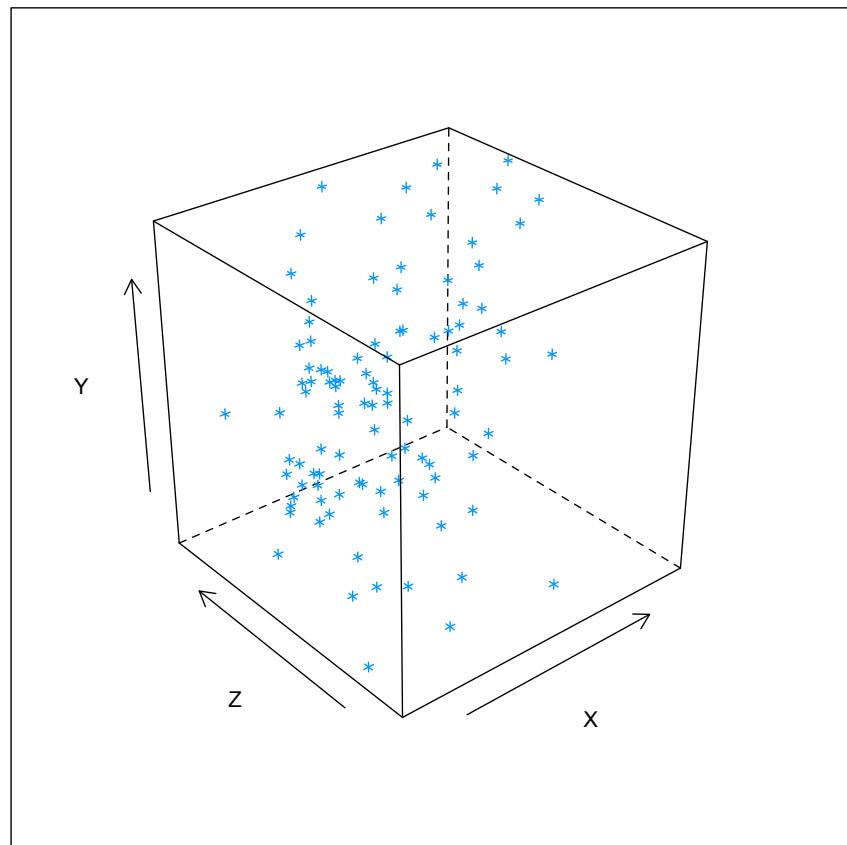


Figure 2.2: Scatter-plot of response variable Y (vertical axis) and two variables which influence it (horizontal axes): X , which is included in the regression, and Z , which is omitted. X and Z have a correlation of +0.1. (Figure created using the `cloud` command in the package `lattice`; see accompanying R file.)

2.2. SHIFTING DISTRIBUTIONS, OMITTED VARIABLES, AND TRANSFORMATIONS

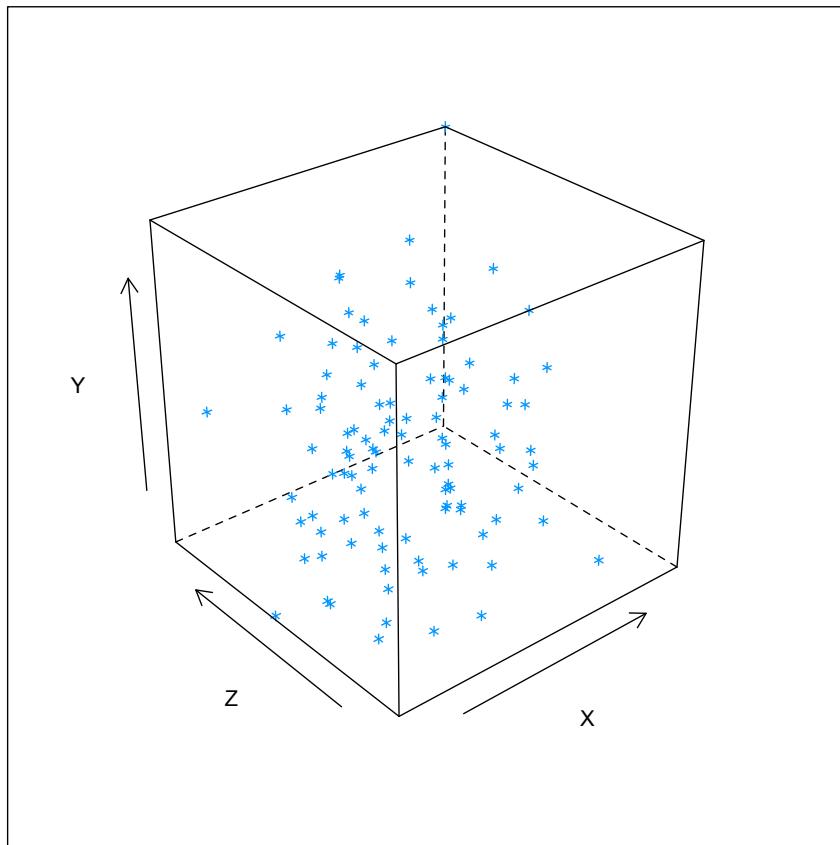


Figure 2.3: As in Figure 2.2, but shifting so that the correlation between X and Z is now -0.1 , though the marginal distributions, and the distribution of Y given X and Z , are unchanged. (See accompanying R file for commands used to make this figure.)

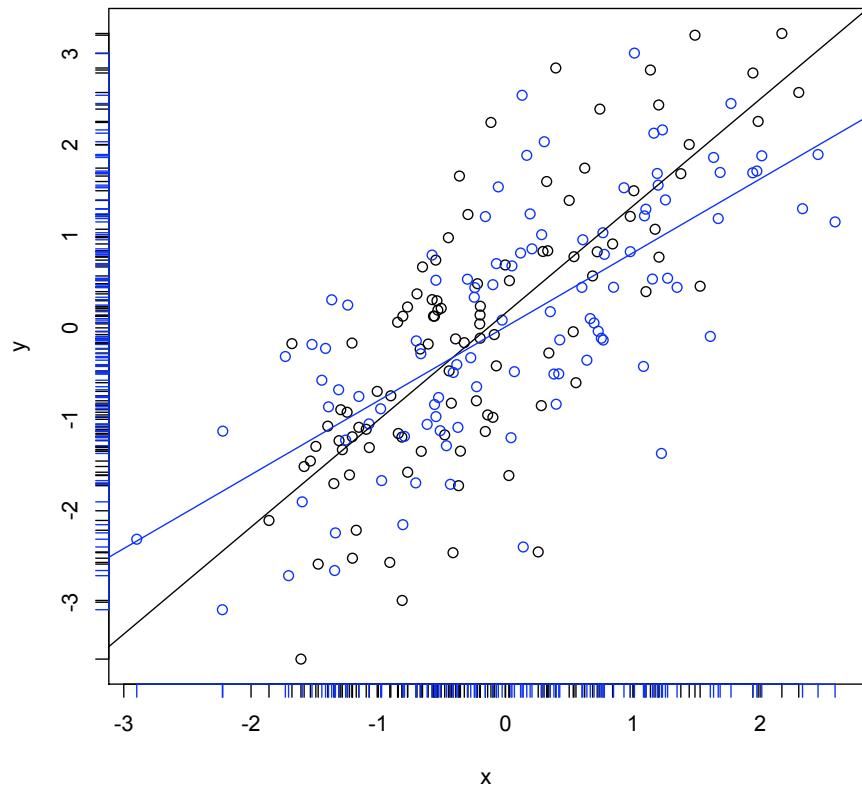


Figure 2.4: Joint distribution of X and Y from Figure 2.2 (black, with a positive correlation between X and Z) and from Figure 2.3 (blue, with a negative correlation between X and Z). Tick-marks on the axes show the marginal distributions, which are manifestly little-changed. (See accompanying R file for commands.)

2.2. SHIFTING DISTRIBUTIONS, OMITTED VARIABLES, AND TRANSFORMATIONS

features makes them less predictive of the response — in linear regression, it tends to push $\hat{\beta}$ closer to zero than it would be if we could regress Y on \vec{U} .

On account of the error-in-variables problem, some people get very upset when they see imprecisely-measured features as inputs to a regression. Some of them, in fact, demand that the input variables be measured *exactly*, with no noise whatsoever.

This position, however, is crazy, and indeed there's a sense where it isn't actually a problem at all. Our earlier reasoning about how to find the optimal linear predictor of Y from \vec{X} remains valid whether something like Eq. 2.31 is true or not. Similarly, the reasoning last time about the actual regression function being the over-all optimal predictor, etc., is unaffected. If in the future we will continue to have \vec{X} rather than \vec{U} available to us for prediction, then Eq. 2.31 is irrelevant *for prediction*. Without better data, the relationship of Y to \vec{U} is just one of the unanswerable questions the world is full of, as much as “what song the sirens sang, or what name Achilles took when he hid among the women”.

Now, if you are willing to assume that $\vec{\eta}$ is a *very* nicely behaved Gaussian and you know its variance, then there are standard solutions to the error-in-variables problem for linear regression — ways of estimating the coefficients you'd get if you could regress Y on \vec{U} . I'm not going to go over them, partly because they're in standard textbooks, but mostly because the assumptions are hopelessly demanding.⁴

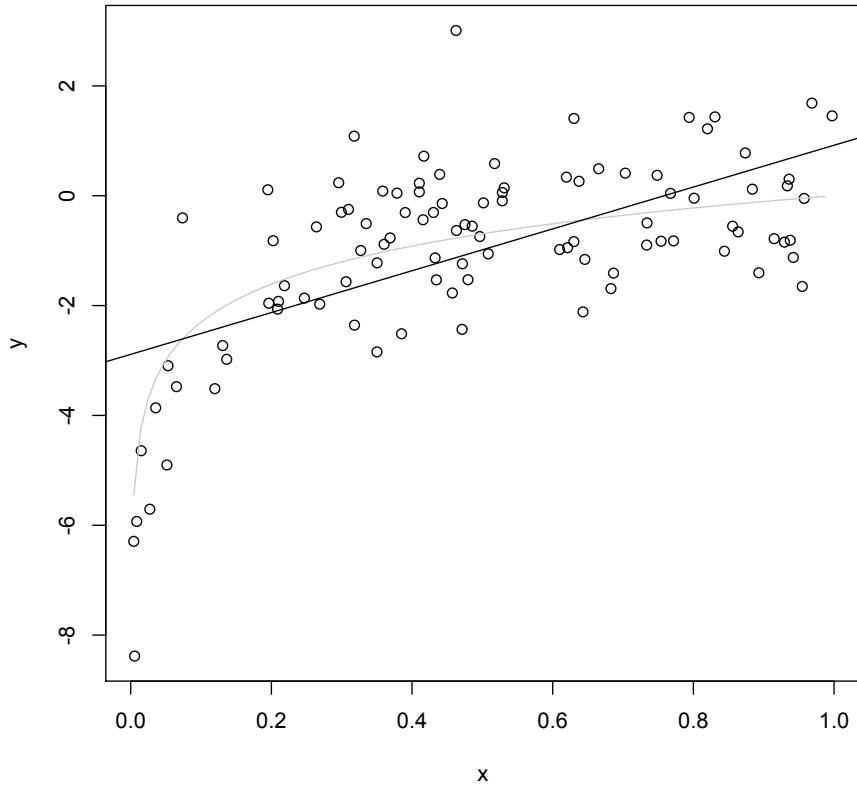
2.2.4 Transformation

Let's look at a simple non-linear example, $Y|X \sim \mathcal{N}(\log X, 1)$. The problem with smoothing data from this source on to a straight line is that the true regression curve isn't very straight, $E[Y|X = x] = \log x$. (Figure 2.5.) This suggests replacing the variables we have with ones where the relationship *is* linear, and then undoing the transformation to get back to what we actually measure and care about.

We have two choices: we can transform the response Y , or the predictor X . Here transforming the response would mean regressing $\exp Y$ on X , and transforming the predictor would mean regressing Y on $\log X$. Both kinds of transformations can be worth trying, but transforming the predictors is, in my experience, often a better bet, for three reasons.

1. Mathematically, $E[f(Y)] \neq f(E[Y])$. A mean-squared optimal prediction of $f(Y)$ is not necessarily close to the transformation of an optimal prediction of Y . And Y is, presumably, what we really want to predict. (Here, however, it works out.)
2. Imagine that $Y = \sqrt{X} + \log Z$. There's not going to be any particularly nice transformation of Y that makes everything linear; though there will be transformations of the features.
3. This generalizes to more complicated models with features built from multiple covariates.

⁴Non-parametric error-in-variable methods are an active topic of research (Carroll *et al.*, 2009).



```
x <- runif(100)
y <- rnorm(100,mean=log(x),sd=1)
plot(y~x)
curve(log(x),add=TRUE,col="grey")
abline(lm(y~x))
```

Figure 2.5: Sample of data for $Y|X \sim \mathcal{N}(\log X, 1)$. (Here $X \sim \text{Unif}(0, 1)$, and all logs are natural logs.) The true, logarithmic regression curve is shown in grey (because it's not really observable), and the linear regression fit is shown in black.

Figure 2.6 shows the effect of these transformations. Here transforming the predictor does, indeed, work out more nicely; but of course I chose the example so that it does so.

To expand on that last point, imagine a model like so:

$$r(\vec{x}) = \sum_{j=1}^q c_j f_j(\vec{x}) \quad (2.32)$$

If we know the functions f_j , we can estimate the optimal values of the coefficients c_j by least squares — this is a regression of the response on new features, which happen to be defined in terms of the old ones. Because the parameters are outside the functions, that part of the estimation works just like linear regression. Models embraced under the heading of Eq. 2.32 include linear regressions with **interactions** between the independent variables (set $f_j = x_i x_k$, for various combinations of i and k), and **polynomial regression**. There is however nothing magical about using products and powers of the independent variables; we could regress Y on $\sin x$, $\sin 2x$, $\sin 3x$, etc.

To apply models like Eq. 2.32, we can either (a) fix the functions f_j in advance, based on guesses about what should be good features for this problem; (b) fix the functions in advance by always using some “library” of mathematically convenient functions, like polynomials or trigonometric functions; or (c) try to *find* good functions from the data. Option (c) takes us beyond the realm of linear regression as such, into things like **splines** (Chapter 8) and **additive models** (Chapter 9). Later, after we have seen how additive models work, we’ll examine how to automatically search for transformations of *both* sides of a regression model.

2.3 Adding Probabilistic Assumptions

The usual treatment of linear regression adds many more probabilistic assumptions. Specifically, the assumption is that

$$Y|\vec{X} \sim \mathcal{N}(\vec{X} \cdot \beta, \sigma^2) \quad (2.33)$$

with all Y values being independent conditional on their \vec{X} values. So now we are *assuming* that the regression function is exactly linear; we are *assuming* that at each \vec{X} the scatter of Y around the regression function is Gaussian; we are *assuming* that the variance of this scatter is constant; and we are *assuming* that there is no dependence between this scatter and anything else.

None of these assumptions was needed in deriving the optimal linear predictor. None of them is so mild that it should go without comment or without at least some attempt at testing.

Leaving that aside just for the moment, why make those assumptions? As you know from your earlier classes, they let us write down the likelihood of the observed responses y_1, y_2, \dots, y_n (conditional on the covariates $\vec{x}_1, \dots, \vec{x}_n$), and then estimate β and σ^2 by maximizing this likelihood. As you also know, the maximum likelihood

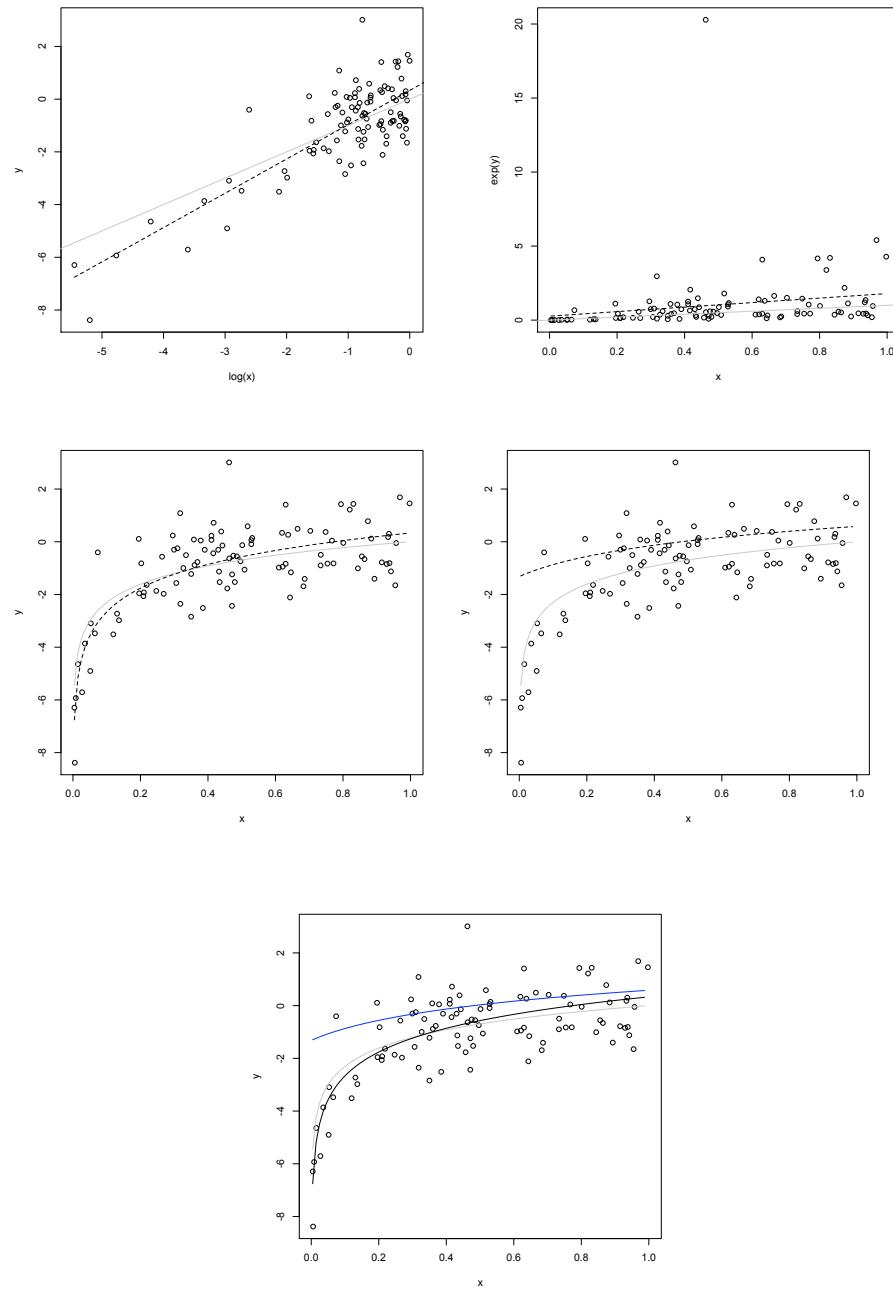


Figure 2.6: Transforming the predictor (left column) and the response (right column) in the data from Figure 2.5, displayed in both the transformed coordinates (top row) and the original coordinates (middle row). The bottom row super-imposes the two estimated curves (transformed X in black, transformed Y in blue). The true regression curve is always shown in grey. (R code deliberately omitted; can you reproduce this?)

estimate of β is exactly the same as the β obtained by minimizing the residual sum of squares. This coincidence would not hold in other models, with non-Gaussian noise.

We saw earlier that $\hat{\beta}$ is consistent under comparatively weak assumptions — that it converges to the optimal coefficients. But then there might, possibly, still be other estimators are also consistent, but which converge faster. If we make the extra statistical assumptions, so that $\hat{\beta}$ is also the maximum likelihood estimate, we can lay that worry to rest. The MLE is generically (and certainly here!) **asymptotically efficient**, meaning that it converges as fast as any other consistent estimator, at least in the long run. So we are not, so to speak, wasting any of our data by using the MLE.

A further advantage of the MLE is that, as $n \rightarrow \infty$, its sampling distribution is itself a Gaussian, centered around the true parameter values. This lets us calculate standard errors and confidence intervals quite easily. Here, with the Gaussian assumptions, much more exact statements can be made about the distribution of $\hat{\beta}$ around β . You can find the formulas in any textbook on regression, so I won't get into that.

We can also use a general property of MLEs for model testing. Suppose we have two classes of models, Ω and ω . Ω is the general case, with p parameters, and ω is a special case, where some of those parameters are constrained, but $q < p$ of them are left free to be estimated from the data. The constrained model class ω is then **nested** within Ω . Say that the MLEs with and without the constraints are, respectively, $\hat{\Theta}$ and $\hat{\theta}$, so the maximum log-likelihoods are $L(\hat{\Theta})$ and $L(\hat{\theta})$. Because it's a maximum over a larger parameter space, $L(\hat{\Theta}) \geq L(\hat{\theta})$. On the other hand, if the true model really is in ω , we'd expect the unconstrained estimate and the constrained estimate to be coming closer and closer. It turns out that the difference in log-likelihoods has an asymptotic distribution which doesn't depend on any of the model details, namely

$$2 \left[L(\hat{\Theta}) - L(\hat{\theta}) \right] \rightsquigarrow \chi^2_{p-q} \quad (2.34)$$

That is, a χ^2 distribution with one degree of freedom for each extra parameter in Ω (that's why they're called "degrees of freedom").⁵

This approach can be used to test particular restrictions on the model, and so it is sometimes used to assess whether certain variables influence the response. This, however, gets us into the concerns of the next section.

2.3.1 Examine the Residuals

By construction, the residuals of a fitted linear regression have mean zero and are uncorrelated with the independent variables. If the usual probabilistic assumptions hold, however, they have many other properties as well.

1. The residuals have a Gaussian distribution at each \vec{x} .

⁵If you assume the noise is Gaussian, the left-hand side of Eq. 2.34 can be written in terms of various residual sums of squares. However, the equation itself remains valid under other noise distributions, which just change the form of the likelihood function. See Appendix C.

2. The residuals have the *same* Gaussian distribution at each \vec{x} , i.e., they are *independent* of the predictor variables. In particular, they must have the same variance (i.e., they must be homoskedastic).
3. The residuals are *independent of each other*. In particular, they must be *uncorrelated* with each other.

These properties — Gaussianity, homoskedasticity, lack of correlation — are all *testable* properties. When they all hold, we say that the residuals are **white noise**. One would never expect them to hold exactly in any finite sample, but if you do test for them and find them strongly violated, you should be extremely suspicious of your model. These tests are much more important than checking whether the coefficients are significantly different from zero.

Every time someone uses linear regression with the standard assumptions for inference and does *not* test whether the residuals are white noise, an angel loses its wings.

2.3.2 On Significant Coefficients

If all the usual distributional assumptions hold, then *t*-tests can be used to decide whether particular coefficients are statistically-significantly different from zero. Pretty much any piece of statistical software, R very much included, reports the results of these tests automatically. It is far too common to seriously over-interpret those results, for a variety of reasons.

Begin with what hypothesis, exactly, is being tested when R (or whatever) runs those *t*-tests. Say, without loss of generality, that there are p predictor variables, $\vec{X} = (X_1, \dots, X_p)$, and that we are testing the coefficient on X_p . Then the null hypothesis is not just “ $\beta_p = 0$ ”, but “ $\beta_p = 0$ in a linear model which also includes X_1, \dots, X_p ”. The alternative hypothesis is not “ $\beta_p \neq 0$ ”, but “ $\beta_p \neq 0$ in a model which also includes X_1, \dots, X_p ”. The optimal linear coefficient on X_p will depend on not just on the relationship between X_p and the response Y , but also on what other variables are included in the model. The *t*-test checks whether adding X_p really improves predictions if one is already using all the other variables — whether it helps prediction “at the margin”, not whether X_p is important in any absolute sense.

Even if you are willing to say “Yes, all I really want to know about this variable is whether adding it to the model really helps me predict”, bear in mind that the question being addressed by the *t*-test is whether adding that variable will help *at all*. Of course, as you know from your regression class, and as we’ll see in more detail next time, expanding the model never hurts its performance on the *training* data. The point of the *t*-test is to gauge whether the improvement in prediction is small enough to be due to chance, or so large, *compared to what noise could produce*, that one could confidently say the variable adds *some* predictive ability. This has several implications which are insufficiently appreciated among users.

In the first place, tests on individual coefficients can seem to contradict tests on groups of coefficients. Adding a set of variables together to the model could significantly improve the fit (as checked by, say, a partial *F* test), even if none of the

coefficients is itself significant. In fact, every single coefficient in the model could be insignificant, while the model as a whole is highly significant.

In the second place, it's worth thinking about which variables will show up as statistically significant. Remember that the t -statistic is $\hat{\beta}_i/\text{se}(\hat{\beta}_i)$, the ratio of the estimated coefficient to its standard error. We saw above that $\text{Var}[\hat{\beta}] = \frac{\sigma^2}{n} \mathbf{v}^{-1}$. This means that the standard errors will be shrink as the sample size grows, so more and more variables will become significant as we get more data — but how much data we collect is irrelevant to how the process we're studying actually works. Moreover, at a fixed sample size, the coefficients with smaller standard errors will tend to be the ones whose variables have more variance, and whose variables are less correlated with the other predictors. High input variance and low correlation help us estimate the coefficient precisely, but, again, they have nothing to do with whether the input variable is actually very closely related to the response.

To sum up, it is *never* the case that statistical significance is the same as scientific, real-world significance. Statistical significance is always about what “signals” can be picked out clearly from background noise. In the case of linear regression coefficients, statistical significance runs together the size of the coefficients, how bad the linear regression model is, the sample size, the variance in the input variable, and the correlation of that variable with all the others.

Of course, even the limited “does it help predictions enough to bother with?” utility of the usual t -test (and F -test) calculations goes away if the standard distributional assumptions do not hold, so that the calculated p -values are just wrong. One can sometimes get away with using bootstrapping (Chapter 6) to get accurate p -values for standard tests under non-standard conditions.

2.4 Linear Regression Is Not the Philosopher's Stone

The philosopher's stone, remember, was supposed to be able to transmute base metals (e.g., lead) into the perfect metal, gold (Eliade, 1971). Many people treat linear regression as though it had a similar ability to transmute a correlation matrix into a scientific theory. In particular, people often argue that:

1. because a variable has a non-zero regression coefficient, it must influence the response;
2. because a variable has a zero regression coefficient, it must not influence the response;
3. if the independent variables change, we can predict how much the response will change by plugging in to the regression.

All of this is wrong, or at best right only under very particular circumstances.

We have already seen examples where influential variables have regression coefficients of zero. We have also seen examples of situations where a variable with no influence has a non-zero coefficient (e.g., because it is correlated with an omitted variable which does have influence). *If* there are no nonlinearities and *if* there are

no omitted influential variables and *if* the noise terms are always independent of the predictor variables, are we good?

No. Remember from Equation 2.5 that the optimal regression coefficients depend on both the marginal distribution of the predictors and the joint distribution (covariances) of the response and the predictors. There is no reason whatsoever to suppose that if we *change* the system, this will leave the conditional distribution of the response alone.

A simple example may drive the point home. Suppose we surveyed all the cars in Pittsburgh, recording the maximum speed they reach over a week, and how often they are waxed and polished. I don't think anyone doubts that there will be a positive correlation here, and in fact that there will be a positive regression coefficient, even if we add in many other variables as predictors. Let us even postulate that the relationship is linear (perhaps after a suitable transformation). Would anyone believe that waxing cars will make them go faster? Manifestly not. But this is exactly how people interpret regressions in all kinds of applied fields — instead of saying waxing makes cars go faster, it might be saying that receiving targeted ads makes customers buy more, or that consuming dairy foods makes diabetes progress faster, or Those claims might be *true*, but the regressions could easily come out the same way if the claims were false. Hence, the regression results provide little or no *evidence* for the claims.

Similar remarks apply to the idea of using regression to "control for" extra variables. If we are interested in the relationship between one predictor, or a few predictors, and the response, it is common to add a bunch of other variables to the regression, to check both whether the apparent relationship might be due to correlations with something else, and to "control for" those other variables. The regression coefficient this is interpreted as how much the response would change, on average, if the independent variable were increased by one unit, "holding everything else constant". There is a very particular sense in which this is true: it's a prediction about the changes in the conditional of the response (conditional on the given values for the other predictors), assuming that observations are randomly drawn from the same population we used to fit the regression.

In a word, what regression does is *probabilistic* prediction. It says what will happen if we keep drawing from the same population, but *select* a sub-set of the observations, namely those with given values of the independent variables. A *causal* or *counterfactual* prediction would say what would happen if we (or Someone) *made* those variables take on those values. There may be no difference between selection and intervention, in which case regression can work as a tool for causal inference⁶; but in general there is. Probabilistic prediction is a worthwhile endeavor, but it's important to be clear that this is what regression does. There are techniques for doing actually causal prediction, which we will explore in Part III.

Every time someone thoughtlessly uses regression for causal inference, an angel not only loses its wings, but is cast out of Heaven and falls in most extreme agony into the everlasting fire.

⁶In particular, if we *assign* values of the independent variables in a way which breaks possible dependencies with omitted variables and noise — either by randomization or by experimental control — then regression can, in fact, work for causal inference.

2.5 Exercises

1. Convince yourself that if the real regression function is linear, β does not depend on the marginal distribution of X . You may want to start with the case of one independent variable.
2. Run the code from Figure 2.5. Then replicate the plots in Figure 2.6.
3. Which kind of transformation is superior for the model where $Y|X \sim \mathcal{N}(\sqrt{X}, 1)$?

Chapter 3

Evaluating Statistical Models: Error and Inference

3.1 What Are Statistical Models For? Summaries, Forecasts, Simulators

There are (at least) three levels at which we can use statistical models in data analysis: as summaries of the data, as predictors, and as simulators.

The lowest and least demanding level is just to use the model as a summary of the data — to use it for **data reduction**, or **compression**. Just as one can use the sample mean or sample quantiles as descriptive statistics, recording some features of the data and saying nothing about a population or a generative process, we could use estimates of a model’s parameters as descriptive summaries. Rather than remembering all the points on a scatter-plot, say, we’d just remember what the OLS regression surface was.

It’s hard to be wrong about a summary, unless we just make a mistake. (It may or may not be *helpful* for us later, but that’s different.) When we say “the slope which minimized the sum of squares was 4.02”, we make no claims about anything *but* the training data. It relies on no assumptions, beyond our doing the calculations right. But it also asserts nothing about the rest of the world. As soon as we try to connect our training data to the rest of the world, we start relying on assumptions, and we run the risk of being wrong.

Probably the most common connection to want to make is to say what *other* data will look like — to make predictions. In a statistical model, with random noise terms, we do not anticipate that our predictions will ever be *exactly* right, but we also anticipate that our mistakes will show stable probabilistic patterns. We can evaluate predictions based on those patterns of error — how big is our typical mistake? are we biased in a particular direction? do we make a lot of little errors or a few huge ones?

Statistical inference about model parameters — estimation and hypothesis testing — can be seen as a kind of prediction, extrapolating from what we saw in a small

piece of data to what we would see in the whole population, or whole process. When we *estimate* the regression coefficient $\hat{b} = 4.02$, that involves predicting new values of the dependent variable, but also predicting that if we repeated the experiment and re-estimated \hat{b} , we'd get a value close to 4.02.

Using a model to summarize old data, or to predict new data, doesn't commit us to assuming that the model describes the process which generates the data. But we often want to do that, because we want to interpret parts of the model as aspects of the real world. We think that in neighborhoods where people have more money, they spend more on houses — perhaps each extra \$1000 in income translates into an extra \$4020 in house prices. Used this way, statistical models become stories about how the data were generated. If they are accurate, we should be able to use them to *simulate* that process, to step through it and produce something that looks, probabilistically, just like the actual data. This is often what people have in mind when they talk about *scientific* models, rather than just statistical ones.

An example: if you want to predict where in the night sky the planets will be, you can actually do very well with a model where the Earth is at the center of the universe, and the Sun and everything else revolve around it. You can even estimate, from data, how fast Mars (for example) goes around the Earth, or where, in this model, it should be tonight. But, since the Earth is *not* at the center of the solar system, those parameters don't actually refer to anything in reality. They are just mathematical fictions. On the other hand, we can also predict where the planets will appear in the sky using models where all the planets orbit the Sun, and the parameters of the orbit of Mars in that model *do* refer to reality.¹

This chapter focuses on evaluating predictions, for three reasons. First, often we just want prediction. Second, if a model can't even predict well, it's hard to see how it could be right scientifically. Third, often the best way of checking a scientific model is to turn some of its implications into statistical predictions.

3.2 Errors, In and Out of Sample

With any predictive model, we can gauge how well it works by looking at its errors. We want these to be small; if they can't be small all the time we'd like them to be small on average. We may also want them to be patternless or unsystematic (because if there was a pattern to them, why not adjust for that, and make smaller mistakes). We'll come back to patterns in errors later, when we look at specification testing (Chapter 10). For now, we'll concentrate on the size of the errors.

To be a little more mathematical, we have a data set with points $\mathbf{z}_n = z_1, z_2, \dots, z_n$. (For regression problems, think of each data point as the pair of input and output values, so $z_i = (x_i, y_i)$, with x_i possibly a vector.) We also have various possible models, each with different parameter settings, conventionally written θ . For regression, θ tells us which regression function to use, so $m_\theta(x)$ or $m(x; \theta)$ is the prediction we make at point x with parameters set to θ . Finally, we have a **loss function** L which

¹We can be pretty confident of this, because we use our parameter estimates to send our robots to Mars, and they get there.

tells us how big the error is when we use a certain θ on a certain data point, $L(z, \theta)$. For mean-squared error, this would just be

$$L(z, \theta) = (y - m_\theta(x))^2 \quad (3.1)$$

But we could also use the mean absolute error

$$L(z, \theta) = |y - m_\theta(x)| \quad (3.2)$$

or many other loss functions. Sometimes we will actually be able to measure how costly our mistakes are, in dollars or harm to patients. If we had a model which gave us a distribution for the data, then $p_\theta(z)$ would a probability density at z , and a typical loss function would be the negative log-likelihood, $-\log p_\theta(z)$. No matter what the loss function is, I'll abbreviate the sample average of the loss over the whole data set by $L(\mathbf{z}_n, \theta)$.

What we would like, ideally, is a predictive model which has zero error on future data. We basically never achieve this:

- The world just really is a noisy and stochastic place, and this means even the true, ideal model has non-zero error.² This corresponds to the first, σ_x^2 , term in the bias-variance decomposition, Eq. 1.26 from Chapter 1.
- Our models are usually more or less **mis-specified**, or, in plain words, wrong. We hardly ever get the functional form of the regression, the distribution of the noise, the form of the causal dependence between two factors, etc., *exactly* right.³ This is the origin of the bias term in the bias-variance decomposition. Of course we can get any of the details in the model specification *more or less* wrong, and we'd prefer to be less wrong.
- Our models are never perfectly estimated. Even if our data come from a perfect IID source, we only ever have a finite sample, and so our parameter estimates are (almost!) never quite the true, infinite-limit values. This is the origin of the variance term in the bias-variance decomposition. But as we get more and more data, the sample should become more and more representative of the whole process, and estimates should converge too.

So, because our models are flawed, we have limited data and the world is stochastic, we cannot expect even the best model to have zero error. Instead, we would like to minimize the **expected error**, or **risk**, or **generalization error**, on new data.

What we would like to do is to minimize the risk or expected loss

$$\mathbb{E}[L(Z, \theta)] = \int dz p(z)L(z, \theta) \quad (3.3)$$

²This is so even if you believe in some kind of ultimate determinism, because the variables we plug in to our predictive models are not complete descriptions of the physical state of the universe, but rather immensely coarser, and this coarseness shows up as randomness.

³Except maybe in fundamental physics, and even there our predictions are about our fundamental theories *in the context of experimental set-ups*, which we never model in complete detail.

To do this, however, we'd have to be able to calculate that expectation. Doing that would mean knowing the distribution of Z — the joint distribution of X and Y , for the regression problem. Since we don't know the true joint distribution, we need to approximate it somehow.

A natural approximation is to use our training data \mathbf{z}_n . For each possible model θ , we can calculate the sample mean of the error on the data, $\bar{L}(\mathbf{z}_n, \theta)$, called the **in-sample loss** or the **empirical risk**. The simplest strategy for estimation is then to pick the model, the value of θ , which minimizes the in-sample loss. This strategy is imaginatively called **empirical risk minimization**. Formally,

$$\widehat{\theta}_n \equiv \operatorname{argmin}_{\theta \in \Theta} \bar{L}(\mathbf{z}_n, \theta) \quad (3.4)$$

This means picking the regression which minimizes the sum of squared errors, or the density with the highest likelihood⁴. This what you've usually done in statistics courses so far, and it's very natural, but it does have some issues, notably optimism and over-fitting.

The problem of optimism comes from the fact that our training data isn't perfectly representative. The in-sample loss is a sample average. By the law of large numbers, then, we anticipate that, for each θ ,

$$\bar{L}(\mathbf{z}_n, \theta) \rightarrow \mathbb{E}[L(Z, \theta)] \quad (3.5)$$

as $n \rightarrow \infty$. This means that, with enough data, the in-sample error is a good approximation to the generalization error of any given model θ . (Big samples are representative of the underlying population or process.) But this does *not* mean that the in-sample performance of $\widehat{\theta}$ tells us how well it will generalize, because we purposely picked it to match the training data \mathbf{z}_n . To see this, notice that the in-sample loss equals the risk plus sampling noise:

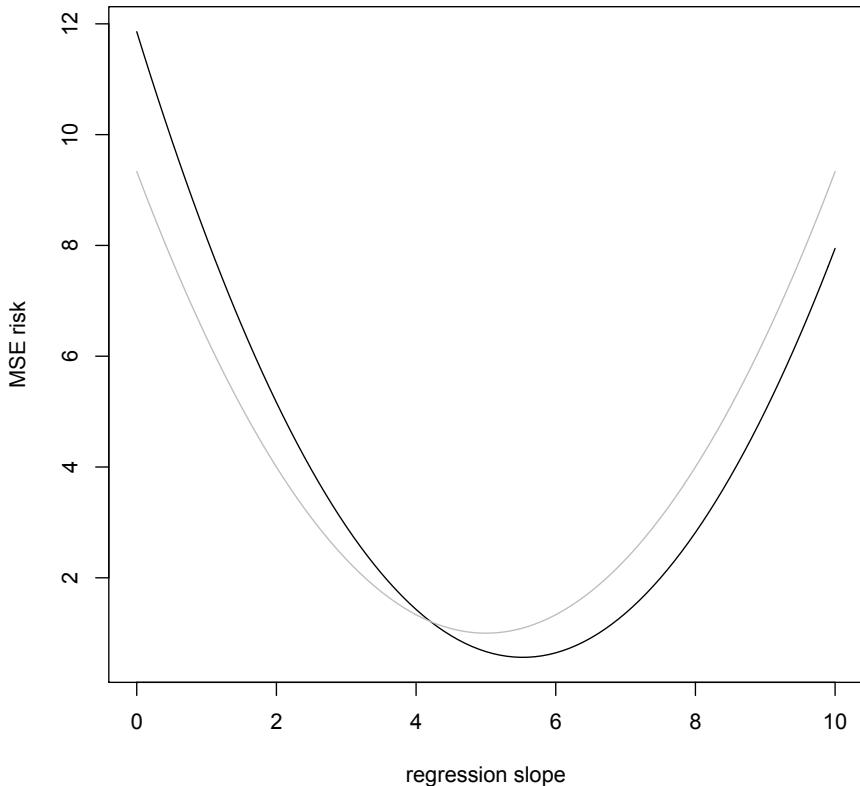
$$\bar{L}(\mathbf{z}_n, \theta) = \mathbb{E}[L(Z, \theta)] + \eta_n(\theta) \quad (3.6)$$

Here $\eta(\theta)$ is a random term which has mean zero, and represents the effects of having only a finite quantity of data, of size n , rather than the complete probability distribution. (I write it $\eta_n(\theta)$ as a reminder that different values of θ are going to be affected differently by the same sampling fluctuations.) The problem, then, is that the model which minimizes the in-sample loss could be one with good generalization performance ($\mathbb{E}[L(Z, \theta)]$ is small), or it could be one which got very lucky ($\eta_n(\theta)$ was large and negative):

$$\widehat{\theta}_n = \operatorname{argmin}_{\theta \in \Theta} (\mathbb{E}[L(Z, \theta)] + \eta_n(\theta)) \quad (3.7)$$

We only want to minimize $\mathbb{E}[L(Z, \theta)]$, but we can't separate it from $\eta_n(\theta)$, so we're almost surely going to end up picking a $\widehat{\theta}_n$ which was more or less lucky ($\eta_n < 0$) as well as good ($\mathbb{E}[L(Z, \theta)]$ small). This is the reason why picking the model which best fits the data tends to exaggerate how well it will do in the future (Figure 3.1).

⁴Remember, maximizing the likelihood is the same as maximizing the log-likelihood, because log is an increasing function. Therefore maximizing the likelihood is the same as *minimizing the negative log-likelihood*.



```

n<-20; theta<-5
x<-runif(n); y<-x*theta+rnorm(n)
empirical.risk <- function(b) { mean((y-b*x)^2) }
true.risk <- function(b) { 1 + (theta-b)^2*(0.5^2+1/12) }
curve(Vectorize(empirical.risk)(x),from=0,to=2*theta,
      xlab="regression slope",ylab="MSE risk")
curve(true.risk,add=TRUE,col="grey")

```

Figure 3.1: Plots of empirical and generalization risk for a simple case of regression through the origin, $Y = \theta X + \epsilon$, $\epsilon \sim \mathcal{N}(0, 1)$, with the true $\theta = 5$. X is uniformly distributed on the unit interval $[0, 1]$. The black curve is the mean squared error on one particular training sample (of size $n = 20$) as we vary the guessed slope; here the minimum is at $\hat{\theta} = 5.53$. The grey curve is the true or generalization risk. (See EXERCISE 2.) The gap between the grey and the black curves is what the text calls $\eta_n(\theta)$.

Again, by the law of large numbers $\eta_n(\theta) \rightarrow 0$ for each θ , but now we need to worry about how fast it's going to zero, and whether that rate depends on θ . Suppose we knew that $\min_\theta \eta_n(\theta) \rightarrow 0$, or $\max_\theta |\eta_n(\theta)| \rightarrow 0$. Then it would follow that $\eta_n(\widehat{\theta}_n) \rightarrow 0$, and the over-optimism in using the in-sample error to approximate the generalization error would at least be shrinking. If we knew how fast $\max_\theta |\eta_n(\theta)|$ was going to zero, we could even say something about how much bigger the true risk was likely to be. A lot of more advanced statistics and machine learning theory is thus about uniform laws of large numbers (showing $\max_\theta |\eta_n(\theta)| \rightarrow 0$) and rates of convergence.

Learning theory is a beautiful, deep, and practically important subject, but also subtle and involved one.⁵ To stick closer to analyzing real data, and to not turn this into an advanced probability class, I will only talk about some more-or-less heuristic methods, which are good enough for many purposes.

3.3 Over-Fitting and Model Selection

The big problem with using the in-sample error is related to over-optimism, but at once trickier to grasp and more important. This is the problem of **over-fitting**. To illustrate it, let's start with Figure 3.2. This has the twenty X values from a Gaussian distribution, and $Y = 7X^2 - 0.5X + \epsilon$, $\epsilon \sim \mathcal{N}(0, 1)$. That is, the true regression curve is a parabola, with additive and independent Gaussian noise. Let's try fitting this — but pretend that we didn't know that the curve was a parabola. We'll try fitting polynomials of different orders in x — order 0 (a flat line), order 1 (a linear regression), order 2 (quadratic regression), up through order 9. Figure 3.3 shows the data with the polynomial curves, and Figure 3.4 shows the in-sample mean squared error as a function of the order of the polynomial.

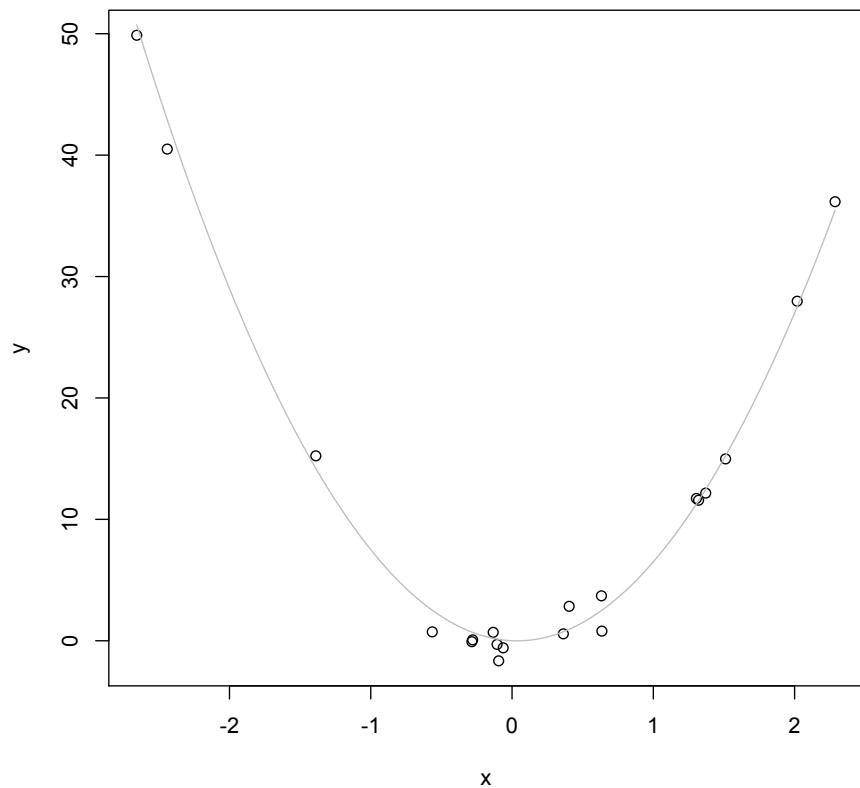
Notice that the in-sample error goes down as the order of the polynomial increases; it has to. Every polynomial of order p is also a polynomial of order $p+1$, so going to a higher-order model can only reduce the in-sample error. Quite generally, in fact, as one uses more and more complex and flexible models, the in-sample error will get smaller and smaller.⁶

Things are quite different if we turn to the generalization error. In principle, I could calculate that for any of the models, since I know the true distribution, but it would involve calculating things like $E[X^{18}]$, which won't be very illuminating. Instead, I will just draw a lot more data from the same source, twenty thousand data points in fact, and use the error of the old models on the new data as their generalization error⁷. The results are in Figure 3.5.

⁵Some comparatively easy starting points are Kearns and Vazirani (1994), Cristianini and Shawe-Taylor (2000) and Mohri *et al.* (2012). At a more advanced level, look at the tutorial papers by Bousquet *et al.* (2004); von Luxburg and Schölkopf (2008), or the textbooks by Vidyasagar (2003), or read the book by Vapnik (2000) (one of the founders).

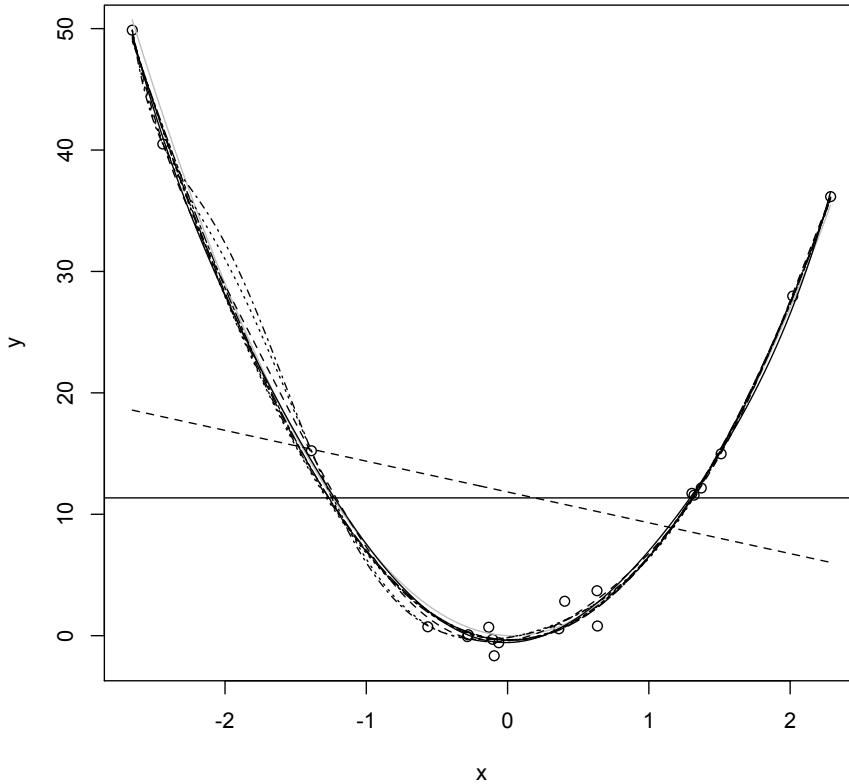
⁶In fact, since there are only 20 data points, they could all be fit exactly if the order of the polynomials went up to 19. (Remember that any two points define a line, any three points a parabola, etc. — $p+1$ points define a polynomial of order p which passes through them.)

⁷This works, yet again, because of the law of large numbers. In Chapters 5 and especially 6, we will see much more about replacing complicated probabilistic calculations with simple simulations.



```
plot(x,y2)
curve(7*x^2-0.5*x,add=TRUE,col="grey")
```

Figure 3.2: Scatter-plot showing sample data and the true, quadratic regression curve (grey parabola).

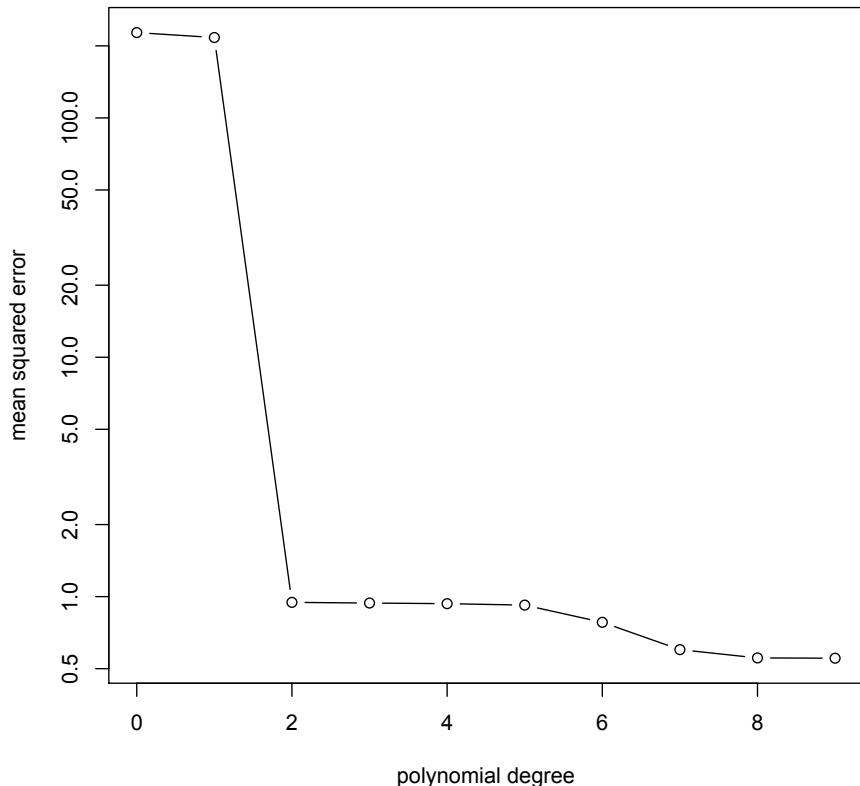


```

y2.0 = lm(y2 ~ 1)
abline(h=y2.0$coefficients[1])
d = seq(-2,2,length.out=200)
for (degree in 1:9) {
  fm = lm(y2 ~ poly(x,degree))
  assign(paste("y2",degree,sep=".") , fm)
  lines(d, predict(fm,data.frame(x=d)),lty=(degree+1))
}

```

Figure 3.3: Twenty training data points (dots), and ten different fitted regression lines (polynomials of order 0 to 9, indicated by different line types). R NOTES: The `poly` command constructs orthogonal (uncorrelated) polynomials of the specified degree from its first argument; regressing on them is conceptually equivalent to regressing on $1, x, x^2, \dots x^{\text{degree}}$, but more numerically stable. (See `help(poly)`.) This use of the `assign` and `paste` functions together is helpful for storing results which don't fit well into arrays.

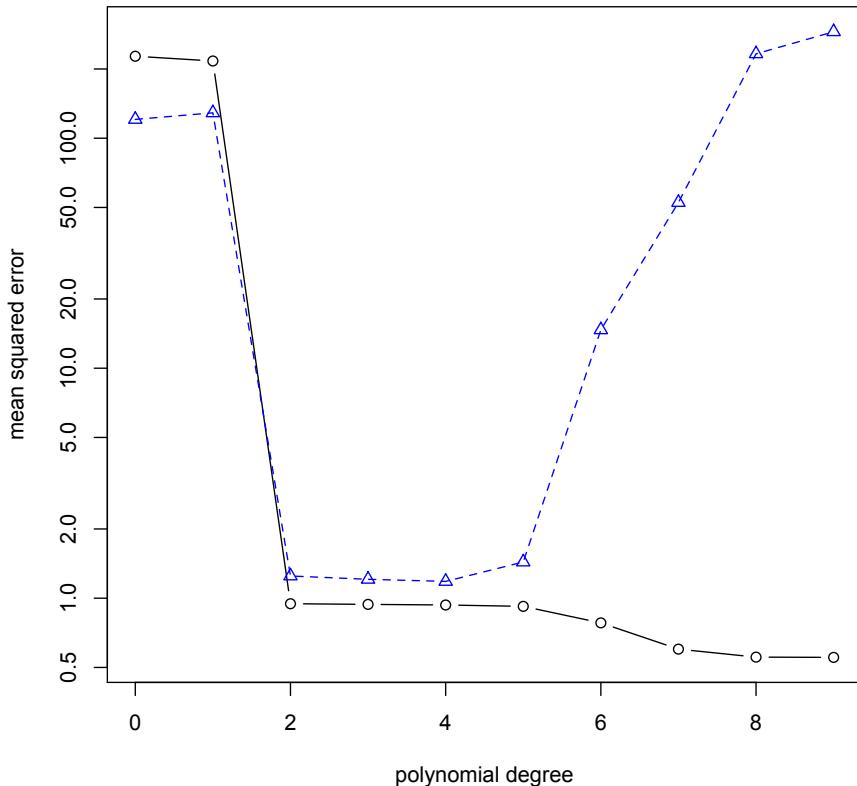


```

mse.q = vector(length=10)
for (degree in 0:9) {
  # The get() function is the inverse to assign()
  fm = get(paste("y",degree,sep="."))
  mse.q[degree+1] = mean(residuals(fm)^2)
}
plot(0:9,mse.q,type="b",xlab="polynomial degree",ylab="mean squared error",
log="y")

```

Figure 3.4: Degree of polynomial vs. its in-sample mean-squared error on the data from the previous figure. Note the logarithmic scale for the vertical axis.



```

gmse.q = vector(length=10)
for (degree in 0:9) {
  fm = get(paste("y",degree,sep="."))
  predictions = predict(fm,data.frame(x=x.new))
  resids = y.new - predictions
  gmse.q[degree+1] = mean(resids^2)
}
plot(0:9,mse.q,type="b",xlab="polynomial degree",
     ylab="mean squared error",log="y",ylim=c(min(mse.q),max(gmse.q)))
lines(0:9,gmse.q,lty=2,col="blue")
points(0:9,gmse.q,pch=24,col="blue")

```

Figure 3.5: In-sample error (black dots) compared to generalization error (blue triangles). Note the logarithmic scale for the vertical axis.

What is happening here is that the higher-order polynomials — beyond order 2 — are not just a *little* optimistic about how well they fit, they are *wildly* over-optimistic. The models which seemed to do notably better than a quadratic actually do much, much worse. If we picked a polynomial regression model based on in-sample fit, we'd chose the highest-order polynomial available, and suffer for it.

What's going on here is that the more complicated models — the higher-order polynomials, with more terms and parameters — were not actually fitting the *generalizable* features of the data. Instead, they were fitting the sampling noise, the accidents which don't repeat. That is, the more complicated models **over-fit** the data. In terms of our earlier notation, η is bigger for the more flexible models. The model which does best here is the quadratic, because the true regression function happens to be of that form. The more powerful, more flexible, higher-order polynomials were able to get closer to the training data, but that just meant matching the noise better. In terms of the bias-variance decomposition, the bias shrinks with the model order, but the variance of estimation grows.

Notice that the models of order 0 and order 1 also do worse than the quadratic model — their problem is not over-fitting but *under-fitting*; they would do better if they were more flexible. Plots of generalization error like this usually have a minimum. If we have a choice of models — if we need to do **model selection** — we would like to find the minimum. Even if we do not have a *choice* of models, we might like to know how big the gap between our in-sample error and our generalization error is likely to be.

There is nothing special about polynomials here. All of the same lessons apply to variable selection in linear regression, to k -nearest neighbors (where we need to choose k), to kernel regression (where we need to choose the bandwidth), and to other methods we'll see later. In every case, there is going to be a minimum for the generalization error curve, which we'd like to find.

(A minimum with respect to what, though? In Figure 3.5, the horizontal axis is the model order, which here is the number of parameters (minus one). More generally, however, what we care about is some measure of how complex the model space is, which is not necessarily the same thing as the number of parameters. What's more relevant is how flexible the class of models is, how many different functions it can approximate. Linear polynomials can approximate a smaller set of functions than quadratics can, so the latter are more complex, or have higher **capacity**. More advanced learning theory has a number of ways of quantifying this, but the details get pretty arcane, and we will just use the concept of complexity or capacity informally.)

3.4 Cross-Validation

The most straightforward way to find the generalization error would be to do what I did above, and to use fresh, independent data from the same source — a **testing** or **validation** data-set. Call this \mathbf{z}'_m , as opposed to our training data \mathbf{z}_n . We fit our model to \mathbf{z}_n , and get $\widehat{\boldsymbol{\theta}}_n$. The loss of this on the validation data is

$$\mathbb{E} \left[L(Z, \widehat{\boldsymbol{\theta}}_n) \right] + \eta'_m(\widehat{\boldsymbol{\theta}}_n) \quad (3.8)$$

where now the sampling noise on the *validation* set, η'_m , is independent of $\widehat{\theta}_m$. So this gives us an unbiased estimate of the generalization error, and, if m is large, a precise one. If we need to select one model from among many, we can pick the one which does best on the validation data, with confidence that we are not just over-fitting.

The problem with this approach is that we absolutely, positively, cannot use any of the validation data in estimating the model. Since collecting data is expensive — it takes time, effort, and usually money, organization, effort and skill — this means getting a validation data set is expensive, and we often won't have that luxury.

3.4.1 Data-set Splitting

The next logical step, however, is to realize that we don't strictly need a separate validation set. We can just take our data and *split* it ourselves into training and testing sets. If we divide the data into two parts at random, we ensure that they have (as much as possible) the same distribution, and that they are independent of each other. Then we can act just as though we had a real validation set. Fitting to one part of the data, and evaluating on the other, gives us an unbiased estimate of generalization error. Of course it doesn't matter which half of the data is used to train and which half is used to test, so we can do it both ways and average.

Figure 3.6 illustrates the idea with a bit of the data and linear models from the first homework set, and Code Example 1 shows the code used to make Figure 3.6.

3.4.2 *k*-Fold Cross-Validation (CV)

The problem with data-set splitting is that, while it's an unbiased estimate of the risk, it is often a very noisy one. If we split the data evenly, then the test set has $n/2$ data points — we've cut in half the number of sample points we're averaging over. It would be nice if we could reduce that noise somewhat, especially if we are going to use this for model selection.

One solution to this, which is pretty much the industry standard, is what's called ***k*-fold cross-validation**. Pick a small integer k , usually 5 or 10, and divide the data at random into k equally-sized subsets. (The subsets are sometimes called “folds”.) Take the first subset and make it the test set; fit the models to the rest of the data, and evaluate their predictions on the test set. Now make the second subset the test set and the rest of the training sets. Repeat until each subset has been the test set. At the end, average the performance across test sets. This is the cross-validated estimate of generalization error for each model. Model selection then picks the model with the smallest estimated risk.⁸

The reason cross-validation works is that it uses the existing data to simulate the process of generalizing to new data. If the full sample is large, then even the smaller portion of it in the training data is, with high probability, fairly representative of the data-generating process. *Randomly* dividing the data into training and test sets makes

⁸A closely related procedure, sometimes also called “*k*-fold CV”, is to pick $1/k$ of the data points at random to be the test set (using the rest as a training set), and then pick an *independent* $1/k$ of the data points as the test set, etc., repeating k times and averaging. The differences are subtle, but what's described in the main text makes sure that each point is used in the test set just once.

	Median_house_value	Median_household_income	Median_rooms	
1	909600	111667	6.0	
2	748700	66094	4.6	
3	773600	87306	5.0	
4	579200	62386	4.5	
5	480800	55658	4.8	
6	460800	38646	4.3	
...	
10605	253400	71638	6.6	

	Median_house_value	Median_household_income	Median_rooms	
2	748700	66094	4.6	
3	773600	87306	5.0	
(A) 5	480800	55658	4.8	
6	460800	38646	4.3	
8	439300	59091	4.4	
...	
10605	253400	71638	6.6	

	Median_house_value	Median_household_income	Median_rooms	
1	909600	111667	6.0	
4	579200	62386	4.5	
(B) 7	473500	52837	4.3	
9	369800	40402	4.6	
10	467200	44140	3.5	
14	353700	27138	3.9	
...	
10604	209500	56667	6.0	

	$\widehat{\beta}_{\text{intercept}}$	$\widehat{\beta}_{\text{income}}$	$\widehat{\beta}_{\text{rooms}}$	MSE
(A) Income only	$(2.74 \pm 0.56) \times 10^4$	5.252 ± 0.085	NA	2.62×10^{10}
Income + Rooms	$(4.772 \pm 0.093) \times 10^5$	7.748 ± 0.081	$(-1.125 \pm 0.020) \times 10^5$	1.66×10^{10}
(B) Income only	$\widehat{\beta}_{\text{intercept}}$	$\widehat{\beta}_{\text{income}}$	$\widehat{\beta}_{\text{rooms}}$	MSE
Income + Rooms	$(3.99 \pm .55) \times 10^4$	4.99 ± 0.8	NA	2.59×10^{10}
	$(5.040 \pm 0.089) \times 10^5$	7.609 ± 0.079	$(-1.162 \pm 0.020) \times 10^5$	1.56×10^{10}

	MSE($A \rightarrow B$)	MSE($B \rightarrow A$)	average
Income only	2.60×10^{10}	2.62×10^{10}	2.61×10^{10}
Income + Rooms	1.56×10^{10}	1.67×10^{10}	1.61×10^{10}

Figure 3.6: Example of data-set splitting. The top table shows three columns and seven rows of the housing-price data used in homework 1. This is then randomly split into two equally sized parts (tables in the second row). I estimate a linear model which predicts house value from income alone, and another model which predicts from income and the median number of rooms, on each half (parameter estimates and in-sample MSEs in the third row). The fourth row shows the performance of each estimate on the other half of the data, and the average for each of the two models. Note that the larger model *always* has a lower in-sample error, whether or not it is really better, so the in-sample MSEs provide no *evidence* that we should use the larger model. Having a lower score under data-set splitting, however, *is* evidence that the larger model generalizes better. (For R commands used to get these numbers, see Code Example 1.) — Can you explain why the coefficient on the number of rooms is negative?

```

half_A <- sample(1:nrow(housing),size=nrow(housing)/2,replace=FALSE)
half_B <- setdiff(1:nrow(housing),half_A)
small_formula = "Median_house_value ~ Median_household_income"
large_formula = "Median_house_value ~ Median_household_income + Median_rooms"
small_formula <- as.formula(small_formula)
large_formula <- as.formula(large_formula)
mAsmall <- lm(small_formula,data=housing,subset=half_A)
mBsmall <- lm(small_formula,data=housing,subset=half_B)
mAlarge <- lm(large_formula,data=housing,subset=half_A)
mBlarge <- lm(large_formula,data=housing,subset=half_B)
in.sample.mse <- function(model) { mean(residuals(model)^2) }
in.sample.mse(mAsmall); in.sample.mse(mAlarge)
in.sample.mse(mBsmall); in.sample.mse(mBlarge)
new.sample.mse <- function(model,half) {
  test <- housing[half,]
  predictions <- predict(model,newdata=test)
  return(mean((test$Median_house_value - predictions)^2))
}
new.sample.mse(mAsmall,half_B); new.sample.mse(mBsmall,half_A)
new.sample.mse(mBlarge,half_A); new.sample.mse(mAlarge,half_B)

```

Code Example 1: Code used to generate the numbers in Figure 3.6. (Code used to display values from the data frames omitted.)

it very unlikely that the division is rigged to favor any one model class, over and above what it would do on real new data. Of course the original data set is never *perfectly* representative of the full data, and a smaller testing set is even less representative, so this isn't ideal, but the approximation is often quite good. It is especially good at getting the *relative* order of different models right, that is, at controlling over-fitting.⁹

Cross-validation is probably the most widely-used method for model selection, and for picking control settings, in modern statistics. There are circumstances where it can fail — especially if you give it *too many* models to pick among — but it's the first thought of seasoned practitioners, and it should be your first thought, too. The assignments to come will make you *very* familiar with it.

3.4.3 Leave-one-out Cross-Validation

Suppose we did k -fold cross-validation, but with $k = n$. Our testing sets would then consist of single points, and each point would be used in testing once. This is called **leave-one-out cross-validation**. It actually came before k -fold cross-validation, and has two advantages. First, it doesn't require any random number generation, or keeping track of which data point is in which subset. Second, and more importantly, because we are only testing on *one* data point, it's often possible to find what the prediction on the left-out point would be by doing calculations on a model fit to the *whole* data. This means that we only have to fit each model once, rather than k times, which can be a big savings of computing time.

The drawback to leave-one-out CV is subtle but often decisive. Since each training set has $n - 1$ points, any two training sets must share $n - 2$ points. The models fit to those training sets tend to be strongly correlated with each other. Even though we are averaging n out-of-sample forecasts, those are correlated forecasts, so we are not really averaging away all that much noise. With k -fold CV, on the other hand, the fraction of data shared between any two training sets is just $\frac{k-2}{k-1}$, not $\frac{n-2}{n-1}$, so even though the number of terms being averaged is smaller, they are less correlated.

There are situations where this issue doesn't really matter, or where it's overwhelmed by leave-one-out's advantages in speed and simplicity, so there is certainly still a place for it, but one subordinate to k -fold CV.¹⁰

3.5 Warnings

Some caveats are in order.

⁹The cross-validation score for the selected model still tends to be somewhat over-optimistic, because it's still picking the luckiest model — though the influence of luck is much attenuated. Tibshirani and Tibshirani (2009) provides a simple correction.

¹⁰At this point, it may be appropriate to say a few words about the Akaike information criterion, or AIC. AIC also tries to estimate how well a model will generalize to new data. One can show that, under standard assumptions, as the sample size gets large, leave-one-out CV actually gives the same estimate as AIC (Claeskens and Hjort, 2008, §2.9). However, there do not seem to be any situations where AIC works where leave-one-out CV does not work at least as well. So AIC should really be understood as a very fast, but often very crude, approximation to the more accurate cross-validation.

1. All of these model selection methods aim at getting models which will generalize well to new data, *if it follows the same distribution* as old data. Generalizing well even when distributions change is a much harder and much less well-understood problem (Quiñonero-Candela *et al.*, 2009). It is particularly troublesome for a lot of applications involving large numbers of human beings, because society keeps changing all the time — it's natural for the variables to vary, but the *relationships* between variables also change. (That's history.)
2. All the model selection methods we have discussed aim at getting models which *predict well*. This is not necessarily the same as getting the *true theory of the world*. Presumably the true theory will also predict well, but the converse does not necessarily follow. We will see examples later where false but low-capacity models, because they have such low variance of estimation, actually out-predict correctly specified models.

3.5.1 Parameter Interpretation

The last item is worth elaborating on. In many situations, it is very natural to want to attach some substantive, real-world meaning to the parameters of our statistical model, or at least to some of them. I have mentioned examples above like astronomy, and it is easy to come up with many others from the natural sciences. This is also extremely common in the social sciences. It is fair to say that this is much less carefully attended to than it should be.

To take just one example, consider the paper “Luther and Suleyman” by Prof. Murat Iyigun (Iyigun, 2008). The major idea of the paper is to try to help explain why the Protestant Reformation was not wiped out during the European wars of religion (or alternately, why the Protestants did not crush all the Catholic powers), leading western Europe to have a mixture of religions, with profound consequences. Iyigun’s contention is that all of the Christians were so busy fighting the Ottoman Turks, or perhaps so afraid of what might happen if they did not, that conflicts among the European Christians were suppressed. To quote his abstract:

at the turn of the sixteenth century, Ottoman conquests lowered the number of all newly initiated conflicts among the Europeans roughly by 25 percent, while they dampened all longer-running feuds by more than 15 percent. The Ottomans’ military activities influenced the length of intra-European feuds too, with each Ottoman-European military engagement shortening the duration of intra-European conflicts by more than 50 percent.

To back this up, and provide those quantitative figures, Prof. Iyigun estimates linear regression models, of the form¹¹

$$Y_t = \beta_0 + \beta_1 X_t + \beta_2 Z_t + \beta_3 U_t + \epsilon_t \quad (3.9)$$

where Y_t is “the number of violent conflicts initiated among or within continental European countries at time t ”¹², X_t is “the number of conflicts in which the Ottoman

¹¹His Eq. 1 on pp. 1473; I have modified the notation to match mine.

¹²In one part of the paper; he uses other dependent variables elsewhere.

Empire confronted European powers at time t ", Z_t is "the count at time t of the newly initiated number of Ottoman conflicts with others and its own domestic civil discords", U_t is control variables reflecting things like the availability of harvests to feed armies, and ϵ_t is Gaussian noise.

The qualitative idea here, about the influence of the Ottoman Empire on the European wars of religion, has been suggested by quite a few historians before¹³. The point of this paper is to support this rigorously, and make it precise. That support and precision requires Eq. 3.9 to be an accurate depiction of at least part of the process which lead European powers to fight wars of religion. Prof. Iyigun, after all, wants to be able to interpret a negative estimate of β_1 as saying that fighting off the Ottomans *kept* Christians from fighting each other. If Eq. 3.9 is inaccurate, if the model is badly mis-specified, however, β_1 becomes the best approximation to the truth within a systematically wrong model, and the support for claims like "Ottoman conquests lowered the number of all newly initiated conflicts among the Europeans roughly by 25 percent" drains away.

To back up the use of Eq. 3.9, Prof. Iyigun looks at a range of slightly different linear-model specifications (e.g., regress the number of intra-Christian conflicts this year on the number of Ottoman attacks last year), and slightly different methods of estimating the parameters. What he does *not* do is look at the other implications of the model: that residuals should be (at least approximately) Gaussian, that they should be unpredictable from the regressor variables. He does not look at whether the relationships he thinks are linear really are linear (see Chapters 4, 9, and 10). He does not try to simulate his model and look at whether the patterns of European wars it produces resemble actual history (see Chapter 5). He does not try to check whether he has a model which really supports causal inference, though he has a causal question (see Part III).

I do not say any of this to denigrate Prof. Iyigun. His paper is actually *much better* than most quantitative work in the social sciences. This is reflected by the fact that it was published in the *Quarterly Journal of Economics*, one of the most prestigious, and rigorously-reviewed, journals in the field. The point is that by the end of this course, you will have the tools to do *better*.

3.6 Exercises

To think through, not to hand in.

1. Suppose that one of our model classes contains the true and correct model, but we also consider more complicated and flexible model classes. Does the bias-variance trade-off mean that we will over-shoot the true model, and always go for something more flexible, when we have enough data? (This would mean there was such a thing as *too much* data to be reliable.)
2. Derive the formula for the generalization risk in the situation depicted in Figure 3.1, as given by the `true.risk` function in the code for that figure. In particular, explain to yourself where the constants 0.5^2 and $1/12$ come from.

¹³See §1–2 of Iyigun (2008), and MacCulloch (2004, *passim*).

Chapter 4

Using Nonparametric Smoothing in Regression

Having spent long enough running down linear regression, and thought through evaluating predictive models, it is time to turn to constructive alternatives, which are (also) based on smoothing.

Recall the basic kind of smoothing we are interested in: we have a response variable Y , some input variables which we bind up into a vector X , and a collection of data values, $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. By “smoothing”, I mean that predictions are going to be weighted averages of the observed responses in the training data:

$$\hat{r}(x) = \sum_{i=1}^n y_i w(x, x_i, h) \quad (4.1)$$

Most smoothing methods have a control setting, which here I write h , that determines *how much* smoothing we do. With k nearest neighbors, for instance, the weights are $1/k$ if x_i is one of the k -nearest points to x , and $w = 0$ otherwise, so large k means that each prediction is an average over many training points. Similarly with kernel regression, where the degree of smoothing is controlled by the bandwidth h .

Why do we want to do this? How do we pick how much smoothing to do?

4.1 How Much Should We Smooth?

When we smooth very little ($h \rightarrow 0$), then we can match very small, fine-grained or sharp aspects of the true regression function, if there are such. That is, less smoothing leads to less bias. At the same time, less smoothing means that each of our predictions is going to be an average over (in effect) fewer observations, making the prediction noisier. Smoothing less increases the variance of our estimate. Since

$$(\text{total error}) = (\text{noise}) + (\text{bias})^2 + (\text{variance}) \quad (4.2)$$

(Eq. 1.26), if we plot the different components of error as a function of h , we typically get something that looks like Figure 4.1. Because changing the amount of smoothing has opposite effects on the bias and the variance, there is an optimal amount of smoothing, where we can't reduce one source of error without increasing the other. We therefore want to find that optimal amount of smoothing, which is where cross-validation comes in.

You should note, at this point, that the optimal amount of smoothing depends on the real regression curve, our smoothing method, *and* how much data we have. This is because the variance contribution generally shrinks as we get more data.¹ If we get more data, we go from Figure 4.1 to Figure 4.2. The minimum of the over-all error curve has shifted to the left, and we should smooth less.

Strictly speaking, **parameters** are properties of the data-generating process alone, so the optimal amount of smoothing is not really a parameter. If you do think of it as a parameter, you have the problem of why the “true” value changes as you get more data. It's better thought of as a setting or control variable in the smoothing method, to be adjusted as convenient.

4.2 Adapting to Unknown Roughness

Consider Figure 4.3, which graphs two functions, f and g . Both are “smooth” functions in the qualitative, mathematical sense². We could Taylor-expand both functions to approximate their values anywhere, just from knowing enough derivatives at one point x_0 .³ Alternately, if instead of knowing the derivatives at x_0 , we have the values of the functions at a sequence of points x_1, x_2, \dots, x_n , we could use interpolation to fill out the rest of the curve. Quantitatively, however, $f(x)$ is less smooth than $g(x)$ — it changes much more rapidly, with many reversals of direction. For the same degree of inaccuracy in the interpolation $f(\cdot)$ needs more, and more closely spaced, training points x_i than goes $g(\cdot)$.

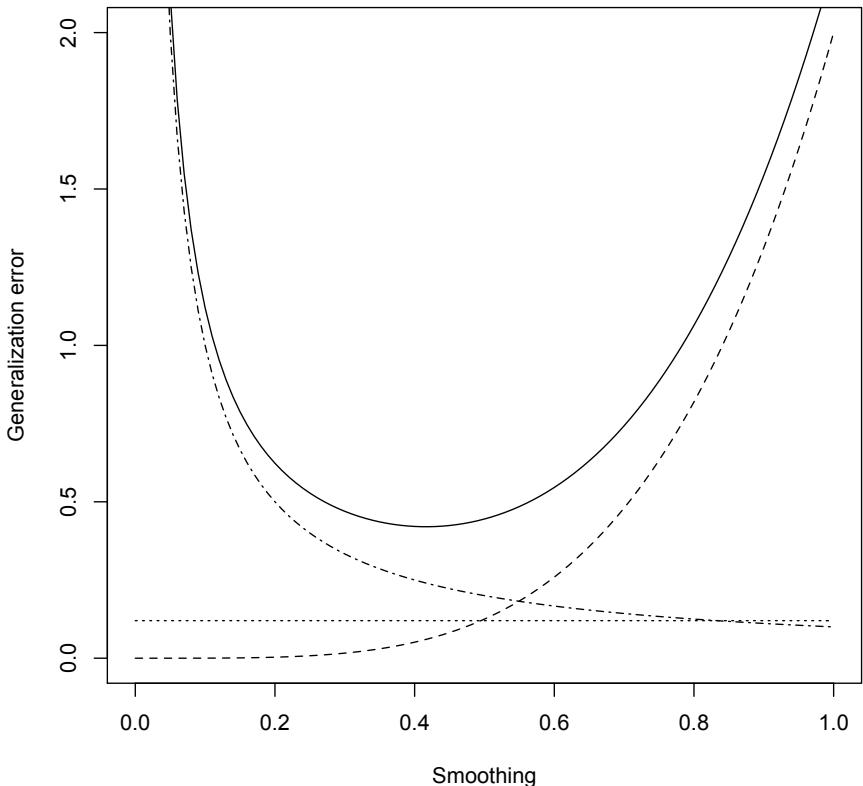
Now suppose that we don't get to actually get to see $f(x)$ and $g(x)$, but rather just $f(x) + \epsilon$ and $g(x) + \eta$, where ϵ and η are noise. (To keep things simple I'll assume they're the usual mean-zero, constant-variance, IID Gaussian noises, say with $\sigma = 0.15$.) The data now look something like Figure 4.4. Can we now recover the curves?

As remarked in Chapter 1, if we had multiple measurements at the same x , then we could recover the expectation value by averaging: since the regression function $r(x) = E[Y|X = x]$, if we had many observations all with $x_i = x$, the average of the corresponding y_i would (by the law of large numbers) converge on $r(x)$. Generally, however, we have at most one measurement per value of x , so simple averaging won't work. Even if we just confine ourselves to the x_i where we have observations, the mean-squared error will always be σ^2 , the noise variance. However, our estimate would be unbiased.

¹Sometimes bias changes as well. Noise does not (why?).

²They are “ C^∞ ”: they're not only continuous, but their derivatives exist and are continuous to all orders.

³Technically, a function whose Taylor series converges everywhere is **analytic**.

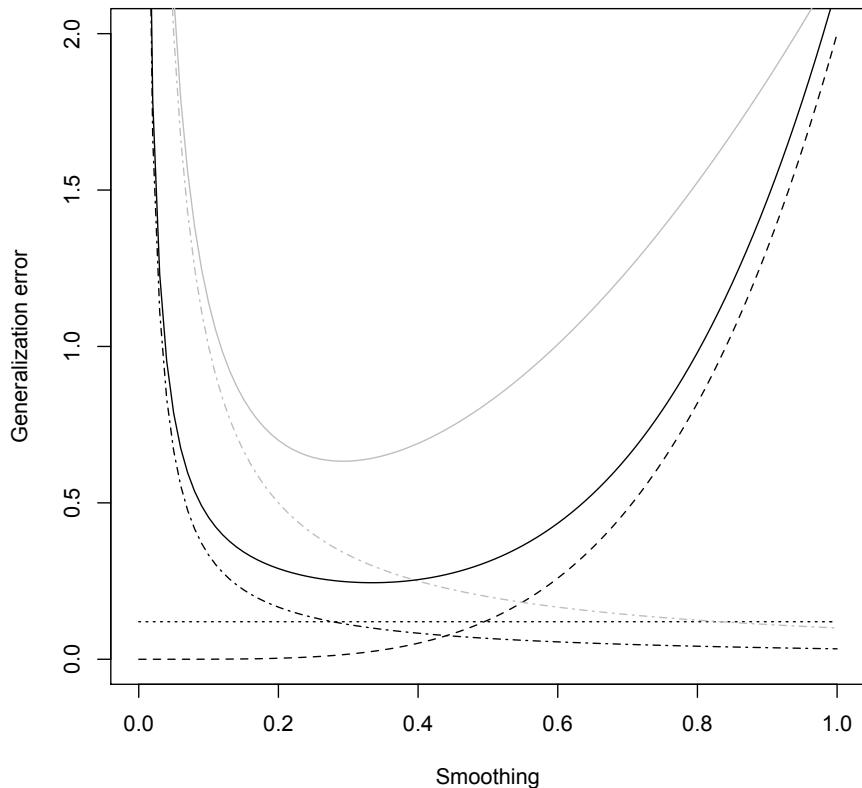


```

curve(2*x^4,from=0,to=1,lty=2,xlab="Smoothing",ylab="Generalization error")
curve(0.12+x-x,lty=3,add=TRUE)
curve(1/(10*x),lty=4,add=TRUE)
curve(0.12+2*x^4+1/(10*x),add=TRUE)

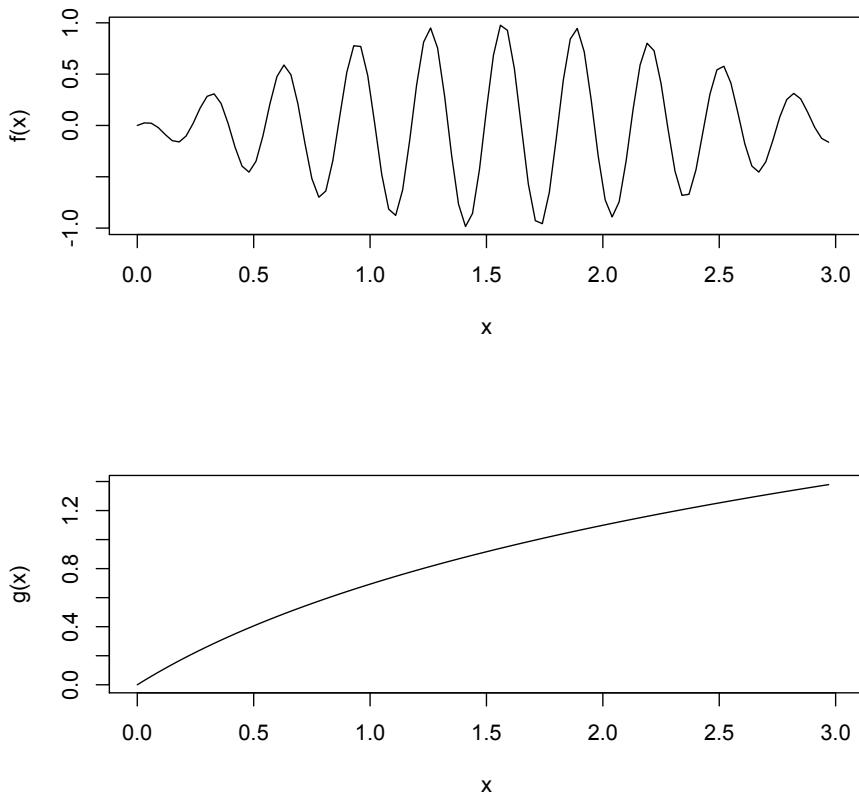
```

Figure 4.1: Over-all generalization error for different amounts of smoothing (solid curve) decomposed into process noise (dotted line), approximation error introduced by smoothing (=squared bias; dashed curve), and estimation variance (dot-and-dash curve). The numerical values here are arbitrary, but the functional forms (squared bias $\propto h^4$, variance $\propto n^{-1}h^{-1}$) are representative of typical results for non-parametric smoothing.



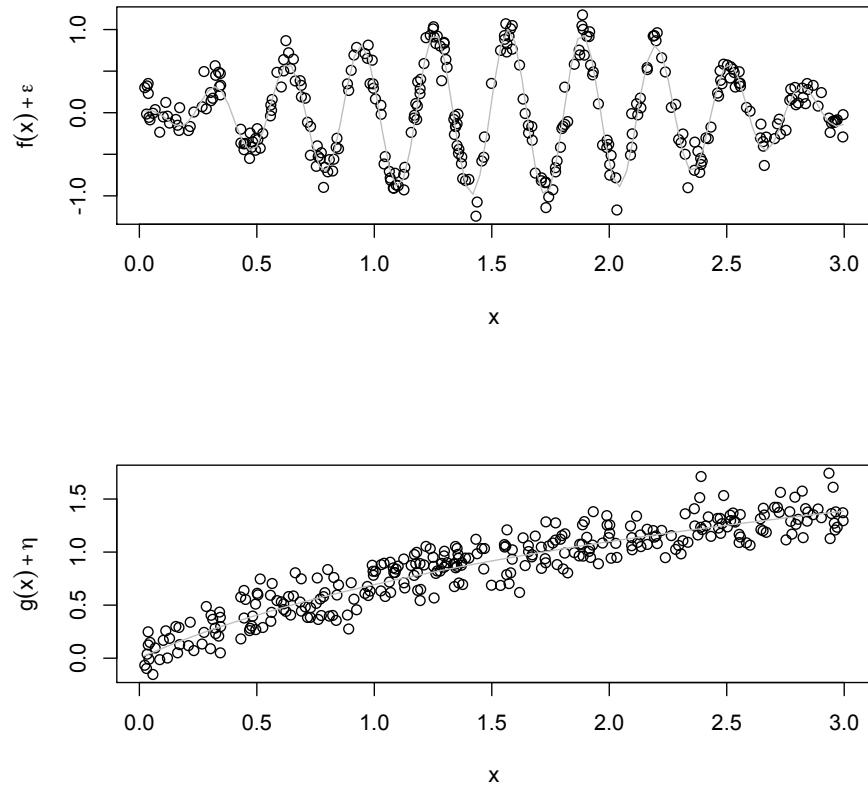
```
curve(2*x^4,from=0,to=1,lty=2,xlab="Smoothing",ylab="Generalization error")
curve(0.12+x-x,lty=3,add=TRUE)
curve(1/(10*x),lty=4,add=TRUE,col="grey")
curve(0.12+2*x^2+1/(10*x),add=TRUE,col="grey")
curve(1/(30*x),lty=4,add=TRUE)
curve(0.12+2*x^4+1/(30*x),add=TRUE)
```

Figure 4.2: Consequences of adding more data to the components of error: noise (dotted) and bias (dashed) are unchanged, but the new variance curve (dotted and dashed, black) is to the left of the old (greyed), so the new over-all error curve (solid black) is lower, and has its minimum at a smaller amount of smoothing than the old (solid grey).



```
par(mfcol=c(2,1))
curve(sin(x)*cos(20*x),from=0,to=3,xlab="x",ylab=expression(f(x)))
curve(log(x+1),from=0,to=3,xlab="x",ylab=expression(g(x)))
```

Figure 4.3: Two curves for the running example. Above, $f(x)$; below, $g(x)$. (As it happens, $f(x) = \sin x \cos 20x$, and $g(x) = \log x + 1$, but that doesn't really matter.)



```

x = runif(300,0,3)
yf = sin(x)*cos(20*x)+rnorm(length(x),0,0.15)
yg = log(x+1)+rnorm(length(x),0,0.15)
par(mfcol=c(2,1))
plot(x,yf,xlab="x",ylab=expression(f(x)+epsilon))
curve(sin(x)*cos(20*x),col="grey",add=TRUE)
plot(x,yg,xlab="x",ylab=expression(g(x)+eta))
curve(log(x+1),col="grey",add=TRUE)

```

Figure 4.4: The same two curves as before, but corrupted by IID Gaussian noise with mean zero and standard deviation 0.15. (The x values are the same, but there are different noise realizations for the two curves.) The light grey line shows the noiseless curves.

What smoothing methods try to use is that we may have multiple measurements at points x_i which are *near* the point of interest x . If the regression function is smooth, as we're assuming it is, $r(x_i)$ will be close to $r(x)$. Remember that the mean-squared error is the sum of bias (squared) and variance. Averaging values at $x_i \neq x$ is going to introduce bias, but averaging many independent terms together also reduces variance. If by smoothing we get rid of more variance than we gain bias, we come out ahead.

Here's a little math to see it. Let's assume that we can do a first-order Taylor expansion (Figure 4.5), so

$$r(x_i) \approx r(x) + (x_i - x)r'(x) \quad (4.3)$$

and

$$y_i \approx r(x) + (x_i - x)r'(x) + \epsilon_i \quad (4.4)$$

Now we average: to keep the notation simple, abbreviate the weight $w(x_i, x, h)$ by just w_i .

$$\hat{r}(x) = \frac{1}{n} \sum_{i=1}^n y_i w_i \quad (4.5)$$

$$= \frac{1}{n} \sum_{i=1}^n (r(x) + (x_i - x)r'(x) + \epsilon_i) w_i \quad (4.6)$$

$$= r(x) + \sum_{i=1}^n w_i \epsilon_i + r'(x) \sum_{i=1}^n w_i (x_i - x) \quad (4.7)$$

$$\hat{r}(x) - r(x) = \sum_{i=1}^n w_i \epsilon_i + r'(x) \sum_{i=1}^n w_i (x_i - x) \quad (4.8)$$

$$\mathbb{E} [(\hat{r}(x) - r(x))^2] = \sigma^2 \sum_{i=1}^n w_i^2 + \mathbb{E} \left[\left(r'(x) \sum_{i=1}^n w_i (x_i - x) \right)^2 \right] \quad (4.9)$$

(Remember that: $\sum w_i = 1$; $\mathbb{E} [\epsilon_i] = 0$; the noise is uncorrelated with everything; and $\mathbb{E} [\epsilon_i^2] = \sigma^2$.)

The first term on the final right-hand side is an estimation variance, which will tend to shrink as n grows. (If $w_i = 1/n$, the unweighted averaging case, we get back the familiar σ^2/n .) The second term, expectation, on the other hand, is bias, which grows as x_i gets further from x , and as the magnitudes of the derivatives grow, i.e., how smooth or wiggly the regression function is. For this to work, w_i had better shrink as $x_i - x$ and $r'(x)$ grow.⁴ Finally, all else being equal, w_i should also shrink with n , so that the over-all size of the sum shrinks as we get more data.

To illustrate, let's try to estimate $f(1.6)$ and $g(1.6)$ from the noisy observations. We'll try a simple approach, just averaging all values of $f(x_i) + \epsilon_i$ and $g(x_i) + \eta_i$

⁴The higher derivatives of r also matter, since we should really be keeping more than just the first term in the Taylor expansion. The details get messy, but Eq. 4.12 below gives the upshot for kernel smoothing in particular.

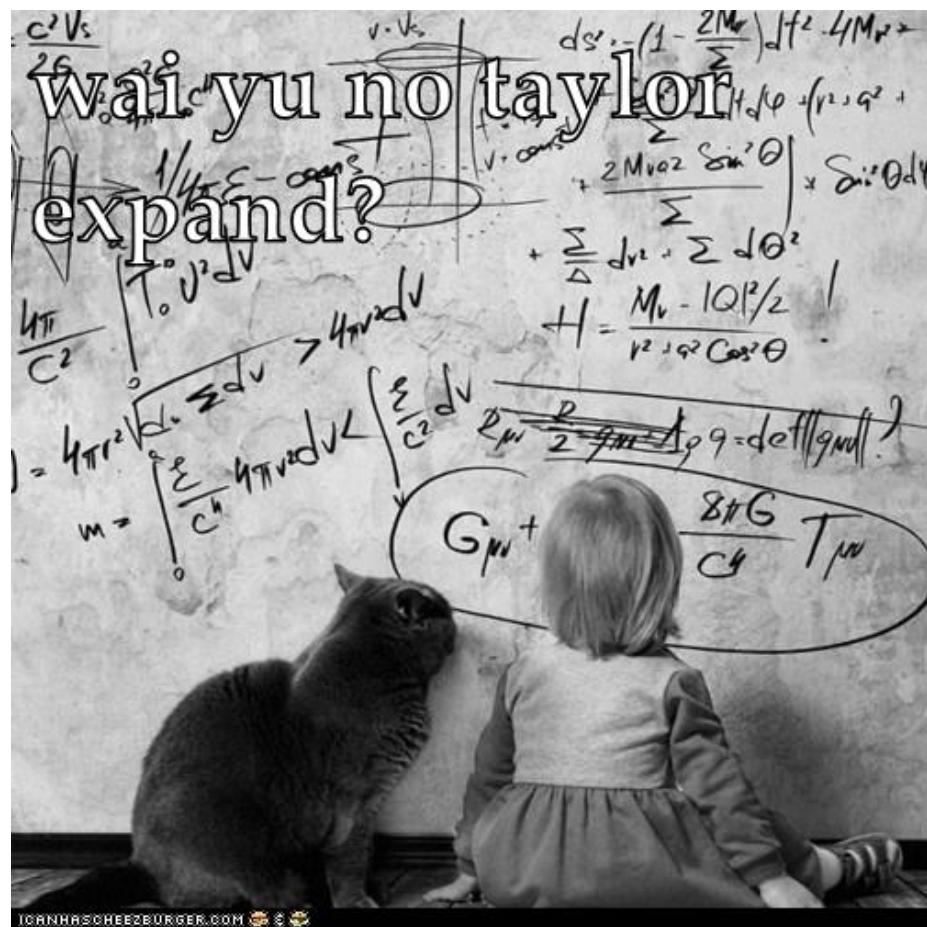


Figure 4.5: Sound advice when stuck on almost any problem in statistical theory.

for $1.5 < x_i < 1.7$ with equal weights. For f , this gives 0.46, while $f(1.6) = 0.89$. For g , this gives 0.98, with $g(1.6) = 0.95$. (See figure 4.6). The same size window introduces a much larger bias with the rougher, more rapidly changing f than with the smoother, more slowly changing g . Varying the size of the averaging window will change the amount of error, and it will change it in different ways for the two functions.

If one does a more careful second-order Taylor expansion like that leading to Eq. 4.9, specifically for kernel regression, one can show that the bias at x is

$$\mathbb{E} [\hat{r}(x) - r(x) | X_1 = x_1, \dots, X_n = x_n] = h^2 \left[\frac{1}{2} r''(x) + \frac{r'(x)f'(x)}{f(x)} \right] \sigma_K^2 + o(h^2) \quad (4.10)$$

where f is the density of x , and $\sigma_K^2 = \int u^2 K(u) du$, the variance of the probability density corresponding to the kernel⁵. The r'' term just comes from the second-order part of the Taylor expansion. To see where the $r'f'$ term comes from, imagine first that x is a mode of the distribution, so $f'(x) = 0$. As h shrinks, only training points where X_i is very close to x will have any weight in $\hat{r}(x)$, and their distribution will be roughly symmetric around x (at least once h is sufficiently small). So, at mode, $\mathbb{E} [w(X_i, x, h)(X_i - x)\hat{r}(x)] \approx 0$. Away from a mode, there will tend to be more training points on one side or the other of x , depending on the sign of $f'(x)$, and this induces a bias. The tricky part of the analysis is concluding that the bias has exactly the form given above.⁶

One can also work out the variance of the kernel regression estimate,

$$\text{Var} [\hat{r}(x) | X_1 = x_1, \dots, X_n = x_n] = \frac{\sigma^2(x)R(K)}{nhf(x)} + o((nb)^{-1}) \quad (4.11)$$

where $R(K) = \int K^2(u) du$. Roughly speaking, the width of the region where the kernel puts non-trivial weight is about h , so there will be about $nhf(x)$ training points available to estimate $\hat{r}(x)$. Each of these has a y_i value, equal to $r(x)$ plus noise of variance $\sigma^2(x)$. The final factor of $R(K)$ accounts for the average weight.

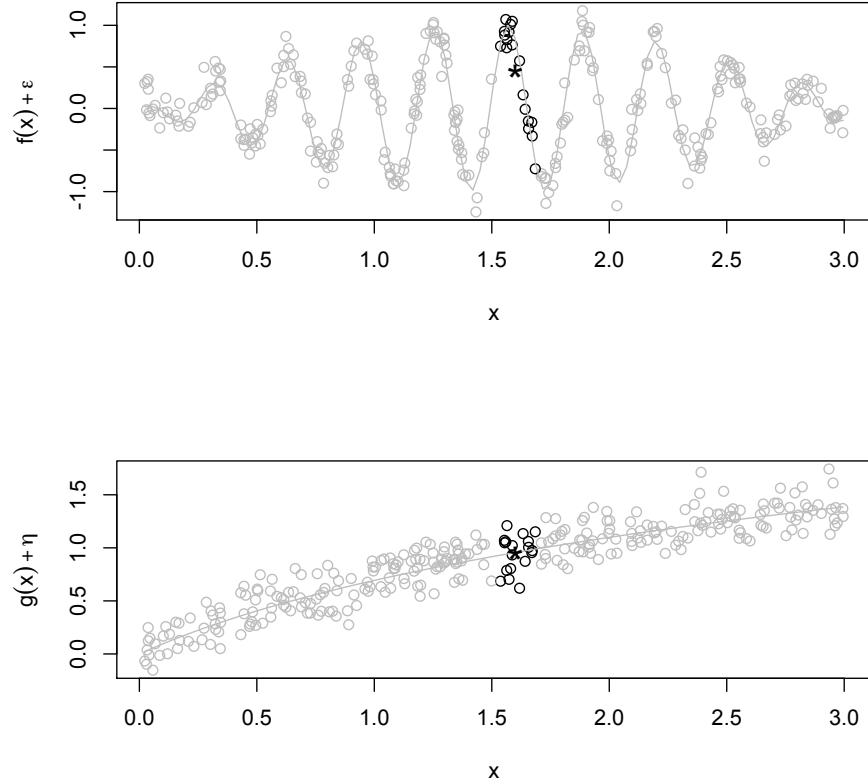
Putting the bias together with the variance, we get an expression for the mean squared error of the kernel regression at x :

$$MSE(x) = \sigma^2(x) + h^4 \left[\frac{1}{2} r''(x) + \frac{r'(x)f'(x)}{f(x)} \right]^2 (\sigma_K^2)^2 + \frac{\sigma^2(x)R(K)}{nhf(x)} + o(h^4) + o(1/nb) \quad (4.12)$$

Eq. 4.12 tells us that, in principle, there is a single optimal choice of bandwidth h , an optimal degree of smoothing. We could find it by taking Eq. 4.12, differentiating with

⁵If you are not familiar with the “order” symbols O and o , now would be a good time to read Appendix B.

⁶Exercise 1 shows how to do a bit of the demonstration for the special case of the uniform (boxcar) kernel.

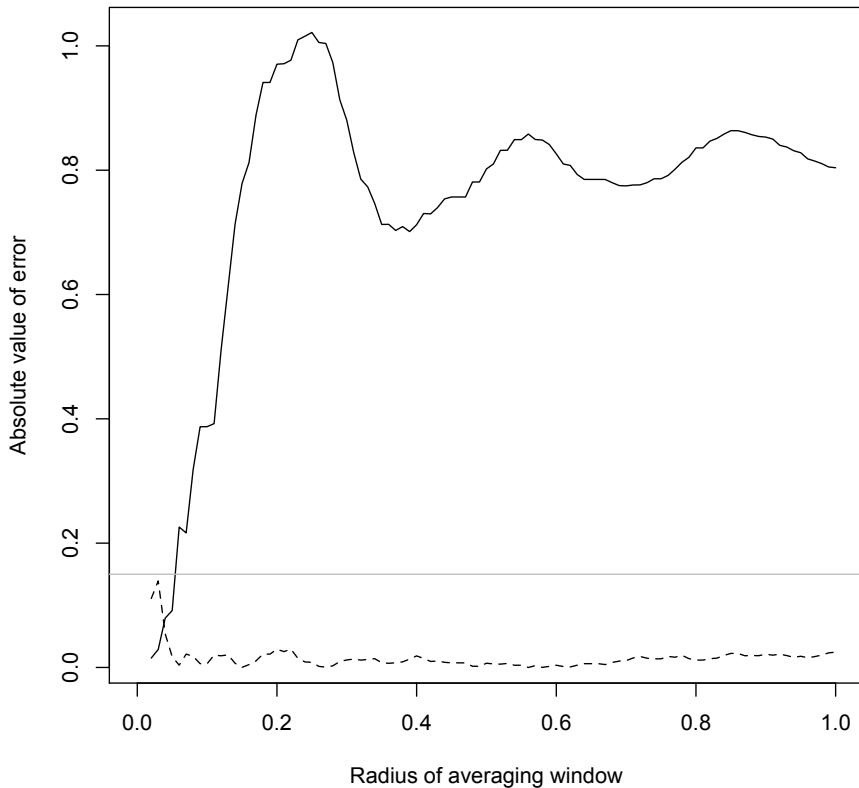


```

par(mfcol=c(2,1))
colors=ifelse((x<1.7)&(x>1.5),"black","grey")
plot(x,yf,xlab="x",ylab=expression(f(x)+epsilon),col=colors)
curve(sin(x)*cos(20*x),col="grey",add=TRUE)
points(1.6,mean(yf[(x<1.7)&(x>1.5)]),pch="*",cex=2)
plot(x,yg,xlab="x",ylab=expression(g(x)+eta),col=colors)
curve(log(x+1),col="grey",add=TRUE)
points(1.6,mean(yg[(x<1.7)&(x>1.5)]),pch="*",cex=2)

```

Figure 4.6: Relationship between smoothing and function roughness. In both the upper and lower panel we are trying to estimate the value of the regression function at $x = 1.6$ from averaging observations taken with $1.5 < x_i < 1.7$ (black points, others are “ghosted” in grey). The location of the average is shown by the large black X . Averaging over this window works poorly for the rough function $f(x)$ in the upper panel (the bias is large), but much better for the smoother function in the lower panel (the bias is small).



```

loc_ave_err <- function(h,y,y0) {abs(y0-mean(y[(1.6-h < x) & (1.6+h>x)]))}

yf0=sin(1.6)*cos(20*1.6)
yg0=log(1+1.6)

f.LAE = sapply(0:100/100,loc_ave_err,y=yf,y0=yf0)
g.LAE = sapply(0:100/100,loc_ave_err,y=yg,y0=yg0)
plot(0:100/100,f.LAE,xlab="Radius of averaging window",
     ylab="Absolute value of error",type="l")
lines(0:100/100,g.LAE,lty=2)
abline(h=0.15,col="grey")

```

Figure 4.7: Estimating $f(1.6)$ and $g(1.6)$ from averaging observed values at $1.6 - h < x < 1.6 + h$, for different radii h . Solid line: error of estimates of $f(1.6)$; dashed line: error of estimates of $g(1.6)$; grey line: σ , the standard deviation of the noise.

respect to the bandwidth, and setting everything to zero (neglecting the o terms):

$$0 = 4b^3 \left[\frac{1}{2} r''(x) + \frac{r'(x)f'(x)}{f(x)} \right]^2 (\sigma_K^2)^2 - \frac{\sigma^2(x)R(K)}{nb^2 f(x)} \quad (4.13)$$

$$b = \left(n \frac{4f(x)(\sigma_K^2)^2 \left[\frac{1}{2} r''(x) + \frac{r'(x)f'(x)}{f(x)} \right]^2}{\sigma^2(x)R(K)} \right)^{-1/5} \quad (4.14)$$

Of course, this expression for the optimal b involves the unknown derivatives $r'(x)$ and $r''(x)$, plus the unknown density $f(x)$ and its unknown derivative $f'(x)$. But if we knew the derivative of the regression function, we would basically know the function itself (just integrate), so we seem to be in a vicious circle, where we need to know the function before we can learn it.⁷

One way of expressing this is to talk about how well a smoothing procedure *would* work, if an Oracle were to tell us the derivatives, or (to cut to the chase) the optimal bandwidth b_{opt} . Since most of us do not have access to such oracles, we need to *estimate* b_{opt} . Once we have this estimate, \hat{b} , then we get out weights and our predictions, and so a certain mean-squared error. Basically, our MSE will be the Oracle's MSE, plus an extra term which depends on how far \hat{b} is to b_{opt} , and how sensitive the smoother is to the choice of bandwidth.

What would be really nice would be an *adaptive* procedure, one where our actual MSE, using \hat{b} , approaches the Oracle's MSE, which it gets from b_{opt} . This would mean that, in effect, we are *figuring out* how rough the underlying regression function is, and so how much smoothing to do, rather than having to guess or be told. An adaptive procedure, if we can find one, is a partial⁸ substitute for prior knowledge.

4.2.1 Bandwidth Selection by Cross-Validation

The most straight-forward way to pick a bandwidth, and one which generally manages to be adaptive, is in fact cross-validation; k -fold CV is usually somewhat better than leave-one-out, but the latter often works acceptably too. The usual procedure is to come up with an initial grid of candidate bandwidths, and then use cross-validation to estimate how well each one of them would generalize. The one with the lowest error under cross-validation is then used to fit the regression curve to the whole data⁹.

Code Example 2 shows how it would work in R, with the values of the input variable being in the vector x (one dimensional) and the response in the vector y

⁷You may be wondering, at this point, why I keep talking about *the* optimal bandwidth, when it would seem, from Eq. 4.14, that the bandwidth should vary with x . One can actually go through pretty much the same sort of analysis in terms of the *expected* values of the derivatives, and the qualitative conclusions will be the same, but the notational overhead is even worse. Alternatively, there are techniques for variable-bandwidth smoothing.

⁸Only partial, because we'd *always* do better if the Oracle would just tell us b_{opt} .

⁹Since the optimal bandwidth is $\propto n^{-1/5}$, and the training sets in cross-validation are smaller than the whole data set, one might adjust the bandwidth proportionally. However, if n is small enough that this makes a big difference, the sheer noise in bandwidth estimation usually overwhelms this.

(also one dimensional), using the `npreg` function from the `np` library (Hayfield and Racine, 2008).¹⁰

The return value has three parts. The first is the actual best bandwidth. The second is a vector which gives the cross-validated mean-squared mean-squared errors of all the different bandwidths in the vector `bandwidths`. The third component is an array which gives the MSE for each bandwidth on each fold. It can be useful to know things like whether the difference between the CV score of the best bandwidth and the runner-up is bigger than their fold-to-fold variability.

Figure 4.8 plots the CV estimate of the (root) mean-squared error versus bandwidth for our two curves. Figure 4.9 shows the data, the actual regression functions and the estimated curves with the CV-selected bandwidths. This illustrates *why* picking the bandwidth by cross-validation works: the curve of CV error against bandwidth is actually a pretty good approximation to the true curve of generalization error against bandwidth (which would look like Figure 4.1), and so optimizing over the CV curve is close to optimizing over the generalization error curve. If we had a problem where cross-validation didn't give good estimates of generalization error, this wouldn't work.

Notice, by the way, in Figure 4.8, that the rougher curve is more sensitive to the choice of bandwidth, and that the smoother curve always has a lower mean-squared error. Also notice that, at the minimum, one of the cross-validation estimates of generalization error is smaller than the true system noise level; this shows that cross-validation doesn't completely correct for optimism¹¹.

We still need to come up with an initial set of candidate bandwidths. For reasons which will drop out of the math in Chapter 15, it's often reasonable to start around $1.06s_X/n^{1/5}$, where s_X is the sample standard deviation of X . However, it is hard to be very precise about this, and good results often require some honest trial and error.

4.2.2 Convergence of Kernel Smoothing and Bandwidth Scaling

Go back to Eq. 4.12 for the mean squared error of kernel regression. As we said, it involves some unknown constants, but we can bury them inside big- O order symbols, which also absorb the little- o remainder terms:

$$MSE(b) = \sigma^2(x) + O(b^4) + O(1/nb) \quad (4.15)$$

The $\sigma^2(x)$ term is going to be there no matter what, so let's look at the excess risk over and above the intrinsic noise:

$$MSE(b) - \sigma^2(x) = O(b^4) + O(1/nb) \quad (4.16)$$

That is, the (squared) bias from the kernel's only approximately getting the curve is proportional to the fourth power of the bandwidth, but the variance is inversely

¹⁰The `np` package actually has a function, `npregbw`, which automatically selects bandwidths through a sophisticated combination of cross-validation and optimization techniques. The default settings for this function make it very slow, by trying very, very hard to optimize the bandwidth.

¹¹Tibshirani and Tibshirani (2009) gives a fairly straightforward way to adjust the estimate of the generalization error for the selected model or bandwidth, but that doesn't influence the choice of the best bandwidth.

```

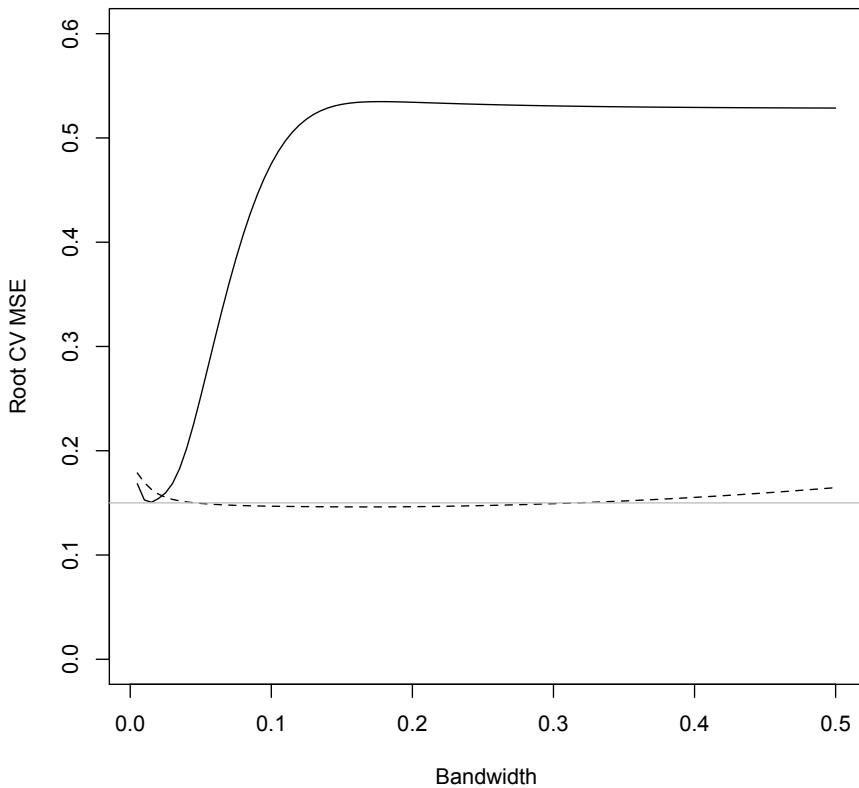
# Cross-validation for univariate kernel regression
cv_bws_npreg <- function(x,y,bandwidths=(1:50)/50,
  num.folds=10) {
  require(np)
  n <- length(x)
  stopifnot(n > 1, length(y) == n)
  stopifnot(length(bandwidths) > 1)
  stopifnot(num.folds > 0, num.folds==trunc(num.folds))

  fold_MSEs <- matrix(0,nrow=num.folds,
    ncol=length(bandwidths))
  colnames(fold_MSEs) = bandwidths

  case.folds <- rep(1:num.folds,length.out=n)
  case.folds <- sample(case.folds)
  for (fold in 1:num.folds) {
    train.rows = which(case.folds==fold)
    x.train = x[train.rows]
    y.train = y[train.rows]
    x.test = x[-train.rows]
    y.test = y[-train.rows]
    for (bw in bandwidths) {
      fit <- npreg(txdat=x.train,tydat=y.train,
        exdat=x.test,eydat=y.test,bws=bw)
      fold_MSEs[fold,paste(bw)] <- fit$MSE
    }
  }
  CV_MSEs = colMeans(fold_MSEs)
  best.bw = bandwidths[which.min(CV_MSEs)]
  return(list(best.bw=best.bw,
    CV_MSEs=CV_MSEs,
    fold_MSEs=fold_MSEs))
}

```

Code Example 2: Comments omitted here to save space; see the accompanying R file on the class website. The `colnames` trick: component names have to be character strings; other data types will be coerced into characters when we assign them to be names. Later, when we want to refer to a bandwidth column by its name, we wrap the name in another coercing function, such as `paste`. — The vector of default bandwidths is arbitrary and only provided for illustration; it should not be blindly copied and used on data (or on homework problems).

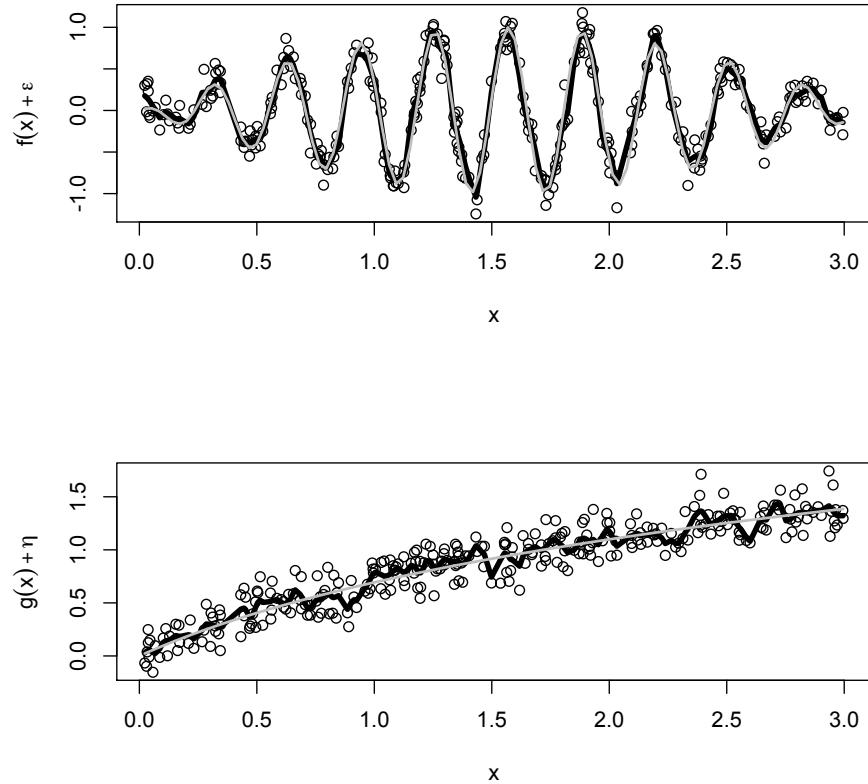


```

fbws <- cv_bws_npreg(x,yf,bandwidths=(1:100)/200
gbws <- cv_bws_npreg(x,yg,bandwidths=(1:100)/200
plot(1:100/200,sqrt(fbws$CV_MSEs),xlab="Bandwidth",
     ylab="Root CV MSE",type="l",ylim=c(0,0.6))
lines(1:100/200,sqrt(gbws$CV_MSEs),lty=2)
abline(h=0.15,col="grey")

```

Figure 4.8: Cross-validated estimate of the (root) mean-squared error as a function of the bandwidth. Solid curve: data from $f(x)$; dashed curve: data from $g(x)$; grey line: true σ . Notice that the rougher curve is more sensitive to the choice of bandwidth, and that the smoother curve is more predictable at every choice of bandwidth. Also notice that CV does not *completely* compensate for the optimism of in-sample fitting (see where the dashed curve falls below the grey line). CV selects bandwidths of 0.015 for f and 0.165 for g .



```

x.ord=order(x)
par(mfcol=c(2,1))
plot(x,yf,xlab="x",ylab=expression(f(x)+epsilon))
fhat <- npreg(bws=fbws$best.bw,txdat=x,tydat=yf)
lines(x[x.ord],fitted(fhat)[x.ord],lwd=4)
curve(sin(x)*cos(20*x),col="grey",add=TRUE,lwd=2)
plot(x,yg,xlab="x",ylab=expression(g(x)+eta))
ghat <- npreg(bws=fbws$best.bw,txdat=x,tydat=yg)
lines(x[x.ord],fitted(ghat)[x.ord],lwd=4)
curve(log(x+1),col="grey",add=TRUE,lwd=2)

```

Figure 4.9: Data from the running examples (circles), true regression functions (grey) and kernel estimates of regression functions with CV-selected bandwidths (black). The widths of the regression functions are exaggerated. Since the x values aren't sorted, we need to put them in order if we want to draw lines connecting the fitted values; then we need to put the fitted values in the same order. An alternative would be to use predict on the sorted values, as in the next section.

proportional to the product of sample size and bandwidth. If we kept h constant and just let $n \rightarrow \infty$, we'd get rid of the variance, but we'd be left with the bias. To get the MSE to go to zero, we need to let the bandwidth h change with n — call it \hat{h}_n . Specifically, suppose $\hat{h}_n \rightarrow 0$ as $n \rightarrow \infty$, but $n\hat{h}_n \rightarrow \infty$. Then, by Eq. 4.16, the risk (generalization error) of kernel smoothing is approaching that of the ideal predictor or regression function.

What is the best bandwidth? We saw in Eq. 4.14 that it is (up to constants)

$$h_{\text{opt}} = O(n^{-1/5}) \quad (4.17)$$

If we put this bandwidth into Eq. 4.16, we get

$$\text{MSE}(h) - \sigma^2(x) = O\left(\left(n^{-1/5}\right)^4\right) + O\left(n^{-1}\left(n^{-1/5}\right)^{-1}\right) = O\left(n^{-4/5}\right) + O\left(n^{-4/5}\right) = O\left(n^{-4/5}\right) \quad (4.18)$$

That is, the excess prediction error of kernel smoothing over and above the system noise goes to zero as $1/n^{0.8}$. Notice, by the way, that the contributions of bias and variance to the generalization error are both of the same order, $n^{-0.8}$.

Is this fast or small? We can compare it to what would happen with a parametric model, say with parameter θ . (Think, for instance, of linear regression, but not only linear regression.) There is an optimal value of the parameter, θ_0 , which would minimize the mean-squared error. At θ_0 , the parametric model has MSE

$$\text{MSE}(\theta_0) = \sigma^2(x) + b(x, \theta_0) \quad (4.19)$$

where b is the bias of the parametric model; this is zero when the parametric model is true¹². Since θ_0 is unknown and must be estimated, one typically has $\hat{\theta} - \theta_0 = O(1/\sqrt{n})$. A first-order Taylor expansion of the parametric model contributes an error $O(\hat{\theta} - \theta_0)$, so altogether

$$\text{MSE}(\hat{\theta}) - \sigma^2(x) = b(x, \theta_0) + O(1/n) \quad (4.20)$$

So parametric models converge more quickly (n^{-1} goes to zero faster than $n^{-0.8}$), but they will typically converge to the wrong answer ($b^2 > 0$). Kernel smoothing converges a bit more slowly, but always converges to the right answer¹³.

How does all this change if h must be found by cross-validation? If we write \widehat{h}_{CV} for the bandwidth picked by cross-validation, the one can show (Simonoff, 1996, ch. 5) that

$$\frac{\widehat{h}_{CV} - h_{\text{opt}}}{h_{\text{opt}}} - 1 = O(n^{-1/10}) \quad (4.21)$$

¹²When the parametric model is not true, the optimal parameter value θ_0 is often called the **pseudo-truth**.

¹³It is natural to wonder if one couldn't do better than kernel smoothing's $O(n^{-4/5})$ while still having no asymptotic bias. Resolving this is very difficult, but the answer turns out to be "no" in the following sense (Wasserman, 2006). Any curve-fitting method which can learn arbitrary smooth regression functions will have some curves where it cannot converge any faster than $O(n^{-4/5})$. (In the jargon, that is the **minimax rate**.) Methods which converge faster than this for some kinds of curves have to converge more slowly for others. So this is the best rate we can hope for on truly unknown curves.

Given this, one concludes (EXERCISE 2) that the MSE of using \widehat{h}_{CV} is also $O(n^{-4/5})$.

4.2.3 Summary on Kernel Smoothing

Suppose that X and Y are both one-dimensional, and the true regression function $r(x) = E[Y|X=x]$ is continuous and has first and second derivatives¹⁴. Suppose that the noise around the true regression function is uncorrelated between different observations. Then the bias of kernel smoothing, when the kernel has bandwidth h , is $O(h^2)$, and the variance, after n samples, is $O(1/nh)$. The optimal bandwidth is $O(n^{-1/5})$, and the excess mean squared error of using this bandwidth is $O(n^{-4/5})$. If the bandwidth is selected by cross-validation, the excess risk is still $O(n^{-4/5})$.

4.3 Kernel Regression with Multiple Inputs

For the most part, when I've been writing out kernel regression I have been treating the input variable x as a scalar. There's no reason to insist on this, however; it could equally well be a vector. If we want to enforce that in the notation, say by writing $\vec{x} = (x^1, x^2, \dots, x^d)$, then the kernel regression of y on \vec{x} would just be

$$\hat{r}(\vec{x}) = \sum_{i=1}^n y_i \frac{K(\vec{x} - \vec{x}_i)}{\sum_{j=1}^n K(\vec{x} - \vec{x}_j)} \quad (4.22)$$

In fact, if we want to predict a vector, we'd just substitute \vec{y}_i for y_i above.

To make this work, we need kernel functions for vectors. For scalars, I said that any probability density function would work so long as it had mean zero, and a finite, strictly positive (not 0 or ∞) variance. The same conditions carry over: any distribution over vectors can be used as a multivariate kernel, provided it has mean zero, and the variance matrix is finite and strictly positive¹⁵. In practice, the overwhelmingly most common and practical choice is to use **product kernels**¹⁶.

A product kernel simply uses a different kernel for each component, and then multiplies them together:

$$K(\vec{x} - \vec{x}_i) = K_1(x^1 - x_i^1)K_2(x^2 - x_i^2)\dots K_d(x^d - x_i^d) \quad (4.23)$$

Now we just need to pick a bandwidth for each kernel, which in general should not be equal — say $\vec{h} = (h_1, h_2, \dots, h_d)$. Instead of having a one-dimensional error curve, as in Figure 4.1 or 4.2, we will have a d -dimensional error surface, but we can still use cross-validation to find the vector of bandwidths that generalizes best. We generally can't, unfortunately, break the problem up into somehow picking the best bandwidth for each variable without considering the others. This makes it slower to select good bandwidths in multivariate problems, but still often feasible.

¹⁴Or can be approximated to arbitrarily closely by such functions.

¹⁵Remember that for a matrix v to be “strictly positive”, it must be the case that for any vector \vec{a} , $\vec{a} \cdot v\vec{a} > 0$. Covariance matrices are automatically non-negative, so we're just ruling out the case of some weird direction along which the distribution has zero variance.

¹⁶People do sometimes use multivariate Gaussians; we'll glance at this in Chapter 14.

(We can actually turn the need to select bandwidths together to our advantage. If one or more of the variables are irrelevant to our prediction given the others, cross-validation will tend to give them the maximum possible bandwidth, and smooth away their influence. In Chapter 15, we'll look at formal tests based on this idea.)

Kernel regression will recover almost any regression function. This is true even when the true regression function involves lots of interactions among the input variables, perhaps in complicated forms that would be very hard to express in linear regression. For instance, Figure 4.10 shows a contour plot of a reasonably complicated regression surface, at least if one were to write it as polynomials in x^1 and x^2 , which would be the usual approach. Figure 4.12 shows the estimate we get with a product of Gaussian kernels and only 1000 noisy data points. It's not perfect, of course (in particular the estimated contours aren't as perfectly smooth and round as the true ones), but the important thing is that we got this without having to know, and describe in Cartesian coordinates, the type of shape we were looking for. Kernel smoothing *discovered* the right general form.

There are limits to these abilities of kernel smoothers; the biggest one is that they require more and more data as the number of predictor variables increases. We will see later (Chapter 9) exactly how much data is required, generalizing the kind of analysis done §4.2.2, and some of the compromises this can force us into.

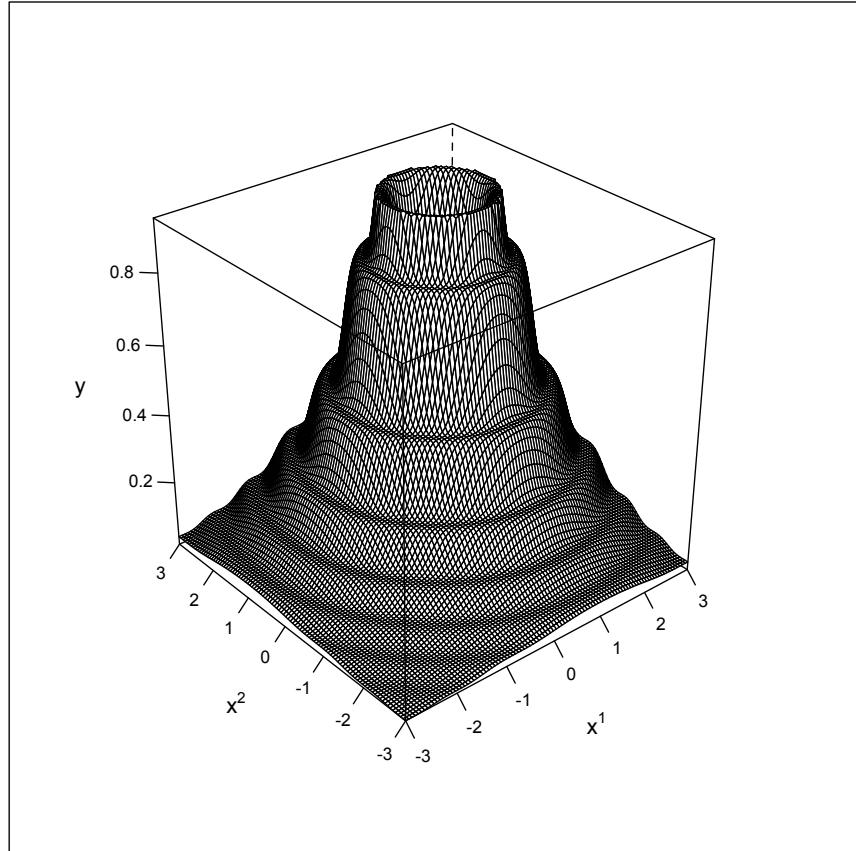
4.4 Interpreting Smoothers: Plots

In a linear regression without interactions, it is fairly easy to interpret the coefficients. The expected response changes by β_i for a one-unit change in the i^{th} input variable. The coefficients are also the derivatives of the expected response with respect to the inputs. And it is easy to draw pictures of how the output changes as the inputs are varied, though the pictures are somewhat boring (straight lines or planes).

As soon as we introduce interactions, all this becomes harder, even for parametric regression. If there is an interaction between two components of the input, say x^1 and x^2 , then we can't talk about the change in the expected response for a one-unit change in x^1 without saying what x^2 is. We might *average* over x^2 values, and we'll see next time a reasonable way of doing this, but the flat statement "increasing x^1 by one unit increases the response by β_1 " is just false, no matter what number we fill in for β_1 . Likewise for derivatives; we'll come back to them next time as well.

What about pictures? If there are only two input variables, then we can make plots like the wireframes in the previous section, or contour- or level- plots, which will show the predictions for different combinations of the two variables. But suppose we want to look at one variable at a time? Suppose there are more than two input variables?

A reasonable way of producing a curve for each input variable is to set all the others to some "typical" value, such as the mean or the median, and to then plot the predicted response as a function of the one remaining variable of interest. See Figure 4.13 for an example of this. Of course, when there are interactions, changing the values of the other inputs will change the response to the input of interest, so it may be a good idea to produce a couple of curves, possibly super-imposed (again, see

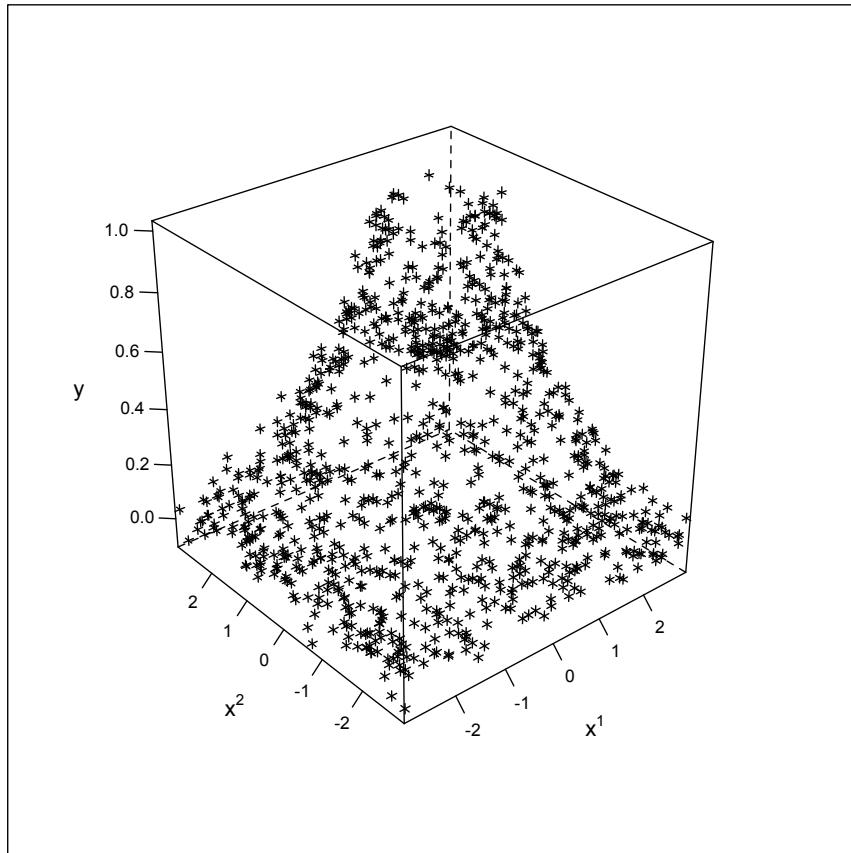


```

x1.points <- seq(-3,3,length.out=100)
x2.points <- x1.points
x12grid <- expand.grid(x1=x1.points,x2=x2.points)
y <- matrix(0,nrow=100,ncol=100)
y <- outer(x1.points,x2.points,f)
library(lattice)
wireframe(y~x12grid$x1*x12grid$x2,scales=list(arrows=FALSE),
          xlab=expression(x^1),ylab=expression(x^2),zlab="y")

```

Figure 4.10: An example of a regression surface that would be very hard to learn by piling together interaction terms in a linear regression framework. (Can you guess what the mystery function f is?) — `wireframe` is from the graphics library `lattice`.

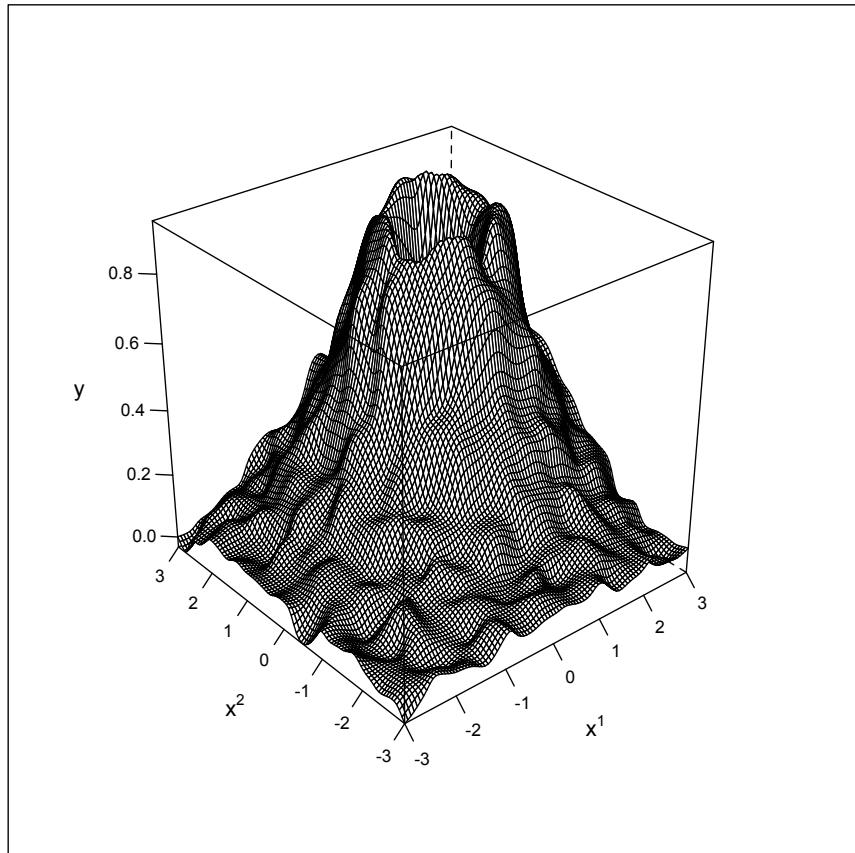


```

x1.noise <- runif(1000,min=-3,max=3)
x2.noise <- runif(1000,min=-3,max=3)
y.noise <- f(x1.noise,x2.noise)+rnorm(1000,0,0.05)
noise <- data.frame(y=y.noise,x1=x1.noise,x2=x2.noise)
cloud(z~x*y,data=noise,col="black",scales=list(arrows=FALSE),
      xlab=expression(x^1),ylab=expression(x^2),zlab="y")

```

Figure 4.11: 1000 data points, randomly sampled from the surface in Figure 4.10, plus independent Gaussian noise (s.d. = 0.05).



```

noise.np <- npreg(y~x1+x2,data=noise)
y.out <- matrix(0,100,100)
y.out <- predict(noise.np,newdata=x12grid)
wireframe(y.out~x12grid$x1*x12grid$x2,scales=list(arrows=FALSE),
          xlab=expression(x^1),ylab=expression(x^2),zlab="y")

```

Figure 4.12: Gaussian kernel regression of the points in Figure 4.11. Notice that the estimated function will make predictions at arbitrary points, not just the places where there was training data.

Figure 4.13).

If there are three or more input variables, we can look at the interactions of any two of them, taken together, by fixing the others and making three-dimensional or contour plots, along the same principles.

The fact that smoothers don't give us a simple story about how each input is associated with the response may seem like a disadvantage compared to using linear regression. Whether it really is a disadvantage depends on whether there really is a simple story to be told — and, if there isn't, how big a lie you are prepared to tell in order to keep your story simple.

4.5 Average Predictive Comparisons

Suppose we have a linear regression model

$$Y = \beta_1 X^1 + \beta_2 X^2 + \epsilon \quad (4.24)$$

and we want to know how much Y changes, on average, for a one-unit increase in X^1 . The answer, as you know very well, is just β_1 :

$$[\beta_1(X^1 + 1) + \beta_2 X^2] - [\beta_1 X^1 + \beta_2 X^2] = \beta_1 \quad (4.25)$$

This is an interpretation of the regression coefficients which you are very used to giving. But it fails as soon as we have interactions:

$$Y = \beta_1 X^1 + \beta_2 X^2 + \beta_3 X^1 X^2 + \epsilon \quad (4.26)$$

Now the effect of increasing X^1 by 1 is

$$[\beta_1(X^1 + 1) + \beta_2 X^2 + \beta_3(X^1 + 1)X^2] - [\beta_1 X^1 + \beta_2 X^2 + \beta_3 X^1 X^2] = \beta_1 + \beta_3 X^2 \quad (4.27)$$

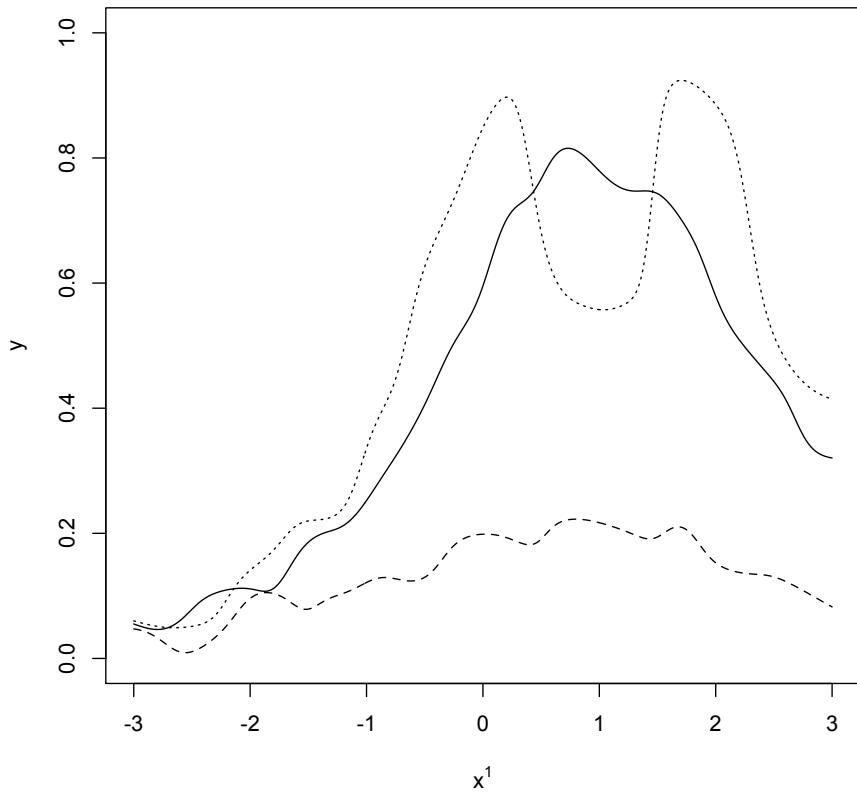
There just isn't one answer “how much does the response change when X^1 is increased by one unit?”, it depends on the value of X^2 . We certainly can't just answer “ β_1 ”.

We also can't give just a single answer if there are nonlinearities. Suppose that the true regression function is this:

$$Y = \frac{e^{\beta X}}{1 + e^{\beta X}} + \epsilon \quad (4.28)$$

which looks like Figure 4.14, setting $\beta = 7$ (for luck). Moving x from -4 to -3 increases the response by 7.57×10^{-10} , but the increase in the response from $x = -1$ to $x = 0$ is 0.499. Functions like this are very common in psychology, medicine (dose-response curves for drugs), biology, etc., and yet we cannot sensibly talk about *the* response to a one-unit increase in x . (We will come back to curves which look like this in Chapter 12.)

More generally, let's say we are regressing Y on a vector \vec{X} , and want to assess the impact of one component of the input on Y . To keep the use of subscripts and

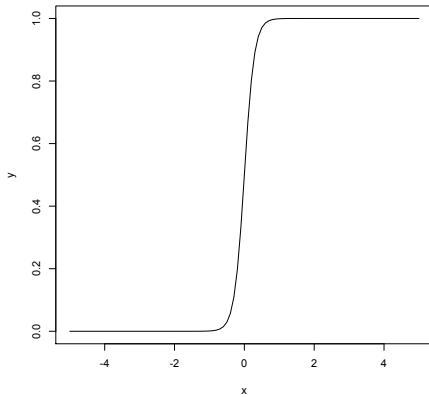


```

new.frame <- data.frame(x=seq(-3,3,length.out=300),y=median(y.noise))
plot(new.frame$x,predict(noise.np,newdata=new.frame),
     type="l",xlab=expression(x^1),ylab="y",ylim=c(0,1.0))
new.frame$y <- quantile(y.noise,0.25)
lines(new.frame$x,predict(noise.np,newdata=new.frame),lty=2)
new.frame$y <- quantile(y.noise,0.75)
lines(new.frame$x,predict(noise.np,newdata=new.frame),lty=3)

```

Figure 4.13: Predicted mean response as function of the first input coordinate x^1 for the example data, evaluated with the second coordinate x^2 set to the median (solid), its 25th percentile (dashed) and its 75th percentile (dotted). Note that the changing shape of the partial response curve indicates an interaction between the two inputs. Also, note that the model is able to make predictions at arbitrary coordinates, whether or not there were any training points there. (It happened that no observation was exactly at the median, the 25th or the 75th percentile for the second input.)



```
curve(exp(7*x)/(1+exp(7*x)), from=-5, to=5, ylab="y")
```

Figure 4.14: The function of Eq. 4.28, with $\beta = 7$.

superscripts to a minimum, we'll write $\vec{X} = (u, \vec{V})$, where u is the coordinate we're really interested in. (It doesn't have to come first, of course.) We would like to know how much the prediction changes as we change u ,

$$EY|\vec{X} = (u^{(2)}, \vec{v}) - EY|\vec{X} = (u^{(1)}, \vec{v}) \quad (4.29)$$

and the change in the response per unit change in u ,

$$\frac{EY|\vec{X} = (u^{(2)}, \vec{v}) - EY|\vec{X} = (u^{(1)}, \vec{v})}{u^{(2)} - u^{(1)}} \quad (4.30)$$

Both of these, but especially the latter, are called the **predictive comparison**. Note that both of them, as written, depend on $u^{(1)}$ (the starting value for the variable of interest), on $u^{(2)}$ (the ending value), and on \vec{v} (the other variables, held fixed during this comparison). We have just seen that in a linear model without interactions, $u^{(1)}$, $u^{(2)}$ and \vec{v} all go away and leave us with the regression coefficient on u . In nonlinear or interacting models, we can't simplify so much.

Once we have estimated a regression model, we can chose our starting point, ending point and context, and just plug in to Eq. 4.29 or Eq. 4.30. (Take a look again at problem 6 on Homework 2.) But suppose we do want to boil this down into a single number for each input variable — how might we go about this?

One good answer, which comes from Gelman and Pardoe (2007), is just to average 4.30 over the data¹⁷ More specifically, we have as our **average predictive comparison**

¹⁷Actually, they propose something very slightly more complicated, which takes into account the uncertainty in our estimate of the regression function. We'll come back to this in a few lectures when we see how to quantify uncertainty in complex models.

for u

$$\frac{\sum_{i=1}^n \sum_{j=1}^n \hat{r}(u_j, \vec{v}_i) - \hat{r}(u_i, \vec{v}_j) \text{sign}(u_j - u_i)}{(u_j - u_i) \text{sign}(u_j - u_i)} \quad (4.31)$$

where i and j run over data points, \hat{r} is our estimated regression function, and the sign function is defined by $\text{sign}(x) = +1$ if $x > 0$, $= 0$ if $x = 0$, and $= -1$ if $x < 0$. We use the sign function this way to make sure we are always looking at the consequences of *increasing* u .

The average predictive comparison provides a reasonable summary measure of how one should expect the response to vary as u changes slightly. But, once the model is nonlinear or allows interactions, it's just not possible to summarize the predictive relationship between u and y with a single number, and so the value of Eq. 4.31 is going to depend on the distribution of u (and possibly v), even when the regression function is unchanged. (See Exercise 3.)

4.6 Exercises

- Suppose we use a uniform (“boxcar”) kernel extending over the region $(-h/2, h/2)$. Show that

$$\mathbb{E}[\hat{r}(0)] = \mathbb{E}\left[r(X) \mid -\frac{h}{2} < X < \frac{h}{2}\right] \quad (4.32)$$

$$\begin{aligned} &= r(0) + r'(0)\mathbb{E}\left[X \mid -\frac{h}{2} < X < \frac{h}{2}\right] \\ &\quad + \frac{r''(0)}{2}\mathbb{E}\left[X^2 \mid -\frac{h}{2} < X < \frac{h}{2}\right] + o(h^2) \end{aligned} \quad (4.33)$$

Show that $\mathbb{E}\left[X \mid -\frac{h}{2} < X < \frac{h}{2}\right] = O(r'(0)f'(0)h^2)$, and that $\mathbb{E}\left[X^2 \mid -\frac{h}{2} < X < \frac{h}{2}\right] = O(h^2)$. Conclude that the overall bias is $O(h^2)$.

- Use Eqs. 4.21, 4.17 and 4.16 to show that the excess risk of the kernel smoothing, when the bandwidth is selected by cross-validation, is also $O(n^{-4/5})$.
- Generate 1000 data points where X is uniformly distributed between -4 and 4 , and $Y = e^{7x}/(1 + e^{7x}) + \epsilon$, with ϵ Gaussian and with variance 0.01 . Use non-parametric regression to estimate $\hat{r}(x)$, and then use Eq. 4.31 to find the average predictive comparison. Now re-run the simulation with X uniform on the interval $[0, 0.5]$ and re-calculate the average predictive comparison. What happened?

Chapter 5

Simulation

You will recall from your previous statistics courses that quantifying uncertainty in statistical inference requires us to get at the **sampling distributions** of things like estimators. When the very strong simplifying assumptions of basic statistics courses do not apply¹, or when estimators are themselves complex objects, like kernel regression curves or even histograms, there is little hope of being able to write down sampling distributions in closed form. We get around this by using simulation to approximate the sampling distributions we can't calculate.

5.1 What Do We Mean by “Simulation”?

A stochastic model is a mathematical story about how the data could have been generated. Simulating the model means implementing it, step by step, in order to produce something which should look like the data — what's sometimes called **synthetic data**, or **surrogate data**, or a **realization** of the model. In a stochastic model, some of the steps we need to follow involve a random component, and so multiple simulations starting from exactly the same initial conditions will not give exactly the same outputs or realizations. Rather, will be a distribution over the realizations. Doing large numbers of simulations gives us a good estimate of this distribution.

For a trivial example, consider a model with three random variables, $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$, $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, with $X_1 \perp\!\!\!\perp X_2$, and $X_3 = X_1 + X_2$. Simulating from this model means drawing a random value from the first normal distribution for X_1 , drawing a second random value for X_2 , and adding them together to get X_3 . The marginal distribution of X_3 , and the joint distribution of (X_1, X_2, X_3) , are implicit in this specification of the model, and we can find them by running the simulation.

In this particular case, we could also find the distribution of X_3 , and the joint distribution, by probability calculations of the kind you learned how to do in your basic probability courses. For instance, X_3 is $\mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. These analytical

¹In 36-401, you will have seen results about the sampling distribution of linear regression coefficients when the linear model is true, and the noise is Gaussian with constant variance. As an exercise, try to get parallel results when the noise has a t distribution with 10 degrees of freedom.

probability calculations can usually be thought of as just short-cuts for exhaustive simulations.

5.2 How Do We Simulate Stochastic Models?

5.2.1 Chaining Together Random Variables

Stochastic models are usually specified by sets of conditional distributions for one random variable, given some other variable or variables. For instance, a simple linear regression model might have the specification

$$X \sim \mathcal{N}(\mu_x, \sigma_1^2) \quad (5.1)$$

$$Y|X \sim \mathcal{N}(\beta_0 + \beta_1 X, \sigma_2^2) \quad (5.2)$$

If we knew how to generate a random variable from the distributions given on the right-hand sides, we could simulate the whole model by chaining together draws from those conditional distributions. This is in fact the general strategy for simulating any sort of stochastic model, by chaining together random variables.²

What this means is that we can reduce the problem of simulating to that of generating random variables.

5.2.2 Random Variable Generation

5.2.2.1 Built-in Random Number Generators

R provides random number generators for most of the most common distributions. By convention, the names of these functions all begin with the letter “r”, followed by the abbreviation of the functions, and the first argument is always the number of draws to make, followed by the parameters of the distribution:

```
rnorm(n, mean=0, sd=1) # Gaussian
runif(n, min=0, max=1) # Uniform
rexp(n, rate=1)         # Exponential, rate is 1/mean
rpois(n, lambda)       # Poisson, lambda is mean
rbinom(n, size, prob)  # Binomial
```

etc., etc. A further convention is that these parameters can be *vectorized*. Rather than giving a single mean and standard deviation (say) for multiple draws from the Gaussian distribution, each draw can have its own:

```
rnorm(10, mean=1:10, sd=1/sqrt(1:10))
```

That instance is rather trivial, but the exact same principle would be at work here:

²In this case, we could in principle first generate Y , and then draw from $Y|X$, but have fun finding those distributions. Especially have fun if, say, X has a t distribution with 5 degrees of freedom — a very small change to the specification.

```
rnorm(nrow(x), mean=predict(regression.model, newdata=x),
      sd=predict(volatility.model, newdata=x))
```

where `regression.model` and `volatility.model` are previously-defined parts of the model which tell us about conditional expectations and conditional variances.

Of course, none of this explains how R actually draws from any of these distributions; it's all at the level of a black box, which is to say black magic. Because ignorance is evil, and, even worse, *unhelpful* when we need to go beyond the standard distributions, it's worth open the black box at least a little.

5.2.2.2 Transformations

If we can generate a random variable Z with some distribution, and $V = g(Z)$, then we can generate V . So one thing which gets a lot of attention is writing random variables as transformations of one another — ideally as transformations of easy-to-generate variables.

Example. Suppose we can generate random numbers from the standard Gaussian distribution $Z \sim \mathcal{N}(0, 1)$. Then we can generate from $\mathcal{N}(\mu, \sigma^2)$ as $\sigma Z + \mu$. We can generate χ^2 random variables with 1 degree of freedom as Z^2 . We can generate χ^2 random variables with d degrees of freedom by summing d independent copies of Z^2 .

In particular, if we can generate random numbers uniformly distributed between 0 and 1, we can use this to generate anything which is a transformation of a uniform distribution. How far does that extend?

5.2.2.3 Quantile Method

Suppose that we know the **quantile function** Q_Z for the random variable X we want, so that $Q_Z(0.5)$ is the median of X , $Q_Z(0.9)$ is the 90th percentile, and in general $Q_Z(p)$ is bigger than or equal to X with probability p . Q_Z comes as a pair with the cumulative distribution function F_Z , since

$$Q_Z(F_Z(a)) = a, F_Z(Q_Z(p)) = p \quad (5.3)$$

In the **quantile method** (or **inverse distribution transform method**), we generate a uniform random number U and feed it as the argument to Q_Z . Now $Q_Z(U)$ has the distribution function F_Z :

$$\Pr(Q_Z(U) \leq a) = \Pr(F_Z(Q_Z(U)) \leq F_Z(a)) \quad (5.4)$$

$$= \Pr(U \leq F_Z(a)) \quad (5.5)$$

$$= F_Z(a) \quad (5.6)$$

where the last line uses the fact that U is uniform on $[0, 1]$, and the first line uses the fact that F_Z is a non-decreasing function, so $b \leq a$ is true if and only if $F_Z(b) \leq F_Z(a)$.

Example. The CDF of the exponential distribution with rate λ is $1 - e^{-\lambda x}$. The quantile function $Q(p)$ is thus $-\frac{\log(1-p)}{\lambda}$. (Notice that this is positive, because $1 - p < 1$ and so $\log(1 - p) < 0$, and that it has units of $1/\lambda$, which are the units of x , as it

should.) Therefore, if $U \sim \text{Unif}(0, 1)$, then $-\frac{\log(1-U)}{\lambda} \sim \text{Exp}(\lambda)$. This is the method used by `rexp()`.

Example. The **Pareto distribution** or **power law** is a two-parameter family, $f(x; \alpha, x_0) = \frac{\alpha-1}{x_0} \left(\frac{x}{x_0}\right)^{-\alpha}$ if $x \geq x_0$, with density 0 otherwise. Integration shows that the cumulative distribution function is $F(x; \alpha, x_0) = 1 - \left(\frac{x}{x_0}\right)^{-\alpha+1}$. The quantile function therefore is $Q(p; \alpha, x_0) = x_0(1-p)^{-\frac{1}{\alpha-1}}$. (Notice that this has the same units as x , as it should.)

Example. The standard Gaussian $\mathcal{N}(0, 1)$ does not have a closed form for its quantile function, but there are fast and accurate ways of calculating it numerically (they're what stand behind `qnorm`), so the quantile method can be used. In practice, there are other transformation methods which are even faster, but rely on special tricks.

Since $Q_Z(U)$ has the same distribution function as Z , we can use the quantile method, as long as we can calculate Q_Z . Since Q_Z always exists, in principle this solves the problem. In practice, we need to calculate Q_Z before we can use it, and this may not have a closed form, and numerical approximations may be intractable.³

5.2.2.4 Rejection Method

Another general approach, which avoids needing the quantile function, is the **rejection method**. Suppose that we want to generate Z , with probability density function f_Z , and we have a method to generate R , with p.d.f. ρ , called the **proposal distribution**. Also suppose that $f_Z(x) \leq \rho(x)M$, for some constant $M > 1$. For instance, if f_Z has a limited range $[a, b]$, we could take ρ to be the uniform distribution on $[a, b]$, and M the maximum density of f_Z .

The rejection method algorithm then goes as follows.

1. Generate a proposal R from ρ .
2. Generate a uniform U , independently of R .
3. Is $MU\rho(R) < f_Z(R)$?
 - If yes, “accept the proposal” by returning R and stopping.
 - If no, “reject the proposal”, discard R and U , and go back to (1)

If ρ is uniform, this just amounts to checking whether $MU < f_Z(R)$, with M the maximum density of Z .

Computationally, the idea looks like Example 3.

One way to understand the rejection method is as follows. Imagine drawing the curve of $f_Z(x)$. The total area under this curve is 1, because $\int dx f_Z(x) = 1$. The area between any two points a and b on the horizontal axis is $\int_a^b dx f_Z(x) = F_Z(b) - F_Z(a)$. It follows that if we could uniformly sample points from the area between the curve and the horizontal axis, their x coordinates would have exactly the distribution

³In essence, we have to solve the nonlinear equation $F_Z(x) = p$ for x over and over — and that assumes we can easily calculate F_Z .

```
rrejection.1 <- function(dttarget,dproposal,rproposal,M) {
  rejected <- TRUE
  while(rejected) {
    R <- rproposal(1)
    U <- runif(1)
    rejected <- (M*U*dproposal(R) < dttarget(R))
  }
  return(R)
}

rrejection <- function(n,dttarget,dproposal,rproposal,M) {
  replicate(n,rrejection.1(dttarget,dproposal,rproposal,M))
}
```

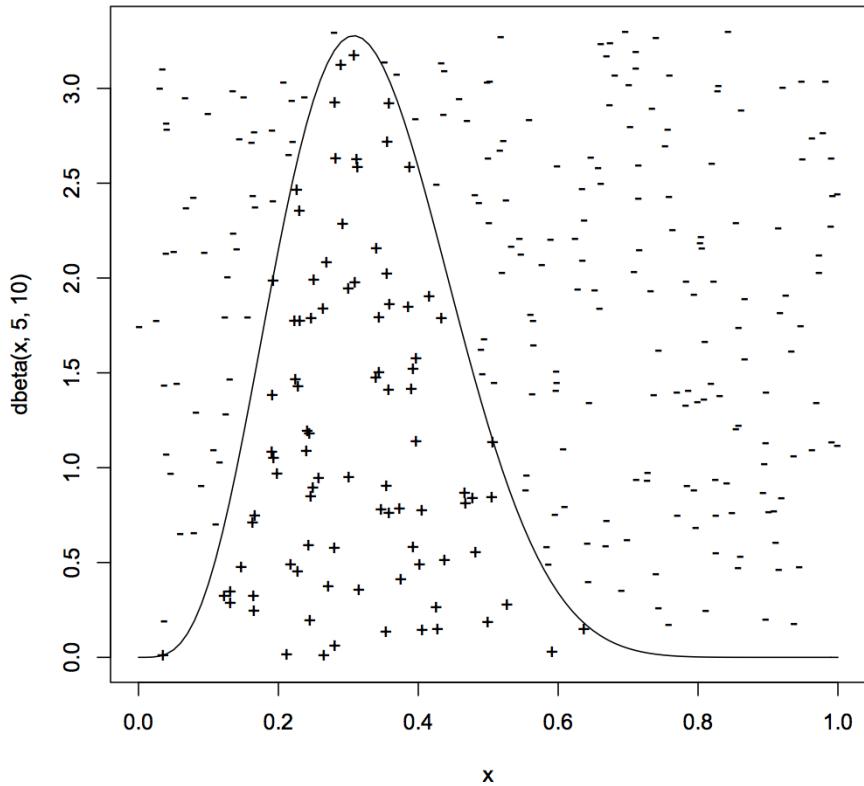
Code Example 3: An example of how the rejection method would be used. The arguments `dttarget`, `dproposal` and `rproposal` would all be functions. This is not quite industrial-strength code, because it does not let us pass arguments to those functions flexibly. See online code for comments.

function we are looking for. If ρ is a uniform distribution, then we are drawing a rectangle which just encloses the curve of f_Z , sampling points uniformly from the rectangle (with x coordinates R and y coordinates MU), and only keeping the ones which fall under the curve. When ρ is not uniform, but we can sample from it nonetheless, then we are uniformly sampling from the area under $M\rho$, and keeping only the points which are also below f_Z .

Example. The beta distribution, $f(x;a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}$, is defined on the unit interval⁴. While its quantile function can be calculated and so we could use the quantile method, we could also use the reject method, taking the uniform distribution for the proposals. Figure 5.1 illustrates how it would go for the Beta(5,10) distribution

The rejection method's main drawback is speed. The probability of accepting on any given pass through the algorithm is $1/M$. (EXERCISE: Why?) Thus produce n random variables from it takes, on average, nM cycles. (EXERCISE: Why?) Clearly, we want M to be as small, which means that we want the proposal distribution ρ to be close to the target distribution f_Z . Of course if we're using the rejection method because it's hard to draw from the target distribution, and the proposal distribution is close to the target distribution, it may be hard to draw from the proposal.

⁴Here $\Gamma(a) = \int_0^\infty dx e^{-x} x^{a-1}$. It is not obvious, but for integer a , $\Gamma(a) = (a-1)!$. The distribution gets its name because $\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$ is called the **beta function** of a and b , a kind of continuous generalization of $\binom{a+b}{a}$. The beta distribution arises in connection with problems about minima and maxima, and inference for binomial distributions.



```

M <- 3.3
curve(dbeta(x,5,10),from=0,to=1,ylim=c(0,M))
r <- runif(300,min=0,max=1)
u <- runif(300,min=0,max=1)
below <- which(M*u*runif(r,min=0,max=1) <= dbeta(r,5,10))
points(r[below],M*u[below],pch="+")
points(r[-below],M*u[-below],pch="-")

```

Figure 5.1: Illustration of the rejection method for generating random numbers from a Beta(5,10) distribution. The proposal distribution is uniform on the range of the beta, which is $[0, 1]$. Points are thus sampled uniformly from the rectangle which runs over $[0, 1]$ on the horizontal axis and $[0, 3.3]$ on the vertical axis, i.e., $M = 3.3$, because the density of the Beta is < 3.3 everywhere. (This is not the lowest possible M but it is close.) Proposed points which fall below the Beta's pdf are marked + and are accepted; those above the pdf curve are marked - and are rejected. In this case, exactly 70% of proposals are rejected.

5.2.2.5 The Metropolis Algorithm and Markov Chain Monte Carlo

One very important, but tricky, way of getting past the limitations of the rejection method is what's called the **Metropolis algorithm**. Once again, we have a density f_Z from which we wish to sample. Once again, we introduce a distribution for "proposals", and accept or reject proposals depending on the density f_Z . The twist now is that instead of making independent proposals each time, the next proposal depends on the last accepted value — the proposal distribution is a conditional pdf $\rho(r|z)$.

Assume for simplicity that $\rho(r|z) = rho(z|r)$. (For instance, we could have a Gaussian proposal distribution centered on z .) Then the Metropolis algorithm goes as follows.

1. Start with value Z_0 (fixed or random).
2. Generate R from the conditional distribution $\rho(\cdot|Z_t)$.
3. Generate a uniform U , independent of R .
4. Is $U \leq f_Z(R)/f_Z(Z_t)$?
 - If yes, set $Z_{t+1} = R$ and go to (2)
 - If not, set $Z_{t+1} = Z_t$ and go to (2)

Mostly simply, the algorithm is run until $t = n$, at which point it returns Z_1, Z_2, \dots, Z_n . In practice, better results are obtained if it's run for $n + n_0$ steps, and the first n_0 values of Z are discarded — this is called "burn-in".

Notice that if $f_Z(R) > f_Z(Z_t)$, then R is always accepted. The algorithm always accepts proposals which move it towards places where the density is higher than where it currently is. If $f_Z(R) < f_Z(Z_t)$, then the algorithm accepts the move with some probability, which shrinks as the density at R gets lower. It should not be hard to persuade yourself that the algorithm will spend more time in places where f_Z is high.

It's possible to say a bit more. Successive values of Z_t are dependent on each other, but $Z_{t+1} \perp\!\!\!\perp Z_{t-1} | Z_t$ — this is a Markov process. The target distribution f_Z is in fact exactly the stationary distribution of the Markov process. If the proposal distributions have broad enough support that the algorithm can get from any z to any z' in a finite number of steps, then the process will "mix". (In fact we only need to be able to visit points where $f_Z > 0$.) This means that if we start with an arbitrary distribution for Z_0 , the distribution of Z_t approaches f_Z and stays there — the point of burn-in is to give this convergence time to happen. The fraction of time Z_t is close to x is in fact proportional to $f_Z(x)$, so we can use the output of the algorithm as, approximately, so many draws from that distribution.⁵

It would seem that the Metropolis algorithm should be superior to the rejection method, since to produce n random values we need only n steps, or $n + n_0$ to handle burn-in, not nM steps. However, this is deceptive, because if the proposal distribution is not well-chosen, the algorithm ends up staying stuck in the same spot for, perhaps, a very long time. Suppose, for instance, that the distribution is bimodal. If

⁵And if the dependence between Z_t and Z_{t+1} bothers us, we can always randomly permute them, once we have them.

Z_0 starts out in between the modes, it's easy for it to move rapidly to one peak or the other, and spend a lot of time there. But to go from one mode to the other, the algorithm has to make a series of moves, all in the same direction, which all reduce f_Z , which happens but is unlikely. It thus takes a very long time to explore the whole distribution. The "best" optimal proposal distribution is make $\rho(r|z) = f_Z(r)$, i.e., to just sample from the target distribution. If we could do that, of course, we wouldn't need the Metropolis algorithm, but trying to make ρ close to f_Z is generally a good idea.

The original **Metropolis algorithm** was invented in the 1950s to facilitate designing the hydrogen bomb. It relies on the assumption that the proposal distribution is symmetric, $\rho(r|z) = \rho(z|r)$. It is sometimes convenient to allow an asymmetric proposal distribution, in which case one accepts R if $U \frac{\rho(R|Z_t)}{\rho(Z_t|R)} \leq \frac{f_Z(R)}{f_Z(Z_t)}$. This is called **Metropolis-Hastings**. Both are examples of the broader class of **Markov Chain Monte Carlo** algorithms, where we give up on getting independent samples from the target distribution, and instead make the target the invariant distribution of a Markov process.

5.2.2.6 Generating Uniform Random Numbers

Everything previously to this rested on being able to generate uniform random numbers, so how do we do that? Well, really that's a problem for computer scientists... But it's good to understand a little bit about the basic ideas.⁶

First of all, the numbers we get will be produced by some deterministic algorithm, and so will be merely **pseudorandom** rather than truly random. But we would like the deterministic algorithm to produce extremely convoluted results, so that its output *looks* random in as many ways that we can test as possible. Dependencies should be complicated, and correlations between easily-calculated functions of successive pseudorandom numbers should be small and decay quickly. (In fact, "truly random" can be defined, more or less, as the limit of the algorithm becoming infinitely complicated.) Typically, pseudorandom number generators are constructed to produce a sequence of uniform values, starting with an initial value, called the **seed**. In normal operation, the seed is set from the computer's clock; when debugging, the seed can be held fixed, to ensure that results can be reproduced exactly.

Probably the simplest example is **incommensurable rotations**. Imagine a watch which fails very slightly, but deterministically, to keep proper time, so that its second hand advances $\phi \neq 1$ seconds in every real second of time. The position of the watch after t seconds is

$$\theta_t = \theta_0 + t\alpha \bmod 60 \quad (5.7)$$

If ϕ is **commensurable** with 60, meaning $\alpha/60 = k/m$ for some integers k, m , then the positions would just repeat every $60k$ seconds. If α is **incommensurable**, because it is an irrational number, then θ_t never repeats. In this case, not only does θ_t never repeat, but it is uniformly distributed between 0 and 60, in the sense that the fraction of time it spends in any sub-interval is just proportional to the length of the interval. (EXERCISE: Why?)

⁶This section is optional for Spring 2013.

You *could* use this as a pseudo-random number generator, with θ_0 as the seed, but it would not be a very good one, for two reasons. First, exactly representing an irrational number α on a digital computer is impossible, so at best you could use a rational number such that the period $60k$ is large. Second, and more pointedly, the successive θ_t are really too close to each other, and too similar. Even if we only took, say, every 50th value, they'd still be quite correlated with each other.

One way this has been improved is to use *multiple* incommensurable rotations. Say we have a second inaccurate watch, $\phi_t = \phi_0 + \beta t \bmod 60$, where β is incommensurable with both 60 and with α . We record θ_t when ϕ_t is within some small window of 0.⁷

Another approach is to use more aggressively complicated deterministic mappings. Take the system

$$\begin{aligned}\theta_{t+1} &= \theta_t + \phi_t \bmod 1 \\ \phi_{t+1} &= \theta_t + 2\phi_t \bmod 1\end{aligned}\tag{5.8}$$

This is known as “Arnold’s cat map”, after the great Soviet mathematician V. I. Arnold, and Figure 5.2. We can think of this as the second-hand θ_t advancing not by a fixed amount α every second, but by a varying amount ϕ_t . The variable ϕ_t , meanwhile, advances by the amount $\phi_t + \theta_t$. The effect of this is that if we look at only one of the two coordinates, say θ_t , we get a sequence of numbers which, while deterministic, is uniformly distributed, and very hard to predict (Figure 5.3).

5.2.3 Sampling

A complement to drawing from given distributions is to **sample** from a given collection of objects. This is such a common task that R has a handy built-in function to do it:

```
sample(x, size, replace=FALSE, prob=NULL)
```

Here `x` is a vector which defines the set of objects we’re going to sample from. `size` is the number of samples we want to draw from `x`. `replace` says whether the samples are drawn with or without replacement. (If `replace=TRUE`, then `size` can be arbitrarily larger than the length of `x`. If `replace=FALSE`, having a larger `size` doesn’t make sense.) Finally, the optional argument `prob` allows for *weighted* sampling; ideally, `prob` is a vector of probabilities as long as `x`, giving the probability of drawing each element of `x`⁸.

As a convenience for a common situation, running `sample` with one argument produces a random permutation of the input, i.e.,

```
sample(x)
```

is equivalent to

⁷The core idea here actually dates back to a medieval astronomer named Nicholas Oresme in the 1300s, as part of an argument that the universe would not repeat exactly (von Plato, 1994, pp. 279–284).

⁸If the elements of `prob` do not add up to 1, but are positive, they will be normalized by their sum, e.g., setting `prob=c(9,9,1)` will assign probabilities $(\frac{9}{19}, \frac{9}{19}, \frac{1}{19})$ to the three elements of `x`.

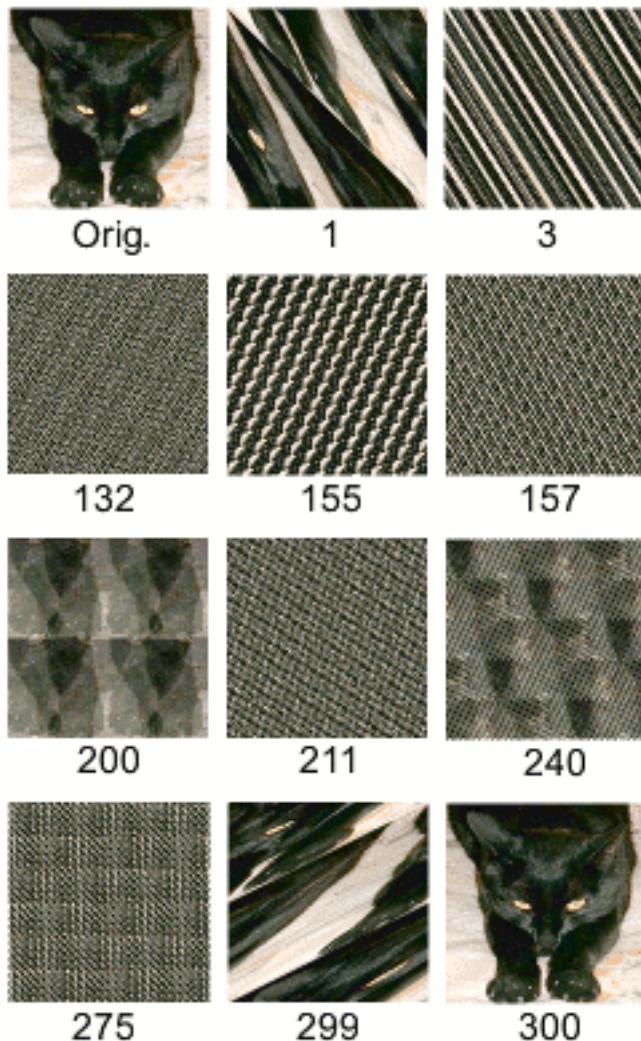


Figure 5.2: Effect of the Arnold cat map. The original image is 300×300 , and mapped into the unit square. The cat map is then applied to the coordinates of each pixel separately, giving a new pixel which inherits the old color. (This can most easily seen in the transition from the original to time 1.) The original image re-assembles itself at time 300 because all the original coordinates were multiples of $1/300$. If we had sampled every, say, 32 time-steps, it would have taken much longer to see a repetition. In the meanwhile, following the x coordinate of a single pixel from the original image would provide a very creditable sequence of pseudo-random values. (Figure from Wikipedia, s.v. “Arnold’s cat map”. See also <http://math.gmu.edu/~sander/movies/arnold.html>.)

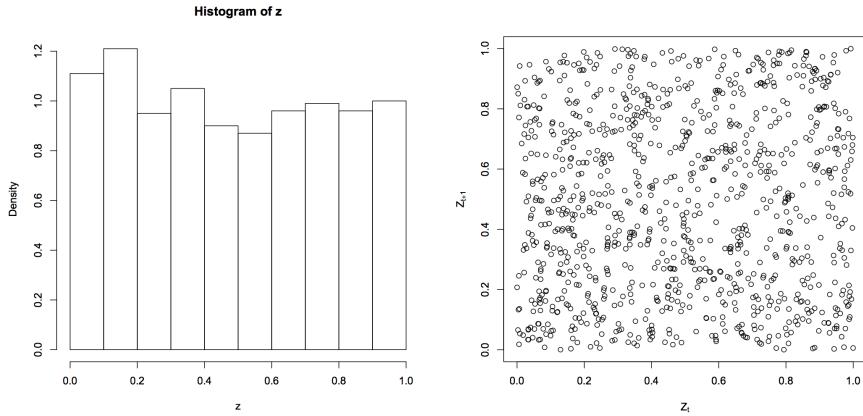
```

arnold.map <- function(v) {
  theta <- v[1]
  phi <- v[2]
  theta.new <- (theta+phi)%%1
  phi.new <- (theta+2*phi)%%1
  return(c(theta.new,phi.new))
}

rarnold <- function(n,seed) {
  z <- vector(length=n)
  for (i in 1:n) {
    seed <- arnold.map(seed)
    z[i] <- seed[1]
  }
  return(z)
}

```

Code Example 4: A function implementing the Arnold cat map (Eq. 5.9), and a second function which uses it as a pseudo-random number generator. See online version for comments.



```

par(mfrow=c(2,1))
z <- rarnold(1000,c(0.11124,0.42111))
hist(z,probability=TRUE)
plot(z[-1000],z[-1],xlab=expression(Z[t]),ylab=expression(Z[t+1]))

```

Figure 5.3: Left: histogram from 1000 samples of the θ coordinate of the Arnold cat map, started from $(0.11124, 0.42111)$. Right: scatter-plot of successive values from the sample, showing that the dependence is very subtle.

```
sample(x, size=length(x), replace=FALSE)
```

For example, if we're doing five-fold cross-validation, then

```
sample(rep(1:5, length.out=nrow(df))
```

will first repeat the numbers 1,2,3,4,5 until we have one number for each row of df, and then shuffle the order of those numbers randomly. This then would give an assignment of each row of df to one (and only one) of five folds.

5.2.3.1 Sampling Rows from Data Frames

When we have multivariate data (which is the usual situation), we typically arrange it into a data-frame, where each row records one unit of observation, with multiple interdependent columns. The natural notion of sampling is then to draw a random sample of the data points, which in that representation amounts to a random sample of the rows. We can implement this simply by sampling row *numbers*. For instance, this command,

```
df[sample(1:nrow(df), size=b), ]
```

will create a new data frame from b, by selecting b rows from df without replacement. It is an easy exercise to figure out how to sample from a data frame *with* replacement, and with unequal probabilities per row.

5.2.3.2 Multinomials and Multinoullis

If we want to draw one value from a multinomial distribution with probabilities $p = (p_1, p_2, \dots, p_k)$, then we can use sample:

```
sample(1:k, size=1, prob=p)
```

If we want to simulate a “multinoulli” process⁹, i.e., a sequence of independent and identically distributed multinomial random variables, then we can easily do so:

```
rmultinoulli <- function(n, prob) {
  k <- length(prob)
  return(sample(1:k, size=n, replace=TRUE, prob=prob))
}
```

5.2.3.3 Probabilities of Observation

Often, our models of how the data are generated will break up into two parts. One part is a model of how actual variables are related to each other out in the world. (E.g., we might model how education and racial categories are related to occupation, and occupation is related to income.) The other part is a model of how variables come to be recorded in our data, and the distortions they might undergo in the course of doing so. (E.g., we might model the probability that someone appears in a survey

⁹A handy term I learned from Gustavo Lacerda.

as a function of race and income.) Plausible sampling mechanisms often make the probability of appearing in the data a function of some of the variables. This can then have important consequences for our inferences from the data we happen to see to the whole population or process.

```
income <- rnorm(n,mean=predict(income.model,x),sd=sigma)
capture.probabilities <- predict(observation.model,x)
observed.income <- sample(income,size=b,prob=capture.probabilities)
```

5.2.4 Repeating Simulations

Because simulations are often most useful when they are repeated many times, R has a command to repeat a whole block of code:

```
replicate(n,expr)
```

Here `expr` is some executable “expression” in R, basically something you could type in the terminal without trouble, and `n` is the number of times to repeat it.

For instance,

```
output <- replicate(1000,rnorm(length(x),beta0+beta1*x,sigma))
```

will replicate, 1000 times, sampling from the predictive distribution of a Gaussian linear regression model. Conceptually, this is equivalent to doing something like

```
output <- matrix(0,nrow=1000,ncol=length(x))
for (i in 1:1000) {
  output[i,] <- rnorm(length(x),beta0+beta1*x,sigma)
}
```

but the `replicate` version has two great advantages. First, it is faster, because R processes it with specially-optimized code. (Loops are especially slow in R.) Second, and *far* more importantly, it is *clearer*: it makes it obvious what is being done, in one line, and leaves the computer to figure out the boring and mundane details of how best to implement it.

5.3 Why Simulate?

There are three major uses for simulation: to understand a model, to check it, and to fit it.

5.3.1 Understanding the Model; Monte Carlo

We understand a model by seeing what it predicts about the variables we care about, and the relationships between them. Sometimes those predictions are easy to extract from a mathematical representation of the model, but often they aren’t. With a model we can simulate, however, we can just run the model and see what happens.

Our stochastic model gives a distribution for some random variable Z , which in general is a complicated, multivariate object with lots of interdependent components. We may also be interested in some complicated function g of Z , such as, say, the ratio of two components of Z , or even some nonparametric curve fit through the data points. How do we know what the model says about g ?

Assuming we can make draws from the distribution of Z , we can find the distribution of any function of it we like, to as much precision as we want. Suppose that $\tilde{Z}_1, \tilde{Z}_2, \dots, \tilde{Z}_b$ are the outputs of b independent runs of the model — b different *replicates* of the model. (I am using the tilde to remind us that these are just simulations.) We can calculate g on each of them, getting $g(\tilde{Z}_1), g(\tilde{Z}_2), \dots, g(\tilde{Z}_b)$. If averaging makes sense for these values, then

$$\frac{1}{b} \sum_{i=1}^b g(\tilde{Z}_i) \xrightarrow{b \rightarrow \infty} \mathbf{E}[g(Z)] \quad (5.9)$$

by the law of large numbers. So simulation and averaging lets us get expectation values. This basic observation is the seed of the **Monte Carlo method**.¹⁰ If our simulations are independent, we can even say that $\frac{1}{b} \sum_{i=1}^b g(\tilde{Z}_i)$ has approximately the distribution $\mathcal{N}(\mathbf{E}[g(Z)], \text{Var}[g(Z)]/b)$ by the central limit theorem. Of course, if you can get expectation values, you can also get variances. (This is handy if trying to apply the central limit theorem!) You can also get any higher moments — if you need the kurtosis for whatever reason, you just have to simulate enough.

You can also pick any set s and get the probability that $g(Z)$ falls into that set:

$$\frac{1}{b} \sum_{i=1}^b \mathbf{1}_s(g(Z_i)) \xrightarrow{b \rightarrow \infty} \Pr(g(Z) \in s) \quad (5.10)$$

The reason this works is of course that $\Pr(g(Z) \in s) = \mathbf{E}[\mathbf{1}_s(g(Z))]$, so it's just the central limit theorem. So we can get the whole distribution of any complicated function of the model that we want, as soon as we can simulate the model. It is really only a little harder to get the complete sampling distribution than it is to get the expectation value, and the exact same ideas apply.

5.3.2 Checking the Model

An important but under-appreciated use for simulation is to *check* models after they have been fit. If the model is right, after all, it represents the mechanism which generates the data. This means that when we simulate, we run that mechanism, and the surrogate data which comes out of the machine should look like the real data. More exactly, the real data should look like a typical realization of the model. If it does not, then the model's account of the data-generating mechanism is systematically wrong

¹⁰The name comes from the physicists who used the method to do calculations relating to designing the hydrogen bomb; see Metropolis *et al.* (1953). Folklore among physicists says that the method goes back at least to Enrico Fermi in the 1930s, without the cutesy name.

```
rgeyser <- function() {
  n <- nrow(geyser)
  sigma <- summary(fit.ols)$sigma
  new.waiting <- rnorm(n, mean=fitted(fit.ols), sd=sigma)
  new.geyser <- data.frame(duration=geyser$duration,
    waiting=new.waiting)
  return(new.geyser)
}
```

Code Example 5: Function for generating surrogate data sets from the linear model fit to `geyser`.

in some way. By carefully choosing the simulations we perform, we can learn a lot about how the model breaks down and how it might need to be improved.¹¹

Often the comparison between simulations and data can be done qualitatively and visually. For example, a classic data set concerns the time between eruptions of the Old Faithful geyser in Yellowstone, and how they relate to the duration of the latest eruption. A common exercise is to fit a regression line to the data by ordinary least squares:

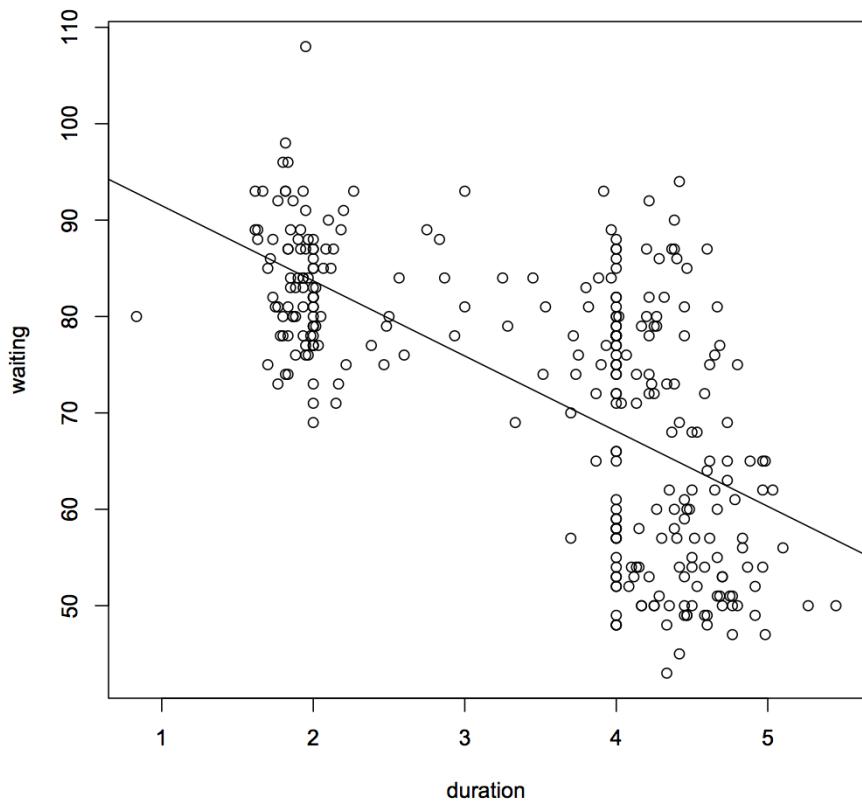
```
library(MASS)
data(geyser)
fit.ols <- lm(waiting~duration,data=geyser)
```

Figure 5.4 shows the data, together with the OLS regression line. It doesn't look that great, but if someone insisted it was a triumph of quantitative vulcanology, how could you show they were wrong?

Well, OLS is usually presented as part of a probability model for the response conditional on the input, with Gaussian and homoskedastic noise. In this case, the probability model is $\text{waiting} = \beta_0 + \beta_1 \text{duration} + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$. If we simulate from this probability model, we'll get something we can compare to the actual data, to help us assess whether the scatter around that regression line is really bothersome. Since OLS doesn't require us to assume a distribution for the input variable (here, `duration`), the simulation function in Code Example 5 leaves those values alone, but regenerates values of the response (`waiting`) according the model assumptions.

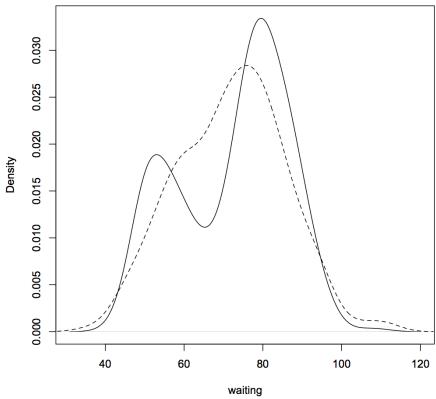
A useful principle for model checking is that if we do some exploratory data analyses of the real data, doing the same analyses to realizations of the model should give roughly the same results. This isn't really the case here. Figure 5.5 shows the actual density of `waiting`, plus the density produced by simulating — reality is clearly bimodal, but the model is unimodal. Similarly, Figure 5.6 shows the real data, the OLS line, and a simulation from the OLS model. It's visually clear that the deviations of the real data from the regression line are both bigger and more patterned than those we get from simulating the model, so something is wrong with the latter.

¹¹"Might", because sometimes we're better off with a model that makes systematic mistakes, if they're small and getting it right would be a hassle.



```
plot(geyser$duration, geyser$waiting, xlab="duration", ylab="waiting")
abline(fit.ols)
```

Figure 5.4: Data for the geyser data set, plus the OLS regression line.



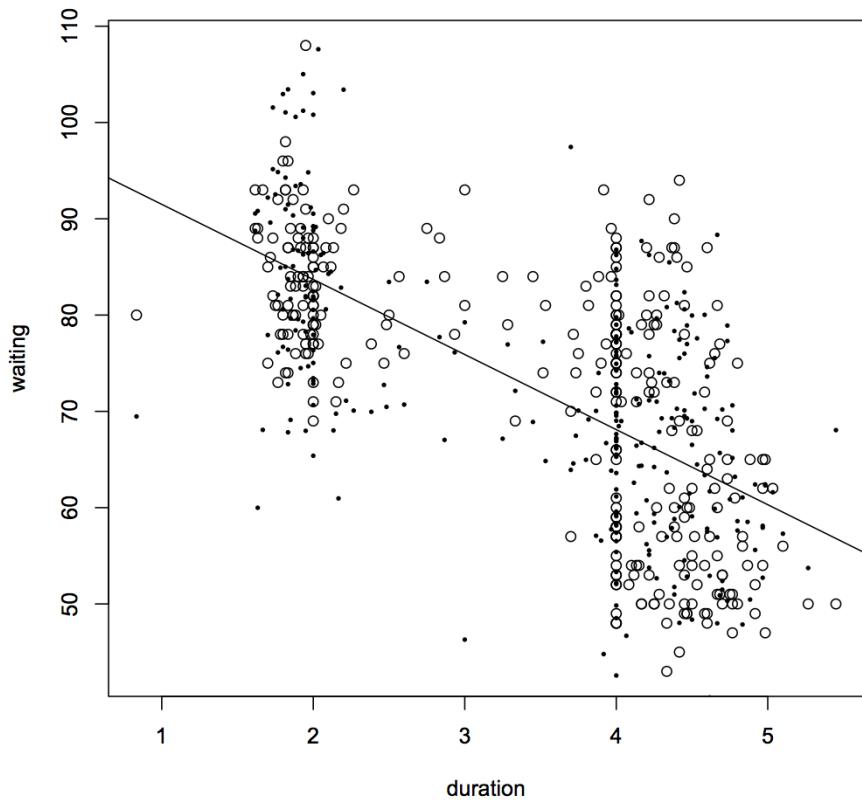
```
plot(density(geyser$waiting),xlab="waiting",main="",sub="")
lines(density(rgeyser()$waiting),lty=2)
```

Figure 5.5: Actual density of the waiting time between eruptions (solid curve) an that produced by simulating the OLS model (dashed).

By itself, just seeing that data doesn't look like a realization of the model isn't super informative, since we'd really like to know *how* the model's broken, and how to fix it. Further simulations, comparing analyses of the data to analyses of the simulation output, are often very helpful here. Looking at Figure 5.6, we might suspect that one problem is heteroskedasticity — the variance isn't constant. This suspicion is entirely correct, and will be explored in §7.3.2.

5.3.3 Sensitivity Analysis

Often, the statistical inference we do on the data is predicated on certain assumptions about how the data is generated. For instance, if we have missing values for some variables and just *ignore* incomplete rows, we are implicitly assuming that data are “missing at random”, rather than in some systematic way. If we are not *totally* confident in such assumptions, we might wish to see what happens they break down. That is, we set up a model where the assumptions are more or less violated, and then run our *original* analysis on the simulation output. Because it's a simulation, we know the complete truth about the data-generating process, and can assess how far off our inferences are. In favorable circumstances, our inferences don't mess up too much even when the assumptions we used to motivate the analysis are badly wrong. Sometimes, however, we discover that even tiny violations of our initial assumptions lead to large errors in our inferences. Then we either need to make some compelling case for those assumptions, or be *very* cautious in our inferences.



```
plot(geyser$duration, geyser$waiting, xlab="duration", ylab="waiting")
abline(fit.ols)
points(rgeyser(), pch=20, cex=0.5)
```

Figure 5.6: As in Figure 5.4, plus one realization of simulating the OLS model (small black dots).

5.4 The Method of Simulated Moments

Checking whether the model's simulation output looks like the data naturally suggests the idea of *adjusting* the model until it does. This becomes a way of estimating the model — in the jargon, **simulation-based inference**. All forms of this involve adjusting parameters of the model until the simulations *do* look like the data. They differ in what “look like” means, concretely. The most straightforward form of simulation-based inference is the **method of simulated moments**.¹²

5.4.1 The Method of Moments

You will have seen the ordinary **method of moments** in earlier statistics classes. Let's recall the general setting. We have a model with a parameter vector θ , and pick a vector m of moments to calculate. The moments, like the expectation of any variables, are functions of the parameters,

$$m = g(\theta) \quad (5.11)$$

for some function g . If that g is invertible, then we can recover the parameters from the moments,

$$\theta = g^{-1}(m) \quad (5.12)$$

The method of moments *estimator* takes the observed, sample moments \hat{m} , and plugs them into Eq. 5.12:

$$\widehat{\theta}_{MM} = g^{-1}(\hat{m}) \quad (5.13)$$

What if g^{-1} is hard to calculate — if it's hard to explicitly solve for parameters from moments? In that case, we can use minimization:

$$\widehat{\theta}_{MM} = \underset{\theta}{\operatorname{argmin}} \|g(\theta) - \hat{m}\|^2 \quad (5.14)$$

For the minimization version, we just have to calculate moments from parameters $g(\theta)$, not vice versa. To see that Eqs. 5.13 and 5.14 do the same thing, notice that (i) the squared¹³ distance $\|g(\theta) - \hat{m}\|^2 \geq 0$, (ii) the distance is only zero when the moments are matched exactly, and (iii) there is only θ which will match the moments.

In either version, the method of moments works *statistically* because the sample moments \hat{m} converge on their expectations $g(\theta)$ as we get more and more data. This is, to repeat, a consequence of the law of large numbers.

It's worth noting that nothing in this argument says that m has to be a vector of *moments* in the strict sense. They could be expectations of any functions of the random variables, so long as $g(\theta)$ is invertible, we can calculate the sample expectations of these functions from the data, and the sample expectations converge. When m isn't just a vector of moments, then, we have the **generalized method of moments**.

¹²This section is optional for spring 2013.

¹³Why squared? Basically because it makes the function we're minimizing smoother, and the optimization nicer.

It is also worth noting that there's a somewhat more general version of the same method, where we minimize

$$(g(\theta) - \hat{m}) \cdot \mathbf{w} (g(\theta) - \hat{m}) \quad (5.15)$$

with some positive-definite weight matrix \mathbf{w} . This can help if some of the moments are much more sensitive to the parameters than others. But this goes beyond what we really need here.

5.4.2 Adding in the Simulation

All of this supposes that we know how to calculate $g(\theta)$ — that we can find the moments exactly. Even if this is too hard, however, we could always *simulate* to approximate these expectations, and try to match the simulated moments to the real ones. Rather than Eq. 5.14, the estimator would be

$$\widehat{\theta}_{SMM} = \operatorname{argmin}_{\theta} \left\| \tilde{g}_{s,T}(\theta) - \hat{m} \right\|^2 \quad (5.16)$$

with s being the number of simulation paths and T being their size. Now consistency requires that $\tilde{g} \rightarrow g$, either as T grows or s or both, but this is generally assured by the law of large numbers, as we talked about earlier. Simulated method of moments estimates like this are generally more uncertain than ones which don't rely on simulation, since it introduces an extra layer of approximation, but this can be reduced by increasing s .¹⁴

5.4.3 An Example: Moving Average Models and the Stock Market

To give a concrete example, we will try fitting a time series model to the stock market: it's a familiar subject which interests *most* students, and we can check the method of simulated moments here against other estimation techniques.¹⁵

Our data will consist of about ten year's worth of daily values for the S& P 500 stock index, available on the class website:

```
sp <- read.csv("SPhistory.short.csv")
# We only want closing prices
sp <- sp[,7]
# The data are in reverse chronological order, which is weird for us
sp <- rev(sp)
# And in fact we only want log returns, i.e., difference in logged prices
sp <- diff(log(sp))
```

¹⁴A common trick is to fix T at the actual sample size n , and then to increase s as much as computationally feasible. By looking at the variance of \tilde{g} across different runs of the model with the same θ , one gets an idea of how much uncertainty there is in \hat{m} itself, and so of how precisely one should expect to be able to match it. If the optimizer has gotten $|\tilde{g}(\theta) - \hat{m}|$ down to 0.02, and the standard deviation of \tilde{g} at constant θ is 0.1, further effort at optimization is probably wasted.

¹⁵Nothing in what follows, or in the homework, could actually be used to make money, however.

Professionals in finance do not care so much about the sequence of **prices** P_t , as the sequence of **returns**, $\frac{P_t - P_{t-1}}{P_{t-1}}$. This is because making \$1000 is a lot better when you invested \$1000 than when you invested \$1,000,000, but 10% is 10%. In fact, it's often easier to deal with the **log returns**, $X_t = \log \frac{P_t}{P_{t-1}}$, as we do here.

The model we will fit is a **first-order moving average**, or MA(1), model:

$$X_t = Z_t + \theta Z_{t-1} \quad (5.17)$$

$$Z_t \sim \mathcal{N}(0, \sigma^2) \text{ i.i.d.} \quad (5.18)$$

The X_t sequence of variables are the returns we see; the Z_t variables are invisible to us. The interpretation of the model is as follows. Prices in the stock market change in response to news that affects the prospects of the companies listed, as well as news about changes in over-all economic conditions. Z_t represents this flow of news, good and bad. It makes sense that Z_t is uncorrelated, because the relevant part of the news is only what everyone hadn't already worked out from older information¹⁶. However, it does take some time for the news to be assimilated, and this is why Z_{t-1} contributes to X_t . A negative contribution, $\theta < 0$, would seem to indicate a "correction" to the reaction to the previous day's news.

Mathematically, notice that since Z_t and θZ_{t-1} are independent Gaussians, X_t is a Gaussian with mean 0 and variance $\sigma^2 + \theta^2 \sigma^2$. The marginal distribution of X_t is therefore the same for all t . For technical reasons¹⁷, we can really only get sensible behavior from the model when $-1 \leq \theta \leq 1$.

There are two parameters, θ and σ^2 , so we need two moments for estimation. Let's try $\text{Var}[X_t]$ and $\text{Cov}[X_t, X_{t-1}]$.

$$\text{Var}[X_t] = \text{Var}[Z_t] + \theta^2 \text{Var}[Z_{t-1}] \quad (5.19)$$

$$= \sigma^2 + \theta^2 \sigma^2 \quad (5.20)$$

$$= \sigma^2(1 + \theta^2) \equiv v(\theta, \sigma) \quad (5.21)$$

(This agrees with our earlier reasoning about Gaussians, but doesn't need it.)

$$\text{Cov}[X_t, X_{t-1}] = E[(Z_t + \theta Z_{t-1})(Z_{t-1} + \theta Z_{t-2})] \quad (5.22)$$

$$= \theta E[Z_{t-1}^2] \quad (5.23)$$

$$= \theta \sigma^2 \equiv c(\theta, \sigma) \quad (5.24)$$

We can solve the system of equations for the parameters, starting with eliminating

¹⁶Nobody will ever say "What? It's snowing in Pittsburgh in February? I must call my broker!"

¹⁷Think about trying to recover Z_t , if we knew θ . One might try $X_t - \theta X_{t-1}$, which is almost right, it's $Z_t + \theta Z_{t-1} - \theta Z_{t-1} - \theta^2 Z_{t-2} = Z_t - \theta^2 Z_{t-2}$. Similarly, $X_t - \theta X_{t-1} + \theta^2 X_{t-2} = Z_t + \theta^3 Z_{t-2}$, and so forth. If $|\theta| < 1$, then this sequence of approximations will converge on Z_t ; if not, then not. It turns out that models which are not "invertible" in this way are very strange — see Shumway and Stoffer (2000).

σ^2 :

$$\frac{c(\theta, \sigma)}{v(\theta, \sigma)} = \frac{\sigma^2 \theta}{\sigma^2(1 + \theta^2)} \quad (5.25)$$

$$= \frac{\theta}{1 + \theta^2} \quad (5.26)$$

$$0 = \theta^2 \frac{c}{v} - \theta + \frac{c}{v} \quad (5.27)$$

This is a quadratic in θ ,

$$\theta = \frac{1 \pm \sqrt{1 - 4 \frac{c^2}{v^2}}}{2c/v} \quad (5.28)$$

and it's easy to confirm¹⁸ that this has only one solution in the meaningful range, $-1 \leq \theta \leq 1$. Having found θ , we solve for σ^2 ,

$$\sigma^2 = c/\theta \quad (5.29)$$

The method of moments estimator takes the sample values of these moments, \hat{v} and \hat{c} , and plugs them in to Eqs. 5.28 and 5.29. With the S& P returns, the sample covariance is -1.61×10^{-5} , and the sample variance 1.96×10^{-4} . This leads to $\hat{\theta}_{MM} = -8.28 \times 10^{-2}$, and $\hat{\sigma^2}_{MM} = 1.95 \times 10^{-4}$. In terms of the model, then, each day's news has a follow-on impact on prices which is about 8% as large as its impact the first day, but with the opposite sign.¹⁹

If we did not know how to solve a quadratic equation, we could use the minimization version of the method of moments estimator:

$$\begin{bmatrix} \hat{\theta}_{MM} \\ \hat{\sigma^2}_{MM} \end{bmatrix} = \underset{\theta, \sigma^2}{\operatorname{argmin}} \left\| \begin{bmatrix} \sigma^2 \theta - \hat{c} \\ \sigma^2(1 + \theta^2) - \hat{v} \end{bmatrix} \right\|^2 \quad (5.30)$$

Computationally, it would go something like Code Example 6.

The parameters estimated by minimization agree with those from direct algebra to four significant figures, which I hope is good enough to reassure you that this works.

Before we can try out the method of simulated moments, we have to figure out how to simulate our model. X_t is a deterministic function of Z_t and Z_{t-1} , so our general strategy says to first generate the Z_t , and then compute X_t from that. But here the Z_t are just a sequence of independent Gaussians, which is a solved problem for us. The one wrinkle is that to get our first value X_1 , we need a previous value Z_0 . Code Example 7 shows the solution.

¹⁸For example, plot c/v as a function of θ , and observe that any horizontal line cuts the graph at only one point.

¹⁹It would be natural to wonder whether $\hat{\theta}_{MM}$ is really significantly different from zero. Assuming Gaussian noise, one could, in principle, calculate the probability that even though $\theta = 0$, by chance \hat{c}/\hat{v} was so far from zero as to give us our estimate. As you will see in the homework, however, Gaussian assumptions are very bad for this data. This sort of thing is why we have bootstrapping.

```

ma.mm.est <- function(c,v) {
  theta.0 <- c/v
  sigma2.0 <- v
  fit <- optim(par=c(theta.0,sigma2.0), fn=ma.mm.objective,
               c=c, v=v)
  return(fit)
}

ma.mm.objective <- function(params,c,v) {
  theta <- params[1]
  sigma2 <- params[2]
  c.pred <- theta*sigma2
  v.pred <- sigma2*(1+theta^2)
  return((c-c.pred)^2 + (v-v.pred)^2)
}

```

Code Example 6: Code for implementing method of moments estimation of a first-order moving average model, as in Eq. 5.30. See Appendix 5.6 for “design notes”, and the online code for comments.

```

rma <- function(n,theta,sigma2,s=1) {
  z <- replicate(s,rnorm(n=n+1,mean=0,sd=sqrt(sigma2)))
  x <- z[-1,] + theta*z[-(n+1),]
  return(x)
}

```

Code Example 7: Function which simulates *s* independent runs of a first-order moving average model, each of length *n*, with given noise variance *sigma2* and after-effect *theta*. See online for the version with comments on the code details.

```

sim.var <- function(n,theta,sigma2,s=1) {
  vars <- apply(rma(n,theta,sigma2,s),2,var)
  return(mean(vars))
}

sim.cov <- function(n,theta,sigma2,s=1) {
  x <- rma(n,theta,sigma2,s)
  covs <- colMeans(x[-1,]*x[-n,])
  return(mean(covs))
}

```

Code Example 8: Functions for calculating the variance and covariance for specified parameter values from simulations.

What we need to extract from the simulation are the variance and the covariance. It will be more convenient to have functions which calculate these call `rma()` themselves (Code Example 8).

Figure 5.7 plots the covariance, the variance, and their ratio as functions of θ with $\sigma^2 = 1$, showing both the values obtained from simulation and the theoretical ones.²⁰ The agreement is quite good, though of course not quite perfect.²¹

Conceptually, we could estimate θ by just taking the observed value $\hat{c}/\hat{\sigma}$, running a horizontal line across Figure 5.7c, and seeing at what θ it hit one of the simulation dots. Of course, there might not be one it hits *exactly*...

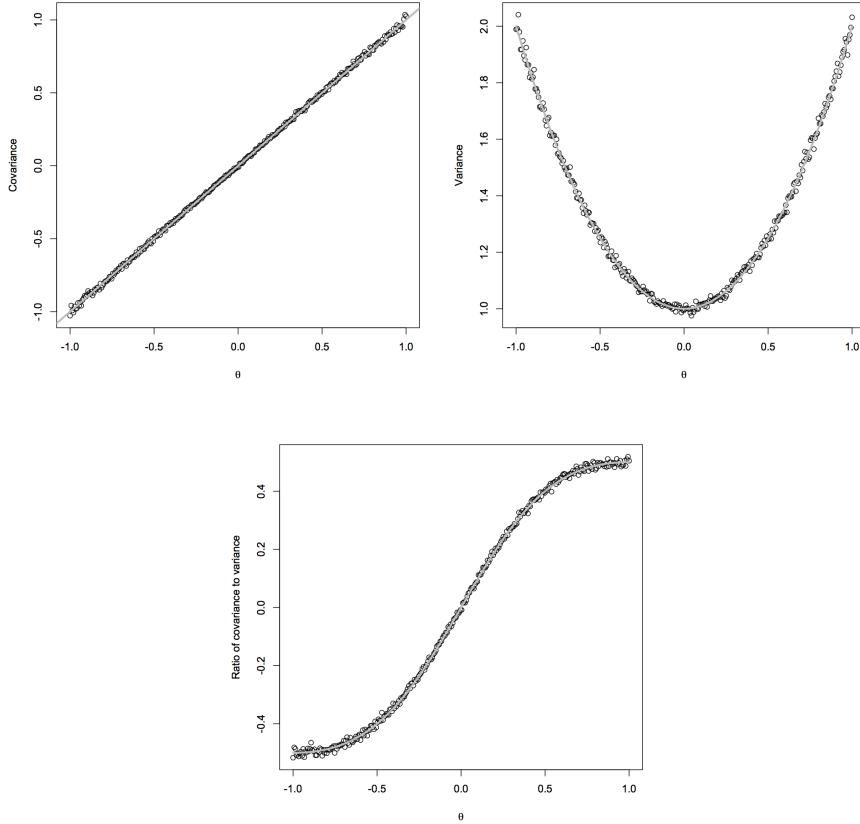
The more practical approach is Code Example 9. The code is practically identical to that in Code Example 6, except that the variance and covariance predicted by given parameter settings now come from simulating those settings, not an exact calculation. Also, we have to say how long a simulation to run, and how many simulations to average over per parameter value.

When I run this, with $s=100$, I get $\hat{\theta}_{MSM} = -8.36 \times 10^{-2}$ and $\hat{\sigma}_{MSM}^2 = 1.94 \times 10^{-4}$, which is quite close to the non-simulated method of moments estimate. In fact, in this case there is actually a maximum likelihood estimator (`arima()`, after the more general class of models including MA models), which claims $\hat{\theta}_{ML} = -9.75 \times 10^{-2}$ and $\hat{\sigma}_{ML}^2 = 1.94 \times 10^{-4}$. Since the standard error of the MLE on θ is ± 0.02 , this is working essentially as well as the method of moments, or even the method of simulated moments.

In this case, because there *is* a very tractable maximum likelihood estimator, one generally wouldn't use the method of simulated moments. But we can in this case check whether it works (it does), and so we can use the same technique for other models, where an MLE is unavailable.

²⁰I could also have varied σ^2 and made 3D plots, but that would have been more work. Also, the variance and covariance are both proportional to σ^2 , so the shapes of the figures would all be the same.

²¹If you look at those figures and think "Why not do a nonparametric regression of the simulated moments against the parameters and use the fitted values as \tilde{g} , it'll get rid of some of the simulation noise?", congratulations, you've just discovered the smoothed method of simulated moments.



```

theta.grid <- seq(from=-1,to=1,length.out=300)
cov.grid <- sapply(theta.grid,sim.cov,sigma2=1,n=length(sp),s=10)
plot(theta.grid,cov.grid,xlab=expression(theta),ylab="Covariance")
abline(0,1,col="grey",lwd=3)
var.grid <- sapply(theta.grid,sim.var,sigma2=1,n=length(sp),s=10)
plot(theta.grid,var.grid,xlab=expression(theta),ylab="Variance")
curve((1+x^2),col="grey",lwd=3,add=TRUE)
plot(theta.grid,cov.grid/var.grid,xlab=expression(theta),
      ylab="Ratio of covariance to variance")
curve(x/(1+x^2),col="grey",lwd=3,add=TRUE)

```

Figure 5.7: Plots of the covariance, the variance, and their ratio as a function of θ , with $\sigma^2 = 1$. Dots show simulation values (averaging 10 realizations each as long as the data), the grey curves the exact calculations.

```
ma msm.est <- function(c,v,n,s) {  
  theta.0 <- c/v  
  sigma2.0 <- v  
  fit <- optim(par=c(theta.0,sigma2.0),fn=ma msm.objective,c=c,v=v,n=n,s=s)  
  return(fit)  
}  
  
ma msm.objective <- function(params,c,v,n,s) {  
  theta <- params[1]  
  sigma2 <- params[2]  
  c.pred <- sim.cov(n,theta,sigma2,s)  
  v.pred <- sim.var(n,theta,sigma2,s)  
  return((c-c.pred)^2 + (v-v.pred)^2)  
}
```

Code Example 9: Code for implementing the method of simulated moments estimation of a first-order moving average model.

5.5 Exercises

To think through, not to hand in.

Section 5.4 explained the method of simulated moments, where we try to match expectations of various functions of the data. Expectations of functions are summary statistics, but they're not the *only* kind of summary statistics. We could try to estimate our model by matching any set of summary statistics, so long as (i) there's a unique way of mapping back from summaries to parameters, and (ii) estimates of the summary statistics converge as we get more data.

A powerful but somewhat paradoxical version of this is what's called **indirect inference**, where the summary statistics are the parameters of a *different* model. This second or **auxiliary** model does *not* have to be correctly specified, it just has to be easily fit to the data, and satisfy (i) and (ii) above. Say the parameters of the auxiliary model are β , as opposed to the θ of our real model. We calculate $\hat{\beta}$ on the real data. Then we simulate from different values of θ , fit the auxiliary to the simulation outputs, and try to match the auxiliary estimates. Specifically, the indirect inference estimator is

$$\hat{\theta}_{II} = \underset{\theta}{\operatorname{argmin}} \|\tilde{\beta}(\theta) - \hat{\beta}\|^2 \quad (5.31)$$

where $\tilde{\beta}(\theta)$ is the value of β we estimate from a simulation of θ , of the same size as the original data. (We might average together a couple of simulation runs for each θ .) If we have a consistent estimator of β , then

$$\hat{\beta} \rightarrow \beta \quad (5.32)$$

$$\tilde{\beta}(\theta) \rightarrow b(\theta) \quad (5.33)$$

If in addition $b(\theta)$ is invertible, then

$$\hat{\theta}_{II} \rightarrow \theta \quad (5.34)$$

For this to work, the auxiliary model needs to have at least as many parameters as the real model, but we can often arrange this by, say, making the auxiliary model a linear regression with a lot of coefficients.

A specific case, often useful for time series, is to make the auxiliary model an **autoregressive model**, where each observation is linearly regressed on the previous ones. A first-order autoregressive model (or "AR(1)") is

$$X_t = \beta_0 + \beta_1 X_{t-1} + \epsilon_t \quad (5.35)$$

where $\epsilon_t \sim \mathcal{N}(0, \beta_3)$. (So an AR(1) has three parameters.)

1. Convince yourself that if X_t comes from an MA(1) process, it can't also be written as an AR(1) model.
2. Write a function, `ar1.fit`, to fit an AR(1) model to a time series, using `lm`, and to return the three parameters (intercept, slope, noise variance).

3. Apply `ar1.fit` to the S&P 500 data; what are the auxiliary parameter estimates?
4. Combine `ar1.fit` with the simulator `rma`, and plot the three auxiliary parameters as functions of θ , holding σ^2 fixed at 1. (This is analogous to Figure 5.7.)
5. Write functions, analogous to `ma.msm.est` and `ma.msm.objective`, for estimating an MA(1) model, using an AR(1) model as the auxiliary function. Does this recover the right parameter values when given data simulated from an MA(1) model?
6. What values does your estimator give for θ and σ^2 on the S& P 500 data? How do they compare to the other estimates?

5.6 Appendix: Some Design Notes on the Method of Moments Code

Go back to Section 5.4.3 and look at the code for the method of moments. There've been a fair amount of questions about writing code, and this is a useful example.

The first function, `ma.mm.est`, estimates the parameters taking as inputs two numbers, representing the covariance and the variance. The real work is done by the built-in `optim` function, which itself takes two major arguments. One, `fn`, is the function to optimize. Another, `par`, is an initial guess about the parameters at which to begin the search for the optimum.²²

The `fn` argument to `optim` must be a function, here `ma.mm.objective`. The first argument to that function has to be a vector, containing all the parameters to be optimized over. (Otherwise, `optim` will quit and complain.) There can be other arguments, not being optimized over, to that function, which `optim` will pass along, as you see here. `optim` will also accept a lot of optional arguments to control the search for the optimum — see `help(optim)`.

All `ma.mm.objective` has to do is calculate the objective function. The first two lines peel out θ and σ^2 from the parameter vector, just to make it more readable. The next two lines calculate what the moments should be. The last line calculates the distance between the model predicted moments and the actual ones, and returns it. The whole thing could be turned into a one-line, like

```
return(t(params-c(c,v)) %*% (params-c(c,v)))
```

or perhaps even more obscure, but that is usually a bad idea.

Notice that I could write these two functions independently of one another, at least to some degree. When writing `ma.mm.est`, I knew I would need the objective function, but all I needed to know about it was its name, and the promise that it would take a parameter vector and give back a real number. When writing `ma.mm.objective`, all I had to remember about the other function was the promise this one needed to fulfill. In my experience, it is usually easiest to do any substantial coding in this “top-down” fashion²³. Start with the high-level goal you are trying to achieve, break it down into a few steps, write something which will put those steps together, presuming other functions or programs can do them. Now go and write the functions to do each of those steps.

The code for the method of simulated moments is entirely parallel to these. Writing it as two separate pairs of functions is therefore somewhat wasteful. If I find a mistake in one pair, or think of a way to improve it, I need to remember to make corresponding changes in the other pair (and not introduce a new mistake). In the long run, when you find yourself writing parallel pieces of code over and over, it is better to try to pull together the common parts and write them *once*. Here, that would mean something like one pair of functions, with the inner one having an argument

²²Here `par` is a very rough guess based on `c` and `v` — it'll actually be right when `c=0`, but otherwise it's not much good. Fortunately, it doesn't have to be! Anyway, let's return to designing the code

²³What qualifies as “substantial coding” depends on how much experience you have

which controlled whether to calculate the predicted moments by simulation or by a formula. You may try your hand at writing this.

Chapter 6

The Bootstrap

We are now several chapters into a statistics class and have said basically nothing about uncertainty. This should seem odd, and may even be disturbing if you are very attached to your p -values and saying variables have “significant effects”. It is time to remedy this, and talk about how we can quantify uncertainty for complex models. The key technique here is what’s called **bootstrapping**, or **the bootstrap**.

6.1 Stochastic Models, Uncertainty, Sampling Distributions

Statistics is the branch of mathematical engineering which studies ways of drawing inferences from limited and imperfect data. We want to know how a neuron in a rat’s brain responds when one of its whiskers gets tweaked, or how many rats live in Pittsburgh, or how high the water will get under the 16^{mathrm{th}} Street bridge during May, or the typical course of daily temperatures in the city over the year, or the relationship between the number of birds of prey in Schenley Park in the spring and the number of rats the previous fall. We have some data on all of these things. But we know that our data is incomplete, and experience tells us that repeating our experiments or observations, even taking great care to replicate the conditions, gives more or less different answers every time. It is foolish to treat any inference from the data in hand as certain.

If all data sources were totally capricious, there’d be nothing to do beyond piously qualifying every conclusion with “but we could be wrong about this”. A mathematical discipline of statistics is possible because while repeating an experiment gives different results, some kinds of results are more common than others; their relative frequencies are reasonably stable. We thus model the data-generating mechanism through probability distributions and stochastic processes. When and why we can use stochastic models are very deep questions, but ones for another time. If we *can* use them in our problem, quantities like the ones I mentioned above are represented as functions of the stochastic model, i.e., of the underlying probability distribution.

Since a function of a function is a “functional”, and these quantities are functions of the true probability distribution function, we’ll call these **functionals** or **statistical functionals**¹. Functionals could be single numbers (like the total rat population), or vectors, or even whole curves (like the expected time-course of temperature over the year, or the regression of hawks now on rats earlier). Statistical inference becomes estimating those functionals, or testing hypotheses about them.

These estimates and other inferences are functions of the data values, which means that they inherit variability from the underlying stochastic process. If we “re-ran the tape” (as the late, great Stephen Jay Gould used to say), we would get different data, with a certain characteristic distribution, and applying a fixed procedure would yield different inferences, again with a certain distribution. Statisticians want to use this distribution to quantify the uncertainty of the inferences. For instance, the standard error is an answer to the question “By how much would our estimate of this functional vary, typically, from one replication of the experiment to another?” (It presumes a particular meaning for “typically vary”, as root mean square deviation around the mean.) A confidence region on a parameter, likewise, is the answer to “What are all the values of the parameter which *could* have produced this data with at least some specified probability?”, i.e., all the parameter values under which our data are not low-probability outliers. The confidence region is a promise that *either* the true parameter point lies in that region, *or* something very unlikely under any circumstances happened — or that our stochastic model is wrong.

To get things like standard errors or confidence intervals, we need to know the distribution of our estimates around the true values of our functionals. These **sampling distributions** follow, remember, from the distribution of the data, since our estimates are functions of the data. Mathematically the problem is well-defined, but actually *computing* anything is another story. Estimates are typically complicated functions of the data, and mathematically-convenient distributions may all be poor approximations to the data source. Saying anything in closed form about the distribution of estimates can be simply hopeless. The two classical responses of statisticians were to focus on tractable special cases, and to appeal to asymptotics.

Your introductory statistics courses mostly drilled you in the special cases. From one side, limit the kind of estimator we use to those with a simple mathematical form — say, means and other linear functions of the data. From the other, assume that the probability distributions featured in the stochastic model take one of a few forms for which exact calculation *is* possible, analytically or via tabulated special functions. Most such distributions have origin myths: the Gaussian arises from averaging many independent variables of equal size (say, the many genes which contribute to height in humans); the Poisson distribution comes from counting how many of a large number of independent and individually-improbable events have occurred (say, radioactive nuclei decaying in a given second), etc. Squeezed from both ends, the sampling distribution of estimators and other functions of the data becomes exactly calculable in terms of the aforementioned special functions.

That these origin myths invoke various limits is no accident. The great results

¹Most writers in theoretical statistics just call them “parameters” in a generalized sense, but I will try to restrict that word to actual parameters specifying statistical models, to minimize confusion. I may slip up.

of probability theory — the laws of large numbers, the ergodic theorem, the central limit theorem, etc. — describe limits in which *all* stochastic processes in broad classes of models display the same asymptotic behavior. The central limit theorem, for instance, says that if we average more and more independent random quantities with a common distribution, and that common distribution isn't too pathological, then the average becomes closer and closer to a Gaussian² Typically, as in the CLT, the limits involve taking more and more data from the source, so statisticians use the theorems to find the asymptotic, large-sample distributions of their estimates. We have been especially devoted to re-writing our estimates as averages of independent quantities, so that we can use the CLT to get Gaussian asymptotics.

Up through about the 1960s, statistics was split between developing general ideas about how to draw and evaluate inferences with stochastic models, and working out the properties of inferential procedures in tractable special cases (especially the linear-and-Gaussian case), or under asymptotic approximations. This yoked a very broad and abstract theory of inference to very narrow and concrete practical formulas, an uneasy combination often preserved in basic statistics classes.

The arrival of (comparatively) cheap and fast computers made it feasible for scientists and statisticians to record lots of data and to fit models to it, so they did. Sometimes the models were conventional ones, including the special-case assumptions, which often enough turned out to be detectably, and consequentially, wrong. At other times, scientists wanted more complicated or flexible models, some of which had been proposed long before, but now moved from being theoretical curiosities to stuff that could run overnight³. In principle, asymptotics might handle either kind of problem, but convergence to the limit could be unacceptably slow, especially for more complex models.

By the 1970s, then, statistics faced the problem of quantifying the uncertainty of inferences without using either implausibly-helpful assumptions or asymptotics; all of the solutions turned out to demand *even more* computation. Here we will examine what may be the most successful solution, Bradley Efron's proposal to combine estimation with simulation, which he gave the less-than-clear but persistent name of "the bootstrap" (Efron, 1979).

6.2 The Bootstrap Principle

Remember (from baby stats.) that the key to dealing with uncertainty in parameters and functionals is the sampling distribution of estimators. Knowing what distribution we'd get for our estimates on repeating the experiment would give us things like standard errors. Efron's insight was that we can *simulate* replication. After all, we have already fitted a model to the data, which is a guess at the mechanism which generated the data. Running that mechanism generates simulated data which, by hypothesis, has the same distribution as the real data. Feeding the simulated data through

²The reason is that the non-Gaussian parts of the distribution wash away under averaging, but the average of two Gaussians is another Gaussian.

³Kernel regression, kernel density estimation, and nearest neighbors prediction were all proposed in the 1950s, but didn't begin to be widely used until the 1970s, or even the 1980s.

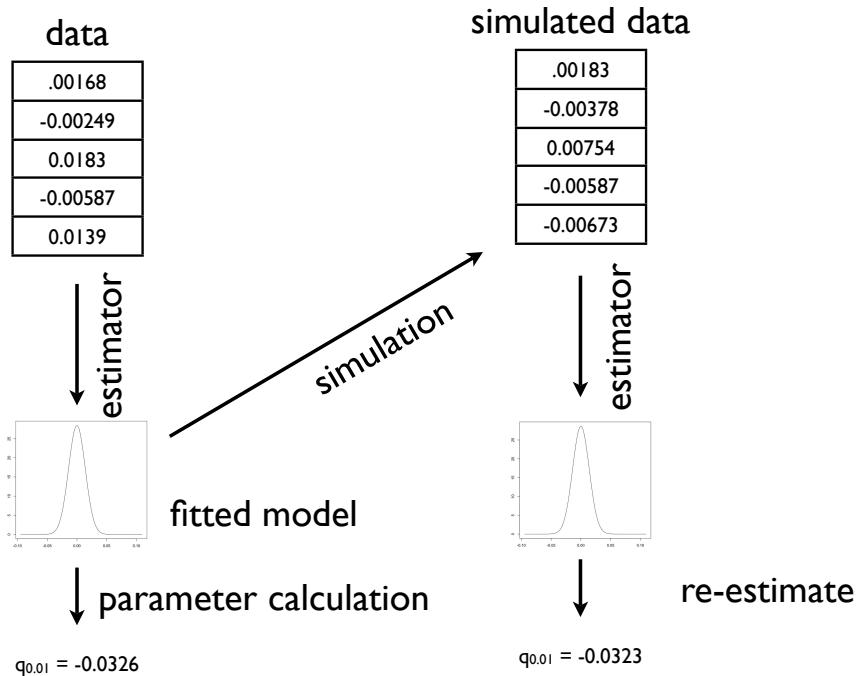


Figure 6.1: Schematic for model-based bootstrapping: simulated values are generated from the fitted model, then treated like the original data, yielding a new estimate of the functional of interest, here called $q_{0.01}$.

our estimator gives us one draw from the sampling distribution; repeating this many times yields the sampling distribution. Since we are using the model to give us its own uncertainty, Efron called this “bootstrapping”; unlike the Baron Munchhausen’s plan for getting himself out of a swamp by pulling himself out by his bootstraps, it works.

Figure 6.1 sketches the over-all process: fit a model to data, use the model to calculate the functional, then get the sampling distribution by generating new, synthetic data from the model and repeating the estimation on the simulation output.

To fix notation, we’ll say that the original data is x . (In general this is a whole data frame, not a single number.) Our parameter estimate from the data is $\hat{\theta}$. Surrogate data sets simulated from the fitted model will be $\tilde{X}_1, \tilde{X}_2, \dots, \tilde{X}_B$. The corresponding re-estimates of the parameters on the surrogate data are $\tilde{\theta}_1, \tilde{\theta}_2, \dots, \tilde{\theta}_B$. The functional of interest is estimated by the statistic T , with sample value $\hat{T} = T(x)$, and values of the surrogates of $\tilde{T}_1 = T(\tilde{X}_1)$, $\tilde{T}_2 = T(\tilde{X}_2)$, \dots , $\tilde{T}_B = T(\tilde{X}_B)$. (The statistic T may be a direct function of the estimated parameters, and only indirectly a function of x .) Everything which follows applies without modification when the functional of interest *is* the parameter, or some component of the parameter.

In this section, we will assume that the model is correct for *some* value of θ , which we will call θ_0 . The true (population or ensemble) values of the functional is likewise

```
rboot <- function(B, statistic, simulator) {
  tboots <- replicate(B, statistic(simulator()))
  return(tboots)
}

bootstrap.se <- function(simulator, statistic, B) {
  tboots <- rboot(B, statistic, simulator)
  se <- sd(tboots)
  return(se)
}
```

Code Example 10: Sketch of code for calculating bootstrap standard errors. The function `rboot` generates B bootstrap samples (using the `simulator` function) and calculates the statistic g on them (using `statistic`). `simulator` needs to be a function which returns a surrogate data set in a form suitable for `statistic`. (How would you modify the code to pass arguments to `simulator` and/or `statistic`?) Because every use of bootstrapping is going to need to do this, it makes sense to break it out as a separate function, rather than writing the same code many times (with many chances of getting it wrong). `bootstrap.se` just calls `rboot` and takes a standard deviation.

t_0 .

6.2.1 Variances and Standard Errors

The simplest thing to do is to get the variance or standard error:

$$\widehat{\text{Var}}[\hat{t}] = \text{Var}[\tilde{t}] \quad (6.1)$$

$$\widehat{\text{se}}(\hat{t}) = \text{sd}(\tilde{t}) \quad (6.2)$$

That is, we approximate the variance of our estimate of t_0 under the true but unknown distribution θ_0 by the variance of re-estimates \tilde{t} on surrogate data from the fitted model $\hat{\theta}$. Similarly we approximate the true standard error by the standard deviation of the re-estimates. The logic here is that the simulated \tilde{X} has about the same distribution as the real X that our data, x , was drawn from, so applying the same estimation procedure to the surrogate data gives us the sampling distribution. This assumes, of course, that our model is right, and that $\hat{\theta}$ is not too far from θ_0 .

Pseudo-code is provided in Code Example 10.

6.2.2 Bias Correction

We can use bootstrapping to correct for a biased estimator. Since the sampling distribution of \tilde{t} is close to that of \hat{t} , and \hat{t} itself is close to t_0 ,

$$\mathbb{E}[\tilde{t}] - t_0 \approx \mathbb{E}[\hat{t}] - \hat{t} \quad (6.3)$$

```
bootstrap.bias <- function(simulator, statistic, B,
  t.hat) {
  tboots <- rboot(B, statistic, simulator)
  bias <- mean(tboots) - t.hat
  return(bias)
}
```

Code Example 11: Sketch of code for bootstrap bias correction. Arguments are as in Code Example 10, except that `t.hat` is the estimate on the original data.

The left hand side is the bias that we want to know, and the right-hand side the was what we can calculate with the bootstrap.

Note, in fact, that Eq. 6.3 remains valid so long as the sampling distribution of $\hat{t} - t_0$ is close to that of $\tilde{t} - \hat{t}$. This is a weaker requirement than asking for \hat{t} and \tilde{t} themselves to have similar distributions, or asking for \hat{t} to be close to t_0 . In statistical theory, a random variable whose distribution does not depend on the parameters is called a **pivot**. (The metaphor is that it stays in one place while the parameters turn around it.) A sufficient (but not necessary) condition for Eq. 6.3 to hold is that $\hat{t} - t_0$ be a pivot, or approximately pivotal.

6.2.3 Confidence Intervals

A confidence interval is a random interval which contains the truth with high probability (the confidence level). If the confidence interval for g is C , and the confidence level is $1 - \alpha$, then we want

$$\Pr(t_0 \in C) = 1 - \alpha \quad (6.4)$$

no matter what the true value of t_0 . When we calculate a confidence interval, our inability to deal with distributions exactly means that the true confidence level, or **coverage** of the interval, is not quite the desired confidence level $1 - \alpha$; the closer it is, the better the approximation, and the more accurate the confidence interval.

When we simulate, we get samples of \tilde{t} , but what we really care about is the distribution of \hat{t} . When we have enough data to start with, those two distributions will be approximately the same. But with equal amounts of data, the distribution of $\tilde{t} - \hat{t}$ will usually be closer to that of $\hat{t} - t_0$ than the distribution of \tilde{t} is to that of \hat{t} . That is, the distribution of fluctuations around the true value usually converges quickly. (Think of the central limit theorem.) We can use this to turn information about the distribution of \tilde{t} into accurate confidence intervals for t_0 , essentially by re-centering \tilde{t} around \hat{t} .

```
bootstrap.ci.basic <- function(simulator, statistic, B,
  t.hat, alpha) {
  tboots <- rboot(B,statistic, simulator)
  ci.lower <- 2*t.hat - quantile(tboots,1-alpha/2)
  ci.upper <- 2*t.hat - quantile(tboots,alpha/2)
  return(list(ci.lower=ci.lower,ci.upper=ci.upper))
}
```

Code Example 12: Sketch of code for calculating the basic bootstrap confidence interval. See Code Examples 11 and 10.

Specifically, let $q_{\alpha/2}$ and $q_{1-\alpha/2}$ be the $\alpha/2$ and $1 - \alpha/2$ quantiles of \tilde{T} . Then

$$1 - \alpha = \Pr(q_{\alpha/2} \leq \tilde{T} \leq q_{1-\alpha/2}) \quad (6.5)$$

$$= \Pr(q_{\alpha/2} - \hat{T} \leq \tilde{T} - \hat{T} \leq q_{1-\alpha/2} - \hat{T}) \quad (6.6)$$

$$\approx \Pr(q_{\alpha/2} - \hat{T} - t_0 \leq q_{1-\alpha/2} - \hat{T}) \quad (6.7)$$

$$= \Pr(q_{\alpha/2} - 2\hat{T} \leq -t_0 \leq q_{1-\alpha/2} - 2\hat{T}) \quad (6.8)$$

$$= \Pr(2\hat{T} - q_{1-\alpha/2} \leq t_0 \leq 2\hat{T} - q_{\alpha/2}) \quad (6.9)$$

The interval $C = [2\hat{T} - q_{\alpha/2}, 2\hat{T} - q_{1-\alpha/2}]$ is random, because \hat{T} is a random quantity, so it makes sense to talk about the probability that it contains the true value t_0 . Also, notice that the upper and lower quantiles of \tilde{T} have, as it were, swapped roles in determining the upper and lower confidence limits. Finally, notice that we do not actually know those quantiles exactly, but they're what we approximate by bootstrapping.

This is the **basic bootstrap confidence interval**, or the **pivotal CI**. It is simple and reasonably accurate, and makes a very good default choice for finding confidence intervals.

6.2.3.1 Other Bootstrap Confidence Intervals

The basic bootstrap CI relies on the distribution of $\tilde{T} - \hat{T}$ being approximately the same as that of $\hat{T} - t_0$. Even when this is false, however, it can be that the distribution of

$$\tau = \frac{\hat{T} - t_0}{\text{se}(\hat{T})} \quad (6.10)$$

is close to that of

$$\tilde{\tau} = \frac{\tilde{T} - \hat{T}}{\text{se}(\tilde{T})} \quad (6.11)$$

This is like what we calculate in a t -test, and since the t -test was invented by “Student”, these are called **studentized** quantities. If τ and $\tilde{\tau}$ have the same distribution,

then we can reason as above and get a confidence interval

$$(\hat{t} - \widehat{\text{se}}(\hat{t})Q_{\tilde{\tau}}(1 - \alpha/2), \hat{t} - \widehat{\text{se}}(\hat{t})Q_{\tilde{\tau}}(\alpha/2)) \quad (6.12)$$

This is the same as the basic interval when $\widehat{\text{se}}(\hat{t}) = \text{se}(\hat{t})$, but different otherwise. To find $\text{se}(\tilde{t})$, we need to actually do a *second* level of bootstrapping, as follows.

1. Fit the model with $\hat{\theta}$, find \hat{t} .
2. For $i \in 1 : B_1$
 - (a) Generate \tilde{X}_i from $\hat{\theta}$
 - (b) Estimate $\tilde{\theta}_i$, \tilde{t}_i
 - (c) For $j \in 1 : B_2$
 - i. Generate X_{ij}^\dagger from $\tilde{\theta}_i$
 - ii. Calculate t_{ij}^\dagger
 - (d) Set $\tilde{\sigma}_i = \text{standard deviation of the } t_{ij}^\dagger$
 - (e) Set $\tilde{\tau}_{ij} = \frac{t_{ij}^\dagger - \tilde{t}_i}{\tilde{\sigma}_i}$ for all j
3. Set $\widehat{\text{se}}(\hat{t}) = \text{standard deviation of the } \tilde{t}_i$
4. Find the $\alpha/2$ and $1 - \alpha/2$ quantiles of the distribution of the $\tilde{\tau}$
5. Plug into Eq. 6.12.

The advantage of the studentized intervals is that they are more accurate than the basic ones; the disadvantage is that they are more work! At the other extreme, the **percentile method** simply sets the confidence interval to

$$(Q_{\tilde{\tau}}(\alpha/2), Q_{\tilde{\tau}}(1 - \alpha/2)) \quad (6.13)$$

This is definitely easier to calculate, but not as accurate as the basic, pivotal CI.

All of these methods have many variations, described in the monographs referred to at the end of this chapter.

6.2.4 Hypothesis Testing

For hypothesis tests, we may want to calculate two sets of sampling distributions: the distribution of the test statistic under the null tells us about the size of the test and significance levels, and the distribution under the alternative tells about power and realized power. We can find either with bootstrapping, by simulating from either the null or the alternative. In such cases, the statistic of interest, which I've been calling T , is the test statistic. Code Example 13 illustrates how to find a p -value by simulating under the null hypothesis. The same procedure would work to calculate power, only we'd need to simulate from the alternative hypothesis, and `testthat` would be set to the critical value of T separating acceptance from rejection, not the observed value.

```
boot.pvalue <- function(test,simulator,B,testthat) {
  testboot <- rboot(B=B, statistic=test, simulator=simulator)
  p <- (sum(test >= testthat)+1)/(B+1)
  return(p)
}
```

Code Example 13: Bootstrap p -value calculation. `testthat` should be the value of the test statistic on the actual data. `test` is a function which takes in a data set and calculates the test statistic, under the presumption that large values indicate departure from the null hypothesis. Note the `+1` in the numerator and denominator of the p -value — it would be more straightforward to leave them off, but this is a little more stable when B is comparatively small. (Also, it keeps us from ever reporting a p -value of exactly 0.)

6.2.4.1 Double bootstrap hypothesis testing

When the hypothesis we are testing involves estimated parameters, we may need to correct for this. Suppose, for instance, that we are doing a goodness-of-fit test. If we estimate our parameters on the data set, we adjust our distribution so that it matches the data. It is thus not surprising if it seems to fit the data well! (Essentially, it's the problem of evaluating performance by looking at in-sample fit, which is more or less where we began the course.)

Some test statistics have distributions which are not affected by estimating parameters, at least not asymptotically. In other cases, one can analytically come up with correction terms. When these routes are blocked, one uses a **double bootstrap**, where a second level of bootstrapping checks how much estimation improves the apparent fit of the model. This is perhaps most easily explained in pseudo-code (Code Example 14).

6.2.5 Parametric Bootstrapping Example: Pareto's Law of Wealth Inequality

The Pareto distribution⁴, or power-law distribution, is a popular model for data with “heavy tails”, i.e. where the probability density $f(x)$ goes to zero only very slowly as $x \rightarrow \infty$. The probability density is

$$f(x) = \frac{\theta - 1}{x_0} \left(\frac{x}{x_0} \right)^{-\theta} \quad (6.14)$$

where x_0 is the minimum scale of the distribution, and θ is the **scaling exponent**. (EXERCISE: show that x_0 is the mode of the distribution.) The Pareto is highly right-skewed, with the mean being much larger than the median.

⁴Named after Vilfredo Pareto, the highly influential late-19th/early-20th century economist, political scientist, and proto-Fascist.

```

doubleboot.pvalue <- function(test,simulator,B1,B2,
  estimator, thetahat, testthat) {
  for (i in 1:B1) {
    xboot <- simulator(theta=thetahat, ...)
    thetaboot <- estimator(xboot)
    testboot[i] <- test(xboot)
    pboot[i] <- boot.pvalue(test,simulator,B2,
      testthat=testboot[i],theta=thetaboot)
  }
  p <- (sum(testboot >= testthat)+1)/(B1+1)
  p.adj <- (sum(pboot <= p)+1)/(B1+1)
}

```

Code Example 14: Code sketch for “double bootstrap” significance testing. The inner or second bootstrap is used to calculate the distribution of nominal bootstrap p-values. For this to work, we need to draw our second-level bootstrap samples from $\tilde{\theta}$, the bootstrap re-estimate, not from $\hat{\theta}$, the data estimate. The code presumes the `simulator` function takes a `theta` argument allowing this.

If we know x_0 , one can show that the maximum likelihood estimator of the exponent θ is

$$\hat{\theta} = 1 + \frac{n}{\sum_{i=1}^n \log \frac{x_i}{x_0}} \quad (6.15)$$

and that this is consistent⁵, and efficient. Picking x_0 is a harder problem (see Clauset *et al.* 2009) — for the present purposes, pretend that the Oracle tells us. The file `pareto.R`, on the class website, contains a number of functions related to the Pareto distribution, including a function `pareto.fit` for estimating it. (There’s an example of its use below.)

Pareto came up with this density when he attempted to model the distribution of wealth. Approximately, but quite robustly across countries and time-periods, the upper tail of the distribution of income and wealth follows a power law, with the exponent varying as money is more or less concentrated among the very richest⁶. Figure 6.2 shows the distribution of net worth for the 400 richest Americans in 2003. Taking $x_0 = 9 \times 10^8$ (again, see Clauset *et al.* 2009), the number of individuals in the tail is 302, and the estimated exponent is $\hat{\theta} = 2.34$.

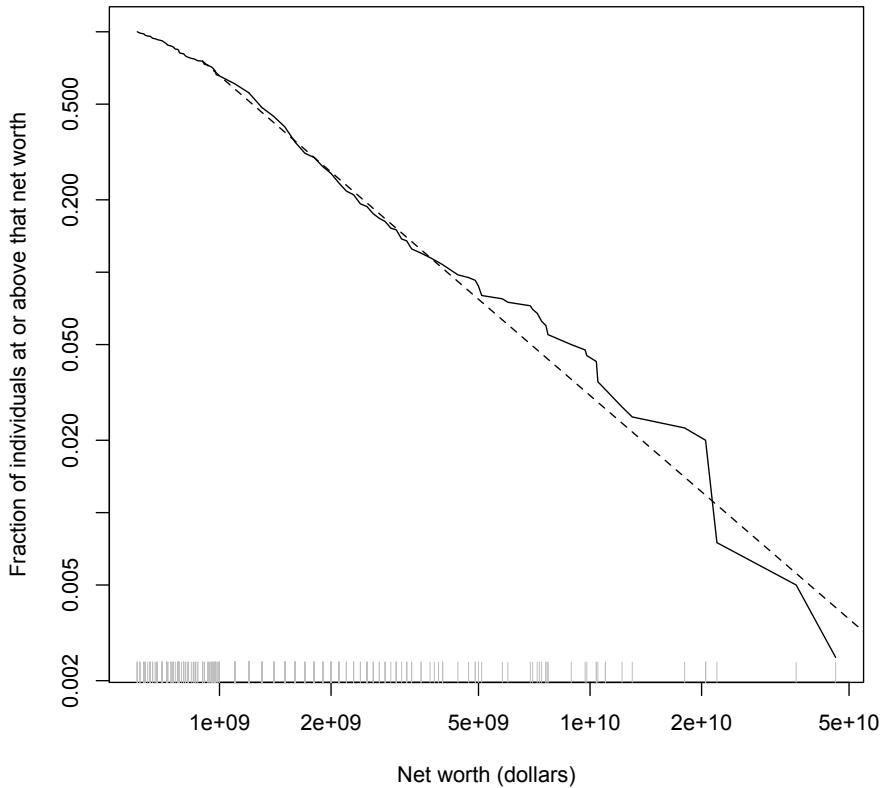
```

> source("pareto.R")
> wealth <- scan("wealth.dat")
> wealth.pareto <- pareto.fit(wealth,threshold=9e8)
> signif(wealth.pareto$exponent,3)
[1] 2.34

```

⁵Because the sample mean of $\log X$ converges, under the law of large numbers

⁶Most of the distribution conforms to a log-normal, at least roughly.



```
plot.survival.loglog(wealth,xlab="Net worth (dollars)",
    ylab="Fraction of individuals at or above that net worth")
rug(wealth,side=1,col="grey")
curve((302/400)*ppareto(x,threshold=9e8,exponent=2.34,lower.tail=FALSE),
    add=TRUE,lty=2,from=9e8,to=2*max(wealth))
```

Figure 6.2: Upper cumulative distribution function (or “survival function”) of net worth for the 400 richest individuals in the US (2000 data). The solid line shows the fraction of the 400 individuals whose net worth W equaled or exceeded a given value w , $\Pr(W \geq w)$. (Note the logarithmic scale for both axes.) The dashed line is a maximum-likelihood estimate of the Pareto distribution, taking $x_0 = \$9 \times 10^8$. (This threshold was picked using the method of Clauset *et al.* 2009.) Since there are 302 individuals at or above the threshold, the cumulative distribution function of the Pareto has to be reduced by a factor of $(302/400)$.

```
rboot.pareto <- function(B,exponent,x0,n) {
  replicate(B,pareto.fit(rpareto(n,x0,exponent),x0)$exponent)
}

pareto.se <- function(B,exponent,x0,n) {
  return(sd(rboot.pareto(B,exponent,x0,n)))
}

pareto.bias <- function(B,exponent,x0,n) {
  return(mean(rboot.pareto(B,exponent,x0,n)) - exponent)
}
```

Code Example 15: Standard error and bias calculation for the Pareto distribution, using parametric bootstrapping.

How much uncertainty is there in this estimate of the exponent? Naturally, we'll bootstrap. We need a function to generate Pareto-distributed random variables; this, along with some related functions, is part of the file `pareto.R` on the course website. With that tool, parametric bootstrapping proceeds as in Code Example 15.

With $\hat{\theta} = 2.34$, $x_0 = 9 \times 10^8$, $n = 302$ and $B = 10^4$, this gives a standard error of ± 0.077 . This matches some asymptotic theory reasonably well⁷, but didn't require asymptotic assumptions.

Asymptotically, the bias is known to go to zero; at this size, bootstrapping gives a bias of 3×10^{-3} , which is effectively negligible.

We can also get the confidence interval (Code Example 16). Using, again, 10^4 bootstrap replications, the 95% CI is $(2.16, 2.47)$. In theory, the confidence interval could be calculated exactly, but it involves the inverse gamma distribution (Arnold, 1983), and it is quite literally faster to write and do the bootstrap than go to look it up.

A more challenging problem is goodness-of-fit; we'll use the Kolmogorov-Smirnov statistic.⁸ Code Example 17 calculates the p -value. With ten thousand bootstrap replications,

```
> ks.pvalue.pareto(1e4,wealth,2.34,9e8)
```

⁷In Asympotopia", the variance of the MLE should be $\frac{(\hat{\theta}-1)^2}{n}$, in this case 0.076. The intuition is that this variance depends on how sharp the maximum of the likelihood function is — if it's sharply peaked, we can find the maximum very precisely, but a broad maximum is hard to pin down. Variance is thus inversely proportional to the second derivative of the negative log-likelihood. (The minus sign is because the second derivative has to be negative at a maximum, while variance has to be positive.) For one sample, the expected second derivative of the negative log-likelihood is $(\theta-1)^{-2}$. (This is called the **Fisher information** of the model.) Log-likelihood adds across independent samples, giving us an over-all factor of n . In the large-sample limit, the actual log-likelihood will converge on the expected log-likelihood, so this gives us the asymptotic variance.

⁸The `pareto.R` file contains a function, `pareto.tail.ks.test`, which does a goodness-of-fit test for fitting a power-law to the tail of the distribution. That differs somewhat from what follows, because it takes into account the extra uncertainty which comes from having to estimate x_0 . Here, I am pretending that an Oracle told us $x_0 = 9 \times 10^8$.

```
pareto.ci <- function(B,exponent,x0,n,alpha) {
  tboot <- rboot.pareto(B,exponent,x0,n)
  ci.lower <- 2*exponent - quantile(tboot,1-alpha/2)
  ci.upper <- 2*exponent - quantile(tboot,alpha/2)
  return(list(ci.lower=ci.lower, ci.upper=ci.upper))
}
```

Code Example 16: Parametric bootstrap confidence interval for the Pareto scaling exponent.

```
[1] 0.0119988
```

Ten thousand replicates is enough that we should be able to accurately estimate probabilities of around 0.01 (since the binomial standard error will be $\sqrt{\frac{0.01}{0.99}} 10^4 \approx 9.9 \times 10^{-4}$; if it weren't, we might want to increase B).

Simply plugging in to the standard formulas, and thereby ignoring the effects of estimating the scaling exponent, gives a p -value of 0.16, which is not outstanding but not awful either. Properly accounting for the flexibility of the model, however, the discrepancy between what it predicts and what the data shows is so large that it would take an awfully big (one-a-hundred) coincidence to produce it.

We have, therefore, detected that the Pareto distribution makes systematic errors for this data, but we don't know much about what they are. In Chapter 16, we'll look at techniques which can begin to tell us something about *how* it fails.

6.3 Non-parametric Bootstrapping

The bootstrap approximates the sampling distribution, with three sources of approximation error. First, **simulation error**: using finitely many replications to stand for the full sampling distribution. Clever simulation design can shrink this, but brute force — just using enough replicates — can also make it arbitrarily small. Second, **statistical error**: the sampling distribution of the bootstrap re-estimates under our estimated model is not exactly the same as the sampling distribution of estimates under the true data-generating process. The sampling distribution changes with the parameters, and our initial estimate is not completely accurate. But it often turns out that distribution of estimates *around* the truth is more nearly invariant than the distribution of estimates themselves, so subtracting the initial estimate from the bootstrapped values helps reduce the statistical error; there are many subtler tricks to the same end. Third, **specification error**: the data source doesn't exactly follow our model at all. Simulating the model then never quite matches the actual sampling distribution.

Efron had a second brilliant idea, which is to address specification error by replacing simulation from the model with re-sampling from the data. After all, our initial collection of data gives us a lot of information about the relative probabilities of different values. In a sense the empirical distribution is the least prejudiced estimate possible of the underlying distribution — anything else imposes biases or

```

ks.stat.pareto <- function(data, exponent, x0) {
  data <- data[data>=x0]
  ks <- ks.test(data, ppareto, exponent=exponent,
    threshold=x0)
  return(ks$statistic)
}

ks.pvalue.pareto <- function(B, data, exponent, x0) {
  testthat <- ks.stat.pareto(data, exponent, x0)
  testboot <- vector(length=B)
  for (i in 1:B) {
    xboot <- rpareto(length(data), exponent=exponent,
      threshold=x0)
    exp.boot <- ppareto.fit(xboot, threshold=x0)$exponent
    testboot[i] <- ks.stat.pareto(xboot, exp.boot, x0)
  }
  p <- (sum(testboot >= testthat)+1)/(B+1)
  return(p)
}

```

Code Example 17: Calculating a p -value for the Pareto distribution, using the Kolmogorov-Smirnov test and adjusting for the way estimating the scaling exponent moves the fitted distribution closer to the data.

pre-conceptions, possibly accurate but also potentially misleading⁹. Lots of quantities can be estimated directly from the empirical distribution, without the mediation of a parametric model. Efron's **non-parametric bootstrap** treats the original data set as a complete population and draws a new, simulated sample from it, picking each observation with equal probability (allowing repeated values) and then re-running the estimation (Figure 6.3). In fact, this is usually what people mean when they talk about "the bootstrap" without any modifier.

Everything we did with parametric bootstrapping can also be done with non-parametric bootstrapping — the only thing that's changing is the distribution the surrogate data is coming from.

The non-parametric bootstrap should remind you of k -fold cross-validation. The analog of leave-one-out CV is a procedure called the **jack-knife**, where we repeat the estimate n times on $n - 1$ of the data points, holding each one out in turn. It's historically important (it dates back to the 1940s), but generally doesn't work as well as the non-parametric bootstrap.

An important variant is the **smoothed bootstrap**, where we re-sample the data points and then perturb each by a small amount of noise, generally Gaussian¹⁰.

Code Example 18 shows how to use re-sampling to get a 95% confidence interval

⁹See §15.6 in Chapter 15.

¹⁰We will see in Chapter 15 that this corresponds to sampling from a kernel density estimate

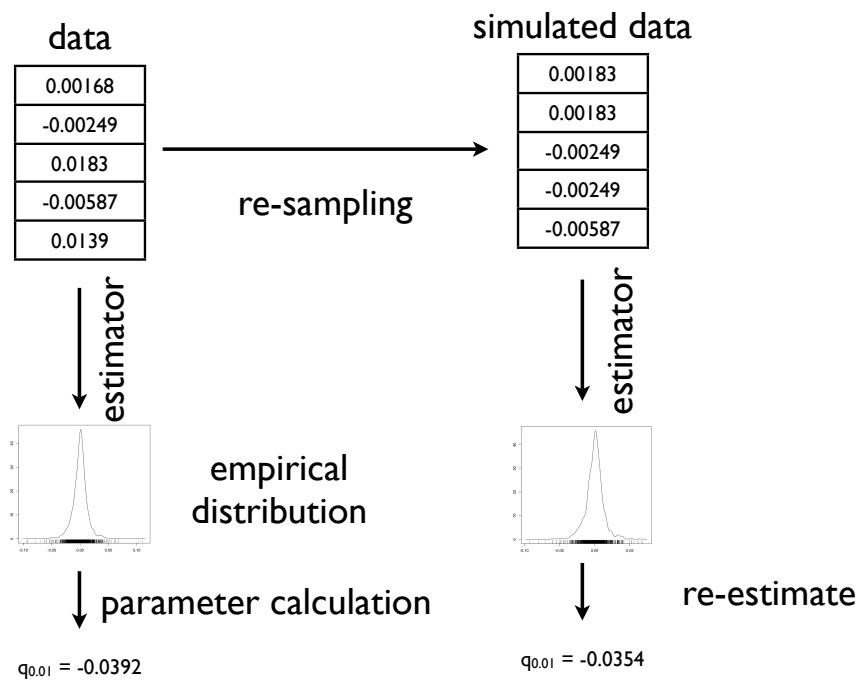


Figure 6.3: Schematic for non-parametric bootstrapping. New data is simulated by re-sampling from the original data (with replacement), and parameters are calculated either directly from the empirical distribution, or by applying a model to this surrogate data.

```

resample <- function(x) {
  sample(x, size=length(x), replace=TRUE)
}

resamp.pareto <- function(B, data, x0) {
  replicate(B,
    pareto.fit(resample(data), threshold=x0)$exponent)
}

resamp.pareto.CI <- function(B, data, alpha, x0) {
  thetahat <- pareto.fit(data, threshold=x0)$exponent
  thetaboot <- resamp.pareto(B, data, x0)
  ci.lower <- 2*thetahat - quantile(thetaboot, 1-alpha/2)
  ci.upper <- 2*thetahat - quantile(thetaboot, alpha/2)
  return(list(ci.lower=ci.lower, ci.upper=ci.upper))
}

```

Code Example 18: Non-parametric bootstrap confidence intervals for the Pareto scaling exponent.

for the Pareto exponent¹¹. With $B = 10^4$, it gives the 95% confidence interval for the scaling exponent as (2.18, 2.48). The fact that this is very close to the interval we got from parametric bootstrapping should actually reassure us about its validity.

6.3.1 Parametric vs. Nonparametric Bootstrapping

When we have a properly specified model, simulating from the model gives more accurate results (at the same n) than does re-sampling the empirical distribution — parametric estimates of the distribution converge faster than the empirical distribution does. If on the other hand the parametric model is mis-specified, then it is rapidly converging to the *wrong* distribution. This is of course just another bias-variance trade-off, like those we've seen in regression.

Since I am suspicious of most parametric modeling assumptions, I prefer re-sampling, when I can figure out how to do it, or at least until I have convinced myself that a parametric model is good approximation to reality.

6.4 Bootstrapping Regression Models

With a regression model, which is fit to a set of input-output pairs, $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, resulting in a regression curve (or surface) $\hat{r}(x)$, fitted values $\hat{y}_i = \hat{r}(x_i)$, and residuals,

¹¹Even if the Pareto model is wrong, the estimator of the exponent will converge on the value which gives, in a certain sense, the best approximation to the true distribution from among all power laws. Econometricians call such parameter values the **pseudo-true**; we are getting a confidence interval for the pseudo-truth. In this case, the pseudo-true scaling exponent can still be a useful way of summarizing *how* heavy tailed the income distribution is, despite the fact that the power law makes systematic errors.

$\epsilon_i = y_i - \hat{y}_i = \hat{r}(x_i)$, we have a choice of several ways of bootstrapping, in decreasing order of relying on the model.

- Simulate new X values from the model's distribution of X , and then draw Y from the specified conditional distribution $Y|X$.
- Hold the x fixed, but draw $Y|X$ from the specified distribution.
- Hold the x fixed, but make Y equal to $\hat{r}(x)$ plus a randomly re-sampled ϵ_j .
- Re-sample (x, y) pairs.

The first case is pure parametric bootstrapping. (So is the second, sometimes, when the regression model is agnostic about X .) The last case is just re-sampling from the joint distribution of (X, Y) . The next-to-last case is called **re-sampling the residuals** or **re-sampling the errors**. When we do that, we rely on the regression model to get the conditional expectation function right, but we don't count on it getting the distribution of the noise around the expectations.

The specific procedure of re-sampling the residuals is to re-sample the ϵ_i , with replacement, to get $\tilde{\epsilon}_1, \tilde{\epsilon}_2, \dots, \tilde{\epsilon}_n$, and then set $\tilde{x}_i = x_i$, $\tilde{y}_i = \hat{r}(\tilde{x}_i) + \tilde{\epsilon}_i$. This surrogate data set is then re-analyzed like new data.

6.4.1 Re-sampling Points: Parametric Example

A classic data set contains the time between 299 eruptions of the Old Faithful geyser in Yellowstone, and the length of the subsequent eruptions; these variables are called `waiting` and `duration`. We'll look at the linear regression of `waiting` on `duration`. We'll re-sample (`duration`, `waiting`) pairs, and would like confidence intervals for the regression coefficients. This is a confidence interval for the coefficients of *best liner predictor*, a functional of the distribution, which, as we saw in Chapters 1 and 2, exists no matter how nonlinear the process really is. It's only a confidence interval for the *true regression parameters* if the real regression function is linear.

Before anything else, look at the model:

```
library(MASS)
data(geyser)
geyser.lm <- lm(waiting~duration,data=geyser)
summary(geyser.lm)
```

The first step in bootstrapping this is to build our simulator, which just means sampling rows from the data frame:

```
resample.geyser <- function() {
  sample.rows <- resample(1:nrow(geyser))
  return(sample.rows)
}
```

```
geyser.lm.cis <- function(B,alpha) {
  tboot <- replicate(B,
    est.waiting.on.duration(resample.geyser()))
  low.quantiles <- apply(tboot,1,quantile,probs=alpha/2)
  high.quantiles <- apply(tboot,1,quantile,probs=1-alpha/2)
  low.cis <- 2*coefficients(geyser.lm) - high.quantiles
  high.cis <- 2*coefficients(geyser.lm) - low.quantiles
  cis <- rbind(low.cis,high.cis)
  rownames(cis) <- as.character(c(alpha/2,1-alpha/2))
  return(cis)
}
```

Code Example 19: Bootstrapped confidence intervals for the linear model of Old Faithful, based on re-sampling data points. Relies on functions defined in the text.

We can check this by running `summary(geyser[resample.geyser(),])`, and seeing that it gives about the same quartiles and mean for both variables as `summary(geyser)`¹².

Next, we define the estimator:

```
est.waiting.on.duration <- function(subset,data=geyser) {
  fit <- lm(waiting ~ duration, data=data,subset=subset)
  return(coefficients(fit))
}
```

This exploits the fact that `lm()`, like many model-estimation functions, can take as an optional argument a vector of row numbers (`subset`), and look only at those rows of the data. We can check that this function works by seeing that `est.waiting.on.duration(1:nrow(geyser))` gives the same results as `coefficients(geyser.lm)`.

Putting the pieces together according to the basic confidence interval recipe (Code Example 19), we get

```
> signif(geyser.lm.cis(B=1e4,alpha=0.05),3)
(Intercept) duration
 0.025      96.5     -8.70
 0.975     102.0     -6.92
```

Notice that we do not have to assume homoskedastic Gaussian noise — fortunately, because that's a very bad assumption here¹³.

¹²The minimum and maximum won't match up well — why not?

¹³We have calculated 95% confidence intervals for the intercept β_0 and the slope β_1 separately. These intervals cover their coefficients all but 5% of the time. Taken together, they give us a rectangle in (β_0, β_1) space, but the coverage probability of *this* rectangle could be anywhere from 95% all the way down to 90%. To get a confidence *region* which simultaneously covers both coefficients 95% of the time, we have two big options. One is to stick to a box-shaped region and just increase the confidence level on each coordinate (to 97.5%). The other is to define some suitable metric of how far apart coefficient vectors are (e.g., ordinary Euclidean distance), find the 95% percentile of the distribution of this metric, and trace the appropriate contour around $\hat{\beta}_0, \hat{\beta}_1$.

6.4.2 Re-sampling Points: Non-parametric Example

Nothing in the logic of re-sampling data points for regression requires us to use a parametric model. Here we'll provide 95% confidence bounds for the kernel smoothing of the geyser data. Since the functional is a whole curve, the confidence set is often called a **confidence band**.

We use the same simulator, but start with a different regression curve, and need a different estimator.

```
library(np)
npr.waiting.on.duration <- function(subset,data=geyser,tol=0.1,ftol=0.1) {
  bw <- npregbw(waiting ~ duration, data=data, subset=subset,
    tol=tol, ftol=ftol)
  fit <- npreg(bw)
  return(fit)
}
geyser.npr <- npr.waiting.on.duration(1:nrow(geyser))
```

Now we construct pointwise 95% confidence bands for the regression curve. For this end, we don't really need to keep around the whole kernel regression object — we'll just use its predicted values on a uniform grid of points, extending slightly beyond the range of the data (Code Example 20). Observe that this will go through bandwidth selection again for each bootstrap sample. This is slow, but it is the most secure way of getting good confidence bands. Applying the bandwidth we found on the data to each re-sample would be faster, but would introduce an extra level of approximation, since we wouldn't be treating each simulation run the same as the original data.

Figure 6.4 shows the curve fit to the data, the 95% confidence limits, and (faintly) all of the bootstrapped curves. Doing the 800 bootstrap replicates took 4 minutes on my laptop¹⁴.

¹⁴Specifically, I ran `system.time(geyser.npr.cis <- npr.cis(B=800,alpha=0.05))`, which not only did the calculations and stored them in `geyser.npr.cis`, but told me how much time it took R to do them.

```

evaluation.points <- seq(from=0.8,to=5.5,length.out=200)
evaluation.points <- data.frame(duration=evaluation.points)

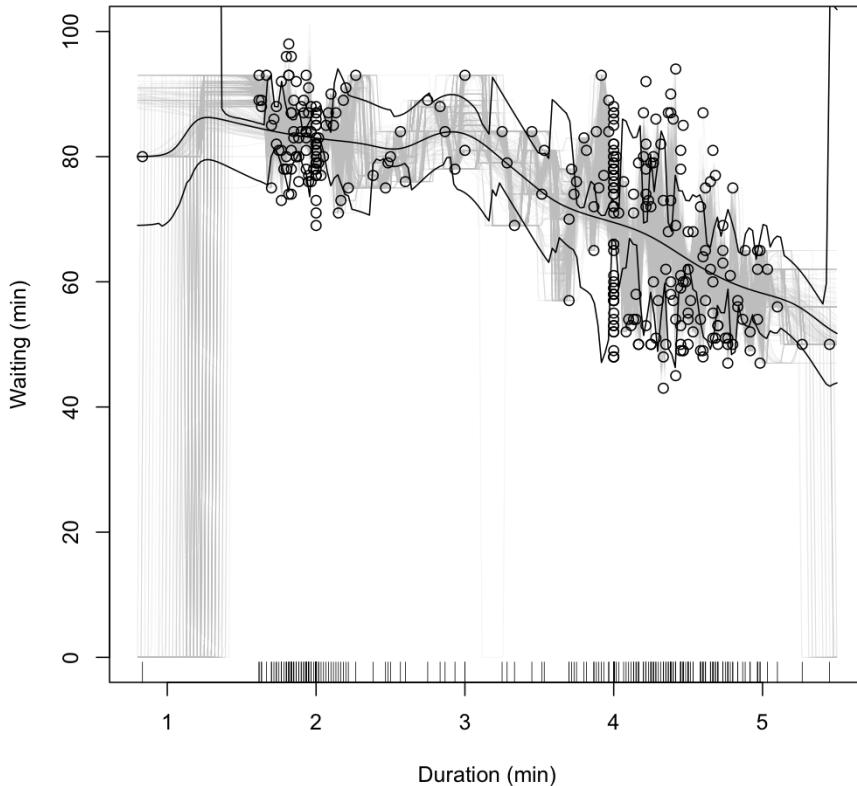
eval.npr <- function(npr) {
  return(predict(npr,newdata=evaluation.points))
}

main.curve <- eval.npr(geyser.npr)

npr.cis <- function(B,alpha) {
  tboot <- replicate(B,
    eval.npr(npr.waiting.on.duration(resample.geyser())))
  low.quantiles <- apply(tboot,1,quantile,probs=alpha/2)
  high.quantiles <- apply(tboot,1,quantile,probs=1-alpha/2)
  low.cis <- 2*main.curve - high.quantiles
  high.cis <- 2*main.curve - low.quantiles
  cis <- rbind(low.cis,high.cis)
  return(list(cis=cis,tboot=t(tboot)))
}

```

Code Example 20: Finding confidence bands around the kernel regression model of Old Faithful by re-sampling data points. Notice that much of `npr.cis` is the same as `geyser.lm.cis`, and the other functions for calculating confidence intervals. It would be better programming practice to extract the common find-the-confidence-limits part as a separate function, which could be called as needed. (Taking the transpose of the `tboot` matrix at the end is just so that it has the same orientation as the matrix of confidence limits.)



```

geyser.npr.cis <- npr.cis(B=800,alpha=0.05)
plot(0,type="n",xlim=c(0.8,5.5),ylim=c(0,100),
     xlab="Duration (min)", ylab="Waiting (min)")
for (i in 1:800) {
  lines(evaluation.points$duration,geyser.npr.cis$tboot[i,],
        lwd=0.1,col="grey")
}
lines(evaluation.points$duration,geyser.npr.cis$cis[1,])
lines(evaluation.points$duration,geyser.npr.cis$cis[2,])
lines(evaluation.points$duration,main.curve)
rug(geyser$duration,side=1)
points(geyser$duration,geyser$waiting)

```

Figure 6.4: Kernel regression curve for Old Faithful (central black line), with 95% confidence bands (other black lines), the 800 bootstrapped curves (thin, grey lines), and the data points. Notice that the confidence bands get wider where there is less data. *Caution:* doing the bootstrap took 4 minutes to run on my computer.

14:41 Thursday 25th April, 2013

6.4.3 Re-sampling Residuals: Example

As an example of re-sampling the residuals, rather than data points, let's take a linear regression, from the homework using the Penn World Tables, of gdp.growth on log(gdp), pop.growth, invest and trade:

```
penn <- read.csv("http://www.stat.cmu.edu/~cshalizi/uADA/13/hw/02/penn-select.csv")
penn.formula <- "gdp.growth ~ log(gdp) + pop.growth + invest + trade"
penn.lm <- lm(penn.formula, data=penn)
```

(Why make the formula a separate object here?) The estimated parameters are

```
> signif(coefficients(penn.lm),3)
(Intercept)    log(gdp)   pop.growth      invest      trade
  5.71e-04    5.07e-04   -1.87e-01    7.15e-04   3.11e-05
```

Code Example 21 shows the new simulator for this set-up (`resample.residuals.penn`)¹⁵, the new estimation function (`penn.estimator`)¹⁶, and the confidence interval calculation (`penn.lm.cis`).

Which delivers our confidence intervals:

```
> signif(penn.lm.cis(1e4,0.05),3)
(Intercept) log(gdp) pop.growth      invest      trade
low.cis     -0.0153 -0.00151    -0.358 0.000499 -2.00e-05
high.cis     0.0175  0.00240    -0.021 0.000937  8.19e-05
```

Doing ten thousand linear regressions took 45 seconds on my computer, as opposed to 4 minutes for eight hundred kernel regressions.

¹⁵How would you check that this was working right?

¹⁶How would you check that this was working right?

```
resample.residuals.penn <- function() {  
  new.frame <- penn  
  new.growths <- fitted(penn.lm) +  
    resample(residuals(penn.lm))  
  new.frame$gdp.growth <- new.growths  
  return(new.frame)  
}  
  
penn.estimator <- function(data) {  
  fit <- lm(penn.formula, data=data)  
  return(coefficients(fit))  
}  
  
penn.lm.cis <- function(B,alpha) {  
  tboot <- replicate(B,  
    penn.estimator(resample.residuals.penn()))  
  low.quantiles <- apply(tboot,1,quantile,probs=alpha/2)  
  high.quantiles <- apply(tboot,1,quantile,probs=1-alpha/2)  
  low.cis <- 2*coefficients(penn.lm) - high.quantiles  
  high.cis <- 2*coefficients(penn.lm) - low.quantiles  
  cis <- rbind(low.cis,high.cis)  
  return(cis)  
}
```

Code Example 21: Re-sampling the residuals to get confidence intervals in a linear model.

6.5 Bootstrap with Dependent Data

If the data point we are looking at are vectors (or more complicated structures) with dependence between components, but each data point is independently generated from the same distribution, then dependence isn't really an issue. We re-sample vectors, or generate vectors from our model, and proceed as usual. In fact, that's what we've done so far in several cases.

If there is dependence *across* data points, things are more tricky. If our model incorporates this dependence, then we can just simulate whole data sets from it. An appropriate re-sampling method is trickier — just re-sampling individual data points destroys the dependence, so it won't do. We will revisit this question when we look at time series and spatial data in Chapters 25–27.

6.6 Things Bootstrapping Does Poorly

The principle behind bootstrapping is that sampling distributions under the true process should be close to sampling distributions under good estimates of the truth. If small perturbations to the data-generating process produce huge swings in the sampling distribution, bootstrapping will not work well, and may fail spectacularly. For parametric bootstrapping, this means that small changes to the underlying parameters must produce small changes to the functionals of interest. Similarly, for non-parametric bootstrapping, it means that adding or removing a few data points must change the functionals only a little¹⁷.

Re-sampling in particular has trouble with extreme values. Here is a simple example: Our data points X_i are IID, with $X_i \sim \text{Unif}(0, \theta_0)$, and we want to estimate θ_0 . The maximum likelihood estimate $\hat{\theta}$ is just the sample maximum of the x_i . We'll use the non-parametric bootstrap to get a confidence interval for this, as above — but I will fix the true $\theta_0 = 1$, and see how often the 95% confidence interval covers the truth.

```
x <- runif(100)
is.covered <- function() {
  max.boot <- replicate(1e3,max(resample(x)))
  all(1 >= 2*max(x) - quantile(max.boot,0.975),
      1 <= 2*max(x) - quantile(max.boot,0.025))
}
sum(replicate(1000,is.covered()))
```

When I run the last line, I get 19, so the true coverage probability is not 95% but 1.9%.

If you suspect that your use of the bootstrap may be setting yourself up for a similar epic fail, your two options are (1) learn some of the theory of the bootstrap

¹⁷More generally, moving from one distribution function f to another $(1 - \epsilon)f + \epsilon g$ mustn't change the functional very much when ϵ is small, no matter in what "direction" g we perturb it. Making this idea precise calls for some fairly deep mathematics, about differential calculus on spaces of functions.

from the references in the “Further Reading” section below, or (2) set up a simulation experiment like this one.

6.7 Further Reading

The original paper on the bootstrap, Efron (1979), is extremely clear, and for the most part presented in the simplest possible terms; it’s worth reading. His later small book (Efron, 1982), while often cited, is not in my opinion so useful nowadays¹⁸. Davison and Hinkley (1997) is both a good textbook, and the reference I consult most often; the CRAN package `boot` is based on the code written for this book. Efron and Tibshirani (1993), while also very good, is more theoretical. Canty *et al.* (2006) has useful advice for serious applications.

6.8 Exercises

To think through, not to hand in.

1. Derive the maximum likelihood estimator for the Pareto distribution (Eq. 6.15) from the density (Eq. 6.14).
2. Find confidence bands for the linear regression model of using
 - (a) The usual Gaussian assumptions (*hint*: try the `intervals="confidence"` option to `predict`)
 - (b) Resampling of residuals
 - (c) Resampling of cases

¹⁸It seems to have done a good job of explaining things to people who were already professional statisticians in 1982.

Chapter 7

Moving Beyond Conditional Expectations: Weighted Least Squares, Heteroskedasticity, Local Polynomial Regression

So far, all our estimates have been based on the mean squared error, giving equal importance to all observations. This is appropriate for looking at conditional expectations. In this chapter, we'll start to work with giving more or less weight to different observations. On the one hand, this will let us deal with other aspects of the distribution beyond the conditional expectation, especially the conditional variance. First we look at weighted least squares, and the effects that ignoring heteroskedasticity can have. This leads naturally to trying to estimate variance functions, on the one hand, and generalizing kernel regression to local polynomial regression, on the other.

7.1 Weighted Least Squares

When we use ordinary least squares to estimate linear regression, we (naturally) minimize the mean squared error:

$$MSE(\beta) = \frac{1}{n} \sum_{i=1}^n (y_i - \vec{x}_i \cdot \beta)^2 \quad (7.1)$$

The solution is of course

$$\hat{\beta}_{OLS} = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y} \quad (7.2)$$

We could instead minimize the *weighted* mean squared error,

$$WMSE(\beta, \vec{w}) = \frac{1}{n} \sum_{i=1}^n w_i (y_i - \vec{x}_i \cdot \beta)^2 \quad (7.3)$$

This includes ordinary least squares as the special case where all the weights $w_i = 1$. We can solve it by the same kind of linear algebra we used to solve the ordinary linear least squares problem. If we write \mathbf{w} for the matrix with the w_i on the diagonal and zeroes everywhere else, the solution is

$$\hat{\beta}_{WLS} = (\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w} \mathbf{y} \quad (7.4)$$

But why would we want to minimize Eq. 7.3?

1. *Focusing accuracy.* We may care very strongly about predicting the response for certain values of the input — ones we expect to see often again, ones where mistakes are especially costly or embarrassing or painful, etc. — than others. If we give the points \vec{x}_i near that region big weights w_i , and points elsewhere smaller weights, the regression will be pulled towards matching the data in that region.
2. *Discounting imprecision.* Ordinary least squares is the maximum likelihood estimate when the ϵ in $Y = \vec{X} \cdot \beta + \epsilon$ is IID Gaussian white noise. This means that the variance of ϵ has to be constant, and we measure the regression curve with the same precision elsewhere. This situation, of constant noise variance, is called **homoskedasticity**. Often however the magnitude of the noise is not constant, and the data are **heteroskedastic**.

When we have heteroskedasticity, even if each noise term is still Gaussian, ordinary least squares is no longer the maximum likelihood estimate, and so no longer efficient. If however we know the noise variance σ_i^2 at each measurement i , and set $w_i = 1/\sigma_i^2$, we get the heteroskedastic MLE, and recover efficiency. (See below.)

To say the same thing slightly differently, there's just no way that we can estimate the regression function as accurately where the noise is large as we can where the noise is small. Trying to give equal attention to all parts of the input space is a waste of time; we should be more concerned about fitting well where the noise is small, and expect to fit poorly where the noise is big.

3. *Doing something else.* There are a number of other optimization problems which can be transformed into, or approximated by, weighted least squares. The most important of these arises from generalized linear models, where the mean response is some nonlinear function of a linear predictor; we will look at them in Chapters 12 and 13.

In the first case, we decide on the weights to reflect our priorities. In the third case, the weights come from the optimization problem we'd really rather be solving. What about the second case, of heteroskedasticity?

7.2 Heteroskedasticity

Suppose the noise variance is itself variable. For example, the figure shows a simple linear relationship between the input X and the response Y , but also a nonlinear

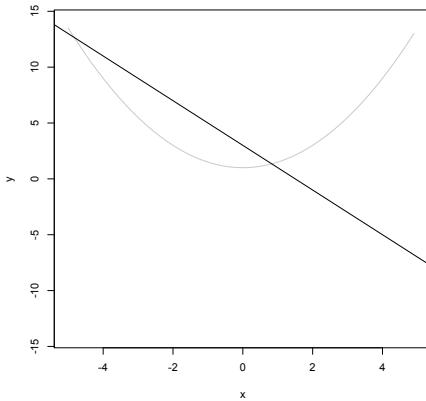


Figure 7.1: Black line: Linear response function ($y = 3 - 2x$). Grey curve: standard deviation as a function of x ($\sigma(x) = 1 + x^2/2$).

relationship between X and $\text{Var}[Y]$.

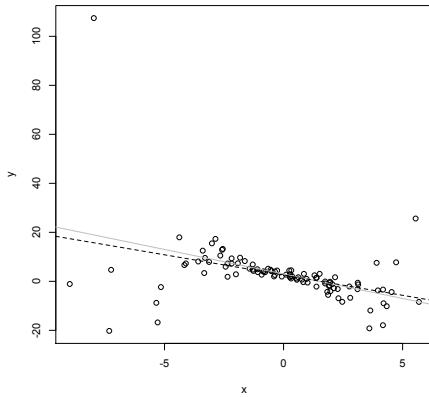
In this particular case, the ordinary least squares estimate of the regression line is $2.56 - 1.65x$, with R reporting standard errors in the coefficients of ± 0.52 and 0.20 , respectively. Those are however calculated under the assumption that the noise is homoskedastic, which it isn't. And in fact we can see, pretty much, that there is heteroskedasticity — if looking at the scatter-plot didn't convince us, we could always plot the residuals against x , which we should do anyway.

To see whether that makes a difference, let's re-do this many times with different draws from the same model (Example 22).

Running `ols.heterosked.error.stats(100)` produces 10^4 random samples which all have the same x values as the first one, but different values of y , generated however from the same model. It then uses those samples to get the standard error of the ordinary least squares estimates. (Bias remains a non-issue.) What we find is the standard error of the intercept is only a little inflated (simulation value of 0.64 versus official value of 0.52), but the standard error of the slope is much larger than what R reports, 0.46 versus 0.20. Since the intercept is fixed by the need to make the regression line go through the center of the data, the real issue here is that our estimate of the slope is much less precise than ordinary least squares makes it out to be. Our estimate is still consistent, but not as good as it was when things were homoskedastic. Can we get back some of that efficiency?

7.2.1 Weighted Least Squares as a Solution to Heteroskedasticity

Suppose we visit the Oracle of Regression (Figure 7.4), who tells us that the noise has a standard deviation that goes as $1 + x^2/2$. We can then use this to improve our regression, by solving the weighted least squares problem rather than ordinary least



```

x = rnorm(100,0,3)
y = 3-2*x + rnorm(100,0,sapply(x,function(x){1+0.5*x^2}))
plot(x,y)
abline(a=3,b=-2,col="grey")
fit.ols = lm(y~x)
abline(fit.ols,lty=2)

```

Figure 7.2: Scatter-plot of $n = 100$ data points from the above model. (Here $X \sim \mathcal{N}(0, 9)$.) Grey line: True regression line. Dashed line: ordinary least squares regression line.

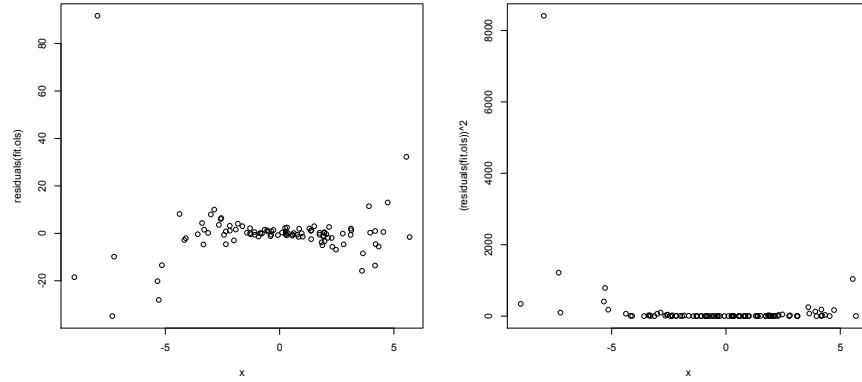
```

ols.heterosked.example = function(n) {
  y = 3-2*x + rnorm(n,0,sapply(x,function(x){1+0.5*x^2}))
  fit.ols = lm(y~x)
  # Return the errors
  return(fit.ols$coefficients - c(3,-2))
}

ols.heterosked.error.stats = function(n,m=10000) {
  ols.errors.raw = t(replicate(m,ols.heterosked.example(n)))
  # transpose gives us a matrix with named columns
  intercept.sd = sd(ols.errors.raw[, "(Intercept)"])
  slope.sd = sd(ols.errors.raw[, "x"])
  return(list(intercept.sd=intercept.sd,slope.sd=slope.sd))
}

```

Code Example 22: Functions to generate heteroskedastic data and fit OLS regression to it, and to collect error statistics on the results.



```
plot(x,residuals(fit.ols))
plot(x,(residuals(fit.ols))^2)
```

Figure 7.3: Residuals (left) and squared residuals (right) of the ordinary least squares regression as a function of x . Note the much greater range of the residuals at large absolute values of x than towards the center; this changing dispersion is a sign of heteroskedasticity.

squares (Figure 7.5).

This not only looks better, it is better: the estimated line is now $2.67 - 1.91x$, with reported standard errors of 0.29 and 0.18. Does this check out with simulation? (Example 23.)

The standard errors from the simulation are 0.22 for the intercept and 0.23 for the slope, so R's internal calculations are working very well.

Why does putting these weights into WLS improve things?

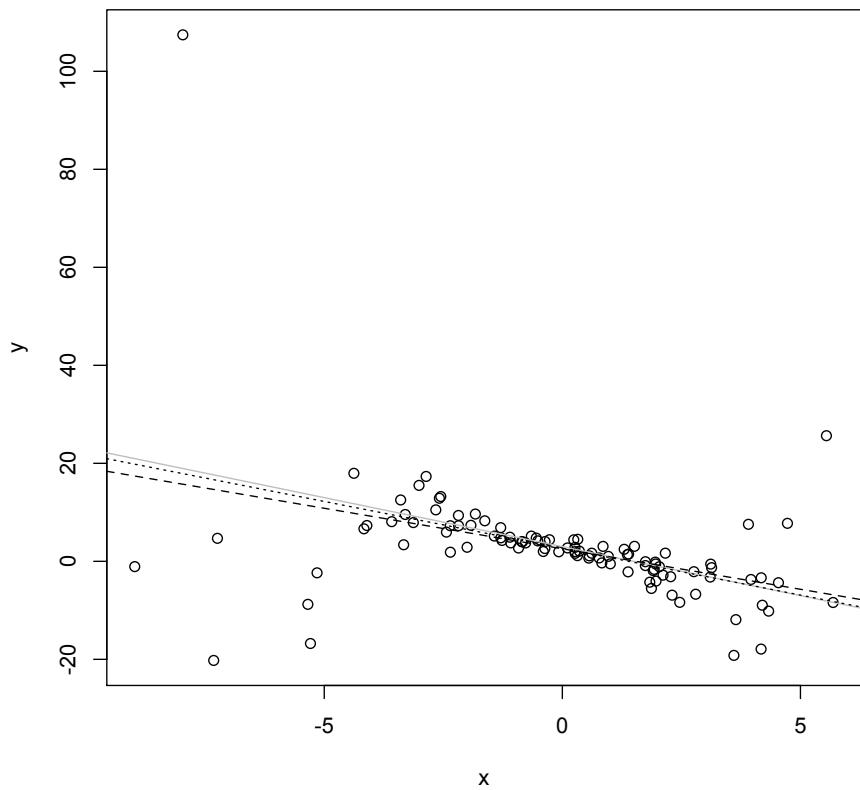
7.2.2 Some Explanations for Weighted Least Squares

Qualitatively, the reason WLS with inverse variance weights works is the following. OLS tries equally hard to match observations at each data point.¹ Weighted least squares, naturally enough, tries harder to match observations where the weights are big, and less hard to match them where the weights are small. But each y_i contains not only the true regression function $r(x_i)$ but also some noise ϵ_i . The noise terms have large magnitudes where the variance is large. So we should want to have small weights where the noise variance is large, because there the data tends to be far from the true regression. Conversely, we should put big weights where the noise variance is small, and the data points are close to the true regression.

¹Less anthropomorphically, the objective function in Eq. 7.1 has the same derivative with respect to the squared error at each point, $\frac{\partial MSE}{\partial (y_i - \bar{x}_i \beta)^2} = \frac{1}{n}$.



Figure 7.4: Statistician (right) consulting the Oracle of Regression (left) about the proper weights to use to overcome heteroskedasticity. (Image from <http://en.wikipedia.org/wiki/Image:Pythia1.jpg>.)



```
fit.wls = lm(y~x, weights=1/(1+0.5*x^2))
abline(fit.wls,lty=3)
```

Figure 7.5: Figure 7.2, with addition of weighted least squares regression line (dotted).

```
wls.heterosked.example = function(n) {
  y = 3-2*x + rnorm(n,0,sapply(x,function(x){1+0.5*x^2}))
  fit.wls = lm(y~x,weights=1/(1+0.5*x^2))
  # Return the errors
  return(fit.wls$coefficients - c(3,-2))
}

wls.heterosked.error.stats = function(n,m=10000) {
  wls.errors.raw = t(replicate(m,wls.heterosked.example(n)))
  # transpose gives us a matrix with named columns
  intercept.sd = sd(wls.errors.raw[, "Intercept"])
  slope.sd = sd(wls.errors.raw[, "x"])
  return(list(intercept.sd=intercept.sd,slope.sd=slope.sd))
}
```

Code Example 23: Linear regression of heteroskedastic data, using weighted least-squared regression.

The qualitative reasoning in the last paragraph doesn't explain why the weights should be inversely proportional to the variances, $w_i \propto 1/\sigma_{x_i}^2$ — why not $w_i \propto 1/\sigma_{\epsilon_i}$, for instance? Look at the equation for the WLS estimates again:

$$\hat{\beta}_{WLS} = (\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w} \mathbf{y} \quad (7.5)$$

Imagine holding \mathbf{x} constant, but repeating the experiment multiple times, so that we get noisy values of \mathbf{y} . In each experiment, $Y_i = \vec{x}_i \cdot \beta + \epsilon_i$, where $E[\epsilon_i] = 0$ and $\text{Var}[\epsilon_i] = \sigma_{\epsilon_i}^2$. So

$$\hat{\beta}_{WLS} = (\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w} \mathbf{x} \beta + (\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w} \epsilon \quad (7.6)$$

$$= \beta + (\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w} \epsilon \quad (7.7)$$

Since $E[\epsilon] = 0$, the WLS estimator is unbiased:

$$E[\hat{\beta}_{WLS}] = \beta \quad (7.8)$$

In fact, for the j^{th} coefficient,

$$\hat{\beta}_j = \beta_j + [(\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w} \epsilon]_j \quad (7.9)$$

$$= \beta_j + \sum_{i=1}^n H_{ji}(w) \epsilon_i \quad (7.10)$$

where in the last line I have bundled up $(\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w}$ as a matrix $\mathbf{H}(w)$, with the argument to remind us that it depends on the weights. Since the WLS estimate is

unbiased, it's natural to want it to also have a small variance, and

$$\text{Var} [\hat{\beta}_j] = \sum_{i=1}^n H_{ji}(w) \sigma_{x_i}^2 \quad (7.11)$$

It can be shown — the result is called the **generalized Gauss-Markov theorem** — that picking weights to minimize the variance in the WLS estimate has the unique solution $w_i = 1/\sigma_{x_i}^2$. It does not require us to assume the noise is Gaussian, but the proof is a bit tricky (see Appendix D).

A less general but easier-to-grasp result comes from adding the assumption that the noise around the regression line is Gaussian — that

$$Y = \vec{x} \cdot \beta + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_x^2) \quad (7.12)$$

The log-likelihood is then (EXERCISE 1)

$$-\frac{n}{2} \ln 2\pi - \frac{1}{2} \sum_{i=1}^n \log \sigma_{x_i}^2 - \frac{1}{2} \sum_{i=1}^n \frac{(y_i - \vec{x}_i \cdot \beta)^2}{\sigma_{x_i}^2} \quad (7.13)$$

If we maximize this with respect to β , everything except the final sum is irrelevant, and so we minimize

$$\sum_{i=1}^n \frac{(y_i - \vec{x}_i \cdot \beta)^2}{\sigma_{x_i}^2} \quad (7.14)$$

which is just weighted least squares with $w_i = 1/\sigma_{x_i}^2$. So, if the probabilistic assumption holds, WLS is the efficient maximum likelihood estimator.

7.2.3 Finding the Variance and Weights

All of this was possible because the Oracle told us what the variance function was. What do we do when the Oracle is not available (Figure 7.6)?

Under some situations we can work things out for ourselves, without needing an oracle.

- We know, empirically, the precision of our measurement of the response variable — we know how precise our instruments are, or each value of the response is really an average of several measurements so we can use their standard deviations, etc.
- We know how the noise in the response must depend on the input variables. For example, when taking polls or surveys, the variance of the proportions we find should be inversely proportional to the sample size. So we can make the weights proportional to the sample size.

Both of these outs rely on kinds of background knowledge which are easier to get in the natural or even the social sciences than in many industrial applications. However, there are approaches for other situations which try to use the observed residuals to get estimates of the heteroskedasticity; this is the topic of the next section.



Figure 7.6: The Oracle may be out (left), or too creepy to go visit (right). What then?
 (Left, the sacred oak of the Oracle of Dodona, copyright 2006 by Flickr user “essayan”, <http://flickr.com/photos/essayen/245236125/>; right, the entrance to the cave of the Sibyl of Cumæ, copyright 2005 by Flickr user “pverdicchio”, <http://flickr.com/photos/occhio/17923096/>. Both used under Creative Commons license.)

7.3 Conditional Variance Function Estimation

Remember that there are two equivalent ways of defining the variance:

$$\text{Var}[X] = E[X^2] - (E[X])^2 = E[(X - E[X])^2] \quad (7.15)$$

The latter is more useful for us when it comes to estimating variance functions. We have already figured out how to estimate means — that’s what all this previous work on smoothing and regression is for — and the deviation of a random variable from its mean shows up as a residual.

There are two generic ways to estimate conditional variances, which differ slightly in how they use non-parametric smoothing. We can call these the **squared residuals method** and the **log squared residuals method**. Here is how the first one goes.

1. Estimate $r(x)$ with your favorite regression method, getting $\hat{r}(x)$.
2. Construct the **squared residuals**, $u_i = (y_i - \hat{r}(x_i))^2$.
3. Use your favorite *non-parametric* method to estimate the conditional mean of the u_i , call it $\hat{q}(x)$.
4. Predict the variance using $\hat{\sigma}_x^2 = \hat{q}(x)$.

The log-squared residuals method goes very similarly.²

1. Estimate $r(x)$ with your favorite regression method, getting $\hat{r}(x)$.
2. Construct the **log squared residuals**, $z_i = \log(y_i - \hat{r}(x_i))^2$.
3. Use your favorite *non-parametric* method to estimate the conditional mean of the z_i , call it $\hat{s}(x)$.

²I learned it from Wasserman (2006, pp. 87–88).

4. Predict the variance using $\hat{\sigma}_x^2 = \exp \hat{s}(x)$.

The quantity $y_i - \hat{r}(x_i)$ is the i^{th} residual. If $\hat{r} \approx r$, then the residuals should have mean zero. Consequently the variance of the residuals (which is what we want) should equal the expected squared residual. So squaring the residuals makes sense, and the first method just smoothes these values to get at their expectations.

What about the second method — why the log? Basically, this is a convenience — squares are necessarily non-negative numbers, but lots of regression methods don't easily include constraints like that, and we really don't want to predict negative variances.³ Taking the log gives us an unbounded range for the regression.

Strictly speaking, we don't need to use non-parametric smoothing for either method. If we had a parametric model for σ_x^2 , we could just fit the parametric model to the squared residuals (or their logs). But even if you think you know what the variance function should look like it, why not check it?

We came to estimating the variance function because of wanting to do weighted least squares, but these methods can be used more generally. It's often important to understand variance in its own right, and this is a general method for estimating it. Our estimate of the variance function depends on first having a good estimate of the regression function

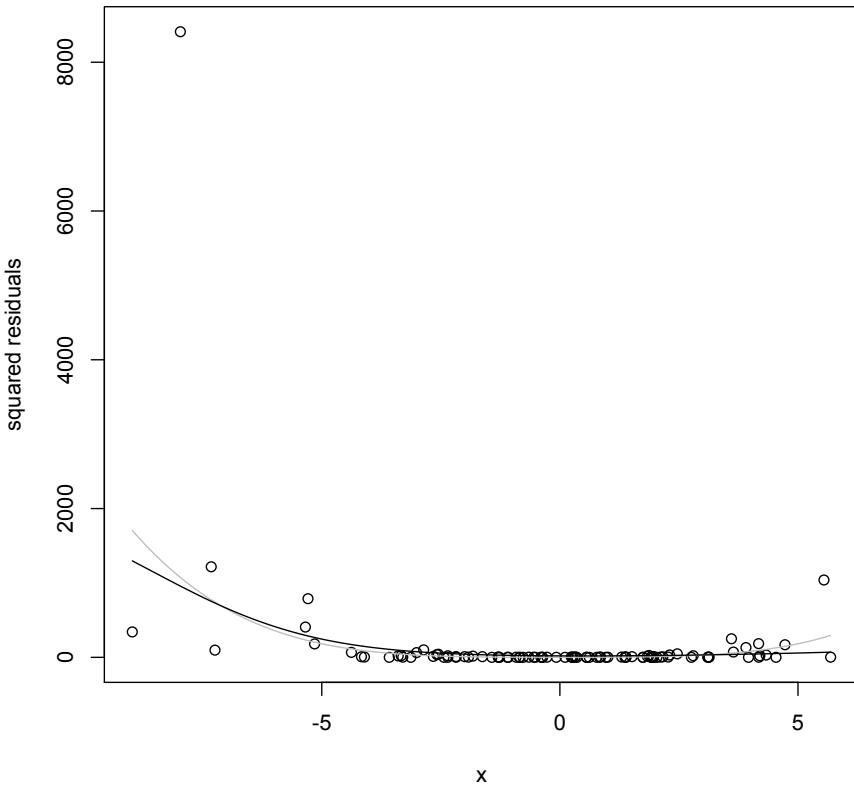
7.3.1 Iterative Refinement of Mean and Variance: An Example

The estimate $\hat{\sigma}_x^2$ depends on the initial estimate of the regression function $\hat{r}(x)$. But, as we saw when we looked at weighted least squares, taking heteroskedasticity into account can change our estimates of the regression function. This suggests an iterative approach, where we alternate between estimating the regression function and the variance function, using each to improve the other. That is, we take either method above, and then, once we have estimated the variance function $\hat{\sigma}_x^2$, we re-estimate \hat{r} using weighted least squares, with weights inversely proportional to our estimated variance. Since this will generally change our estimated regression, it will change the residuals as well. Once the residuals have changed, we should re-estimate the variance function. We keep going around this cycle until the change in the regression function becomes so small that we don't care about further modifications. It's hard to give a strict guarantee, but *usually* this sort of iterative improvement will converge.

Let's apply this idea to our example. Figure 7.3b already plotted the residuals from OLS. Figure 7.7 shows those squared residuals again, along with the true variance function and the estimated variance function.

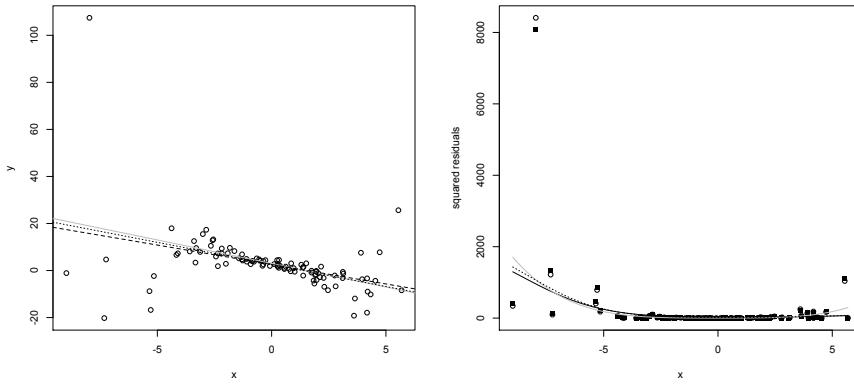
The OLS estimate of the regression line is not especially good ($\hat{\beta}_0 = 2.56$ versus $\beta_0 = 3$, $\hat{\beta}_1 = -1.65$ versus $\beta_1 = -2$), so the residuals are systematically off, but it's clear from the figure that kernel smoothing of the squared residuals is picking up on the heteroskedasticity, and getting a pretty reasonable picture of the variance function.

³Occasionally you do see people doing things like claiming that genetics explains more than 100% of the variance in some psychological trait, and so the contributions of environment and up-bringing have negative variance. Some of them — for instance, Alford *et al.* (2005) — manage to say this with a straight face.



```
plot(x,residuals(fit.ols)^2,ylab="squared residuals")
curve((1+x^2/2)^2,col="grey",add=TRUE)
require(np)
var1 <- npreg(residuals(fit.ols)^2 ~ x)
grid.x <- seq(from=min(x),to=max(x),length.out=300)
lines(grid.x,predict(var1,exdat=grid.x))
```

Figure 7.7: Points: actual squared residuals from the OLS line. Grey curve: true variance function, $\sigma_x^2 = (1 + x^2/2)^2$. Black curve: kernel smoothing of the squared residuals, using npreg.



```

fit.wls1 <- lm(y~x,weights=1/fitted(var1))
plot(x,y)
abline(a=3,b=-2,col="grey")
abline(fit.ols,lty=2)
abline(fit.wls1,lty=3)
plot(x,(residuals(fit.ols))^2,ylab="squared residuals")
points(x,(residuals(fit.wls1))^2,pch=15)
lines(grid.x,predict(var1,exdat=grid.x))
var2 <- npreg(residuals(fit.wls1)^2 ~ x)
curve((1+x^2/2)^2,col="grey",add=TRUE)
lines(grid.x,predict(var2,exdat=grid.x),lty=3)

```

Figure 7.8: Left: As in Figure 7.2, but with the addition of the weighted least squares regression line (dotted), using the estimated variance from Figure 7.7 for weights. Right: As in Figure 7.7, but with the addition of the residuals from the WLS regression (black squares), and the new estimated variance function (dotted curve).

Now we use the estimated variance function to re-estimate the regression line, with weighted least squares.

```

> fit.wls1 <- lm(y~x,weights=1/fitted(var1))
> coefficients(fit.wls1)
(Intercept)          x
  2.595860   -1.876042
> var2 <- npreg(residuals(fit.wls1)^2 ~ x)

```

The slope has changed substantially, and in the right direction (Figure 7.8a). The residuals have also changed (Figure 7.8b), and the new variance function is closer to the truth than the old one.

Since we have a new variance function, we can re-weight the data points and re-estimate the regression:

```

> fit.wls2 <- lm(y~x,weights=1/fitted(var2))
> coefficients(fit.wls2)
(Intercept)          x
  2.625295   -1.914075
> var3 <- npreg(residuals(fit.wls2)^2 ~ x)

```

Since we know that the true coefficients are 3 and -2 , we know that this is moving in the right direction. If I hadn't told you what they were, you could still observe that the difference in coefficients between `fit.wls1` and `fit.wls2` is smaller than that between `fit.ols` and `fit.wls1`, which is a sign that this is converging.

I will spare you the plot of the new regression and of the new residuals. When we update a few more times:

```

> fit.wls3 <- lm(y~x,weights=1/fitted(var3))
> coefficients(fit.wls3)
(Intercept)          x
  2.630249   -1.920476
> var4 <- npreg(residuals(fit.wls3)^2 ~ x)
> fit.wls4 <- lm(y~x,weights=1/fitted(var4))
> coefficients(fit.wls4)
(Intercept)          x
  2.631063   -1.921540

```

By now, the coefficients of the regression are changing in the fourth significant digit, and we only have 100 data points, so the imprecision from a limited sample surely swamps the changes we're making, and we might as well stop.

Manually going back and forth between estimating the regression function and estimating the variance function is tedious. We could automate it with a function, which would look something like this:

```

iterative.wls <- function(x,y,tol=0.01,max.iter=100) {
  iteration <- 1
  old.coefs <- NA
  regression <- lm(y~x)
  coefs <- coefficients(regression)
  while (is.na(old.coefs) || 
         ((max(coefs - old.coefs) > tol) && (iteration < max.iter))) {
    variance <- npreg(residuals(regression)^2 ~ x)
    old.coefs <- coefs
    iteration <- iteration+1
    regression <- lm(y~x,weights=1/fitted(variance))
    coefs <- coefficients(regression)
  }
  return(list(regression=regression,variance=variance,iterations=iteration))
}

```

This starts by doing an unweighted linear regression, and then alternates between WLS for the getting the regression and kernel smoothing for getting the variance. It

stops when no parameter of the regression changes by more than `tol`, or when it's gone around the cycle `max.iter` times.⁴ This code is a bit too inflexible to be really "industrial strength" (what if we wanted to use a data frame, or a more complex regression formula?), but shows the core idea.

7.3.2 Real Data Example: Old Heteroskedastic

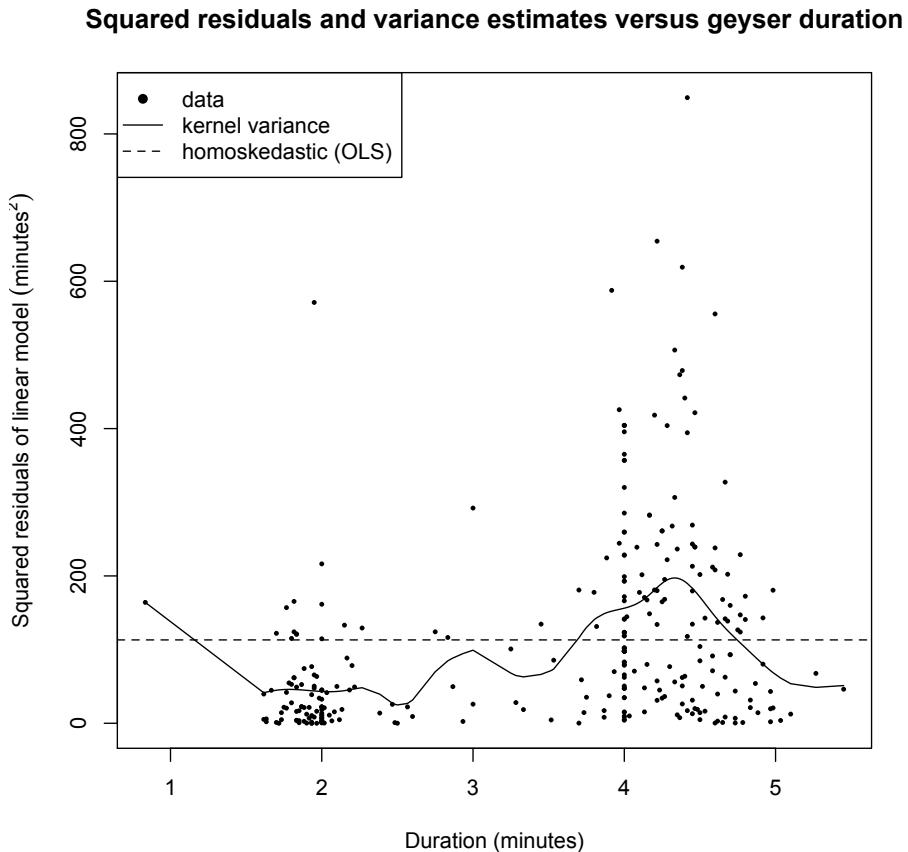
§5.3.2 introduced the `geyser` data set, which is about predicting the waiting time between consecutive eruptions of the "Old Faithful" geyser at Yellowstone National Park from the duration of the latest eruption. Our exploration there showed that a simple linear model (of the kind often fit to this data in textbooks and elementary classes) is not very good, and raised the suspicion that one important problem was heteroskedasticity. Let's follow up on that, building on the computational work done in that section.

The estimated variance function `geyser.var` does not look particularly flat, but it comes from applying a fairly complicated procedure (kernel smoothing with data-driven bandwidth selection) to a fairly limited amount of data (299 observations). Maybe that's the amount of wigglyness we should *expect* to see due to finite-sample fluctuations? To rule this out, we can make surrogate data from the homoskedastic model, treat it the same way as the real data, and plot the resulting variance functions (Figure 7.10). The conditional variance functions estimated from the homoskedastic model are flat or gently varying, with much less range than what's seen in the data.

While that sort of qualitative comparison is genuinely informative, one can also be more quantitative. One might measure heteroskedasticity by, say, evaluating the conditional variance at all the data points, and looking at the ratio of the interquartile range to the median. This would be zero for perfect homoskedasticity, and grow as the dispersion of actual variances around the "typical" variance increased. For the data, this is `IQR(fitted(geyser.var))/median(fitted(geyser.var)) = 0.86`. Simulations from the OLS model give values around 10^{-15} .

There is nothing particularly special about this measure of heteroskedasticity — after all, I just made it up. The broad point it illustrates is that whenever we have some sort of quantitative summary statistic we can calculate on our real data, we can also calculate the same statistic on realizations of the model, and the difference will then tell us something about how close the simulations, and so the model, come to the data. In this case, we learn that the linear, homoskedastic model seriously understates the variability of this data. That leaves open the question of whether the problem is the linearity or the homoskedasticity; I will leave that question to EXERCISE 5.

⁴The condition in the `while` loop is a bit complicated, to ensure that the loop is executed at least once. Some languages have an `until` control structure which would simplify this.

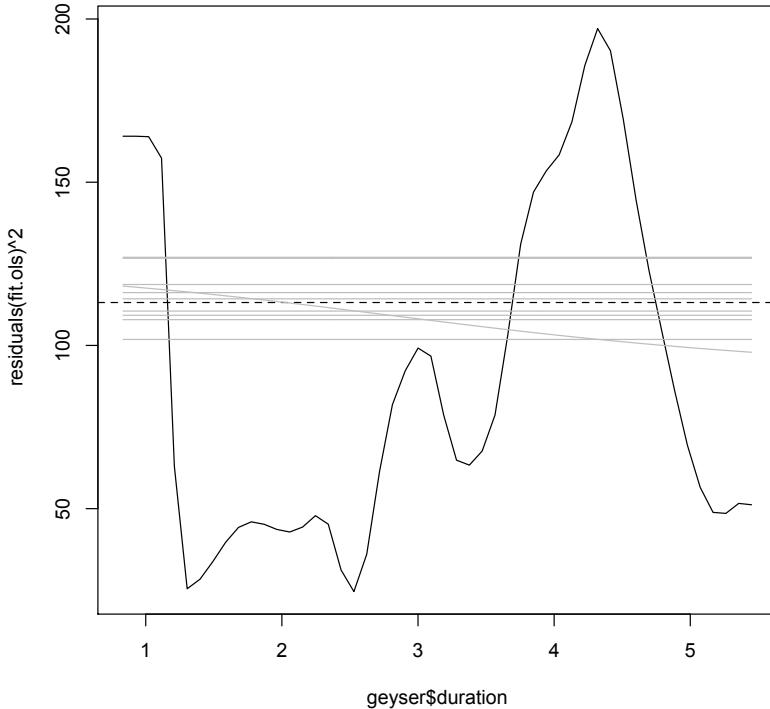


```

plot(geyser$duration, residuals(fit.ols)^2, cex=0.5, pch=16,
      main="Squared residuals and variance estimates versus geyser duration",
      xlab="Duration (minutes)",
      ylab=expression("Squared residuals of linear model "(minutes^2)))
library(np)
geyser.var <- npreg(residuals(fit.ols)^2~geyser$duration)
duration.order <- order(geyser$duration)
lines(geyser$duration[duration.order],fitted(geyser.var)[duration.order])
abline(h=summary(fit.ols)$sigma^2,lty="dashed")
legend("topleft",
      legend=c("data","kernel variance","homoskedastic (OLS)"),
      lty=c(-1,1,2),pch=c(16,-1,-1))

```

Figure 7.9: Squared residuals from the linear model of Figure 5.4, plotted against duration, along with the unconditional, homoskedastic variance implicit in OLS (dashed), and a kernel-regression estimate of the conditional variance (solid).



```

plot(geyser.var)
abline(h=summary(fit.ols)$sigma^2,lty=2)
duration.grid <- seq(from=min(geyser$duration),to=max(geyser$duration),
length.out=300)
one.var.func <- function() {
  fit <- lm(waiting ~ duration, data=rgeyser())
  var.func <- npreg(residuals(fit)^2 ~ geyser$duration)
  lines(duration.grid,predict(var.func,exdat=duration.grid),col="grey")
}
invisible(replicate(10,one.var.func()))

```

Figure 7.10: The actual conditional variance function estimated from the Old Faithful data (and the linear regression), in black, plus the results of applying the same procedure to simulations from the homoskedastic linear regression model (grey lines; see §5.3.2 for the `rgeyser()` function). The fact that the estimates from the simulations are all flat or gently sloped suggests that the changes in variance found in the data are too large to just be sampling noise.

7.4 Re-sampling Residuals with Heteroskedasticity

Re-sampling the residuals of a regression, as described in §6.4, assumes that the distribution of fluctuations around the regression curve is the same for all values of the input x . Under heteroskedasticity, this is of course not the case. Nonetheless, we can still re-sample residuals to get bootstrap confidence intervals, standard errors, and so forth, provided we define and scale them properly. If we have a conditional variance function $\hat{\sigma}^2(x)$, or a conditional standard deviation function $\hat{sigma}(x)$, as well as the estimated regression function $\hat{r}(x)$, we can combine them to re-sample heteroskedastic residuals.

1. Construct the standardized residuals, by dividing the actual residuals by the conditional standard deviation:

$$\eta_i = \epsilon_i / \hat{\sigma}(x_i) \quad (7.16)$$

The η_i should now be all the same size (in distribution!), no matter where x_i is in the space of predictors.

2. Re-sample the η_i with replacement, to get $\tilde{\eta}_1, \dots, \tilde{\eta}_n$.
3. Set $\tilde{x}_i = x_i$.
4. Set $\tilde{y}_i = \hat{r}(\tilde{x}_i) + \hat{\sigma}(\tilde{x}_i)\tilde{\eta}_i$.
5. Analyze the surrogate data $(\tilde{x}_1, \tilde{y}_1), \dots, (\tilde{x}_n, \tilde{y}_n)$ like it was real data.

Of course, this still assumes that the *only* difference in distribution for the noise at different values of x is its scale.

7.5 Local Linear Regression

Switching gears, recall from Chapter 2 that one reason it can be sensible to use a linear approximation to the true regression function $r(x)$ is that we can always⁵ Taylor-expand the latter around any point x_0 ,

$$r(x) = r(x_0) + \sum_{k=1}^{\infty} \frac{(x - x_0)^k}{k!} \left. \frac{d^k r}{d x^k} \right|_{x=x_0} \quad (7.17)$$

and similarly with all the partial derivatives in higher dimensions. If we truncate the series at first order, $r(x) \approx r(x_0) + (x - x_0)r'(x_0)$, we see that the first-order coefficient $r'(x_0)$ is the best linear prediction coefficient, at least when x is sufficiently close to x_0 . The snag in this line of argument is that if $r(x)$ isn't really linear, then r' isn't a constant, and the optimal linear predictor to use depends on where we want to make predictions.

⁵At least if $r(x)$ is differentiable.

However, statisticians are thrifty people, and having assembled all the machinery for linear regression, they are loathe to throw it away just because the fundamental model is wrong. If we can't fit one line, why not fit many? If each point has a different best linear regression, why not estimate them all? Thus the idea of **local** linear regression: fit a different linear regression everywhere, weighting the data points by how close they are to the point of interest⁶.

The very simplest approach we could take would be to divide up the range of x into so many bins, and fit a separate linear regression for each bin. This is unsatisfying for at least three reasons. First, it gives us weird discontinuities at the boundaries between bins. Second, it introduces an odd sort of bias, where our predictions near the boundaries of a bin depend strongly on data from the other side of the bin, and not at all on nearby data points just across the border, which is weird. Third, we need to pick the bins.

The next simplest approach would be to first figure out where we want to make a prediction (say x), and do a linear regression with all the data points which were sufficiently close, $|x_i - x| \leq h$ for some h . Now we are basically using a uniform-density kernel to weight the data points. This eliminates two problems from the binning idea — the examples we include are always centered on the x we're trying to get a prediction for, and we just need to pick one bandwidth h rather than placing all the bin boundaries. But still, each example point always has either weight 0 or weight 1, so our predictions change jerkily as training points fall into or out of the window. It generally works nicer to have the weights change more smoothly with the distance, starting off large and then gradually trailing off to zero.

By now bells may be going off in your head, as this sounds very similar to the kernel regression. In fact, kernel regression is what happens when we truncate Eq. 7.17 at *zeroth* order, getting **locally constant** regression. Here's the problem we're setting up:

$$\operatorname{argmin}_{m(x)} \frac{1}{n} \sum_{i=1}^n w_i(x) (y_i - m(x))^2 \quad (7.18)$$

which has the solution

$$\hat{m}(x) = \frac{\sum_{i=1}^n w_i(x) y_i}{\sum_{j=1}^n w_j(x)} \quad (7.19)$$

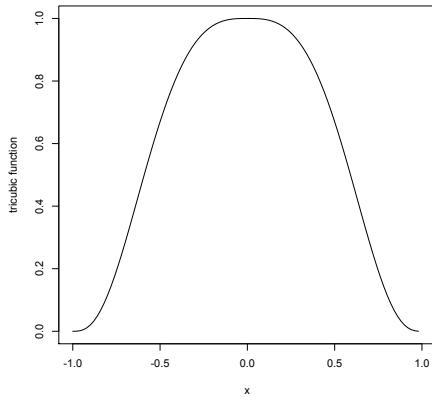
which just is our kernel regression, with the weights being proportional to the kernels, $w_i(x) \propto K(x_i, x)$. (Without loss of generality, we can take the constant of proportionality to be 1.)

What about **locally linear** regression? The optimization problem is

$$\operatorname{argmin}_{m, \beta} \frac{1}{n} \sum_{i=1}^n w_i(x) (y_i - m(x) - (x_i - x) \cdot \beta(x))^2 \quad (7.20)$$

where again we can write $w_i(x)$ as proportional to some kernel function, $w_i(x) \propto K(x_i, x)$. To solve this, abuse notation slightly to define $z_i = (1, x_i - x)$, i.e., the

⁶Some people say "local linear" and some "locally linear".



```
curve((1-abs(x)^3)^3,from=-1,to=1,ylab="tricubic function")
```

Figure 7.11: The tricubic kernel, with broad plateau where $|x| \approx 0$, and the smooth fall-off to zero at $|x| = 1$.

displacement from x , with a 1 stuck at the beginning to (as usual) handle the intercept. Now, by the machinery above,

$$\widehat{(m, \beta(x))} = (\mathbf{z}^T \mathbf{w}(x) \mathbf{z})^{-1} \mathbf{z}^T \mathbf{w}(x) \mathbf{y} \quad (7.21)$$

and the prediction is just the intercept, \hat{m} . If you need an estimate of the first derivatives, those are the $\widehat{\beta}$. Notice, from Eq. 7.21, that if the weights given to each training point change smoothly with x , then the predictions will also change smoothly.⁷

Using a smooth kernel whose density is positive everywhere, like the Gaussian, ensures that the weights will change smoothly. But we could also use a kernel which goes to zero outside some finite range, so long as the kernel rises gradually from zero inside the range. For locally linear regression, a common choice of kernel is therefore the **tri-cubic**,

$$K(x_i, x) = \left(1 - \left(\frac{|x_i - x_0|}{b} \right)^3 \right)^3 \quad (7.22)$$

if $|x - x_i| < b$, and = 0 otherwise (Figure 7.11).

7.5.1 Advantages and Disadvantages of Locally Linear Regression

Why would we use locally linear regression, if we already have kernel regression?

⁷Notice that local linear predictors are still linear smoothers as defined in Chapter 1, (i.e., the predictions are linear in the y_i), but they are not, strictly speaking, *kernel* smoothers, since you can't re-write the last equation in the form of a kernel average.

1. You may recall that when we worked out the bias of kernel smoothers (Eq. 4.10 in Chapter 4), we got a contribution that was proportional to $r'(x)$. If we do an analogous analysis for locally linear regression, the bias is the same, *except* that this derivative term goes away.
2. Relatedly, that analysis we did of kernel regression tacitly assumed the point we were looking at was in the middle of the training data (or at least less than h from the border). The bias gets worse near the edges of the training data. Suppose that the true $r(x)$ is decreasing in the vicinity of the largest x_i . (See the grey curve in Figure 7.12.) When we make our predictions there, in kernel regression we can only average values of y_i which tend to be systematically larger than the value we want to predict. This means that our kernel predictions are systematically biased upwards, and the size of the bias grows with $r'(x)$. (See the black line in Figure 7.12 at the lower right.) If we use a locally linear model, however, it can pick up that there is a trend, and reduce the edge bias by extrapolating it (dashed line in the figure).
3. The predictions of locally linear regression tend to be smoother than those of kernel regression, simply because we are locally fitting a smooth line rather than a flat constant. As a consequence, estimates of the derivative $\frac{d\hat{r}}{dx}$ tend to be less noisy when \hat{r} comes from a locally linear model than a kernel regression.

Of course, total prediction error depends not only on the bias but also on the variance. Remarkably enough, the variance for kernel regression and locally linear regression is the same. Since locally linear regression has smaller bias, the former is often predictively superior.

There are several packages which implement locally linear regression. Since we are already using `np`, one of the simplest is to set the `regtype="1l"` in `npreg`.⁸ There are several other packages which support it, notably `KernSmooth` and `locpoly`.

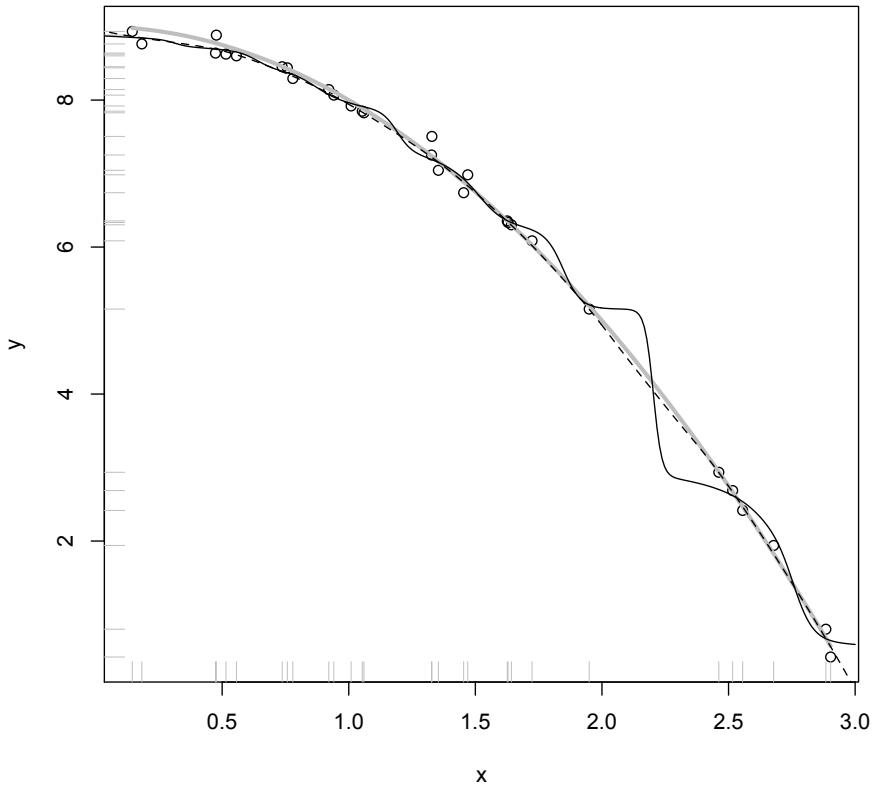
As the name of the latter suggests, there is no reason we *have* to stop at locally linear models, and we could use local polynomials of any order. The main reason to use a higher-order local polynomial, rather than a locally-linear or locally-constant model, is to estimate higher derivatives. Since this is a somewhat specialized topic, I will not say more about it.

7.5.2 Lowess

There is however one additional topic in locally linear models which is worth mentioning. This is the variant called `lowess` or `loess`.⁹ The basic idea is to fit a locally linear model, with a kernel which goes to zero outside a finite window and rises gradually inside it, typically the tri-cubic I plotted earlier. The wrinkle, however, is that rather than solving a least *squares* problem, it minimizes a different and more

⁸"1l" stands for "locally linear", of course; the default is `regtype="1c"`, for "locally constant".

⁹I have heard this name explained as an acronym for both "locally weighted scatterplot smoothing" and "locally weight sum of squares".



```

x <- runif(30,max=3)
y <- 9-x^2 + rnorm(30, sd=0.1)
plot(x,y); rug(x,side=1, col="grey"); rug(y,side=2, col="grey")
curve(9-x^2,col="grey",add=TRUE, lwd=3)
grid.x <- seq(from=0,to=3,length.out=300)
np0 <- npreg(y~x); lines(grid.x,predict(np0, exdat=grid.x))
np1 <- npreg(y~x, regtype="ll"); lines(grid.x,predict(np1, exdat=grid.x),lty=2)

```

Figure 7.12: Points are samples from the true, nonlinear regression function shown in grey. The solid black line is a kernel regression, and the dashed line is a locally linear regression. Note that the locally linear model is smoother than the kernel regression, and less biased when the true curve has a non-zero bias at a boundary of the data (far right).

“robust” loss function,

$$\operatorname{argmin}_{\beta(x)} \frac{1}{n} \sum_{i=1}^n w_i(x) \ell(y - \vec{x}_i \cdot \beta(x)) \quad (7.23)$$

where $\ell(a)$ doesn’t grow as rapidly for large a as a^2 . The idea is to make the fitting less vulnerable to occasional large outliers, which would have very large squared errors, unless the regression curve went far out of its way to accommodate them. For instance, we might have $\ell(a) = a^2$ if $|a| < 1$, and $\ell(a) = 2|a| - 1$ otherwise¹⁰. We will come back to robust estimation later, but I bring it up now because it’s a very common smoothing technique, especially for visualization.

Lowess smoothing is implemented in the default R packages through the function `lowess` (rather basic), and through the function `loess` (more sophisticated), as well as in the CRAN package `locfit` (more sophisticated still). The `lowess` idea can be combined with local fitting of higher-order polynomials; the `loess` and `locfit` commands both support this.

7.6 Exercises

To think through or experiment with, not to hand in.

1. Show that the model of Eq. 7.12 has the log-likelihood given by Eq. 7.13
2. Do the calculus to verify Eq. 7.4.
3. Is $w_i = 1$ a necessary as well as a sufficient condition for Eq. 7.3 and Eq. 7.1 to have the same minimum?
4. The text above looked at whether WLS gives better parameter estimates than OLS when there is heteroskedasticity, and we know and use the variance. Modify the code for to see which one has better generalization error.
5. COMPUTING §7.3.2 looked at the residuals of the linear regression model for the Old Faithful geyser data, and showed that they would imply lots of heteroskedasticity. This might, however, be an artifact of inappropriately using a linear model. Use either kernel regression (cf. §6.4.2) or local linear regression to estimate the conditional mean of waiting given duration, and see whether the apparent heteroskedasticity goes away.
6. Should local linear regression do better or worse than ordinary least squares under heteroskedasticity? What exactly would this mean, and how might you test your ideas?

¹⁰This is called the **Huber loss**; it continuously interpolates between looking like squared error and looking like absolute error. This means that when errors are small, it gives results very like least-squares, but it is resistant to outliers.

Chapter 8

Splines

8.1 Smoothing by Directly Penalizing Curve Flexibility

Let's go back to the problem of smoothing one-dimensional data. We have data points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, and we want to find a good approximation $\hat{r}(x)$ to the true conditional expectation or regression function $r(x)$. Previously, we controlled how smooth we made \hat{r} indirectly, through the bandwidth of our kernels. But why not be more direct, and directly control smoothness?

A natural way to do this, in one dimension, is to minimize the **spline objective function**

$$\mathcal{L}(m, \lambda) \equiv \frac{1}{n} \sum_{i=1}^n (y_i - m(x_i))^2 + \lambda \int (m''(x))^2 dx \quad (8.1)$$

The first term here is just the mean squared error of using the curve $m(x)$ to predict y . We know and like this; it is an old friend.

The second term, however, is something new for us. m'' is the second derivative of m with respect to x — it would be zero if m were linear, so this measures the **curvature** of m at x . The sign of m'' says whether the curvature is concave or convex, but we don't care about that so we square it. We then integrate this over all x to say how curved m is, on average. Finally, we multiply by λ and add that to the MSE. This is adding a **penalty** to the MSE criterion — given two functions with the same MSE, we prefer the one with less average curvature. In fact, we are willing to accept changes in m that increase the MSE by 1 unit if they also reduce the average curvature by at least λ .

The *solution* to this minimization problem,

$$\hat{r}_\lambda = \operatorname{argmin}_m \mathcal{L}(m, \lambda) \quad (8.2)$$

is a function of x , or curve, called a **smoothing spline**, or **smoothing spline function**¹.

¹The name “spline” actually comes from a simple tool used by craftsmen to draw smooth curves, which

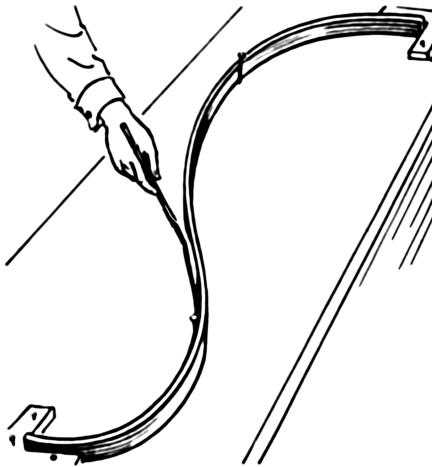


Figure 8.1: A craftsman's spline, from Wikipedia, s.v. "Flat spline".

It is possible to show that all solutions, no matter what the initial data are, are piecewise cubic polynomials which are continuous and have continuous first and second derivatives — i.e., not only is \hat{r} continuous, so are \hat{r}' and \hat{r}'' . The boundaries between the pieces are located at the original data points. These are called, somewhat obscure, the **knots** of the spline. The function continuous beyond the largest and smallest data points, but it is always linear on those regions.² I will not attempt to prove this.

I will also assert, without proof, that such piecewise cubic polynomials can approximate any well-behaved function arbitrarily closely, given enough pieces. Finally, smoothing splines are linear smoothers, in the sense given in Chapter 1: predicted values are always linear combinations of the original response values y_i — see Eq. 8.21 below.

8.1.1 The Meaning of the Splines

Look back to the optimization problem. As $\lambda \rightarrow \infty$, having any curvature at all becomes infinitely penalized, and only linear functions are allowed. But we know how to minimize mean squared error with linear functions, that's OLS. So we understand

was a thin strip of a flexible material like a soft wood, as in Figure 8.1. (A few years ago, when the gas company dug up my front yard, the contractors they hired to put the driveway back used a plywood board to give a smooth, outward-curve edge to the new driveway. The “knots” were metal stakes which the board was placed between, the curve of the board was a spline, and they poured concrete to one side of the board, which they left standing until the concrete dried.) Bending such a material takes energy — the stiffer the material, the more energy has to go into bending it through the same shape, and so the straighter the curve it will make between given points. For smoothing splines, using a stiffer material corresponds to increasing λ .

²Can you explain why it is linear outside the data range, in terms of the optimization problem?

that limit.

On the other hand, as $\lambda \rightarrow 0$, we decide that we don't care about curvature. In that case, we can always come up with a function which just interpolates between the data points, an **interpolation spline** passing exactly through each point. More specifically, of the infinitely many functions which interpolate between those points, we pick the one with the minimum average curvature.

At intermediate values of λ , \hat{r}_λ becomes a function which compromises between having low curvature, and bending to approach all the data points closely (on average). The larger we make λ , the more curvature is penalized. There is a bias-variance trade-off here. As λ grows, the spline becomes less sensitive to the data, with lower variance to its predictions but more bias. As λ shrinks, so does bias, but variance grows. For consistency, we want to let $\lambda \rightarrow 0$ as $n \rightarrow \infty$, just as, with kernel smoothing, we let the bandwidth $h \rightarrow 0$ while $n \rightarrow \infty$.

We can also think of the smoothing spline as the function which minimizes the mean squared error, subject to a constraint on the average curvature. This turns on a general corresponds between penalized optimization and optimization under constraints, which is explored in Appendix E. The short version is that each level of λ corresponds to imposing a cap on how much curvature the function is allowed to have, on average, and the spline we fit with that λ is the MSE-minimizing curve subject to that constraint. As we get more data, we have more information about the true regression function and can relax the constraint (let λ shrink) without losing reliable estimation.

It will not surprise you to learn that we select λ by cross-validation. Ordinary k -fold CV is entirely possible, but leave-one-out CV works quite well for splines. In fact, the default in most spline software is either leave-one-out CV, or an even faster approximation called "generalized cross-validation" or GCV. The details of how to rapidly compute the LOOCV or GCV scores are not especially important for us, but can be found, if you want them, in many books, such as Simonoff (1996, §5.6.3).

8.2 Computational Example: Splines for Stock Returns

The default R function for fitting a smoothing spline is called `smooth.spline`. The syntax is

```
smooth.spline(x, y, cv=FALSE)
```

where `x` should be a vector of values for input variable, `y` is a vector of values for the response (in the same order), and the switch `cv` controls whether to pick λ by generalized cross-validation (the default) or by leave-one-out cross-validation. The object which `smooth.spline` returns has an `$x` component, *re-arranged in increasing order*, a `$y` component of fitted values, a `$yin` component of original values, etc. See `help(smooth.spline)` for more.

As a concrete illustration, Figure 8.2 looks at the same data on stock prices that we saw in homework 3. The vector `sp` contains the log-returns³ of the S & P 500 stock index on 2528 consecutive trading days:

```
sp <- read.csv("http://www.stat.cmu.edu/~cshalizi/uADA/13/hw/03/SPhistory.short.csv")
# We only want closing prices
sp <- sp[,7]
# The data are in reverse chronological order, which is weird for us
sp <- rev(sp)
# And in fact we only want log returns, i.e., difference in logged prices
sp <- diff(log(sp))
```

We want to use the log-returns on one day to predict what they will be on the next. The horizontal axis in the figure shows the log-returns for each of 2527 days t , and the vertical axis shows the corresponding log-return for the succeeding day $t + 1$. A linear model fitted to this data displays a slope of -0.0822 (grey line in the figure). Fitting a smoothing spline with cross-validation selects $\lambda = 0.0513$, and the black curve:

```
> sp.today <- sp[-length(sp)]
> sp.tomorrow <- sp[-1]
> sp.spline <- smooth.spline(x=sp.today,y=sp.tomorrow,cv=TRUE)
Warning message:
In smooth.spline(sp[-length(sp)], sp[-1], cv = TRUE) :
  crossvalidation with non-unique 'x' values seems doubtful
> sp.spline
Call:
smooth.spline(x = sp[-length(sp)], y = sp[-1], cv = TRUE)

Smoothing Parameter  spar= 1.389486  lambda= 0.05129822 (14 iterations)
```

³For a financial asset whose price on day t is p_t , the log-returns on t are $\log p_t / p_{t-1}$. Financiers and other professional gamblers care more about the log returns than about the price change, $p_t - p_{t-1}$. — Actually, in this case it would be even closer to financial practice to add in dividends, $\log(p_t + d_t) / p_{t-1}$, but we will sacrifice realism to simplifying calculations, or imagine that we are working for someone who trades so quickly they never expect to receive dividends.

181 8.2. COMPUTATIONAL EXAMPLE: SPLINES FOR STOCK RETURNS

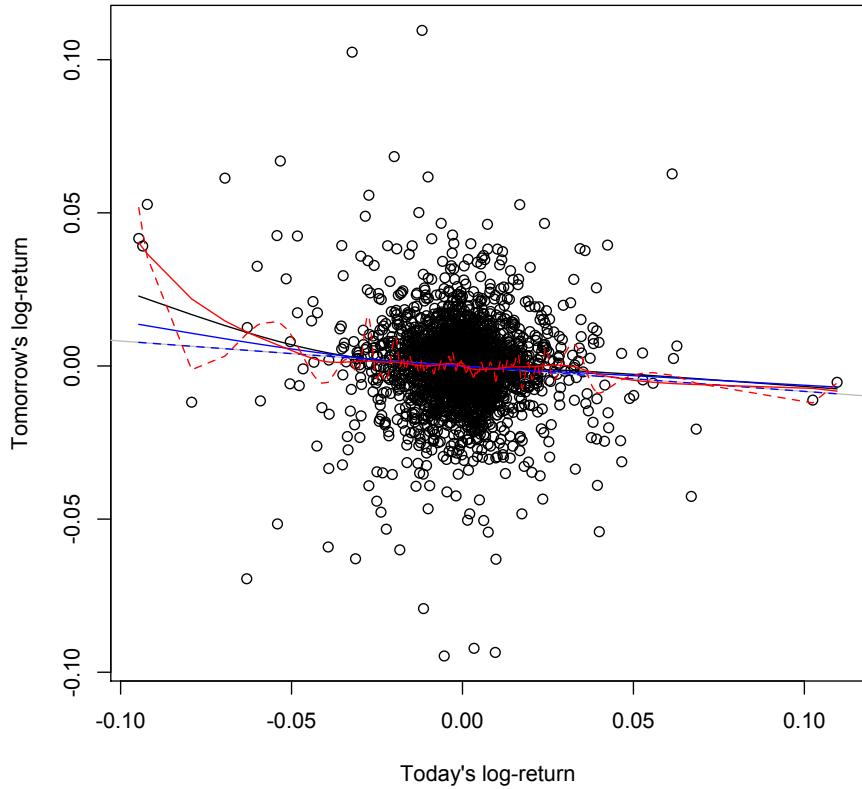
```
Equivalent Degrees of Freedom (Df): 4.206137
Penalized Criterion: 0.4885528
PRESS: 0.0001949005
> sp.spline$lambda
[1] 0.05129822
```

(PRESS is the “prediction sum of squares”, i.e., the sum of the squared leave-one-out prediction errors. Also, the warning about cross-validation, while well-intentioned, is caused here by there being just two days with log-returns of zero.) This is the curve shown in black in the figure. The curves shown in blue are for large values of λ , and clearly approach the linear regression; the curves shown in red are for smaller values of λ .

The spline can also be used for prediction. For instance, if we want to know what the return to expect following a day when the log return was +0.01,

```
> predict(sp.spline,x=0.01)
$x
[1] 0.01
$y
[1] -0.0007169499
```

i.e., a very slightly negative log-return. (Giving both an `x` and a `y` value like this means that we can directly feed this output into many graphics routines, like `points` or `lines`.)



```

plot(sp.today,sp.tomorrow,xlab="Today's log-return",
      ylab="Tomorrow's log-return")
abline(lm(sp.tomorrow ~ sp.today),col="grey")
sp.spline <- smooth.spline(x=sp.today,y=sp.tomorrow,cv=TRUE)
lines(sp.spline)
lines(smooth.spline(sp.today,sp.tomorrow,spar=1.5),col="blue")
lines(smooth.spline(sp.today,sp.tomorrow,spar=2),col="blue",lty=2)
lines(smooth.spline(sp.today,sp.tomorrow,spar=1.1),col="red")
lines(smooth.spline(sp.today,sp.tomorrow,spar=0.5),col="red",lty=2)

```

Figure 8.2: The S& P 500 log-returns data (circles), together with the OLS linear regression (grey line), the spline selected by cross-validation (solid black curve, $\lambda = 0.0513$), some more smoothed splines (blue, $\lambda = 0.322$ and 1320) and some less smooth splines (red, $\lambda = 4.15 \times 10^{-4}$ and 1.92×10^{-8}). Incidentally, `smooth.spline` does not let us control λ directly, but rather a somewhat complicated but basically exponential transformation of it called `spar`. See `help(smooth.spline)` for the gory details. The equivalent λ can be extracted from the return value, e.g., `smooth.spline(sp.today,sp.tomorrow,spar=2)$lambda`.

8.2.1 Confidence Bands for Splines

Continuing the example, the smoothing spline selected by cross-validation has a negative slope everywhere, like the regression line, but it's asymmetric — the slope is more negative to the left, and then levels off towards the regression line. (See Figure 8.2 again.) Is this real, or might the asymmetry be a sampling artifact?

We'll investigate by finding confidence bands for the spline, much as we did in Chapter 6 and homework 3 for kernel regression. Again, we need to bootstrap, and we can do it either by resampling the residuals or resampling whole data points. Let's take the latter approach, which assumes less about the data. We'll need a simulator:

```
sp.frame <- data.frame(today=sp.today,tomorrow=sp.tomorrow)
sp.resampler <- function() {
  n <- nrow(sp.frame)
  resample.rows <- sample(1:n,size=n,replace=TRUE)
  return(sp.frame[resample.rows,])
}
```

This treats the points in the scatterplot as a complete population, and then draws a sample from them, with replacement, just as large as the original⁴. We'll also need an estimator. What we want to do is get a whole bunch of spline curves, one on each simulated data set. But since the values of the input variable will change from one simulation to another, to make everything comparable we'll evaluate each spline function on a fixed grid of points, that runs along the range of the data.

```
# Set up a grid of evenly-spaced points on which to evaluate the spline
grid.300 <- seq(from=min(sp.today),to=max(sp.today),length.out=300)

sp.spline.estimator <- function(data,eval.grid=grid.300) {
  # Fit spline to data, with cross-validation to pick lambda
  fit <- smooth.spline(x=data[,1],y=data[,2],cv=TRUE)
  # Do the prediction on the grid and return the predicted values
  return(predict(fit,x=eval.grid)$y) # We only want the predicted values
}
```

This sets the number of evaluation points to 300, which is large enough to give visually smooth curves, but not so large as to be computationally unwieldy.

Now put these together to get confidence bands:

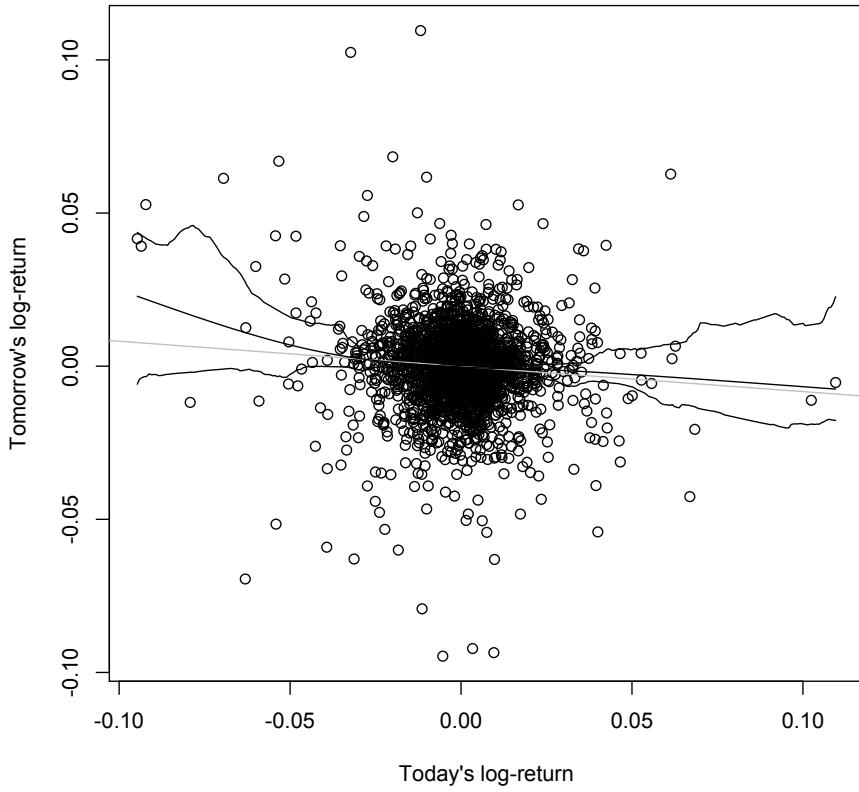
```
sp.spline.cis <- function(B,alpha,eval.grid=grid.300) {
  spline.main <- sp.spline.estimator(sp.frame,eval.grid=eval.grid)
  # Draw B bootstrap samples, fit the spline to each
  # Result has length(eval.grid) rows and B columns
  spline.boots <- replicate(B,
    sp.spline.estimator(sp.resampler(),eval.grid=eval.grid))
  cis.lower <- 2*spline.main - apply(spline.boots,1,quantile,probs=1-alpha/2)
  cis.upper <- 2*spline.main - apply(spline.boots,1,quantile,probs=alpha/2)
```

⁴§25.5 covers more refined ideas about bootstrapping time series.

```
return(list(main.curve=spline.main,lower.ci=cis.lower,upper.ci=cis.upper,
           x=seq(from=min(sp.today),to=max(sp.today),length.out=m)))
}
```

The return value here is a list which includes the original fitted curve, the lower and upper confidence limits, and the points at which all the functions were evaluated.

Figure 8.3 shows the resulting 95% confidence limits, based on $B=1000$ bootstrap replications. (Doing all the bootstrapping took 45 seconds on my laptop.) These are pretty clearly asymmetric in the same way as the curve fit to the whole data, but notice how wide they are, and how they get wider the further we go from the center of the distribution in either direction.



```

sp.cis <- sp.spline.cis(B=1000,alpha=0.05)
plot(sp.today,sp.tomorrow,xlab="Today's log-return",
     ylab="Tomorrow's log-return")
abline(lm(sp.tomorrow ~ sp.today),col="grey")
lines(x=sp.cis$x,y=sp.cis$main.curve)
lines(x=sp.cis$x,y=sp.cis$lower.ci)
lines(x=sp.cis$x,y=sp.cis$upper.ci)

```

Figure 8.3: Bootstrapped pointwise confidence band for the smoothing spline of the S & P 500 data, as in Figure 8.2. The 95% confidence limits around the main spline estimate are based on 1000 bootstrap re-samplings of the data points in the scatterplot.

8.3 Basis Functions and Degrees of Freedom

8.3.1 Basis Functions

Splines, I said, are piecewise cubic polynomials. To see how to fit them, let's think about how to fit a global cubic polynomial. We would define four **basis functions**,

$$B_1(x) = 1 \quad (8.3)$$

$$B_2(x) = x \quad (8.4)$$

$$B_3(x) = x^2 \quad (8.5)$$

$$B_4(x) = x^3 \quad (8.6)$$

with the hypothesis being that the regression function is a weight sum of these,

$$r(x) = \sum_{j=1}^4 \beta_j B_j(x) \quad (8.7)$$

That is, the regression would be linear in the *transformed* variable $B_1(x), \dots, B_4(x)$, even though it is nonlinear in x .

To estimate the coefficients of the cubic polynomial, we would apply each basis function to each data point x_i and gather the results in an $n \times 4$ matrix \mathbf{B} ,

$$B_{ij} = B_j(x_i) \quad (8.8)$$

Then we would do OLS using the \mathbf{B} matrix in place of the usual data matrix \mathbf{x} :

$$\hat{\beta} = (\mathbf{B}^T \mathbf{B})^{-1} \mathbf{B}^T \mathbf{y} \quad (8.9)$$

Since splines are piecewise cubics, things proceed similarly, but we need to be a little more careful in defining the basis functions. Recall that we have n values of the input variable x, x_1, x_2, \dots, x_n . For the rest of this section, I will assume that these are in increasing order, because it simplifies the notation. These n “knots” define $n+1$ pieces or segments $n-1$ of them between the knots, one from $-\infty$ to x_1 , and one from x_n to $+\infty$. A third-order polynomial on each segment would seem to need a constant, linear, quadratic and cubic term per segment. So the segment running from x_i to x_{i+1} would need the basis functions

$$1_{(x_i, x_{i+1})}(x), (x - x_i)1_{(x_i, x_{i+1})}(x), (x - x_i)^21_{(x_i, x_{i+1})}(x), (x - x_i)^31_{(x_i, x_{i+1})}(x) \quad (8.10)$$

where as usual the indicator function $1_{(x_i, x_{i+1})}(x)$ is 1 if $x \in (x_i, x_{i+1})$ and 0 otherwise. This makes it seem like we need $4(n+1) = 4n + 4$ basis functions.

However, we know from linear algebra that the number of basis vectors we need is equal to the number of dimensions of the vector space. The number of adjustable coefficients for an arbitrary piecewise cubic with $n+1$ segments is indeed $4n+4$, but splines are constrained to be smooth. The spline must be continuous, which means that at each x_i , the value of the cubic from the left, defined on (x_{i-1}, x_i) , must

match the value of the cubic from the right, defined on (x_i, x_{i+1}) . This gives us one constraint per data point, reducing the number of adjustable coefficients to at most $3n+4$. Since the first and second derivatives are also continuous, we are down to just $n+4$ coefficients. Finally, we know that the spline function is linear outside the range of the data, i.e., on $(-\infty, x_1)$ and on (x_n, ∞) , lowering the number of coefficients to n . There are no more constraints, so we end up needing only n basis functions. And in fact, from linear algebra, any set of n piecewise cubic functions which are linearly independent⁵ can be used as a basis. One common choice is

$$B_1(x) = 1 \quad (8.11)$$

$$B_2(x) = x \quad (8.12)$$

$$B_{i+2}(x) = \frac{(x - x_i)_+^3 - (x - x_n)_+^3}{x_n - x_i} - \frac{(x - x_{n-1})_+^3 - (x - x_n)_+^3}{x_n - x_{n-1}} \quad (8.13)$$

where $(a)_+ = a$ if $a > 0$, and $= 0$ otherwise. This rather unintuitive-looking basis has the nice property that the second and third derivatives of each B_j are zero outside the interval (x_1, x_n) .

Now that we have our basis functions, we can once again write the spline as a weighted sum of them,

$$m(x) = \sum_{j=1}^m \beta_j B_j(x) \quad (8.14)$$

and put together the matrix \mathbf{B} where $B_{ij} = B_j(x_i)$. We can write the spline objective function in terms of the basis functions,

$$n\mathcal{L} = (\mathbf{y} - \mathbf{B}\beta)^T(\mathbf{y} - \mathbf{B}\beta) + n\lambda\beta^T\Omega\beta \quad (8.15)$$

where the matrix Ω encodes information about the curvature of the basis functions:

$$\Omega_{jk} = \int B_j''(x) B_k''(x) dx \quad (8.16)$$

Notice that only the quadratic and cubic basis functions will make non-zero contributions to Ω . With the choice of basis above, the second derivatives are non-zero on, at most, the interval (x_1, x_n) , so each of the integrals in Ω is going to be finite. This is something we (or, realistically, R) can calculate *once*, no matter what λ is. Now we can find the smoothing spline by differentiating with respect to β :

$$0 = -2\mathbf{B}^T\mathbf{y} + 2\mathbf{B}^T\mathbf{B}\hat{\beta} + 2n\lambda\Omega\hat{\beta} \quad (8.17)$$

$$\mathbf{B}^T\mathbf{y} = (\mathbf{B}^T\mathbf{B} + n\lambda\Omega)\hat{\beta} \quad (8.18)$$

$$\hat{\beta} = (\mathbf{B}^T\mathbf{B} + n\lambda\Omega)^{-1}\mathbf{B}^T\mathbf{y} \quad (8.19)$$

⁵Recall that vectors $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_d$ are linearly independent when there is no way to write any one of the vectors as a weighted sum of the others. The same definition applies to functions.

Notice, incidentally, that we can now show splines are linear smoothers:

$$\hat{y} = \hat{m}(x) = \mathbf{B}\hat{\beta} \quad (8.20)$$

$$= \mathbf{B}(\mathbf{B}^T\mathbf{B} + n\lambda\Omega)^{-1}\mathbf{B}^T\mathbf{y} \quad (8.21)$$

Once again, if this were ordinary linear regression, the OLS estimate of the coefficients would be $(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$. In comparison to that, we've made two changes. First, we've substituted the basis function matrix \mathbf{B} for the original matrix of independent variables, \mathbf{x} — a change we'd have made already for plain polynomial regression. Second, the “denominator” is not $\mathbf{x}^T\mathbf{x}$, but $\mathbf{B}^T\mathbf{B} + n\lambda\Omega$. Since $\mathbf{x}^T\mathbf{x}$ is n times the covariance matrix of the independent variables, we are taking the covariance matrix of the spline basis functions and adding some extra covariance — how much depends on the shapes of the functions (through Ω) and how much smoothing we want to do (through λ). The larger we make λ , the less the actual data matters to the fit.

In addition to explaining how splines can be fit quickly (do some matrix arithmetic), this illustrates two important tricks. One, which we won't explore further here, is to turn a nonlinear regression problem into one which is linear in another set of basis functions. This is like using not just one transformation of the input variables, but a whole library of them, and letting the data decide which transformations are important. There remains the issue of selecting the basis functions, which can be quite tricky. In addition to the spline basis⁶, most choices are various sorts of waves — sine and cosine waves of different frequencies, various wave-forms of limited spatial extent (“wavelets”), etc. The ideal is to chose a function basis where only a few non-zero coefficients would need to be estimated, but this requires some understanding of the data...

The other trick is that of stabilizing an unstable estimation problem by adding a penalty term. This reduces variance at the cost of introducing some bias. Exercise 2 explores this idea.

8.3.2 Degrees of Freedom

You may have noticed that we haven't, so far, talked about the degrees of freedom of our regression models. This is one of those concepts which is much more important for linear regression than elsewhere, but it does still have its uses, and this is a good place to explain how it's calculated for more general models.

First, though, we need to recall how it works for linear regression. We'll start with an $n \times p$ data matrix of predictor variables \mathbf{x} , and an $n \times 1$ column matrix of response values \mathbf{y} . The ordinary least squares estimate of the p -dimensional coefficient vector β is

$$\hat{\beta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \quad (8.22)$$

⁶Or, really, bases; there are multiple sets of basis functions for the splines, just like there are multiple sets of basis vectors for the plane. If you see the phrase “B splines”, it refers to a particular choice of spline basis functions.

This implies, in turn, that we can write the fitted values in terms of \mathbf{x} and \mathbf{y} :

$$\hat{\mathbf{y}} = \mathbf{x}\hat{\beta} \quad (8.23)$$

$$= (\mathbf{x}(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T) \mathbf{y} \quad (8.24)$$

$$= \mathbf{h}\mathbf{y} \quad (8.25)$$

where \mathbf{h} is the $n \times n$ matrix, where h_{ij} says how much of each observed y_j contributes to each fitted \hat{y}_i . This is called the **influence matrix**, or less formally the **hat matrix**.

Notice that \mathbf{h} depends *only* on the predictor variables in \mathbf{x} ; the observed response values in \mathbf{y} don't matter. If we change around \mathbf{y} , the fitted values $\hat{\mathbf{y}}$ will also change, but only within the limits allowed by \mathbf{h} . There are n independent coordinates along which \mathbf{y} can change, so we say the data have n degrees of freedom. Once \mathbf{x} and so \mathbf{h} are fixed, however, $\hat{\mathbf{y}}$ has to lie in an $(n - p)$ -dimensional hyper-plane in this n -dimensional space. There are only $n - p$ *independent* coordinates along which the fitted values can move. Hence we say that the residual degrees of freedom are $n - p$, and p degrees of freedom are captured by the linear regression.

The algebraic expression of this fact is that, for a linear regression, the trace of \mathbf{h} is always p :

$$\text{tr } \mathbf{h} = \text{tr}(\mathbf{x}(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T) \quad (8.26)$$

$$= \text{tr}(\mathbf{x}^T \mathbf{x}(\mathbf{x}^T \mathbf{x})^{-1}) \quad (8.27)$$

$$= \text{tr} \mathbf{I}_p = p \quad (8.28)$$

since for any matrices \mathbf{a}, \mathbf{b} , $\text{tr}(\mathbf{ab}) = \text{tr}(\mathbf{ba})$, and $\mathbf{x}^T \mathbf{x}$ is a $p \times p$ matrix⁷.

For the general class of linear smoothers (Chapter 1), at an arbitrary point x the predicted value of y is a weighted (linear) combination of the observed values,

$$\hat{r}(x) = \sum_{j=1}^n \hat{w}(x, x_j) y_j \quad (8.29)$$

In particular,

$$\hat{y}_i = \hat{r}(x_i) = \sum_{j=1}^n \hat{w}(x_i, x_j) y_j \quad (8.30)$$

and so we can write

$$\hat{\mathbf{y}} = \mathbf{h}\mathbf{y} \quad (8.31)$$

where now, in the general case, $h_{ij} = \hat{w}(x_i, x_j)$. We still call \mathbf{h} the hat or influence matrix. For a kernel smoother, this can be directly calculated from the kernels, but for a spline we need to use Eq. 8.21.

By analogy with Eq. 8.28, we *define* the **effective degrees of freedom** of a linear smoother to be $\text{tr } \mathbf{h}$. Many of the formulas you learned for linear regression, e.g., dividing the residual sum of squares by $n - p$ to get an unbiased estimate of the noise variance, continue to hold approximately for linear smoothers with the effective degrees of freedom in place of p .

⁷This assumes that $\mathbf{x}^T \mathbf{x}$ has an inverse. Can you work out what happens when it does not?

8.4 Splines in Multiple Dimensions

Suppose we have *two* input variables, x and z , and a single response y . How could we do a spline fit?

One approach is to generalize the spline optimization problem so that we penalize the curvature of the spline surface (no longer a curve). The appropriate penalized least-squares objective function to minimize is

$$\mathcal{L}(m, \lambda) = \sum_{i=1}^n (y_i - m(x_i, z_i))^2 + \lambda \int \left[\left(\frac{\partial^2 m}{\partial x^2} \right)^2 + 2 \left(\frac{\partial^2 m}{\partial x \partial z} \right)^2 + \left(\frac{\partial^2 m}{\partial z^2} \right)^2 \right] dx dz \quad (8.32)$$

The solution is called a **thin-plate spline**. This is appropriate when the two input variables x and z should be treated more or less symmetrically⁸.

An alternative is use the spline basis functions from section 8.3. We write

$$m(x) = \sum_{j=1}^{M_1} \sum_{k=1}^{M_2} \beta_{jk} B_j(x) B_k(z) \quad (8.33)$$

Doing all possible multiplications of one set of numbers or functions with another is said to give their **outer product** or **tensor product**, so this is known as a **tensor product spline** or **tensor spline**. We have to chose the number of terms to include for each variable (M_1 and M_2), since using n for each would give n^2 basis functions, and fitting n^2 coefficients to n data points is asking for trouble.

8.5 Smoothing Splines versus Kernel Regression

For one input variable and one output variable, smoothing splines can basically do everything which kernel regression can do⁹. The advantages of splines are their computational speed and (once we've calculated the basis functions) simplicity, as well as the clarity of controlling curvature directly. Kernels however are easier to program (if slower to run), easier to analyze mathematically¹⁰, and extend more straightforwardly to multiple variables, and to combinations of discrete and continuous variables.

8.6 Further Reading

There are good discussions of splines in Simonoff (1996, §5), Hastie *et al.* (2009, ch. 5) and Wasserman (2006, §5.5). Wood (2006, ch. 4) includes a thorough practical

⁸Generalizations to more than two input variables are conceptually straightforward — just keep adding up more partial derivatives — but the book-keeping gets annoying.

⁹In fact, as $n \rightarrow \infty$, smoothing splines approach the kernel regression curve estimated with a specific, rather non-Gaussian kernel. See Simonoff (1996, §5.6.2).

¹⁰Most of the bias-variance analysis for kernel regression can be done with basic calculus, as we did in Chapter 4. The corresponding analysis for splines requires working in infinite-dimensional function spaces called “Hilbert spaces”. It’s a pretty theory, if you like that sort of thing.

treatment of splines as a preparation for additive models (see Chapter 9 in these notes) and generalized additive models (see Chapters 12–13).

The classic reference, by one of the people who developed splines as a useful statistical tool, is Wahba (1990), which is great if you already know what a Hilbert space is and how to navigate one.

The first introduction of spline smoothing in the statistical literature seems to be Whittaker (1922). (“Graduation” was the term often used then for what we call “smoothing”.) He begins with an “inverse probability” (we would now say “Bayesian”) argument for minimizing Eq. 8.1 to find the most probable curve, based on the *a priori* hypothesis of smooth Gaussian curves observed through Gaussian error, and gives tricks for fitting splines more easily with the mathematical technology available in 1922. He does not, however, use the word “spline”, and I am not sure when that analogy was made.

In economics and econometrics, the use of spline smoothing on time series is known as the “Hodrick-Prescott filter”, after two economists who re-discovered the technique in 1981, along with a fallacious argument that λ should always take a particular value (1600), regardless of the data¹¹. See Paige and Trindade (2010) for a (polite) discussion, and demonstration of the advantages of cross-validation.

8.7 Exercises

1. The `smooth.spline` function lets you set the effective degrees of freedom explicitly. Write a function which chooses the number of degrees of freedom by five-fold cross-validation.
2. When we can't measure our predictor variables perfectly, it seems like a good idea to try to include multiple measurements for each one of them. For instance, if we were trying to predict grades in college from grades in high school, we might include the student's grade from each year separately, rather than simply averaging them. Multiple measurements of the same variable will however tend to be strongly correlated, so this means that a linear regression will be *nearly* multi-collinear. This in turn means that it will tend to have multiple, mutually-canceling large coefficients. This makes it hard to interpret the regression and hard to treat the predictions seriously.

One strategy for coping with this situation is to carefully select the variables one uses in the regression. Another, however, is to add a penalty for large coefficient values. For historical reasons, this second strategy is called **ridge regression**, or **Tikhonov regularization**. Specifically, while the OLS estimate is

$$\hat{\beta}_{OLS} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (y_i - x_i \beta)^2, \quad (8.34)$$

¹¹As it were: Hodrick and Prescott re-invented the wheel, and decided that it should be an octagon.

the regularized or penalized estimate is

$$\hat{\beta}_{RR} = \underset{\beta}{\operatorname{argmin}} \left[\frac{1}{n} \sum_{i=1}^n (y_i - x_i \beta)^2 \right] + \sum_{j=1}^p \beta_j^2 \quad (8.35)$$

- (a) Show that the matrix form of the ridge-regression objective function is

$$n^{-1}(\mathbf{y} - \mathbf{x}\beta)^T(\mathbf{y} - \mathbf{x}\beta) + \lambda\beta^T\beta \quad (8.36)$$

- (b) Show that the optimum is

$$\hat{\beta}_{RR} = (\mathbf{x}^T \mathbf{x} + n\lambda\mathbf{I})^{-1} \mathbf{x}^T \mathbf{y} \quad (8.37)$$

- (c) What happens as $\lambda \rightarrow 0$? As $\lambda \rightarrow \infty$? (For the latter, it may help to think about the case of a one-dimensional X first.)
- (d) Let $Y = Z + \epsilon$, with $Z \sim \mathcal{U}(-1, 1)$ and $\epsilon \sim \mathcal{N}(0, 0.05)$. Generate 2000 draws from Z and Y . Now let $X_i = 0.9Z + \eta$, with $\eta \sim \mathcal{N}(0, 0.05)$, for $i \in 1 : 50$. Generate corresponding X_i values. Using the first 1000 rows of the data only, do ridge regression of Y on the X_i (not on Z), plotting the 50 coefficients as functions of λ . Explain why ridge regression is called a **shrinkage estimator**.
- (e) Use cross-validation with the first 1000 rows to pick the optimal value of λ . Compare the out-of-sample performance you get with this penalty to the out-of-sample performance of OLS.

For more on ridge regression, see Appendix E.4.1.

Chapter 9

Additive Models

9.1 Partial Residuals and Back-fitting for Linear Models

The general form of a linear regression model is

$$\mathbf{E}[Y|\vec{X} = \vec{x}] = \beta_0 + \vec{\beta} \cdot \vec{x} = \sum_{j=0}^p \beta_j x_j \quad (9.1)$$

where for $j \in 1 : p$, the x_j are the components of \vec{x} , and x_0 is always the constant 1. (Adding this fictitious constant variable lets us handle the intercept just like any other regression coefficient.)

Suppose we don't condition on all of \vec{X} but just one component of it, say X_k . What is the conditional expectation of Y ?

$$\mathbf{E}[Y|X_k = x_k] = \mathbf{E}\left[\mathbf{E}[Y|X_1, X_2, \dots, X_k, \dots, X_p] | X_k = x_k\right] \quad (9.2)$$

$$= \mathbf{E}\left[\sum_{j=0}^p \beta_j X_j | X_k = x_k\right] \quad (9.3)$$

$$= \beta_k x_k + \mathbf{E}\left[\sum_{j \neq k} \beta_j X_j | X_k = x_k\right] \quad (9.4)$$

where the first line uses the law of total expectation¹, and the second line uses Eq.

¹As you learned in baby prob., this is the fact that $\mathbf{E}[Y|X] = \mathbf{E}[\mathbf{E}[Y|X, Z]|X]$ — that we can always condition on another variable or variables (Z), provided we then average over those extra variables when we're done.

9.1. Turned around,

$$\beta_k x_k = E[Y|X_k = x_k] - E\left[\sum_{j \neq k} \beta_j X_j | X_k = x_k\right] \quad (9.5)$$

$$= E\left[Y - \left(\sum_{j \neq k} \beta_j X_j\right) | X_k = x_k\right] \quad (9.6)$$

The expression in the expectation is the k^{th} **partial residual** — the (total) residual is the difference between Y and its expectation, the partial residual is the difference between Y and what we expect it to be *ignoring* the contribution from X_k . Let's introduce a symbol for this, say $Y^{(k)}$.

$$\beta_k x_k = E[Y^{(k)} | X_k = x_k] \quad (9.7)$$

In words, if the over-all model is linear, then the partial residuals are linear. And notice that X_k is the only input feature appearing here — if we could somehow get hold of the partial residuals, then we can find β_k by doing a simple regression, rather than a multiple regression. Of course to get the partial residual we need to know all the other β_j s...

This suggests the following estimation scheme for linear models, known as the **Gauss-Seidel algorithm**, or more commonly and transparently as **back-fitting**; the pseudo-code is in Example 24.

This is an iterative approximation algorithm. Initially, we look at how far each point is from the global mean, and do a simple regression of those deviations on the first input variable. This then gives us a better idea of what the regression surface really is, and we use the deviations from *that* surface in a simple regression on the next variable; this should catch relations between Y and X_2 that weren't already caught by regressing on X_1 . We then go on to the next variable in turn. At each step, each coefficient is adjusted to fit in with what we have already guessed about the other coefficients — that's why it's called "back-fitting". It is not obvious² that this converges, but it does, and the fixed point on which it converges is the usual least-squares estimate of β .

Back-fitting is not usually how we fit linear models any more, because with modern numerical linear algebra it's actually faster to just calculate $(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y}$. But the cute thing about back-fitting is that it doesn't actually rely on the model being *linear*.

9.2 Additive Models

The **additive model** for regression is

$$E[Y|\vec{X} = \vec{x}] = \alpha + \sum_{j=1}^p f_j(x_j) \quad (9.8)$$

²Unless, I suppose, you're Gauss.

<p>Given: $n \times (p + 1)$ inputs \mathbf{x} (0th column all 1s) $n \times 1$ responses \mathbf{y} small tolerance $\delta > 0$ center \mathbf{y} and each column of \mathbf{x}</p> <pre> $\hat{\beta}_j \leftarrow 0$ for $j \in 1 : p$ until (all $\hat{\beta}_j - \gamma_j \leq \delta$) { for $k \in 1 : p$ { $y_i^{(k)} = y_i - \sum_{j \neq k} \hat{\beta}_j x_{ij}$ $\gamma_k \leftarrow$ regression coefficient of $y^{(k)}$ on $x_{\cdot k}$ $\hat{\beta}_k \leftarrow \gamma_k$ } } $\hat{\beta}_0 \leftarrow (n^{-1} \sum_{i=1}^n y_i) - \sum_{j=1}^p \hat{\beta}_j n^{-1} \sum_{i=1}^n x_{ij}$ Return: $(\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p)$</pre>	
--	--

Code Example 24: Pseudocode for back-fitting linear models. Assume we make at least one pass through the `until` loop. Recall from Chapter 1 that centering the data does not change the β_j ; this way the intercept only have to be calculated once, at the end.

This includes the linear model as a special case, where $f_j(x_j) = \beta_j x_j$, but it's clearly more general, because the f_j 's can be pretty arbitrary nonlinear functions. The idea is still that each input feature makes a separate contribution to the response, and these just add up, but these contributions don't have to be strictly proportional to the inputs. We do need to add a restriction to make it identifiable; without loss of generality, say that $E[Y] = \alpha$ and $E[f_j(X_j)] = 0$.³

Additive models keep a lot of the nice properties of linear models, but are more flexible. One of the nice things about linear models is that they are fairly straightforward to interpret: if you want to know how the prediction changes as you change x_j , you just need to know β_j . The partial response function f_j plays the same role in an additive model: of course the change in prediction from changing x_j will generally depend on the level x_j had before perturbation, but since that's also true of reality that's really a feature rather than a bug. It's true that a set of plots for f_j 's takes more room than a table of β_j 's, but it's also nicer to look at, conveys more information, and imposes fewer systematic distortions on the data.

Now, one of the nice properties which additive models share with linear ones has

³To see why we need to do this, imagine the simple case where $p = 2$. If we add constants c_1 to f_1 and c_2 to f_2 , but subtract $c_1 + c_2$ from α , then nothing *observable* has changed about the model. This degeneracy or lack of identifiability is a little like the way collinearity keeps us from defining true slopes in linear regression. But it's less harmful than collinearity because we really can fix it by the convention given above.

<pre> Given: $n \times p$ inputs \mathbf{x} $n \times 1$ responses \mathbf{y} small tolerance $\delta > 0$ one-dimensional smoother \mathcal{S} $\hat{\alpha} \leftarrow n^{-1} \sum_{i=1}^n y_i$ $\hat{f}_j \leftarrow 0$ for $j \in 1 : p$ until (all $\hat{f}_j - g_j \leq \delta$) { for $k \in 1 : p$ { $y_i^{(k)} = y_i - \sum_{j \neq k} \hat{f}_j(x_{ij})$ $g_k \leftarrow \mathcal{S}(y^{(k)} \sim x_{\cdot k})$ $\hat{f}_k \leftarrow g_k - n^{-1} \sum_{i=1}^n g_k(x_{ik})$ } } Return: $(\hat{\alpha}, \hat{f}_1, \dots, \hat{f}_p)$</pre>	
--	--

Code Example 25: Pseudo-code for back-fitting additive models. Notice the extra step, as compared to back-fitting linear models, which keeps each partial response function centered.

to do with the partial residuals. Defining

$$Y^{(k)} = Y - \left(\alpha + \sum_{j \neq k} f_j(x_j) \right) \quad (9.9)$$

a little algebra along the lines of the last section shows that

$$\mathbb{E} [Y^{(k)} | X_k = x_k] = f_k(x_k) \quad (9.10)$$

If we knew how to estimate arbitrary one-dimensional regressions, we could now use back-fitting to estimate additive models. But we have spent a lot of time talking about how to use smoothers to fit one-dimensional regressions! We could use nearest neighbors, or splines, or kernels, or local-linear regression, or anything else we feel like substituting here.

Our new, improved back-fitting algorithm in Example 25. Once again, while it's not obvious that this converges, it does converge. Also, the back-fitting procedure works well with some complications or refinements of the additive model. If we know the function form of one or another of the f_j , we can fit those parametrically (rather than with the smoother) at the appropriate points in the loop. (This would be a **semiparametric** model.) If we think that there is an interaction between x_j and x_k , rather than their making separate additive contributions, we can smooth them together; etc.

There are actually *two* standard packages for fitting additive models in R: `gam` and `mgcv`. Both have commands called `gam`, which fit **generalized** additive models — the generalization is to use the additive model for things like the probabilities of categorical responses, rather than the response variable itself. If that sounds obscure right now, don't worry — we'll come back to this in Chapters 12–13 after we've looked at generalized linear models. The last section of this chapter illustrates using these packages to fit an additive model.

9.3 The Curse of Dimensionality

Before illustrating how additive models work in practice, let's talk about why we'd want to use them. So far, we have looked at two extremes for regression models; additive models are somewhere in between.

On the one hand, we had linear regression, which is a parametric method (with $p + 1$) parameters. Its weakness is that the true regression function r is hardly ever linear, so even with infinite data it will always make systematic mistakes in its predictions — there's always some approximation bias, bigger or smaller depending on how non-linear r is. The strength of linear regression is that it converges very quickly as we get more data. Generally speaking,

$$MSE_{\text{linear}} = \sigma^2 + \alpha_{\text{linear}} + O(n^{-1}) \quad (9.11)$$

where the first term is the intrinsic noise around the true regression function, the second term is the (squared) approximation bias, and the last term is the estimation variance. Notice that the rate at which the estimation variance shrinks doesn't depend on p — factors like that are all absorbed into the big O .⁴ Other parametric models generally converge at the same rate.

At the other extreme, we've seen a number of completely non-parametric regression methods, such as kernel regression, local polynomials, k -nearest neighbors, etc. Here the limiting approximation bias is actually *zero*, at least for any reasonable regression function r . The problem is that they converge more slowly, because we need to use the data not just to figure out the coefficients of a parametric model, but the sheer shape of the regression function. We saw in Chapter 4 that the mean-squared error of kernel regression in one dimension is $\sigma^2 + O(n^{-4/5})$. Splines, k -nearest-neighbors (with growing k), etc., all attain the same rate. But in p dimensions, this becomes (Wasserman, 2006, §5.12)

$$MSE_{\text{nonpara}} - \sigma^2 = O(n^{-4/(p+4)}) \quad (9.12)$$

There's no ultimate approximation bias term here. Why does the rate depend on p ? Well, to give a very hand-wavy explanation, think of the smoothing methods, where $\hat{r}(\vec{x})$ is an average over y_i for \vec{x}_i near \vec{x} . In a p dimensional space, the volume within ϵ of \vec{x} is $O(\epsilon^p)$, so to get the same density (points per unit volume) around \vec{x} takes exponentially more data as p grows. The appearance of the 4s is a little more

⁴See Appendix B if you are not familiar with “big O ” notation.

mysterious, but can be resolved from an error analysis of the kind we did for kernel density estimation in Chapter 4⁵.

For $p = 1$, the non-parametric rate is $O(n^{-4/5})$, which is of course slower than $O(n^{-1})$, but not all that much, and the improved bias usually more than makes up for it. But as p grows, the non-parametric rate gets slower and slower, and the fully non-parametric estimate more and more imprecise, yielding the infamous **curse of dimensionality**. For $p = 100$, say, we get a rate of $O(n^{-1/26})$, which is not very good at all. Said another way, to get the same precision with p inputs that n data points gives us with one input takes $n^{(4+p)/5}$ data points. For $p = 100$, this is $n^{20.8}$, which tells us that matching the error of $n = 100$ one-dimensional observations requires $O(4 \times 10^{41})$ hundred-dimensional observations.

So completely unstructured non-parametric regressions won't work very well in high dimensions, at least not with plausible amounts of data. The trouble is that there are just *too many* possible high-dimensional functions, and seeing only a trillion points from the function doesn't pin down its shape very well at all.

This is where additive models come in. Not every regression function is additive, so they have, even asymptotically, some approximation bias. But we can estimate each f_j by a simple one-dimensional smoothing, which converges at $O(n^{-4/5})$, almost as good as the parametric rate. So overall

$$MSE_{\text{additive}} - \sigma^2 = \alpha_{\text{additive}} + O(n^{-4/5}) \quad (9.13)$$

Since linear models are a sub-class of additive models, $\alpha_{\text{additive}} \leq \alpha_{\text{lm}}$. From a purely predictive point of view, the only time to prefer linear models to additive models is when n is so small that $O(n^{-4/5}) - O(n^{-1})$ exceeds this difference in approximation biases; eventually the additive model will be more accurate.⁶

⁵More exactly, remember that in one dimension, the bias of a kernel smoother with bandwidth b is $O(b^2)$, and the variance is $O(1/nb)$, because only samples falling in an interval about b across contribute to the prediction at any one point, and when b is small, the number of such samples is proportional to nb . Adding bias squared to variance gives an error of $O(b^4) + O(1/nb)$, solving for the best bandwidth gives $b_{\text{opt}} = O(n^{-1/5})$, and the total error is then $O(n^{-4/5})$. Suppose for the moment that in p dimensions we use the same bandwidth along each dimension. (We get the same end result with more work if we let each dimension have its own bandwidth.) The bias is still $O(b^2)$, because the Taylor expansion still goes through. But now only samples falling into a region of volume $O(b^d)$ around x contribute to the prediction at x , so the variance is $O(1/nb^d)$. The best bandwidth is now $b_{\text{opt}} = O(n^{-1/(p+4)})$, yielding an error of $O(n^{-4/(p+4)})$ as promised.

⁶Unless the best additive approximation to r really is linear; then the linear model has no more bias and better variance.

9.4 Example: California House Prices Revisited

As an example, we'll revisit the housing price data from the homework. This has both California and Pennsylvania, but it's hard to visually see patterns with both states; I'll do California, and let you replicate this all on Pennsylvania, and even on the combined data.

Start with getting the data:

```
housing <- na.omit(read.csv("http://www.stat.cmu.edu/~cshalizi/uADA/13/hw/01/calif_penn_2011.csv"))
calif <- housing[housing$STATEFP==6,]
```

(How do I know that the STATEFP code of 6 corresponds to California?)

We'll fit a linear model for the log price, on the thought that it makes some sense for the factors which raise or lower house values to multiply together, rather than just adding.

```
calif.lm <- lm(log(Median_house_value) ~ Median_household_income
+ Mean_household_income + POPULATION + Total_units + Vacant_units + Owners
+ Median_rooms + Mean_household_size_owners + Mean_household_size_renters
+ LATITUDE + LONGITUDE, data = calif)
```

This is very fast — about a fifth of a second on my laptop.

Here are the summary statistics⁷:

```
> print(summary(calif.lm), signif.stars=FALSE, digits=3)
```

Call:

```
lm(formula = log(Median_house_value) ~ Median_household_income +
+ Mean_household_income + POPULATION + Total_units + Vacant_units +
Owners + Median_rooms + Mean_household_size_owners +
+ Mean_household_size_renters + LATITUDE + LONGITUDE, data = calif)
```

Residuals:

Min	1Q	Median	3Q	Max
-3.855	-0.153	0.034	0.189	1.214

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-5.74e+00	5.28e-01	-10.86	< 2e-16
Median_household_income	1.34e-06	4.63e-07	2.90	0.0038
Mean_household_income	1.07e-05	3.88e-07	27.71	< 2e-16
POPULATION	-4.15e-05	5.03e-06	-8.27	< 2e-16
Total_units	8.37e-05	1.55e-05	5.41	6.4e-08
Vacant_units	8.37e-07	2.37e-05	0.04	0.9719
Owners	-3.98e-03	3.21e-04	-12.41	< 2e-16

⁷I have suppressed the usual stars on “significant” regression coefficients, because, as discussed in Chapter 2, those are not, in fact, the most important variables, and I have reined in R's tendency to use far too many decimal places.

```
predlims <- function(preds,sigma) {
  prediction.sd <- sqrt(preds$se.fit^2+sigma^2)
  upper <- preds$fit+2*prediction.sd
  lower <- preds$fit-2*prediction.sd
  lims <- cbind(lower=lower,upper=upper)
  return(lims)
}
```

Code Example 26: Function for calculating quick-and-dirty prediction limits from a prediction object (`preds`) containing fitted values and their standard errors, and an estimate of the over-all noise level. Because those are two (independent) sources of noise, we need to combine the standard deviations by “adding in quadrature”.

Median_rooms	-1.62e-02	8.37e-03	-1.94	0.0525
Mean_household_size_owners	5.60e-02	7.16e-03	7.83	5.8e-15
Mean_household_size_renters	-7.47e-02	6.38e-03	-11.71	< 2e-16
LATITUDE	-2.14e-01	5.66e-03	-37.76	< 2e-16
LONGITUDE	-2.15e-01	5.94e-03	-36.15	< 2e-16

Residual standard error: 0.317 on 7469 degrees of freedom
 Multiple R-squared: 0.639, Adjusted R-squared: 0.638
 F-statistic: 1.2e+03 on 11 and 7469 DF, p-value: <2e-16

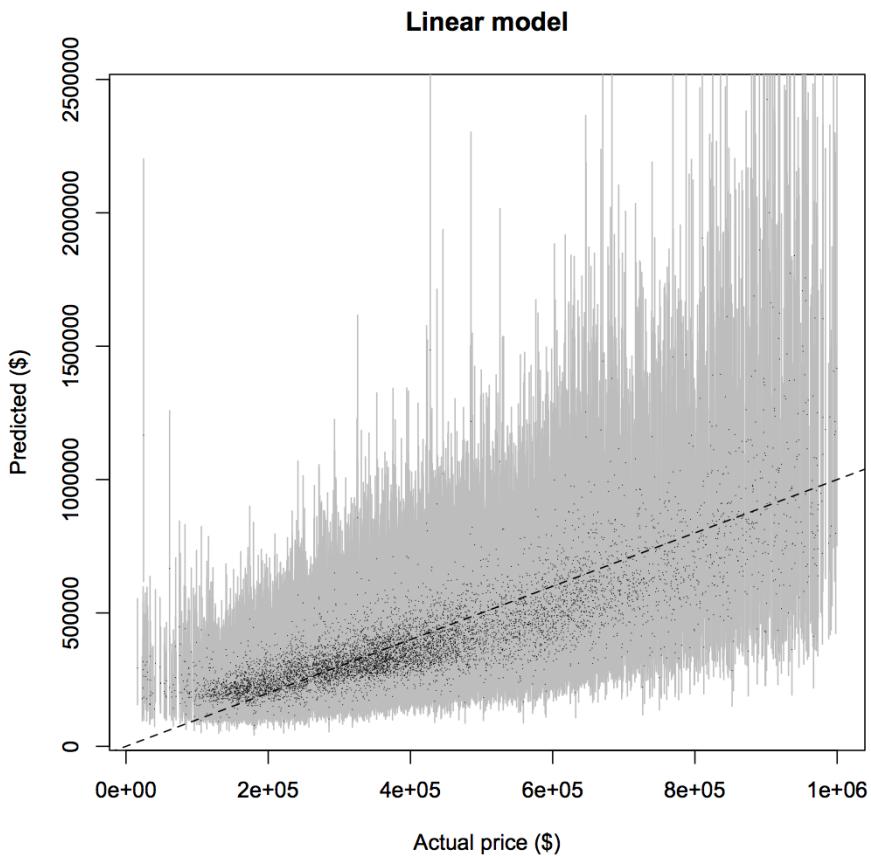
Figure 9.1 plots the predicted prices, ± 2 standard errors, against the actual prices. The predictions are not all that *accurate* — the RMS residual is 0.317 on the log scale (i.e., 37%), but they do have pretty reasonable coverage; about 96% of actual precise fall within the prediction limits⁸

```
sqrt(mean(residuals(calif.lm)^2))
mean((log(calif$Median_house_value) <= predlims.lm[, "upper"])
  & (log(calif$Median_house_value) >= predlims.lm[, "lower"])
```

On the other hand, the predictions *are* quite precise, with the median of the calculated standard errors being 0.011 (i.e., 1.1%). This linear model *thinks* it knows what’s going on.

Next, we’ll fit an additive model, using the `gam` function from the `mgcv` package; this automatically sets the bandwidths using a fast approximation to leave-one-out CV called **generalized cross-validation**, or **GCV**.

⁸Remember from your linear regression class that there are two kinds of confidence intervals we might want to use for prediction. One is a confidence interval for the *conditional mean* at a given value of x ; the other is a confidence interval for the *realized values of Y* at a given x . Earlier examples (and homework) have emphasized the former, but since we don’t know the true conditional means here, we need to use the latter sort of intervals, prediction intervals proper, to evaluate coverage. The `predlims` function in Figure 9.1 calculates a rough prediction interval by taking the standard error of the conditional mean, combining it with the estimated standard deviation, and multiplying by 2. Strictly speaking, we ought to worry about using a t -distribution rather than a Gaussian here, but with 7469 residual degrees of freedom, this isn’t going to matter much. (Assuming Gaussian noise is likely to be more of a concern, but this is only meant to be a rough cut anyway.)



```

preds.lm <- predict(calif.lm, se.fit=TRUE)
predlims.lm <- predlims(preds.lm, sigma=summary(calif.lm)$sigma)
plot(calif$Median_house_value, exp(preds.lm$fit), type="n",
     xlab="Actual price ($)", ylab="Predicted ($)", main="Linear model")
segments(calif$Median_house_value, exp(predlims.lm[, "lower"]),
          calif$Median_house_value, exp(predlims.lm[, "upper"]), col="grey")
abline(a=0, b=1, lty="dashed")
points(calif$Median_house_value, exp(preds.lm$fit), pch=16, cex=0.1)

```

Figure 9.1: Actual median house values (horizontal axis) versus those predicted by the linear model (black dots), plus or minus two *predictive* standard errors (grey bars). The dashed line shows where actual and predicted prices would be equal. Here `predict` gives both a fitted value for each point, and a standard error for that prediction. (There is no `newdata` argument in this call to `predict`, so it defaults to the training data used to learn `calif.lm`, which in this case is what we want.) I've exponentiated the predictions so that they're comparable to the original values (and because it's easier to grasp dollars than log-dollars).

14:41 Thursday 25th April, 2013

```

require(mgcv)
system.time(calif.gam <- gam(log(Median_house_value)
~ s(Median_household_income) + s(Mean_houseould_income) + s(POPULATION)
+ s(Total_units) + s(Vacant_units) + s(Owners) + s(Median_rooms)
+ s(Mean_household_size_owners) + s(Mean_household_size_renters)
+ s(LATITUDE) + s(LONGITUDE), data=calif))
user   system elapsed
5.481   0.441  17.700

```

(That is, it took almost eighteen seconds in total to run this.) The `s()` terms in the `gam` formula indicate which terms are to be smoothed — if we wanted particular parametric forms for some variables, we could do that as well. (Unfortunately we can't just write `MedianHouseValue ~ s(.)`, we have to list all the variables on the right-hand side.) The smoothing here is done by splines, and there are lots of options for controlling the splines, if you know what you're doing.

Figure 9.2 compares the predicted to the actual responses. The RMS error has improved (0.27 on the log scale, or 30%, with 96% of observations falling with ± 2 standard errors of their fitted values), at only a fairly modest cost in the claimed precision (the RMS standard error of prediction is 0.020, or 2.0%). Figure 9.3 shows the partial response functions.

It makes little sense to have latitude and longitude make separate additive contributions here; presumably they interact. We can just smooth them together⁹:

```

calif.gam2 <- gam(log(Median_house_value)
~ s(Median_household_income) + s(Mean_houseould_income) + s(POPULATION)
+ s(Total_units) + s(Vacant_units) + s(Owners) + s(Median_rooms)
+ s(Mean_household_size_owners) + s(Mean_household_size_renters)
+ s(LONGITUDE,LATITUDE), data=calif)

```

This gives an RMS error of ± 0.25 (log-scale) and 96% coverage, with a median standard error of 0.021, so accuracy is improving (at least in sample), with little loss of precision.

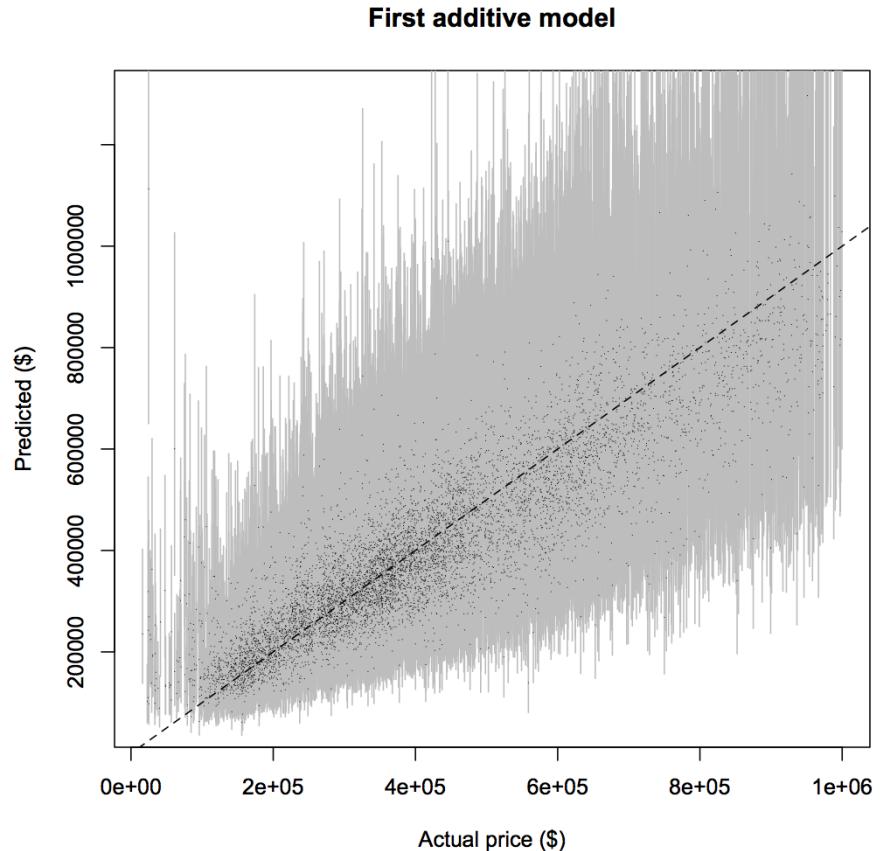
```

preds.gam2 <- predict(calif.gam2, se.fit=TRUE)
predlims.gam2 <- predlims(preds.gam2, sigma=sqrt(calif.gam2$sig2))
mean((log(calif$Median_house_value) <= predlims.gam2[, "upper"]) &
     (log(calif$Median_house_value) >= predlims.gam2[, "lower"]))

```

Figures 9.5 and 9.6 show two different views of the joint smoothing of longitude and latitude. In the perspective plot, it's quite clear that price increases specifically towards the coast, and even more specifically towards the great coastal cities. In the contour plot, one sees more clearly an inward bulge of a negative, but not too very negative, contour line (between -122 and -120 longitude) which embraces Napa, Sacramento, and some related areas, which are comparatively more developed and more expensive than the rest of central California, and so more expensive than one would expect based on their distance from the coast and San Francisco.

⁹If the two variables which interact have very different magnitudes, it's better to smooth them with a `te()` term than an `s()` term — see `help(gam.models)` — but here they are comparable.

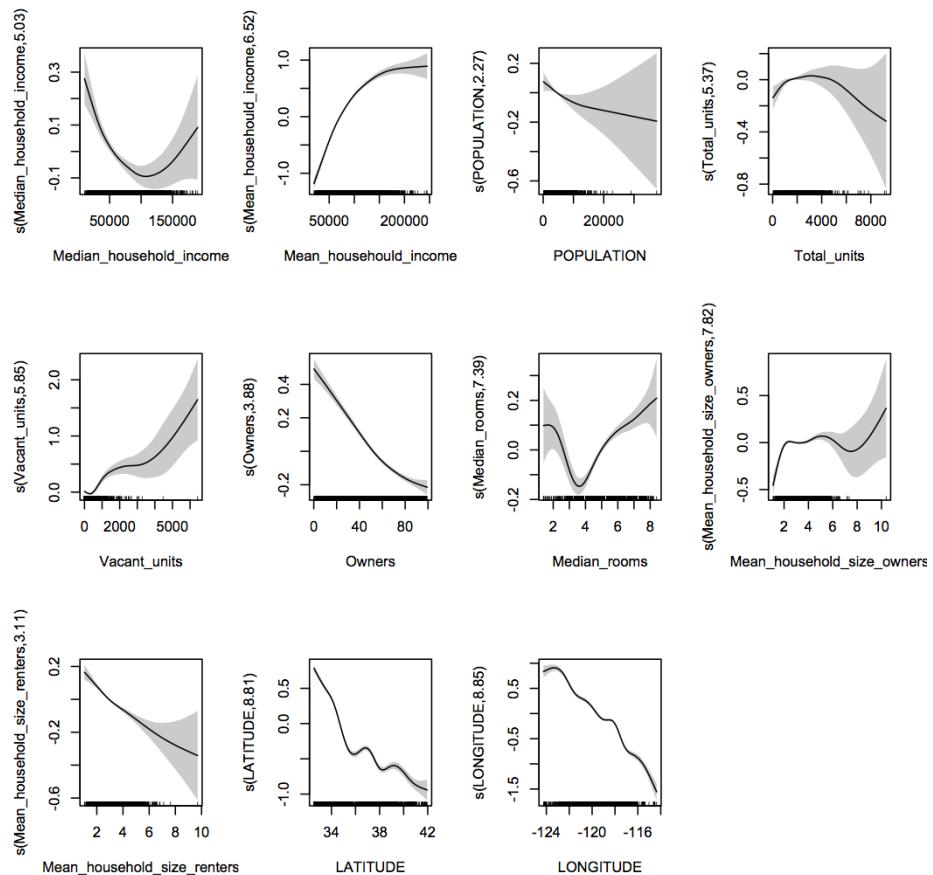


```

preds.gam <- predict(calif.gam, se.fit=TRUE)
predlims.gam <- predlims(preds.gam, sigma=sqrt(calif.gam$sig2))
plot(calif$Median_house_value, exp(preds.gam$fit), type="n",
     xlab="Actual price ($)", ylab="Predicted ($)", main="First additive model")
segments(calif$Median_house_value, exp(predlims.gam[, "lower"]),
          calif$Median_house_value, exp(predlims.gam[, "upper"]), col="grey")
abline(a=0, b=1, lty="dashed")
points(calif$Median_house_value, exp(preds.gam$fit), pch=16, cex=0.1)

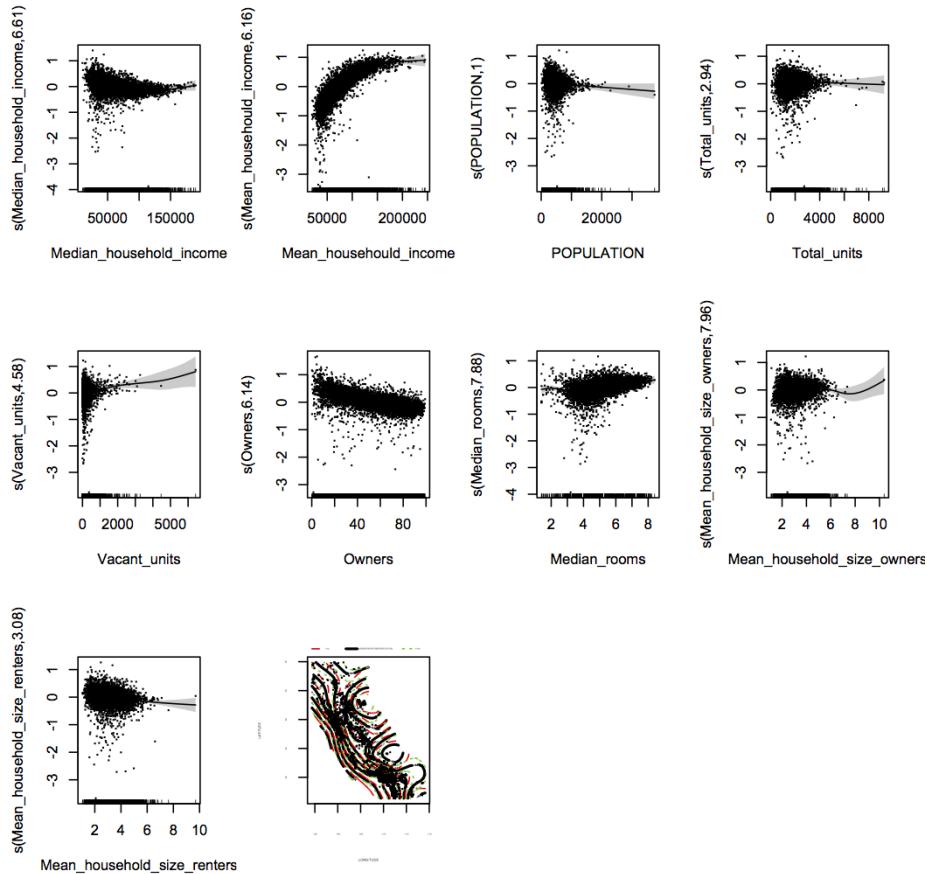
```

Figure 9.2: Actual versus predicted prices for the additive model, as in Figure 9.1. Note that the `sig2` attribute of a model returned by `gam()` is the estimate of the noise around the regression surface (σ^2).



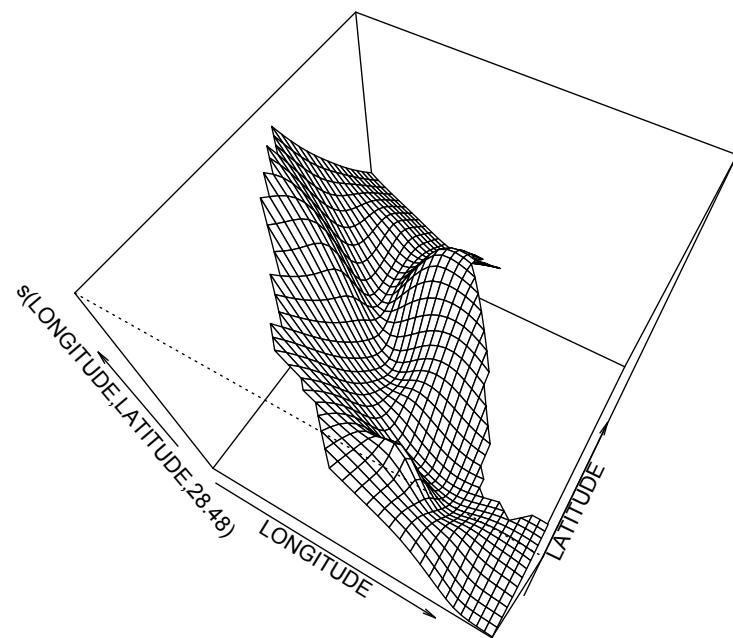
```
plot(calif.gam,scale=0,se=2,shade=TRUE,pages=1)
```

Figure 9.3: The estimated partial response functions for the additive model, with a shaded region showing ± 2 standard errors. The tick marks along the horizontal axis show the observed values of the input variables (a **rug plot**); note that the error bars are wider where there are fewer observations. Setting `pages=0` (the default) would produce eight separate plots, with the user prompted to cycle through them. Setting `scale=0` gives each plot its own vertical scale; the default is to force them to share the same one. Finally, note that here the vertical scales are logarithmic.



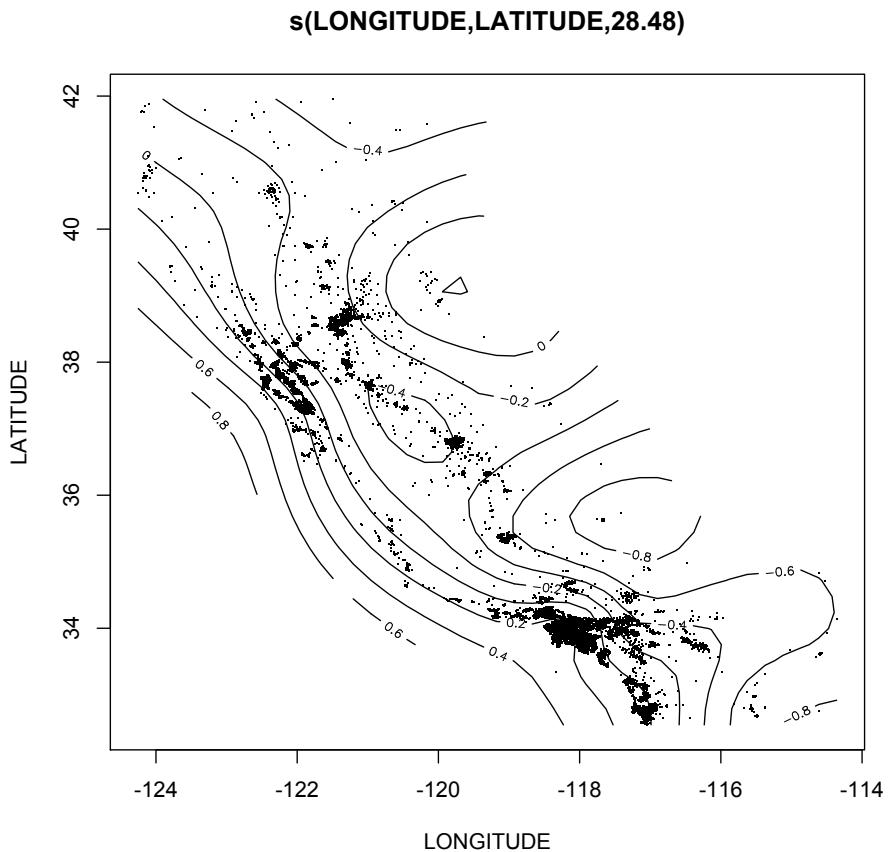
```
plot(calif.gam2,scale=0,se=2,shade=TRUE,resid=TRUE,pages=1)
```

Figure 9.4: Partial response functions and partial residuals for `addfit2`, as in Figure 9.3. See subsequent figures for the joint smoothing of longitude and latitude, which here is an illegible mess. See `help(plot.gam)` for the plotting options used here.



```
plot(calif.gam2,select=10,phi=60,pers=TRUE)
```

Figure 9.5: The result of the joint smoothing of longitude and latitude.



```
plot(calif.gam2,select=10,se=FALSE)
```

Figure 9.6: The result of the joint smoothing of longitude and latitude. Setting `se=TRUE`, the default, adds standard errors for the contour lines in multiple colors. Again, note that these are log units.

```

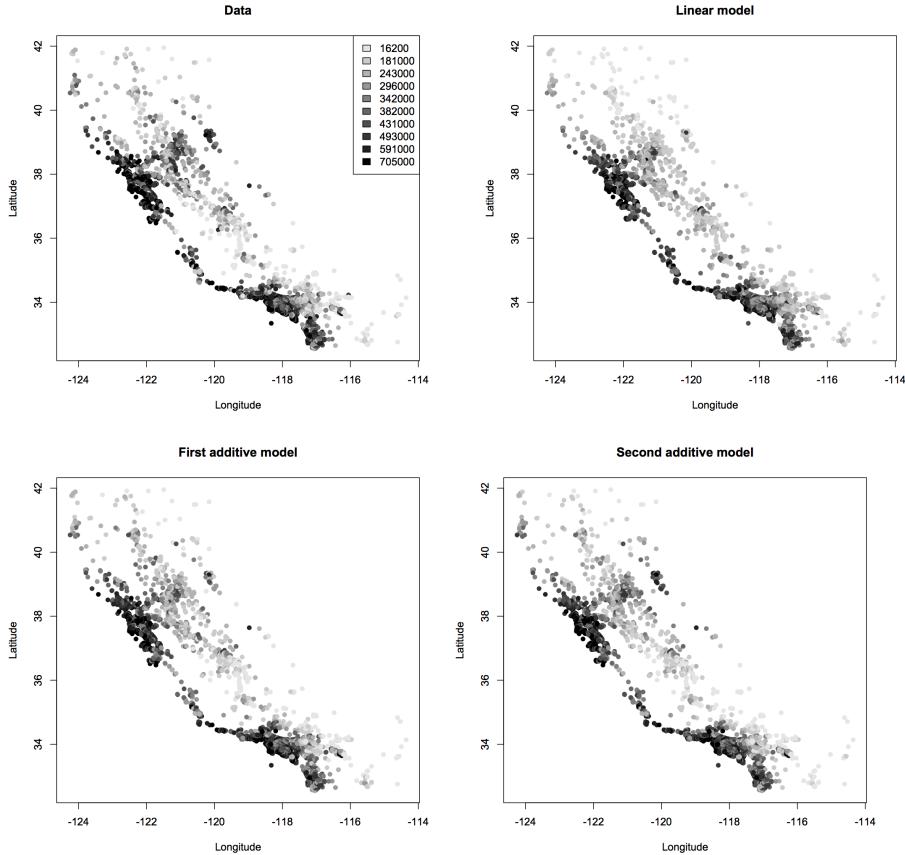
graymapper <- function(z, x=calif$LONGITUDE,y=calif$LATITUDE, n.levels=10,
breaks=NULL,break.by="length",legend.loc="topright",digits=3,...) {
  my.greys = grey(((n.levels-1):0)/n.levels)
  if (!is.null(breaks)) {
    stopifnot(length(breaks) == (n.levels+1))
  }
  else {
    if(identical(break.by,"length")) {
      breaks = seq(from=min(z),to=max(z),length.out=n.levels+1)
    } else {
      breaks = quantile(z,probs=seq(0,1,length.out=n.levels+1))
    }
  }
  z = cut(z,breaks,include.lowest=TRUE)
  colors = my.greys[z]
  plot(x,y,col=colors,bg=colors,...)
  if (!is.null(legend.loc)) {
    breaks.printable <- signif(breaks[1:n.levels],digits)
    legend(legend.loc,legend=breaks.printable,fill=my.greys)
  }
  invisible(breaks)
}

```

Code Example 27: Map-making code. In its basic use, this takes vectors for x and y coordinates, and draws gray points whose color depends on a third vector for z , with darker points indicating higher values of z . Options allow for the control of the number of gray levels, setting the breaks between levels automatically, and using a legend. Returning the break-points makes it easier to use the same scale in multiple maps. See online for commented code.

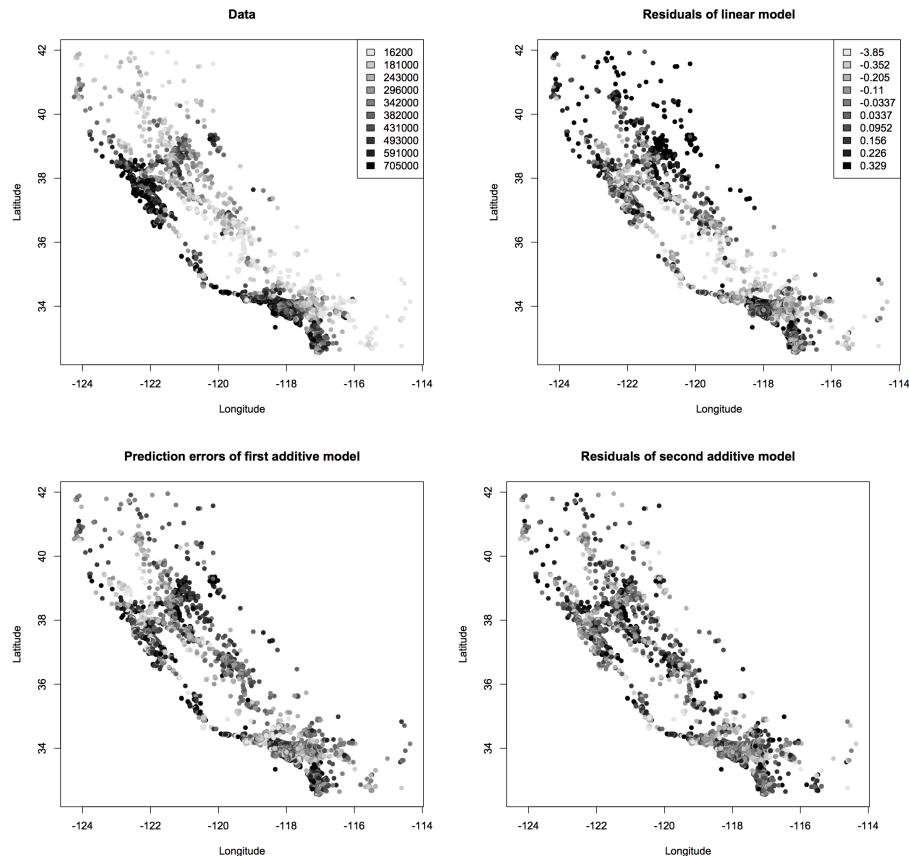
As you will recall from the homework, one of the big things wrong with the linear model was that its errors, i.e., the residuals, were highly structured and very far from random. In essence, it totally missed the existence of cities, and the fact that urban real estate is more expensive. It's a good idea, therefore, to make some maps, showing the actual values, and then, by way of contrast, the residuals of the models. Rather than do the plotting by hand over and over, let's write a function (Code Example 27).

Figures 9.7 and 9.8 show that allowing for the interaction of latitude and longitude (the smoothing term plotted in Figures 9.5–9.6) leads to a much more *random* and less systematic clumping of residuals. This is desirable in itself, even if it does little to improve the mean prediction error. Essentially, what that smoothing term is doing is picking out the existence of California's urban regions, and their distinction from the rural background. Examining the plots of the interaction term should suggest to you how inadequate it would be to just put in a $\text{LONGITUDE} \times \text{LATITUDE}$ term in a linear model.



```
calif.breaks <- graymapper(calif$Median_house_value, pch=16, xlab="Longitude",
  ylab="Latitude", main="Data", break.by="quantiles")
graymapper(exp(preds.lm$fit), breaks=calif.breaks, pch=16, xlab="Longitude",
  ylab="Latitude", legend.loc=NULL, main="Linear model")
graymapper(exp(preds.gam$fit), breaks=calif.breaks, legend.loc=NULL,
  pch=16, xlab="Longitude", ylab="Latitude", main="First additive model")
graymapper(exp(preds.gam2$fit), breaks=calif.breaks, legend.loc=NULL,
  pch=16, xlab="Longitude", ylab="Latitude", main="Second additive model")
```

Figure 9.7: Maps of real prices (top left), and those predicted by the linear model (top right), the purely additive model (bottom left), and the additive model with interaction between latitude and longitude (bottom right). Categories are deciles of the actual prices.



```

graymapper(calif$Median_house_value, pch=16, xlab="Longitude",
           ylab="Latitude", main="Data", break.by="quantiles")
errors.in.dollars <- function(x) { calif$Median_house_value - exp(fitted(x)) }
lm.breaks <- graymapper(residuals(calif.lm), pch=16, xlab="Longitude",
                        ylab="Latitude", main="Residuals of linear model",break.by="quantile")
graymapper(residuals(calif.gam), pch=16, xlab="Longitude",
           ylab="Latitude", main="Residuals errors of first additive model",
           breaks=lm.breaks, legend.loc=NULL)
graymapper(residuals(calif.gam2), pch=16, xlab="Longitude",
           ylab="Latitude", main="Residuals of second additive model",
           breaks=lm.breaks, legend.loc=NULL)

```

Figure 9.8: Actual housing values (top left), and the residuals of the three models. (The residuals are all plotted with the same color codes.) Notice that both the linear model and the additive model without spatial interaction systematically mis-price urban areas. The model with spatial interaction does much better at having randomly-scattered errors, though hardly perfect. — How would you make a map of the magnitude of regression errors?

Including an interaction between latitude and longitude in a spatial problem is pretty obvious. There are other potential interactions which might be important here — for instance, between the two measures of income, or between the total number of housing units available and the number of vacant units. We could, of course, just use a completely unrestricted nonparametric regression — going to the opposite extreme from the linear model. In addition to the possible curse-of-dimensionality issues, however, getting something like `npreg` to run with 7000 data points and 11 predictor variables requires a lot of patience. Other techniques, like nearest neighbor regression or regression trees, may run faster, though cross-validation can be demanding even there.

9.5 Closing Modeling Advice

With modern computing power, there are very few situations in which it is actually better to do linear regression than to fit an additive model. In fact, there seem to be only two good reasons to prefer linear models.

1. Our data analysis is guided by a credible scientific theory which asserts linear relationships *among the variables we measure* (not others, for which our observables serve as imperfect proxies).
2. Our data set is so massive that either the extra processing time, or the extra computer memory, needed to fit and store an additive rather than a linear model is prohibitive.

Even when the first reason applies, and we have good reasons to believe a linear theory, the truly scientific thing to do would be to *check* linearity, by fitting a flexible non-linear model and seeing if it looks close to linear. (We will see formal tests based on this idea in Chapter 10.) Even when the second reason applies, we would like to know how much bias we’re introducing by using linear predictors, which we could do by randomly selecting a subset of the data which is small enough for us to manage, and fitting an additive model.

In the vast majority of cases when users of statistical software fit linear models, neither of these reasons applies: theory doesn’t tell us to expect linearity, and our machines don’t compel us to use it. Linear regression is then employed for no better reason than that users know how to type `lm` but not `gam`. *You* now know better, and can spread the word.

9.6 Further Reading

Simon Wood, who wrote the `mgcv` package, has a very nice book about additive models and their generalizations, Wood (2006); at this level it’s your best source for further information. Buja *et al.* (1989) is a thorough theoretical treatment.

Ezekiel (1924) seems to be the first publication advocating the use of additive models as a general method, which he called “curvilinear multiple correlation”. His paper was complete with worked examples on simulated data (with known answers)

and real data (from economics)¹⁰. He was explicit that any reasonable smoothing or regression technique could be used to determine the partial response functions. He also gave a successive-approximation algorithm for finding partial response functions: start with an initial guess about all the partial responses; plot all the partial residuals; refine the partial responses simultaneously; repeat. This differs from back-fitting in that the partial response functions are updating in parallel within each cycle, not one after the other. This is a subtle difference, and Ezekiel's method will often work, but can run into trouble with correlated predictor variables, when back-fitting will not.

¹⁰“Each of these curves illustrates and substantiates conclusions reached by theoretical economic analysis. Equally important, they provide definite quantitative statements of the relationships. The method of ... curvilinear multiple correlation enable[s] us to use the favorite tool of the economist, *caeteris paribus*, in the analysis of actual happenings equally as well as in the intricacies of theoretical reasoning” (p. 453).

Chapter 10

Testing Parametric Regression Specifications with Nonparametric Regression

10.1 Testing Functional Forms

One important, but under-appreciated, use of nonparametric regression is in testing whether parametric regressions are well-specified.

The typical parametric regression model is something like

$$Y = f(X; \theta) + \epsilon \quad (10.1)$$

where f is some function which is completely specified except for the adjustable parameters θ , and ϵ , as usual, is uncorrelated noise. Usually, but not necessarily, people use a function f that is linear in the variables in X , or perhaps includes some interactions between them.

How can we tell if the specification is right? If, for example, it's a linear model, how can we check whether there might not be some nonlinearity? One common approach is to modify the specification by adding in *specific* departures from the modeling assumptions — say, adding a quadratic term — and seeing whether the coefficients that go with those terms are significantly non-zero, or whether the improvement in fit is significant.¹ For example, one might compare the model

$$Y = \theta_1 x_1 + \theta_2 x_2 + \epsilon \quad (10.2)$$

to the model

$$Y = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \epsilon \quad (10.3)$$

by checking whether the estimated θ_3 is significantly different from 0, or whether the residuals from the second model are significantly smaller than the residuals from the first.

¹In my experience, this is second in popularity only to ignoring the issue.

This can work, *if* you have chosen the right nonlinearity to test. It has the power to detect certain mis-specifications, if they exist, but not others. (What if the departure from linearity is not quadratic but cubic?) If you have good reasons to think that when the model is wrong, it can only be wrong in certain ways, fine; if not, though, why only check for those errors?

Nonparametric regression effectively lets you check for *all* kinds of systematic errors, rather than singling out a particular one. There are three basic approaches, which I give in order of increasing sophistication.

- If the parametric model is right, it should predict as well as, or even better than, the non-parametric one, and we can check whether $MSE_p(\hat{\theta}) - MSE_{np}(\hat{r})$ is sufficiently small.
- If the parametric model is right, the non-parametric estimated regression curve should be very close to the parametric one. So we can check whether $f(x; \hat{\theta}) - \hat{r}(x)$ is approximately zero everywhere.
- If the parametric model is right, then its *residuals* should be patternless and independent of input features, because

$$\mathbf{E}[Y - f(x; \theta)|X] = \mathbf{E}[f(x; \theta) + \epsilon - f(x; \theta)|X] = \mathbf{E}[\epsilon|X] = 0 \quad (10.4)$$

So we can apply non-parametric smoothing to the parametric residuals, $y - f(x; \hat{\theta})$, and see if their expectation is approximately zero everywhere.

We'll stick with the first procedure, because it's simpler for us to implement computationally. However, it turns out to be easier to develop theory for the other two, and especially for the third — see Li and Racine (2007, ch. 12), or Hart (1997).

Here is the basic procedure.

1. Get data $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.
2. Fit the parametric model, getting an estimate $\hat{\theta}$, and in-sample mean-squared error $MSE_p(\hat{\theta})$.
3. Fit your favorite nonparametric regression (using cross-validation to pick control settings as necessary), getting curve \hat{r} and in-sample mean-squared error $MSE_{np}(\hat{r})$.
4. Calculate $\hat{t} = MSE_p(\hat{\theta}) - MSE_{np}(\hat{r})$.
5. Simulate from the parametric model $\hat{\theta}$ to get faked data $(x'_1, y'_1), \dots, (x'_n, y'_n)$.
6. Fit the parametric model to the simulated data, getting estimate $\tilde{\theta}$ and $MSE_p(\tilde{\theta})$.
7. Fit the nonparametric model to the simulated data, getting estimate \tilde{r} and $MSE_{np}(\tilde{r})$.

8. Calculate $\tilde{T} = MSE_p(\hat{\theta}) - MSE_{np}(\tilde{r})$.
9. Repeat steps 5–8 many times to get an estimate of the distribution of T .
10. The p -value is $\frac{1+\#\{\tilde{T} > \hat{T}\}}{1+\#T}$.

Let's step through the logic. In general, the error of the non-parametric model will be converging to the smallest level compatible with the intrinsic noise of the process. What about the parametric model?

Suppose on the one hand that the parametric model is correctly specified. Then its error will also be converging to the minimum — by assumption, it's got the functional form right so bias will go to zero, and as $\hat{\theta} \rightarrow \theta_0$, the variance will also go to zero. In fact, with enough data the correctly-specified parametric model will actually *generalize* better than the non-parametric model².

Suppose on the other hand that the parametric model is mis-specified. Then it is predictions are systematically wrong, even with unlimited amounts of data — there's some bias which never goes away, no matter how big the sample. Since the non-parametric smoother does eventually come arbitrarily close to the true regression function, the smoother will end up predicting better than the parametric model.

Smaller errors for the smoother, then, suggest that the parametric model is wrong. But since the smoother has higher capacity, it could easily get smaller errors on a particular sample by chance and/or over-fitting, so only *big* differences in error count as evidence. Simulating from the parametric model gives us surrogate data which looks just like reality ought to, *if* the model is true. We then see how much better we could expect the non-parametric smoother to fit *under the parametric model*. If the non-parametric smoother fits the actual data much better than this, we can reject the parametric model with high confidence: it's really unlikely that we'd see that big an improvement from using the nonparametric model just by luck.³

As usual, we simulate from the parametric model simply because we have no hope of working out the distribution of the differences in MSEs from first principles. This is an example of our general strategy of bootstrapping.

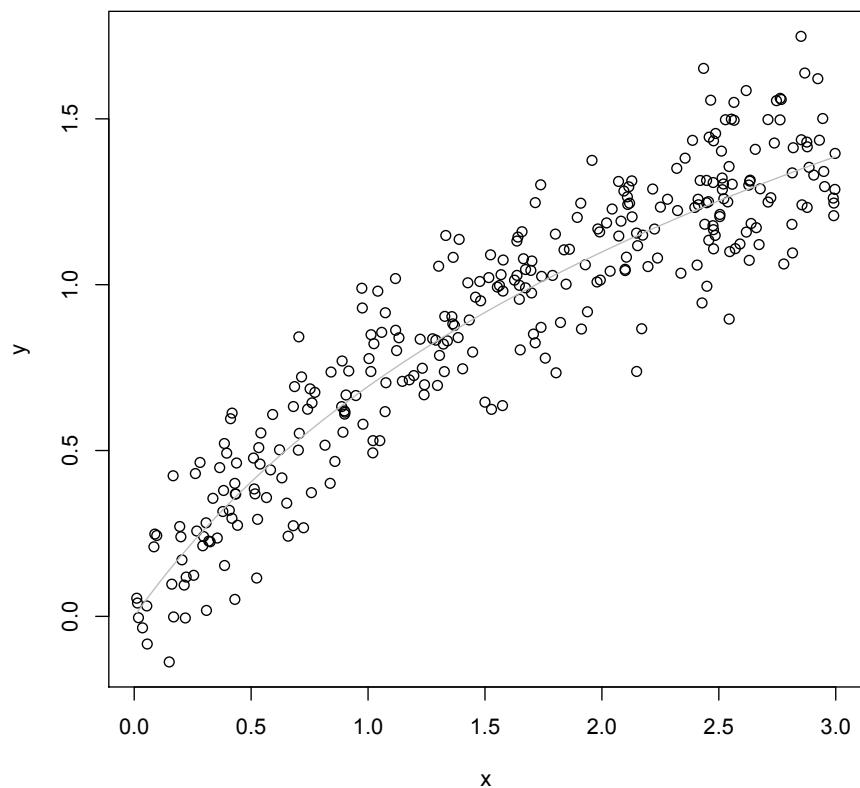
10.1.1 Examples of Testing a Parametric Model

Let's see this in action. First, let's detect a reasonably subtle nonlinearity. Take the non-linear function $g(x) = \log x + 1$, and say that $Y = g(x) + \epsilon$, with ϵ being IID Gaussian noise with mean 0 and standard deviation 0.15. (This is one of the two examples from the notes to Lecture 4.) Figure 10.1 shows the regression function and the data. The nonlinearity is clear with the curve to “guide the eye”, but fairly subtle.

A simple linear regression looks pretty good:

²Remember that the smoother must, so to speak, use up some of the degrees of freedom in the data to figure out the shape of the regression function. The parametric model, on the other hand, takes the shape of the basic shape regression function as given, and uses all the degrees of freedom to tune its parameters.

³As usual with p -values, this is not symmetric. A high p -value might mean that the true regression function is very close to $r(x; \theta)$, or it might just mean that we don't have enough data to draw conclusions.



```

x  <- runif(300,0,3)
yg <- log(x+1)+rnorm(length(x),0,0.15)
gframe <- data.frame(x=x,y=yg)
plot(x,yg,xlab="x",ylab="y")
curve(log(1+x),col="grey",add=TRUE)

```

Figure 10.1: True regression curve (grey) and data points (circles). The curve $g(x) = \log x + 1$.

```

> glinfit = lm(y~x,data=gframe)
> print(summary(glinfit),signif.stars=FALSE,digits=2)

Call:
lm(formula = y ~ x, data = gframe)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.416 -0.115  0.004  0.118  0.387 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.208     0.019     11 <2e-16    
x            0.434     0.011     41 <2e-16    

Residual standard error: 0.16 on 298 degrees of freedom
Multiple R-squared:  0.85, Adjusted R-squared:  0.85 
F-statistic: 1.7e+03 on 1 and 298 DF,  p-value: <2e-16

```

R^2 is ridiculously high — the regression line preserves 85% of the variance in the data. The p -value reported by R is also very, very low, which seems good, but remember all this really means is “you’d have to be crazy to think a flat line fit better than one with a slope” (Figure 10.2)

The in-sample MSE of the linear fit⁴

```

> mean(residuals(glinfit)^2)
[1] 0.02617729

```

The nonparametric regression has a somewhat smaller MSE⁵

```

> gnpr <- npreg(y~x,data=gframe)

> gnpr$MSE
[1] 0.02163506

```

So $\hat{t} = 0.0045$:

```

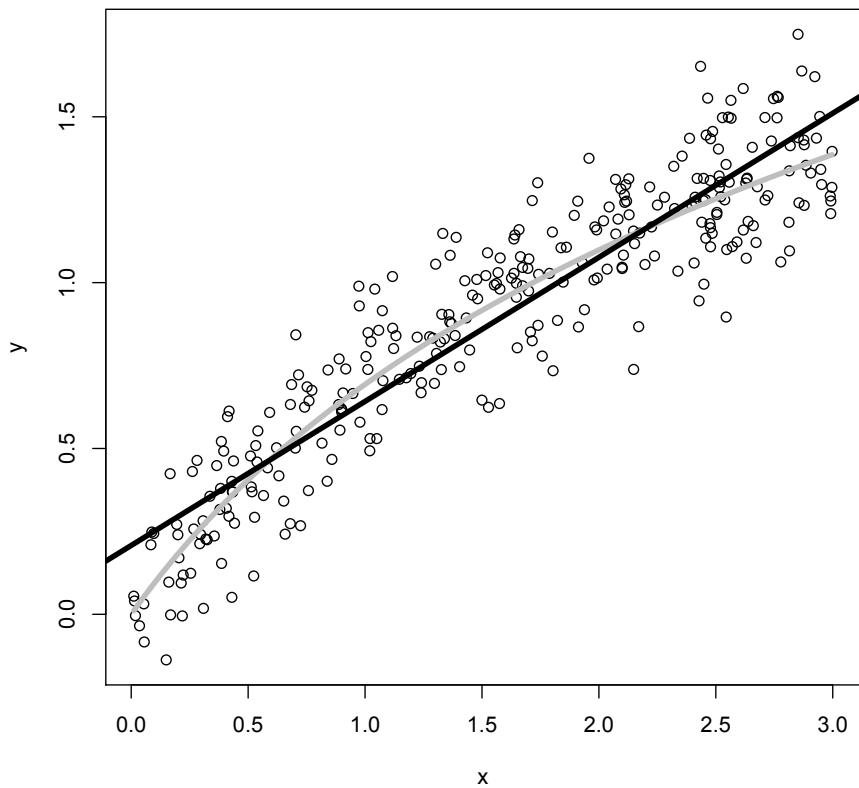
> t.hat = mean(glinfit$residual^2) - gnpr$MSE
> t.hat
[1] 0.004542232

```

Now we need to simulate from the fitted parametric model, using its estimated coefficients and noise level. We have seen several times now how to do this. The function `sim.lm` in Example 28 does this, along the same lines as the examples in

⁴If we ask R for the MSE, by doing `summary(glinfit)$sigma2`, we get 0.02635298. These differ by a factor of $n/(n - 2) = 300/298 = 1.0067$, because R is trying to estimate the out-of-sample error by scaling up the in-sample error, the same way the estimated population variance scales up the sample variance. We want to compare in-sample fits.

⁵`npreg` does not apply the kind of correction mentioned in the previous footnote.



```
plot(x,yg,xlab="x",ylab="y")
curve(log(1+x),col="grey",add=TRUE,lwd=4)
abline(glinfit,lwd=4)
```

Figure 10.2: As previous figure, but adding the least-squares regression line (black). Line widths exaggerated for clarity.

```
# One surrogate data set for simple linear regression
# Inputs: linear model (linfit), x values at which to
# simulate (test.x)
# Outputs: Data frame with columns x and y
sim.lm <- function(linfit, test.x) {
  n <- length(test.x)
  sim.frame <- data.frame(x=test.x)
  # Add the y column later
  sigma <- summary(linfit)$sigma*(n-2)/n # MLE value
  y.sim <- predict(linfit,newdata=sim.frame)
  y.sim <- y.sim + rnorm(n,0,sigma) # Add noise
  sim.frame <- data.frame(sim.frame,y=y.sim) # Adds column
  return(sim.frame)
}
```

Code Example 28: Simulate a new data set from a linear model, assuming homoskedastic Gaussian noise. It also assumes that there is one input variable, x , and that the response variable is called y . Could you modify it to work with multiple regression?

Chapter 6; it assumes homoskedastic Gaussian noise. Again, as before, we need a function which will calculate the difference in MSEs between a linear model and a kernel smoother fit to the same data set — which will do automatically what we did by hand above. This is `calc.T` in Example 29. Note that the kernel bandwidth has to be re-tuned to each new data set.

If we call `calc.T` on the output of `sim.lm`, we get one value of the test statistic under the null distribution:

```
> calc.T(sim.lm(glinfit,x))
[1] 0.001513319
```

Now we just repeat this a lot to get a good approximation to the sampling distribution of T under the null hypothesis:

```
null.samples.T <- replicate(200,calc.T(sim.lm(glinfit,x)))
```

This takes some time, because each replication involves not just generating a new simulation sample, but also cross-validation to pick a bandwidth. This adds up to about a second per replicate on my laptop, and so a couple of minutes for 200 replicates.

(While the computer is thinking, look at the command a little more closely. It leaves the x values alone, and only uses simulation to generate new y values. This is appropriate here because our model doesn't really *say* where the x values came from; it's just about the conditional distribution of Y given X . If the model we were testing specified a distribution for x , we should generate x each time we invoke `calc.T`. If the specification is vague, like “ x is IID” but with no particular distribution, then use the nonparametric bootstrap. The command would be something like

```

# Calculate the difference-in-MSEs test statistic
# Inputs: A data frame (my.data)
# Presumes: data has columns "x" and "y", which are input
# and response
# Calls: np::npreg
# Output: Difference in MSEs between linear model and
# kernel smoother
calc.T <- function(data) {
  # Fit the linear model, extract residuals, calculate MSE
  MSE.p <- mean((lm(y~x, data=data)$residuals)^2)
  # npreg gets unhappy when called with a "data" argument
  # that is defined inside this function; npregbw does
  # not complain
  MSE.np.bw <- npregbw(y~x,data=data)
  MSE.np <- npreg(MSE.np.bw)$MSE
  return(MSE.p - MSE.np)
}

```

Code Example 29: Calculate the difference-in-MSEs test statistic.

```
replicate(200,calc.T(sim.lm(glinfit,resample(x))))
```

using the `resample` function from lecture 8, to draw a different bootstrap sample of x each time.)

When it's done, we can plot the distribution and see that the observed value \hat{t} is pretty far out along the right tail (Figure 10.3). This tells us that it's very unlikely that `npreg` would improve so much on the linear model if the latter were true. In fact, *none* of the bootstrap replicates were that big:

```
> sum(null.samples.T > t.hat)
[1] 0
```

so our estimated p -value is $\frac{1}{201}$. We can thus reject the linear model pretty confidently.⁶

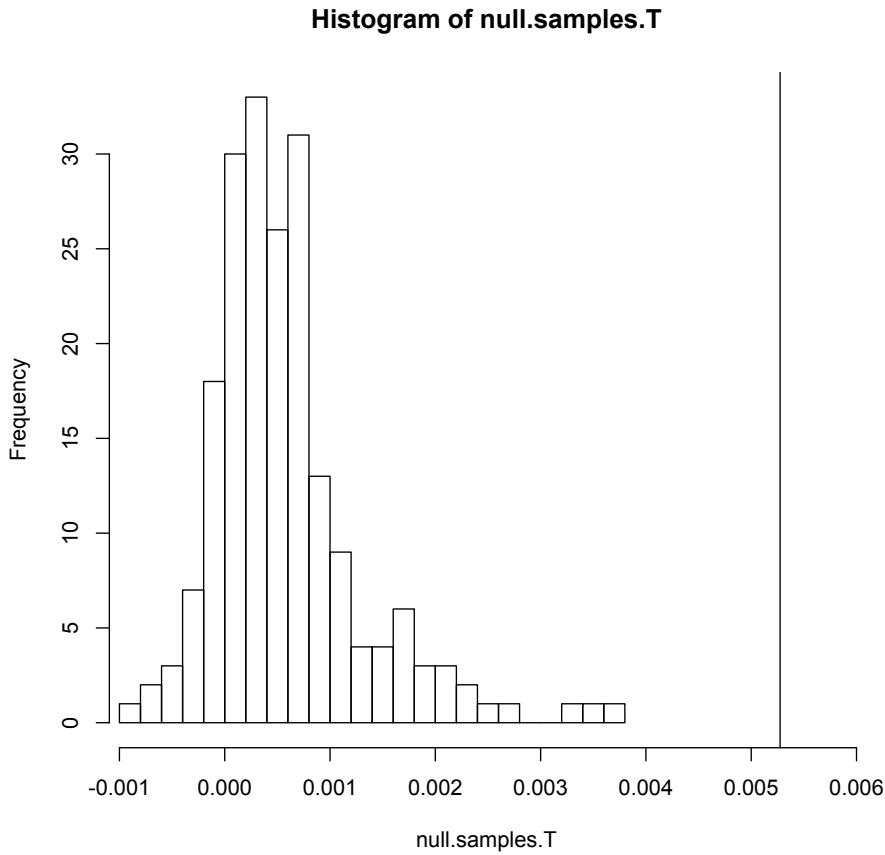
As a second example, let's suppose that the linear model *is* right — then the test should give us a high p -value. So let us stipulate that in reality

$$Y = 0.2 + 0.5x + \eta \quad (10.5)$$

with $\eta \sim \mathcal{N}(0, 0.15^2)$. Figure 10.4 shows data from this, of the same size as before.

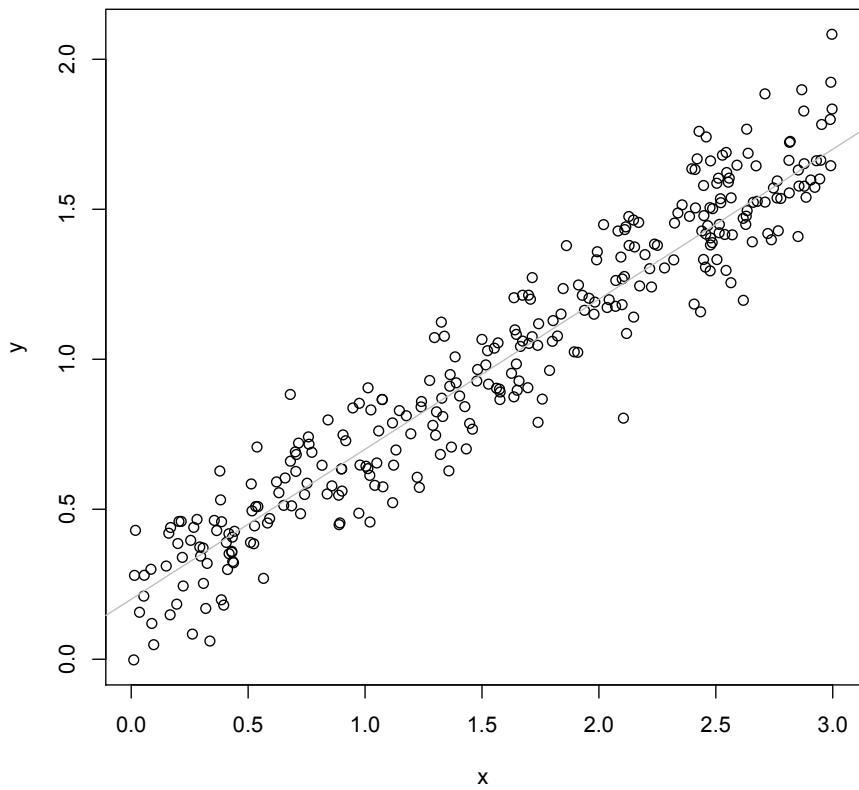
Repeating the same exercise as before, we get that $\hat{t} = 6.8 \times 10^{-4}$, together with a slightly different null distribution (Figure 10.5). Now the p -value is 32%, which one would be quite rash to reject.

⁶If we wanted a more precise estimate of the p -value, we'd need to use more bootstrap samples.



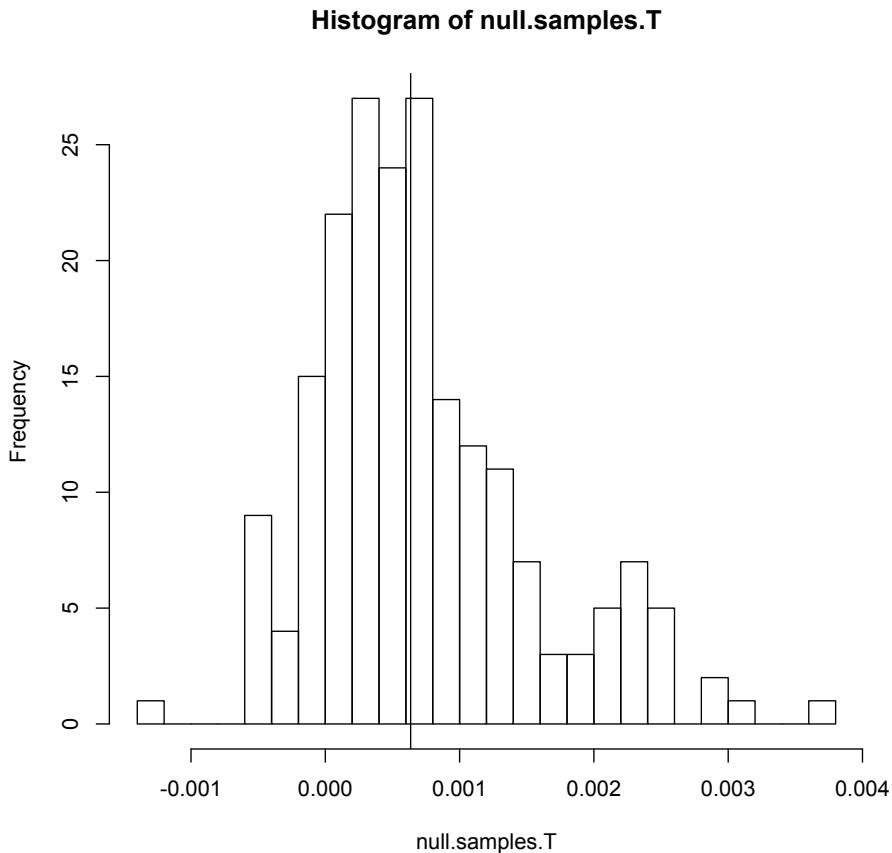
```
hist(null.samples.T,n=31,xlim=c(min(null.samples.T),1.1*t.hat),probability=TRUE)
abline(v=t.hat)
```

Figure 10.3: Histogram of the distribution of $T = MSE_p - MSE_{np}$ for data simulated from the parametric model. The vertical line mark the observed value. Notice that the mode is positive and the distribution is right-skewed; this is typical.



```
y2 = 0.2+0.5*x + rnorm(length(x),0,0.15)
y2.frame <- data.frame(x=x,y=y2)
plot(x,y2,xlab="x",ylab="y")
abline(0.2,0.5,col="grey")
```

Figure 10.4: Data from the linear model (true regression line in grey).



```

y2.fit <- lm(y~x,data=y2.frame)
null.samples.T.y2 <- replicate(200,calc.T(sim.lm(y2.fit,x)))
t.hat2 <- calc.T(y2.frame)
hist(null.samples.T.y2,n=31,probability=TRUE)
abline(v=t.hat2)

```

Figure 10.5: As in Figure 10.3, but using the data and fits from Figure 10.4.

10.1.2 Remarks

10.1.2.0.1 Other Nonparametric Regressions There is nothing especially magical about using kernel regression here. Any consistent nonparametric estimator (say, your favorite spline) would work. They may differ somewhat in their answers on particular cases.

10.1.2.0.2 Additive Alternatives For multivariate regressions, testing against a fully nonparametric alternative can be very time-consuming, as well as running up against curse-of-dimensionality issues⁷. A compromise is to test the parametric regression against an additive model. Essentially nothing has to change.

10.1.2.0.3 Testing $E[\hat{\epsilon}|X] = 0$ I mentioned at the beginning of the chapter that one way to test whether the parametric model is correctly specified is to test whether the residuals have expectation zero everywhere. This amounts to (i) finding the residuals by fitting the parametric model, and (ii) comparing the MSE of the “model” that they have expectation zero with a nonparametric smoothing of the residuals. We just have to be careful that we simulate from the fitted parametric model, and not just by resampling the residuals.

10.1.2.0.4 Stabilizing the Sampling Distribution of the Test Statistic I have just looked at the difference in MSEs. The bootstrap principle being invoked is that the sampling distribution of the test statistic, under the estimated parametric model, should be close the distribution under the true parameter value. As discussed in Chapter 6, sometimes some massaging of the test statistic helps bring these distributions closer. Some modifications to consider:

- Divide the MSE difference by an estimate of the noise σ .
- Divide by an estimate of the noise σ times the difference in degrees of freedom, using the estimated, effective degrees of freedom of the nonparametric regression.
- Use the log ratio of MSEs instead of the MSE difference.

Doing a double bootstrap can help you assess whether these are necessary.

⁷This curse manifests itself here as a loss of power in the test.

10.2 Why Use Parametric Models At All?

It might seem by this point that there is little point to using parametric models at all. Either our favorite parametric model is right, or it isn't. If it is right, then a consistent nonparametric estimate will eventually approximate it arbitrarily closely. If the parametric model is wrong, it will not self-correct, but the non-parametric estimate will eventually show us that the parametric model doesn't work. Either way, the parametric model seems superfluous.

There are two things wrong with this line of reasoning — two good reasons to use parametric models.

1. One use of statistical models, like regression models, is to connect scientific theories to data. The theories are about the mechanisms generating the data. Sometimes these hypotheses are “tight” enough to tell us what the functional form of the regression should be, or even what the distribution of noise terms should be, but still contain unknown parameters. In this case, the parameters themselves are substantively meaningful and interesting — we *don't* just care about prediction. It can be very hard to relate non-parametric smoothing curves to aspects of scientific theories in the same way.⁸
2. Even if all we care about *is* prediction accuracy, there is still the bias-variance trade-off to consider. Non-parametric smoothers will have larger variance in their predictions, at the same sample size, than correctly-specified parametric models, simply because the former are more flexible. Both models are converging on the true regression function, but the parametric model converges faster, because it searches over a more confined space. In terms of total prediction error, the parametric model's low variance plus vanishing bias beats the non-parametric smoother's larger variance plus vanishing bias. (Remember that this is part of the logic of testing parametric models in the previous section.) In the next section, we will see that this argument can actually be pushed further, to work with not-quite-correctly specified models.

Of course, both of these advantages of parametric models only obtain *if* they are well-specified. If we want to claim those advantages, we need to check the specification.

⁸On the other hand, it is not uncommon for scientists to write down theories positing linear relationships between variables, not because they actually believe that, but because that's the only thing they know how to estimate statistically.

10.3 Why We Sometimes Want Mis-Specified Parametric Models

Low-dimensional parametric models have potentially high bias (if the real regression curve is very different from what the model posits), but low variance (because there isn't that much to estimate). Non-parametric regression models have low bias (they're flexible) but high variance (they're flexible). If the parametric model is true, it can converge *faster* than the non-parametric one. Even if the parametric model isn't quite true, a small bias plus low variance can sometimes still beat a non-parametric smoother's smaller bias and substantial variance. With enough data the non-parametric smoother will eventually over-take the mis-specified parametric model, but with small samples we might be better off embracing bias.

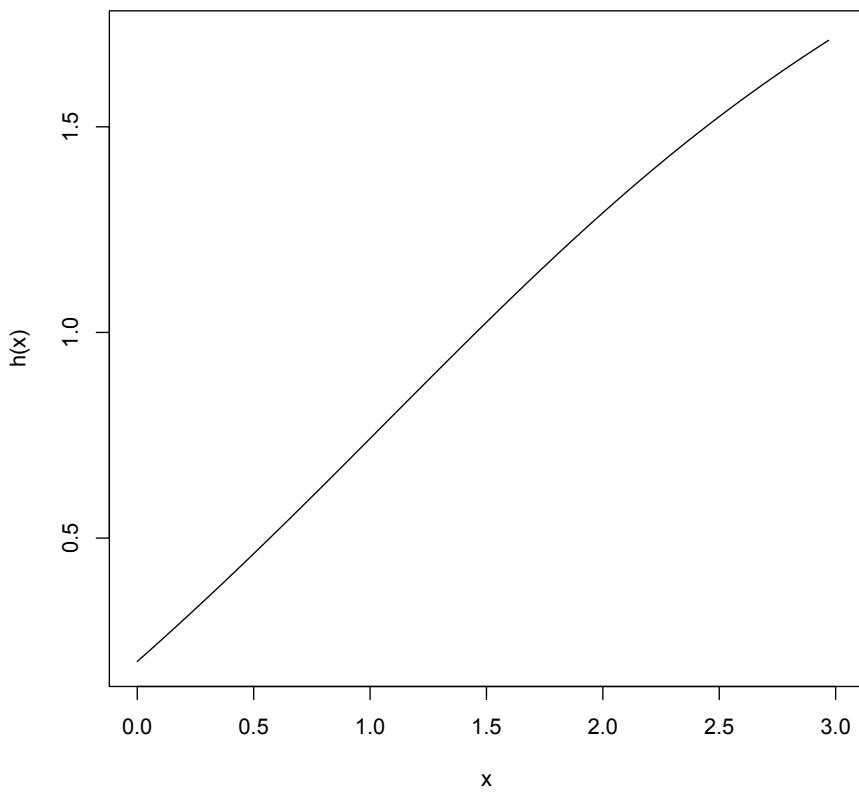
To illustrate, suppose that the true regression function is

$$E[Y|X=x] = 0.2 + \frac{1}{2} \left(1 + \frac{\sin x}{10} \right) x \quad (10.6)$$

This is very nearly linear over small ranges — say $x \in [0, 3]$ (Figure 10.6).

I will use the fact that I know the true model here to calculate the actual expected generalization error, by averaging over many samples (Example 30).

Figure 10.7 shows that, out to a fairly substantial sample size (≈ 500), the lower bias of the non-parametric regression is systematically beaten by the lower variance of the linear model — though admittedly not by much.

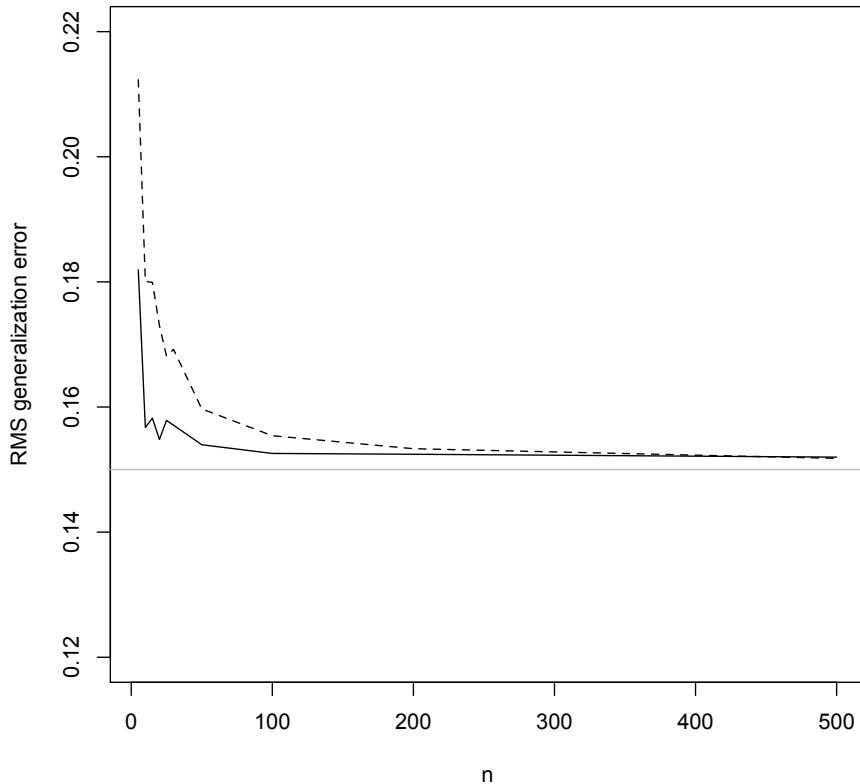


```
h <- function(x) { 0.2 + 0.5*(1+sin(x)/10)*x }
curve(h(x),from=0,to=3)
```

Figure 10.6: Graph of $h(x) = 0.2 + \frac{1}{2} \left(1 + \frac{\sin x}{10}\right) x$ over $[0, 3]$.

```
nearly.linear.out.of.sample = function(n) {  
  # Combines simulating the true model with fitting  
  # parametric model and smoother, calculating MSEs  
  x=seq(from=0,to=3,length.out=n)  
  y = h(x) + rnorm(n,0,0.15)  
  data <- data.frame(x=x,y=y)  
  y.new = h(x) + rnorm(n,0,0.15)  
  sim.lm <- lm(y~x,data=data)  
  lm.mse = mean((fitted(sim.lm) - y.new )^2)  
  sim.np.bw <- npregbw(y~x,data=data)  
  sim.np <- npreg(sim.np.bw)  
  np.mse = mean((sim.np$mean - y.new)^2)  
  mses <- c(lm.mse,np.mse)  
  return(mses)  
}  
  
nearly.linear.generalization = function(n,m=100) {  
  raw = replicate(m,nearly.linear.out.of.sample(n))  
  reduced = rowMeans(raw)  
  return(reduced)  
}
```

Code Example 30: Evaluating the out-of-sample error for the nearly-linear problem as a function of n , and evaluting the generalization error by averaging over many samples.



```

sizes = c(5,10,15,20,25,30,50,100,200,500)
generalizations = supply(sizes,nearly.linear.generalization)
plot(sizes,sqrt(generalizations[1,]),ylim=c(0.12,0.22),type="l",
     xlab="n",ylab="RMS generalization error")
lines(sizes,sqrt(generalizations[2,]),lty=2)
abline(h=0.15,col="grey")
    
```

Figure 10.7: Root-mean-square generalization error for linear model (solid line) and kernel smoother (dashed line), fit to the same sample of the indicated size. The true regression curve is as in 10.6, and observations are corrupted by IID Gaussian noise with $\sigma = 0.15$ (grey horizontal line). The cross-over after which the nonparametric regressor has better generalization performance happens shortly before $n = 500$.

Chapter 11

More about Hypothesis Testing

[[To come]]

[[The logic of hypothesis testing: learning by eliminating alternatives. Error by leaping to conclusions and error by refusing truths, or significance, power, and the will to believe. Choice of test statistic as a filter on the data. P-values and their calculation; the importance of sampling distributions. Exact formulas for sampling distributions as short-cuts for exhaustive simulation. Gygax tests: properly sized, correctly calculated p -values, utterly useless as evidence. The importance of power, and of sensitivity of the sampling distribution to the truth. Size-power trade-off; ROC curves. Ways of increasing power or easing the size-power trade-off. Role of modeling assumptions in controlling power. Substantive vs. statistical significance once more. Confidence sets: the confidence set as a bet; turning hypothesis tests into confidence sets; turning confidence sets into hypothesis tests. Why it is generally better to report confidence intervals than p-values. Some strategic advice: don't use statistical significance as a substitute for thought; avoid dead-salmon testing; test *important* null hypotheses against interesting alternatives; test background assumptions; try sensitivity analyses; be careful of interpretations.]]

Chapter 12

Logistic Regression

12.1 Modeling Conditional Probabilities

So far, we either looked at estimating the conditional expectations of continuous variables (as in regression), or at estimating distributions. There are many situations where however we are interested in input-output relationships, as in regression, but the output variable is discrete rather than continuous. In particular there are many situations where we have binary outcomes (it snows in Pittsburgh on a given day, or it doesn't; this squirrel carries plague, or it doesn't; this loan will be paid back, or it won't; this person will get heart disease in the next five years, or they won't). In addition to the binary outcome, we have some input variables, which may or may not be continuous. How could we model and analyze such data?

We could try to come up with a rule which guesses the binary output from the input variables. This is called **classification**, and is an important topic in statistics and machine learning. However, simply guessing “yes” or “no” is pretty crude — especially if there is no perfect rule. (Why should there be a perfect rule?) Something which takes noise into account, and doesn't just give a binary answer, will often be useful. In short, we want probabilities — which means we need to fit a stochastic model.

What would be nice, in fact, would be to have conditional distribution of the response Y , given the input variables, $\Pr(Y|X)$. This would tell us about how precise our predictions are. If our model says that there's a 51% chance of snow and it doesn't snow, that's better than if it had said there was a 99% chance of snow (though even a 99% chance is not a sure thing). We will see, in Chapter 15, general approaches to estimating conditional probabilities non-parametrically, which can use the kernels for discrete variables from Chapter 4. While there are a lot of merits to this approach, it does involve coming up with a model for the joint distribution of outputs Y and inputs X , which can be quite time-consuming.

Let's pick one of the classes and call it “1” and the other “0”. (It doesn't matter which is which. Then Y becomes an **indicator variable**, and you can convince yourself that $\Pr(Y=1) = \mathbb{E}[Y]$. Similarly, $\Pr(Y=1|X=x) = \mathbb{E}[Y|X=x]$. (In

a phrase, “conditional probability is the conditional expectation of the indicator”.) This helps us because by this point we know all about estimating conditional expectations. The most straightforward thing for us to do at this point would be to pick out our favorite smoother and estimate the regression function for the indicator variable; this will be an estimate of the conditional probability function.

There are two reasons not to just plunge ahead with that idea. One is that probabilities must be between 0 and 1, but our smoothers will not necessarily respect that, even if all the observed y_i they get are either 0 or 1. The other is that we might be better off making more use of the fact that we are trying to estimate probabilities, by more explicitly modeling the probability.

Assume that $\Pr(Y = 1|X = x) = p(x; \theta)$, for some function p parameterized by θ . parameterized function θ , and further assume that observations are independent of each other. The the (conditional) likelihood function is

$$\prod_{i=1}^n \Pr(Y = y_i|X = x_i) = \prod_{i=1}^n p(x_i; \theta)^{y_i} (1 - p(x_i; \theta))^{1-y_i} \quad (12.1)$$

Recall that in a sequence of Bernoulli trials y_1, \dots, y_n , where there is a constant probability of success p , the likelihood is

$$\prod_{i=1}^n p^{y_i} (1 - p)^{1-y_i} \quad (12.2)$$

As you learned in basic statistics, this likelihood is maximized when $p = \hat{p} = n^{-1} \sum_{i=1}^n y_i$. If each trial had its own success probability p_i , this likelihood becomes

$$\prod_{i=1}^n p_i^{y_i} (1 - p_i)^{1-y_i} \quad (12.3)$$

Without some constraints, estimating the “inhomogeneous Bernoulli” model by maximum likelihood doesn’t work; we’d get $\hat{p}_i = 1$ when $y_i = 1$, $\hat{p}_i = 0$ when $y_i = 0$, and learn nothing. If on the other hand we assume that the p_i aren’t just arbitrary numbers but are linked together, if we *model* the probabilities, those constraints give non-trivial parameter estimates, and let us generalize. In the kind of model we are talking about, the constraint, $p_i = p(x_i; \theta)$, tells us that p_i must be the same whenever x_i is the same, and if p is a continuous function, then similar values of x_i must lead to similar values of p_i . Assuming p is known (up to parameters), the likelihood is a function of θ , and we can estimate θ by maximizing the likelihood. This chapter will be about this approach.

12.2 Logistic Regression

To sum up: we have a binary output variable Y , and we want to model the conditional probability $\Pr(Y = 1|X = x)$ as a function of x ; any unknown parameters in the function are to be estimated by maximum likelihood. By now, it will not surprise you to learn that statisticians have approach this problem by asking themselves “how can we use linear regression to solve this?”

1. The most obvious idea is to let $p(x)$ be a linear function of x . Every increment of a component of x would add or subtract so much to the probability. The conceptual problem here is that p must be between 0 and 1, and linear functions are unbounded. Moreover, in many situations we empirically see “diminishing returns” — changing p by the same amount requires a bigger change in x when p is already large (or small) than when p is close to 1/2. Linear models can’t do this.
2. The next most obvious idea is to let $\log p(x)$ be a linear function of x , so that changing an input variable *multiplies* the probability by a fixed amount. The problem is that logarithms are unbounded in only one direction, and linear functions are not.
3. Finally, the easiest modification of $\log p$ which has an unbounded range is the **logistic (or logit) transformation**, $\log \frac{p}{1-p}$. We can make *this* a linear function of x without fear of nonsensical results. (Of course the results could still happen to be *wrong*, but they’re not *guaranteed* to be wrong.)

This last alternative is **logistic regression**.

Formally, the model logistic regression model is that

$$\log \frac{p(x)}{1 - p(x)} = \beta_0 + x \cdot \beta \quad (12.4)$$

Solving for p , this gives

$$p(x; b, w) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_0 + x \cdot \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \cdot \beta)}} \quad (12.5)$$

Notice that the over-all specification is a lot easier to grasp in terms of the transformed probability than in terms of the untransformed probability.¹

To minimize the mis-classification rate, we should predict $Y = 1$ when $p \geq 0.5$ and $Y = 0$ when $p < 0.5$. This means guessing 1 whenever $\beta_0 + x \cdot \beta$ is non-negative, and 0 otherwise. So logistic regression gives us a **linear classifier**. The **decision boundary** separating the two predicted classes is the solution of $\beta_0 + x \cdot \beta = 0$, which is a point if x is one dimensional, a line if it is two dimensional, etc. One can show (exercise!) that the distance from the decision boundary is $\beta_0/\|\beta\| + x \cdot \beta/\|\beta\|$. Logistic regression not only says where the boundary between the classes is, but also says (via Eq. 12.5) that the class probabilities depend on distance from the boundary, in a particular way, and that they go towards the extremes (0 and 1) more rapidly when $\|\beta\|$ is larger. It’s these statements about probabilities which make logistic regression more than just a classifier. It makes stronger, more detailed predictions, and can be fit in a different way; but those strong predictions could be wrong.

Using logistic regression to predict class probabilities is a *modeling choice*, just like it’s a modeling choice to predict quantitative variables with linear regression.

¹Unless you’ve taken statistical mechanics, in which case you recognize that this is the Boltzmann distribution for a system with two states, which differ in energy by $\beta_0 + x \cdot \beta$.

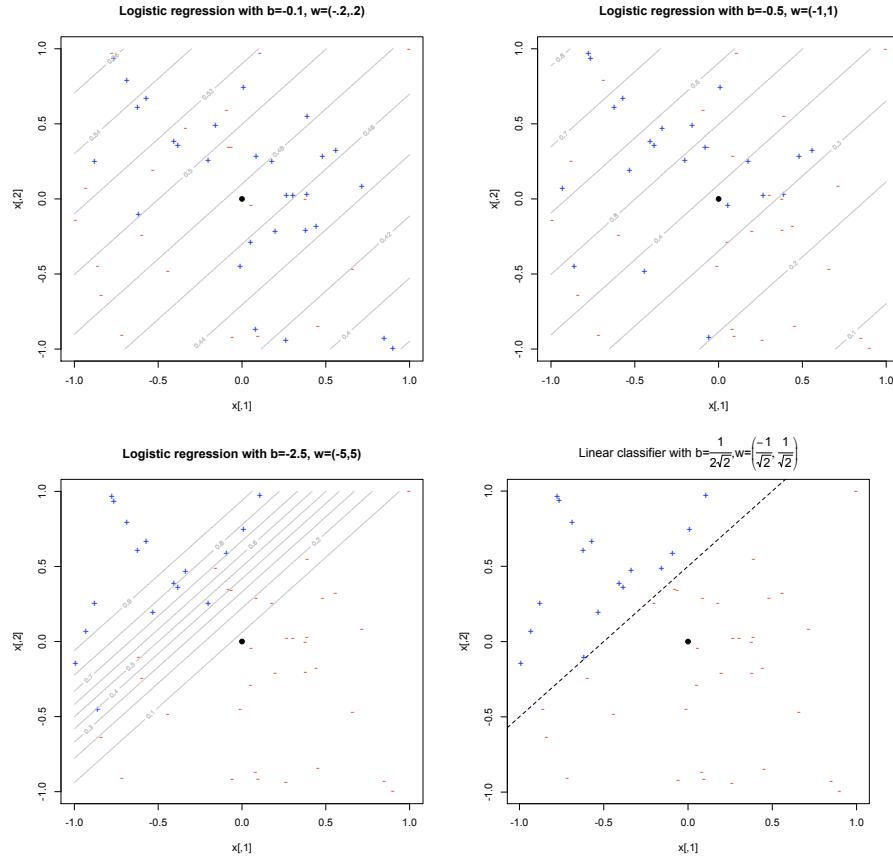


Figure 12.1: Effects of scaling logistic regression parameters. Values of x_1 and x_2 are the same in all plots ($\sim \text{Unif}(-1, 1)$ for both coordinates), but labels were generated randomly from logistic regressions with $\beta_0 = -0.1$, $\beta = (-0.2, 0.2)$ (top left); from $\beta_0 = -0.5$, $\beta = (-1, 1)$ (top right); from $\beta_0 = -2.5$, $\beta = (-5, 5)$ (bottom left); and from a perfect linear classifier with the same boundary. The large black dot is the origin.

In neither case is the appropriateness of the model guaranteed by the gods, nature, mathematical necessity, etc. We begin by positing the model, to get something to work with, and we end (if we know what we're doing) by checking whether it really does match the data, or whether it has systematic flaws.

Logistic regression is one of the most commonly used tools for applied statistics and discrete data analysis. There are basically four reasons for this.

1. Tradition.
2. In addition to the heuristic approach above, the quantity $\log p/(1 - p)$ plays an important role in the analysis of contingency tables (the “log odds”). Classification is a bit like having a contingency table with two columns (classes) and infinitely many rows (values of x). With a finite contingency table, we can estimate the log-odds for each row empirically, by just taking counts in the table. With infinitely many rows, we need some sort of interpolation scheme; logistic regression is linear interpolation for the log-odds.
3. It's closely related to “exponential family” distributions, where the probability of some vector v is proportional to $\exp \beta_0 + \sum_{j=1}^m f_j(v)\beta_j$. If one of the components of v is binary, and the functions f_j are all the identity function, then we get a logistic regression. Exponential families arise in many contexts in statistical theory (and in physics!), so there are lots of problems which can be turned into logistic regression.
4. It often works surprisingly well as a classifier. But, many simple techniques often work surprisingly well as classifiers, and this doesn't really testify to logistic regression getting the probabilities right.

12.2.1 Likelihood Function for Logistic Regression

Because logistic regression predicts probabilities, rather than just classes, we can fit it using likelihood. For each training data-point, we have a vector of features, x_i , and an observed class, y_i . The probability of that class was either p , if $y_i = 1$, or $1 - p$, if $y_i = 0$. The likelihood is then

$$L(\beta_0, \beta) = \prod_{i=1}^n p(x_i)^{y_i} (1 - p(x_i))^{1-y_i} \quad (12.6)$$

(I could substitute in the actual equation for p , but things will be clearer in a moment if I don't.) The log-likelihood turns products into sums:

$$\ell(\beta_0, \beta) = \sum_{i=1}^n y_i \log p(x_i) + (1 - y_i) \log 1 - p(x_i) \quad (12.7)$$

$$= \sum_{i=1}^n \log(1 - p(x_i)) + \sum_{i=1}^n y_i \log \frac{p(x_i)}{1 - p(x_i)} \quad (12.8)$$

$$= \sum_{i=1}^n \log(1 - p(x_i)) + \sum_{i=1}^n y_i (\beta_0 + x_i \cdot \beta) \quad (12.9)$$

$$= \sum_{i=1}^n -\log(1 + e^{\beta_0 + x_i \cdot \beta}) + \sum_{i=1}^n y_i (\beta_0 + x_i \cdot \beta) \quad (12.10)$$

where in the next-to-last step we finally use equation 12.4.

Typically, to find the maximum likelihood estimates we'd differentiate the log likelihood with respect to the parameters, set the derivatives equal to zero, and solve. To start that, take the derivative with respect to one component of β , say β_j .

$$\frac{\partial \ell}{\partial \beta_j} = -\sum_{i=1}^n \frac{1}{1 + e^{\beta_0 + x_i \cdot \beta}} e^{\beta_0 + x_i \cdot \beta} x_{ij} + \sum_{i=1}^n y_i x_{ij} \quad (12.11)$$

$$= \sum_{i=1}^n (y_i - p(x_i; \beta_0, \beta)) x_{ij} \quad (12.12)$$

We are not going to be able to set this to zero and solve exactly. (That's a transcendental equation, and there is no closed-form solution.) We can however approximately solve it numerically.

12.2.2 Logistic Regression with More Than Two Classes

If Y can take on more than two values, say k of them, we can still use logistic regression. Instead of having one set of parameters β_0, β , each class c in $0 : (k-1)$ will have its own offset $\beta_0^{(c)}$ and vector $\beta^{(c)}$, and the predicted conditional probabilities will be

$$\Pr(Y = c | \vec{X} = x) = \frac{e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_c e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}} \quad (12.13)$$

You can check that when there are only two classes (say, 0 and 1), equation 12.13 reduces to equation 12.5, with $\beta_0 = \beta_0^{(1)} - \beta_0^{(0)}$ and $\beta = \beta^{(1)} - \beta^{(0)}$. In fact, no matter how many classes there are, we can always pick one of them, say $c = 0$, and fix its parameters at exactly zero, without any loss of generality².

²Since we can arbitrarily chose which class's parameters to "zero out" without affecting the predicted probabilities, strictly speaking the model in Eq. 12.13 is **unidentified**. That is, different parameter settings lead to *exactly* the same outcome, so we can't use the data to tell which one is right. The usual response here is to deal with this by a convention: we *decide* to zero out the parameters of the first class, and then estimate the contrasting parameters for the others.

Calculation of the likelihood now proceeds as before (only with more book-keeping), and so does maximum likelihood estimation.

12.3 Newton's Method for Numerical Optimization

There are a huge number of methods for numerical optimization; we can't cover all bases, and there is no magical method which will always work better than anything else. However, there are some methods which work very well on an awful lot of the problems which keep coming up, and it's worth spending a moment to sketch how they work. One of the most ancient yet important of them is Newton's method (alias "Newton-Raphson").

Let's start with the simplest case of minimizing a function of one scalar variable, say $f(\beta)$. We want to find the location of the global minimum, β^* . We suppose that f is smooth, and that β^* is a regular interior minimum, meaning that the derivative at β^* is zero and the second derivative is positive. Near the minimum we could make a Taylor expansion:

$$f(\beta) \approx f(\beta^*) + \frac{1}{2}(\beta - \beta^*)^2 \left. \frac{d^2 f}{d\beta^2} \right|_{\beta=\beta^*} \quad (12.14)$$

(We can see here that the second derivative has to be positive to ensure that $f(\beta) > f(\beta^*)$.) In words, $f(\beta)$ is close to quadratic near the minimum.

Newton's method uses this fact, and minimizes a quadratic *approximation* to the function we are really interested in. (In other words, Newton's method is to replace the problem we want to solve, with a problem which we *can* solve.) Guess an initial point $\beta^{(0)}$. If this is close to the minimum, we can take a second order Taylor expansion around $\beta^{(0)}$ and it will still be accurate:

$$f(\beta) \approx f(\beta^{(0)}) + (\beta - \beta^{(0)}) \left. \frac{df}{d\beta} \right|_{\beta=\beta^{(0)}} + \frac{1}{2}(\beta - \beta^{(0)})^2 \left. \frac{d^2 f}{d\beta^2} \right|_{\beta=\beta^{(0)}} \quad (12.15)$$

Now it's easy to minimize the right-hand side of equation 12.15. Let's abbreviate the derivatives, because they get tiresome to keep writing out: $\left. \frac{df}{d\beta} \right|_{\beta=\beta^{(0)}} = f'(\beta^{(0)})$, $\left. \frac{d^2 f}{d\beta^2} \right|_{\beta=\beta^{(0)}} = f''(\beta^{(0)})$. We just take the derivative with respect to β , and set it equal to zero at a point we'll call $\beta^{(1)}$:

$$0 = f'(\beta^{(0)}) + \frac{1}{2}f''(\beta^{(0)})2(\beta^{(1)} - \beta^{(0)}) \quad (12.16)$$

$$\beta^{(1)} = \beta^{(0)} - \frac{f'(\beta^{(0)})}{f''(\beta^{(0)})} \quad (12.17)$$

The value $\beta^{(1)}$ should be a better guess at the minimum β^* than the initial one $\beta^{(0)}$ was. So if we use $\beta^{(1)}$ to make a quadratic approximation to f , we'll get a better approximation, and so we can *iterate* this procedure, minimizing one approximation

and then using that to get a new approximation:

$$\beta^{(n+1)} = \beta^{(n)} - \frac{f'(\beta^{(n)})}{f''(\beta^{(n)})} \quad (12.18)$$

Notice that the true minimum β^* is a **fixed point** of equation 12.18: if we happen to land on it, we'll stay there (since $f'(\beta^*)=0$). We won't show it, but it can be proved that if $\beta^{(0)}$ is close enough to β^* , then $\beta^{(n)} \rightarrow \beta^*$, and that in general $|\beta^{(n)} - \beta^*| = O(n^{-2})$, a very rapid rate of convergence. (Doubling the number of iterations we use doesn't reduce the error by a factor of two, but by a factor of four.)

Let's put this together in an algorithm.

```
my.newton = function(f,f.prime,f.prime2,beta0,tolerance=1e-3,max.iter=50) {
  beta = beta0
  old.f = f(beta)
  iterations = 0
  made.changes = TRUE
  while(made.changes & (iterations < max.iter)) {
    iterations <- iterations +1
    made.changes <- FALSE
    new.beta = beta - f.prime(beta)/f.prime2(beta)
    new.f = f(new.beta)
    relative.change = abs(new.f - old.f)/old.f -1
    made.changes = (relative.changes > tolerance)
    beta = new.beta
    old.f = new.f
  }
  if (made.changes) {
    warning("Newton's method terminated before convergence")
  }
  return(list(minimum=beta,value=f(beta),deriv=f.prime(beta),
              deriv2=f.prime2(beta),iterations=iterations,
              converged=!made.changes))
}
```

The first three arguments here have to all be *functions*. The fourth argument is our initial guess for the minimum, $\beta^{(0)}$. The last arguments keep Newton's method from cycling forever: `tolerance` tells it to stop when the function stops changing very much (the relative difference between $f(\beta^{(n)})$ and $f(\beta^{(n+1)})$ is small), and `max.iter` tells it to never do more than a certain number of steps no matter what. The return value includes the estimated minimum, the value of the function there, and some diagnostics — the derivative should be very small, the second derivative should be positive, etc.

You may have noticed some potential problems — what if we land on a point where f'' is zero? What if $f(\beta^{(n+1)}) > f(\beta^{(n)})$? Etc. There are ways of handling these issues, and more, which are incorporated into real optimization algorithms from numerical analysis — such as the `optim` function in R; I strongly recommend

you use that, or something like that, rather than trying to roll your own optimization code.³

12.3.1 Newton's Method in More than One Dimension

Suppose that the objective f is a function of multiple arguments, $f(\beta_1, \beta_2, \dots, \beta_p)$. Let's bundle the parameters into a single vector, w . Then the Newton update is

$$\beta^{(n+1)} = \beta^{(n)} - \mathbf{H}^{-1}(\beta^{(n)}) \nabla f(\beta^{(n)}) \quad (12.19)$$

where ∇f is the **gradient** of f , its vector of partial derivatives $[\partial f / \partial \beta_1, \partial f / \partial \beta_2, \dots, \partial f / \partial \beta_p]$, and \mathbf{H} is the **Hessian** of f , its matrix of second partial derivatives, $H_{ij} = \partial^2 f / \partial \beta_i \partial \beta_j$.

Calculating \mathbf{H} and ∇f isn't usually very time-consuming, but taking the inverse of \mathbf{H} is, unless it happens to be a diagonal matrix. This leads to various **quasi-Newton** methods, which either approximate H by a diagonal matrix, or take a proper inverse of H only rarely (maybe just once), and then try to update an estimate of $\mathbf{H}^{-1}(\beta^{(n)})$ as $\beta^{(n)}$ changes.

12.3.2 Iteratively Re-Weighted Least Squares

This discussion of Newton's method is quite general, and therefore abstract. In the particular case of logistic regression, we can make everything look much more "statistical".

Logistic regression, after all, is a linear model for a transformation of the probability. Let's call this transformation g :

$$g(p) \equiv \log \frac{p}{1-p} \quad (12.20)$$

So the model is

$$g(p) = \beta_0 + x \cdot \beta \quad (12.21)$$

and $Y|X=x \sim \text{Binom}(1, g^{-1}(\beta_0 + x \cdot \beta))$. It seems that what we should want to do is take $g(y)$ and regress it linearly on x . Of course, the variance of Y , according to the model, is going to chance depending on x — it will be $(g^{-1}(\beta_0 + x \cdot \beta))(1 - g^{-1}(\beta_0 + x \cdot \beta))$ — so we really ought to do a weighted linear regression, with weights inversely proportional to that variance. Since writing $g^{-1}(\beta_0 + x \cdot \beta)$ is getting annoying, let's abbreviate it by μ (for "mean"), and let's abbreviate that variance as $V(\mu)$.

The problem is that y is either 0 or 1, so $g(y)$ is either $-\infty$ or $+\infty$. We will evade this by using Taylor expansion.

$$g(y) \approx g(\mu) + (y - \mu)g'(\mu) \equiv z \quad (12.22)$$

The right hand side, z will be our *effective* response variable, which we will regress on x . To see why this should give us the right coefficients, substitute for $g(\mu)$ in the

³`optim` actually is a wrapper for several different optimization methods; `method=BFGS` selects a Newtonian method; BFGS is an acronym for the names of the algorithm's inventors.

definition of z ,

$$z = \beta_0 + x \cdot \beta + (\gamma - \mu)g'(\mu) \quad (12.23)$$

and notice that, if we've got the coefficients right, $E[Y|X=x] = \mu$, so $(\gamma - \mu)$ should be mean-zero noise. In other words, when we have the right coefficients, z is a linear function of x plus mean-zero noise. That noise doesn't necessarily have constant variance, but we can work it out by propagation of error, getting $(g'(\mu))^2 V(\mu)$, and so use that in weighted least squares to recover β .

Notice that both the weights *and* z depend on the parameters of our logistic regression, through μ . So having done this once, we should really use the new parameters to update z and the weights, and do it again. Eventually, we come to a fixed point, where the parameter estimates no longer change.

The treatment above is rather heuristic⁴, but it turns out to be equivalent to using Newton's method, only with the expected second derivative of the log likelihood, instead of its actual value.⁵ Since, with a large number of observations, the observed second derivative should be close to the expected second derivative, this is only a small approximation.

12.4 Generalized Linear Models and Generalized Additive Models

Logistic regression is part of a broader family of **generalized linear models** (GLMs), where the conditional distribution of the response falls in some parametric family, and the parameters are set by the linear predictor. Ordinary, least-squares regression is the case where response is Gaussian, with mean equal to the linear predictor, and constant variance. Logistic regression is the case where the response is binomial, with n equal to the number of data-points with the given x (usually but not always 1), and p is given by Equation 12.5. Changing the relationship between the parameters and the linear predictor is called changing the **link function**. For computational reasons, the link function is actually the function you apply to the mean response to get back the linear predictor, rather than the other way around — (12.4) rather than (12.5). There are thus other forms of binomial regression besides logistic regression.⁶ There is also Poisson regression (appropriate when the data are counts without any upper limit), gamma regression, etc.; we will say more about these in Chapter 13.

In R, any standard GLM can be fit using the (base) `glm` function, whose syntax is very similar to that of `lm`. The major wrinkle is that, of course, you need

⁴That is, mathematically incorrect.

⁵This takes a reasonable amount of algebra to show, so we'll skip it. The key point however is the following. Take a single Bernoulli observation with success probability p . The log-likelihood is $Y \log p + (1-Y) \log 1-p$. The first derivative with respect to p is $Y/p - (1-Y)/(1-p)$, and the second derivative is $-Y/p^2 - (1-Y)/(1-p)^2$. Taking expectations of the second derivative gives $-1/p - 1/(1-p) = -1/p(1-p)$. In other words, $V(p) = -1/E[\ell'']$. Using weights inversely proportional to the variance thus turns out to be equivalent to dividing by the expected second derivative.

⁶My experience is that these tend to give similar error rates as classifiers, but have rather different guesses about the underlying probabilities.

to specify the family of probability distributions to use, by the `family` option — `family=binomial` defaults to logistic regression. (See `help(glm)` for the gory details on how to do, say, probit regression.) All of these are fit by the same sort of numerical likelihood maximization.

One caution about using maximum likelihood to fit logistic regression is that it can seem to work badly when the training data *can* be linearly separated. The reason is that, to make the likelihood large, $p(x_i)$ should be large when $y_i = 1$, and p should be small when $y_i = 0$. If β_0, β_1 is a set of parameters which perfectly classifies the training data, then $c\beta_0, c\beta_1$ is too, for any $c > 1$, but in a logistic regression the second set of parameters will have more extreme probabilities, and so a higher likelihood. For linearly separable data, then, there is no parameter vector which *maximizes* the likelihood, since ℓ can always be increased by making the vector larger but keeping it pointed in the same direction.

You should, of course, be so lucky as to have this problem.

12.4.1 Generalized Additive Models

A natural step beyond generalized linear models is **generalized additive models** (GAMs), where instead of making the transformed mean response a *linear* function of the inputs, we make it an *additive* function of the inputs. This means combining a function for fitting additive models with likelihood maximization. This is actually done in R with the same `gam` function we used for additive models (hence the name). We will look at how this works in some detail in Chapter 13.

GAMs can be used to check GLMs in much the same way that smoothers can be used to check parametric regressions: fit a GAM and a GLM to the same data, then simulate from the GLM, and re-fit both models to the simulated data. Repeated many times, this gives a distribution for how much better the GAM will seem to fit than the GLM does, *even when the GLM is true*. You can then read a p -value off of this distribution.

12.4.2 An Example (Including Model Checking)

Here's a worked R example, using the data from the upper right panel of Figure 12.1. The 50×2 matrix `x` holds the input variables (the coordinates are independently and uniformly distributed on $[-1, 1]$), and `y.1` the corresponding class labels, themselves generated from a logistic regression with $\beta_0 = -0.5$, $\beta = (-1, 1)$.

```
> logr = glm(y.1 ~ x[,1] + x[,2], family=binomial)
> logr

Call: glm(formula = y.1 ~ x[, 1] + x[, 2], family = binomial)

Coefficients:
(Intercept)      x[, 1]      x[, 2]
-0.410        -1.050       1.366
```

```
Degrees of Freedom: 49 Total (i.e. Null); 47 Residual
Null Deviance: 68.59
Residual Deviance: 58.81 AIC: 64.81
> sum(ifelse(logr$fitted.values<0.5,0,1) != y.1)/length(y.1)
[1] 0.32
```

The **deviance** of a model fitted by maximum likelihood is (twice) the difference between its log likelihood and the maximum log likelihood for a **saturated** model, i.e., a model with one parameter per observation. Hopefully, the saturated model can give a perfect fit.⁷ Here the saturated model would assign probability 1 to the observed outcomes⁸, and the logarithm of 1 is zero, so $D = 2\ell(\hat{\beta}_0, \hat{\beta})$. The null deviance is what's achievable by using just a constant bias b and setting $w = 0$. The fitted model definitely improves on that.⁹

The fitted values of the logistic regression are the class probabilities; this shows that the error rate of the logistic regression, if you force it to predict actual classes, is 32%. This sounds bad, but notice from the contour lines in the figure that lots of the probabilities are near 0.5, meaning that the classes are just genuinely hard to predict.

To see how well the logistic regression assumption holds up, let's compare this to a GAM.¹⁰

```
> library(gam)
> gam.1 = gam(y.1 ~ lo(x[,1]) + lo(x[,2]), family="binomial")
> gam.1
Call:
gam(formula = y.1 ~ lo(x[, 1]) + lo(x[, 2]), family = "binomial")
```

```
Degrees of Freedom: 49 total; 41.39957 Residual
Residual Deviance: 49.17522
```

This fits a GAM to the same data, using lowess smoothing of both input variables. Notice that the residual deviance is lower. That is, the GAM fits this data better. We expect this; the question is whether the difference is significant, or within the range of what we should expect when logistic regression is valid. To test this, we need to simulate from the logistic regression model.

```
simulate.from.logr = function(x, coefs) {
  require(faraway) # For accessible logit and inverse-logit functions
```

⁷The factor of two is so that the deviance will have a χ^2 distribution. Specifically, if the model with p parameters is right, the deviance will have a χ^2 distribution with $n - p$ degrees of freedom.

⁸This is not possible when there are multiple observations with the same input features, but different classes.

⁹AIC is of course the Akaike information criterion, $-2\ell + 2q$, with q being the number of parameters (here, $q = 3$). AIC has some truly devoted adherents, especially among non-statisticians, but I have been deliberately ignoring it and will continue to do so. Basically, to the extent AIC succeeds, it works as fast, large-sample approximation to doing leave-one-out cross-validation. Claeskens and Hjort (2008) is a thorough, modern treatment of AIC and related model-selection criteria from a statistical viewpoint.

¹⁰Previous examples of using GAMs have mostly used the mgcv package and spline smoothing. There is no particular reason to switch to the gam library and lowess smoothing here, but there's also no real reason not to.

12.4. GENERALIZED LINEAR MODELS AND GENERALIZED ADDITIVE MODELS

243

```
n = nrow(x)
linear.part = coefs[1] + x %*% coefs[-1]
probs = ilogit(linear.part) # Inverse logit
y = rbinom(n, size=1, prob=probs)
return(y)
}
```

Now we simulate from our fitted model, and re-fit both the logistic regression and the GAM.

```
delta.deviance.sim = function (x,logistic.model) {
  y.new = simulate.from.logr(x,logistic.model$coefficients)
  GLM.dev = glm(y.new ~ x[,1] + x[,2], family="binomial")$deviance
  GAM.dev = gam(y.new ~ lo(x[,1]) + lo(x[,2])), family="binomial")$deviance
  return(GLM.dev - GAM.dev)
}
```

Notice that in this simulation we are not generating new \vec{X} values. The logistic regression and the GAM are both models for the response *conditional* on the inputs, and are agnostic about how the inputs are distributed, or even whether it's meaningful to talk about their distribution.

Finally, we repeat the simulation a bunch of times, and see where the observed difference in deviances falls in the sampling distribution.

```
> delta.dev = replicate(1000,delta.deviance.sim(x,logr))
> delta.dev.observed = logr$deviance - gam.1$deviance # 9.64
> sum(delta.dev.observed > delta.dev)/1000
[1] 0.685
```

In other words, the amount by which a GAM fits the data better than logistic regression is pretty near the middle of the null distribution. Since the example data really *did* come from a logistic regression, this is a relief.

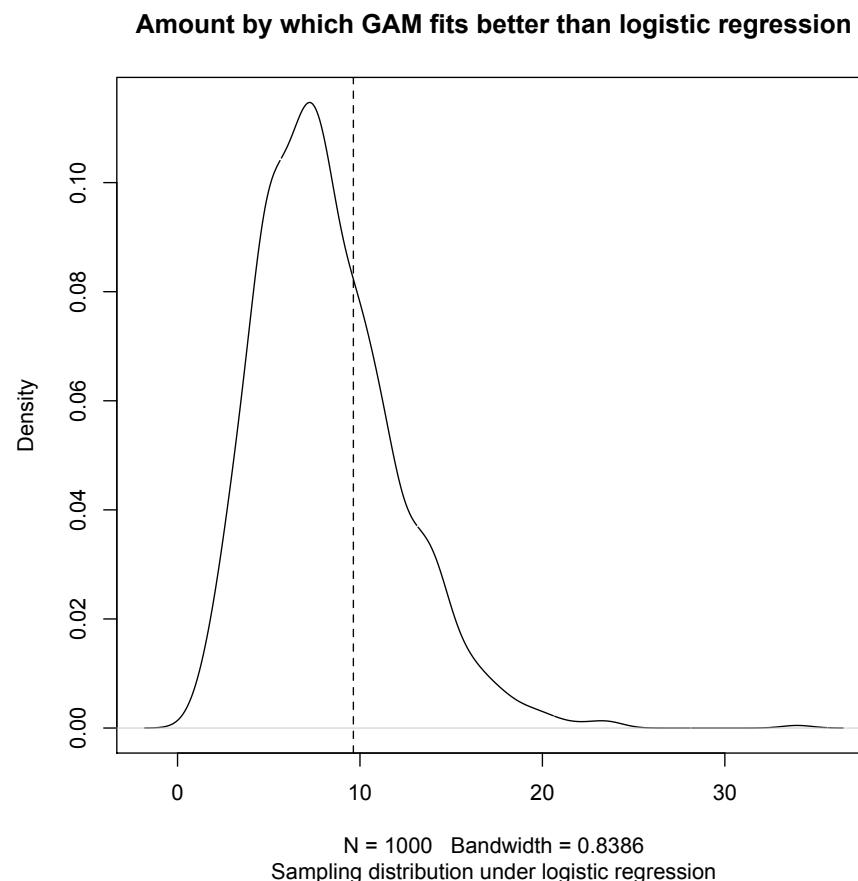


Figure 12.2: Sampling distribution for the difference in deviance between a GAM and a logistic regression, on data generated from a logistic regression. The observed difference in deviances is shown by the dashed horizontal line.

12.5 Exercises

To think through, not to hand in.

1. A multiclass logistic regression, as in Eq. 12.13, has parameters $\beta_0^{(c)}$ and $\beta^{(c)}$ for each class c . Show that we can always get the same predicted probabilities by setting $\beta_0^{(c)} = 0$, $\beta^{(c)} = 0$ for any one class c , and adjusting the parameters for the other classes appropriately.
2. Find the first and second derivatives of the log-likelihood for logistic regression with one predictor variable. Explicitly write out the formula for doing one step of Newton's method. Explain how this relates to re-weighted least squares.

Chapter 13

Generalized Linear Models and Generalized Additive Models

13.1 Generalized Linear Models and Iterative Least Squares

Logistic regression is a particular instance of a broader kind of model, called a **generalized linear model** (GLM). You are familiar, of course, from your regression class with the idea of transforming the response variable, what we've been calling Y , and then predicting the transformed variable from X . This was *not* what we did in logistic regression. Rather, we transformed the conditional expected value, and made that a linear function of X . This seems odd, because it *is* odd, but it turns out to be useful.

Let's be specific. Our usual focus in regression modeling has been the conditional expectation function, $r(x) = E[Y|X=x]$. In plain linear regression, we try to approximate $r(x)$ by $\beta_0 + x \cdot \beta$. In logistic regression, $r(x) = E[Y|X=x] = \Pr(Y=1|X=x)$, and it is a transformation of $r(x)$ which is linear. The usual notation says

$$\eta(x) = \beta_0 + x \cdot \beta \quad (13.1)$$

$$\eta(x) = \log \frac{r(x)}{1 - r(x)} \quad (13.2)$$

$$= g(r(x)) \quad (13.3)$$

defining the logistic **link function** by $g(m) = \log m / (1 - m)$. The function $\eta(x)$ is called the **linear predictor**.

Now, the first impulse for estimating this model would be to apply the transformation g to the response. But Y is always zero or one, so $g(Y) = \pm\infty$, and regression will not be helpful here. The standard strategy is instead to use (what else?) Taylor expansion. Specifically, we try expanding $g(Y)$ around $r(x)$, and stop at first order:

$$g(Y) \approx g(r(x)) + (Y - r(x))g'(r(x)) \quad (13.4)$$

$$= \eta(x) + (Y - r(x))g'(r(x)) \equiv z \quad (13.5)$$

13.1. GENERALIZED LINEAR MODELS AND ITERATIVE LEAST SQUARES

We *define* this to be our effective response after transformation. Notice that if there were no noise, so that y was always equal to its conditional mean $r(x)$, then regressing z on x would give us back the coefficients β_0, β . What this suggests is that we can *estimate* those parameters by regressing z on x .

The term $Y - r(x)$ has expectation zero, so it acts like the noise, with the factor of g' telling us about how the noise is scaled by the transformation. This lets us work out the variance of z :

$$\text{Var}[Z|X=x] = \text{Var}[\eta(x)|X=x] + \text{Var}[(Y - r(x))g'(r(x))|X=x] \quad (13.6)$$

$$= 0 + (g'(r(x)))^2 \text{Var}[Y|X=x] \quad (13.7)$$

For logistic regression, with Y binary, $\text{Var}[Y|X=x] = r(x)(1-r(x))$. On the other hand, with the logistic link function, $g'(r(x)) = \frac{1}{r(x)(1-r(x))}$. Thus, for logistic regression, $\text{Var}[Z|X=x] = [r(x)(1-r(x))]^{-1}$.

Because the variance of Z changes with X , this is a heteroskedastic regression problem. As we saw in chapter 7, the appropriate way of dealing with such a problem is to use weighted least squares, with weights inversely proportional to the variances. This means that the weight at x should be proportional to $r(x)(1-r(x))$. Notice two things about this. First, the weights depend on the current guess about the parameters. Second, we give little weight to cases where $r(x) \approx 0$ or where $r(x) \approx 1$, and the most weight when $r(x) = 0.5$. This focuses our attention on places where we have a lot of potential information — the distinction between a probability of 0.499 and 0.501 is just a lot easier to discern than that between 0.000 and 0.002!

We can now put all this together into an estimation strategy for logistic regression.

1. Get the data $(x_1, y_1), \dots, (x_n, y_n)$, and some initial guesses β_0, β .
2. until β_0, β converge
 - (a) Calculate $\eta(x_i) = \beta_0 + x_i \cdot \beta$ and the corresponding $r(x_i)$
 - (b) Find the effective transformed responses $z_i = \eta(x_i) + \frac{y_i - r(x_i)}{r(x_i)(1-r(x_i))}$
 - (c) Calculate the weights $w_i = r(x_i)(1-r(x_i))$
 - (d) Do a weighted linear regression of z_i on x_i with weights w_i , and set β_0, β to the intercept and slopes of this regression

Our initial guess about the parameters tells us about the heteroskedasticity, which we use to improve our guess about the parameters, which we use to improve our guess about the variance, and so on, until the parameters stabilize. This is called **iterative reweighted least squares** (or “iterative weighted least squares”, “iteratively weighted least squares”, “iteratively reweighted least squares”, etc.), abbreviated IRLS, IRWLS, IWLS, etc. As mentioned in the last chapter, this turns out to be *almost* equivalent to Newton’s method, at least for this problem.

13.1.1 GLMs in General

The set-up for an arbitrary GLM is a generalization of that for logistic regression. We need

- A **linear predictor**, $\eta(x) = \beta_0 + x c \dot{\beta}$
- A **link function** g , so that $\eta(x) = g(r(x))$. For logistic regression, we had $g(r) = \log r / (1 - r)$.
- A **dispersion scale function** V , so that $\text{Var}[Y|X=x] = \sigma^2 V(r(x))$. For logistic regression, we had $V(r) = r(1 - r)$, and $\sigma^2 = 1$.

With these, we know the conditional mean and conditional variance of the response for each value of the input variables x .

As for estimation, basically everything in the IRWLS set up carries over unchanged. In fact, we can go through this algorithm:

1. Get the data $(x_1, y_1), \dots, (x_n, y_n)$, fix link function $g(r)$ and dispersion scale function $V(r)$, and make some initial guesses β_0, β .
2. Until β_0, β converge
 - (a) Calculate $\eta(x_i) = \beta_0 + x_i \cdot \beta$ and the corresponding $r(x_i)$
 - (b) Find the effective transformed responses $z_i = \eta(x_i) + (y_i - r(x_i))g'(r(x_i))$
 - (c) Calculate the weights $w_i = [(g'(r(x_i))^2 V(r(x_i))]^{-1}$
 - (d) Do a weighted linear regression of z_i on x_i with weights w_i , and set β_0, β to the intercept and slopes of this regression

Notice that even if we don't know the over-all variance scale σ^2 , that's OK, because the weights just have to be *proportional* to the inverse variance.

13.1.2 Examples of GLMs

13.1.2.1 Vanilla Linear Models

To re-assure ourselves that we are not doing anything crazy, let's see what happens when $g(r) = r$ (the "identity link"), and $\text{Var}[Y|X=x] = \sigma^2$, so that $V(r) = 1$. Then $g' = 1$, all weights $w_i = 1$, and the effective transformed response $z_i = y_i$. So we just end up regressing y_i on x_i with no weighting at all — we do ordinary least squares. Since neither the weights nor the transformed response will change, IRWLS will converge exactly after one step. So if we get rid of all this nonlinearity and heteroskedasticity and go all the way back to our very first days of doing regression, we get the OLS answers we know and love.

13.1.2.2 Binomial Regression

In many situations, our response variable y_i will be an integer count running between 0 and some pre-determined upper limit n_i . (Think: number of patients in a hospital ward with some condition, number of children in a classroom passing a test, number of widgets produced by a factory which are defective, number of people in a village with some genetic mutation.) One way to model this would be as a binomial random variable, with n_i trials, and a success probability p_i which was a logistic function of predictors x . The logistic regression we have done so far is the special case where $n_i = 1$ always. I will leave it as an EXERCISE (1) for you to work out the link function and the weights for general binomial regression, where the n_i are treated as known.

One implication of this model is that each of the n_i “trials” aggregated together in y_i is independent of all the others, at least once we condition on the predictors x . (So, e.g., whether any student passes the test is independent of whether any of their classmates pass, once we have conditioned on, say, teacher quality and average previous knowledge.) This may or may not be a reasonable assumption. When the successes or failures are dependent, even after conditioning on the predictors, the binomial model will be mis-specified. We can either try to get more information, and hope that conditioning on a richer set of predictors makes the dependence go away, or we can just try to account for the dependence by modifying the variance (“overdispersion” or “underdispersion”); we’ll return to both topics later.

13.1.2.3 Poisson Regression

Recall that the Poisson distribution has probability mass function

$$p(y) = \frac{e^{-\mu} \mu^y}{y!} \quad (13.8)$$

with $E[Y] = \text{Var}[Y] = \mu$. As you remember from basic probability, a Poisson distribution is what we get from a binomial if the probability of success per trial shrinks towards zero but the number of trials grows to infinity, so that we keep the mean number of successes the same:

$$\text{Binom}(n, \mu/n) \rightsquigarrow \text{Pois}(\mu) \quad (13.9)$$

This makes the Poisson distribution suitable for modeling counts with no fixed upper limit, but where the probability that any one of the many individual trials is a success is fairly low. If μ is allowed to be depend on the predictor variables, we get Poisson regression. Since the variance is equal to the mean, Poisson regression is always going to be heteroskedastic.

Since μ has to be non-negative, a natural link function is $g(\mu) = \log \mu$. This produces $g'(\mu) = 1/\mu$, and so weights $w = \mu$. When the expected count is large, so is the variance, which normally would reduce the weight put on an observation in regression, but in this case large expected counts also provide more information about the coefficients, so they end up getting increasing weight.

13.1.3 Uncertainty

Standard errors for coefficients can be worked out as in the case of weighted least squares for linear regression. Confidence intervals for the coefficients will be approximately Gaussian in large samples, for the usual likelihood-theory reasons, when the model is properly specified. One can, of course, also use either a parametric bootstrap, or resampling of cases/data-points to assess uncertainty.

Resampling of residuals can be trickier, because it is not so clear what counts as a residual. When the response variable is continuous, we can get “standardized” or “Pearson” residuals, $\hat{\epsilon}_i = \frac{y_i - \hat{\mu}(x_i)}{\sqrt{\hat{V}(\mu(x_i))}}$, resample them to get $\tilde{\epsilon}_i$, and then add $\tilde{\epsilon}_i \sqrt{\hat{V}(\mu(x_i))}$ to the fitted values. This does not really work when the response is discrete-valued, however.

13.2 Generalized Additive Models

In the development of generalized linear models, we use the link function g to relate the conditional mean $\mu(x)$ to the linear predictor $\eta(x)$. But really nothing in what we were doing required η to be *linear* in x . In particular, it all works perfectly well if η is an additive function of x . We form the effective responses z_i as before, and the weights w_i , but now instead of doing a linear regression on x_i we do an additive regression, using backfitting (or whatever). This gives us a generalized additive model (GAM).

Essentially everything we know about the relationship between linear models and additive models carries over. GAMs converge somewhat more slowly as n grows than do GLMs, but the former have less bias, and strictly include GLMs as special cases. The transformed (mean) response is related to the predictor variables not just through coefficients, but through whole partial response functions. If we want to test whether a GLM is well-specified, we can do so by comparing it to a GAM, and so forth.

In fact, one could even make $\eta(x)$ an arbitrary smooth function of x , to be estimated through (say) kernel smoothing of z_i on x_i . This is rarely done, however, partly because of curse-of-dimensionality issues, but also because, if one is going to go that far, one might as well just use kernels to estimate conditional distributions, as we will see in Chapter 15.

13.3 Weather Forecasting in Snoqualmie Falls

To make the use of logistic regression and GLMs concrete, we are going to build a simple weather forecaster. Our data consist of daily records, from the beginning of 1948 to the end of 1983, of precipitation at Snoqualmie Falls, Washington (Figure 13.1)¹. Each row of the data file is a different year; each column records, for that day of the year, the day's precipitation (rain or snow), in units of $\frac{1}{100}$ inch. Because of leap-days, there are 366 columns, with the last column having an NA value for three out of four years.

```
snoqualmie <- read.csv("snoqualmie.csv", header=FALSE)
# Turn into one big vector without year breaks
snoqualmie <- unlist(snoqualmie)
# Remove NAs from non-leap-years
snoqualmie <- na.omit(snoqualmie)
```

What we want to do is predict tomorrow's weather from today's. This would be of interest if we lived in Snoqualmie Falls, or if we operated either one of the local hydroelectric power plants, or the tourist attraction of the Falls themselves. Examining the distribution of the data (Figures 13.2 and 13.3) shows that there is a big spike in the distribution at zero precipitation, and that days of no precipitation can follow days of any amount of precipitation but seem to be less common after heavy precipitation.

These facts suggest that “no precipitation” is a special sort of event which would be worth predicting in its own right (as opposed to just being when the precipitation happens to be zero), so we will attempt to do so with logistic regression. Specifically, the input variable X_i will be the amount of precipitation on the i^{th} day, and the response Y_i will be the indicator variable for whether there was any precipitation on day $i + 1$ — that is, $Y_i = 1$ if $X_{i+1} > 0$, an $Y_i = 0$ if $X_{i+1} = 0$. We expect from Figure 13.3, as well as common experience, that the coefficient on X should be positive.²

Before fitting the logistic regression, it's convenient to re-shape the data:

```
vector.to.pairs <- function(v) {
  v <- as.numeric(v)
  n <- length(v)
  return(cbind(v[-1], v[-n]))
}
snoq.pairs <- vector.to.pairs(snoqualmie)
colnames(snoq.pairs) <- c("tomorrow", "today")
snoq <- as.data.frame(snoq.pairs)
```

This creates a two-column array, where the first column is the precipitation on day $i + 1$, and the second column is the precipitation on day i (hence the column names). Finally, I turn the whole thing into a data frame.

¹I learned of this data set from Guttorp (1995); the data file is available from <http://www.stat.washington.edu/peter/stoch.mod.data.html>.

²This does not attempt to model *how much* precipitation there will be tomorrow, if there is any. We could make that a separate model, if we can get this part right.

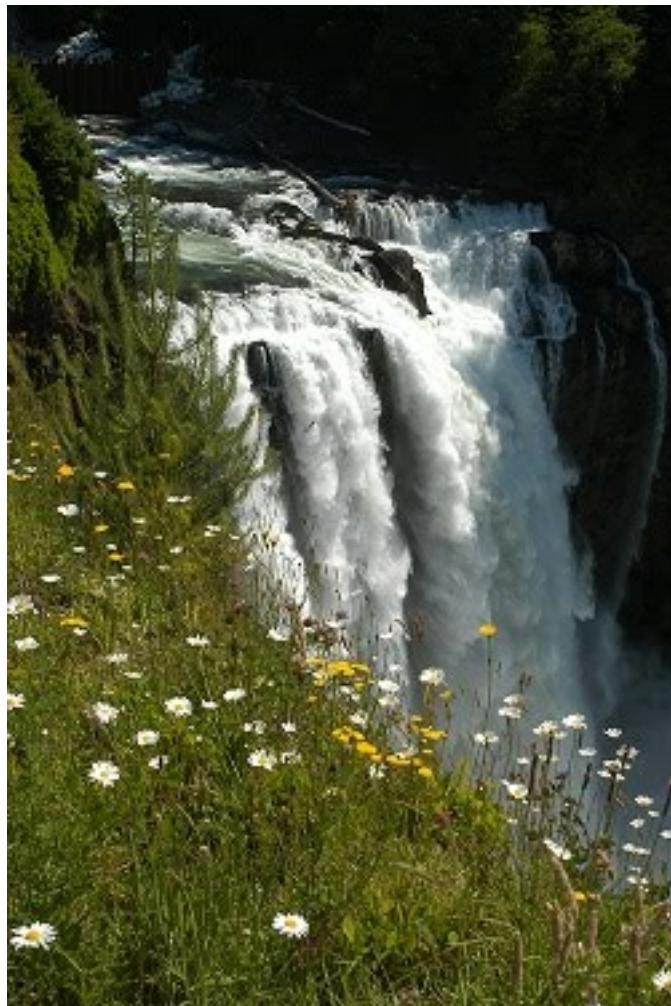
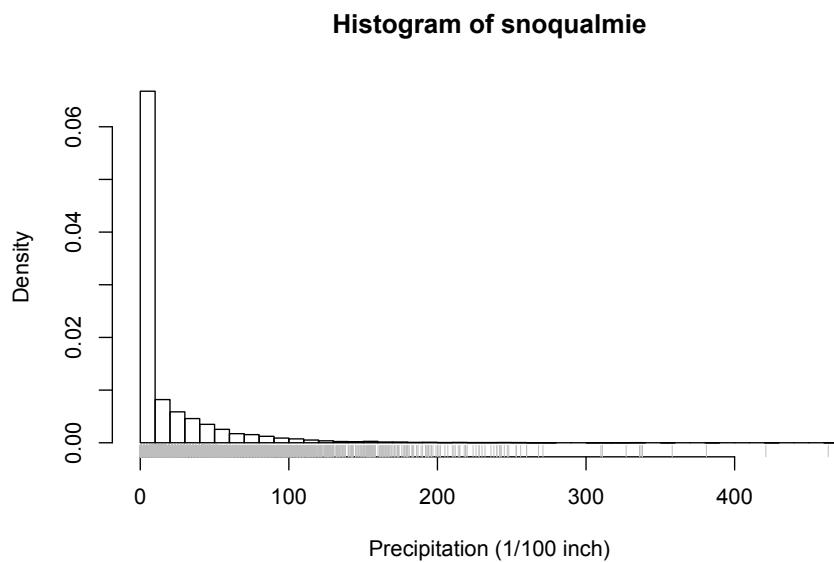
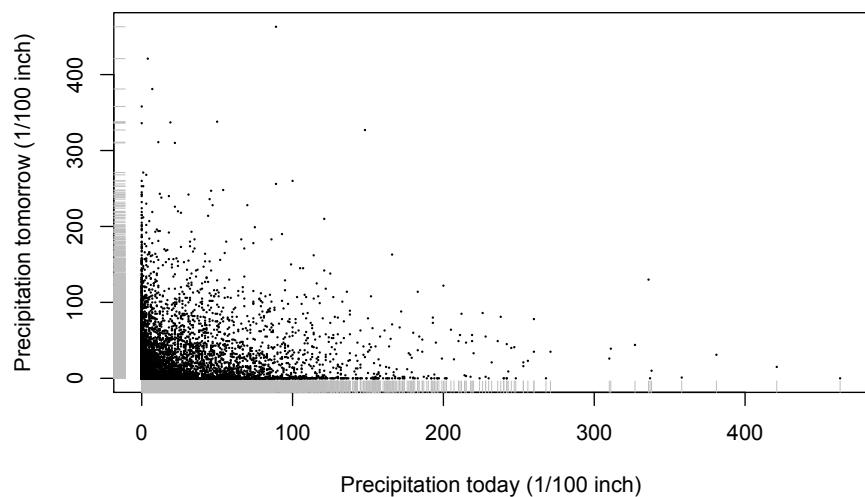


Figure 13.1: Snoqualmie Falls, Washington, on a sunny day. Photo by Jeannine Hall Gailey, from <http://myblog.webbish6.com/2011/07/17-years-and-hoping-for-another-17.html>. [[TODO: Get permission for photo use!]]



```
plot(hist(snoqualmie,n=50,probability=TRUE),xlab="Precipitation (1/100 inch)"  
rug(snoqualmie,col="grey")
```

Figure 13.2: Histogram of the amount of daily precipitation at Snoqualmie Falls



```
plot(snoqualmie[-length(snoqualmie)],snoqualmie[-1],  
      xlab="Precipitation today (1/100 inch)",  
      ylab="Precipitation tomorrow (1/100 inch)",cex=0.1)  
rug(snoqualmie[-length(snoqualmie)],side=1,col="grey")  
rug(snoqualmie[-1],side=2,col="grey")
```

Figure 13.3: Scatterplot showing relationship between amount of precipitation on successive days. Notice that days of no precipitation can follow days of any amount of precipitation, but seem to be more common when there is little or no precipitation to start with.

Now fitting is straightforward:

```
snoq.logistic <- glm((tomorrow > 0) ~ today, data=snoq, family=binomial)
```

To see what came from the fitting, run `summary`:

```
> summary(snoq.logistic)
```

Call:

```
glm(formula = (tomorrow > 0) ~ today, family = binomial, data = snoq)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.3713	-1.1805	0.9536	1.1693	1.1744

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	0.0071899	0.0198430	0.362	0.717
today	0.0059232	0.0005858	10.111	<2e-16 ***

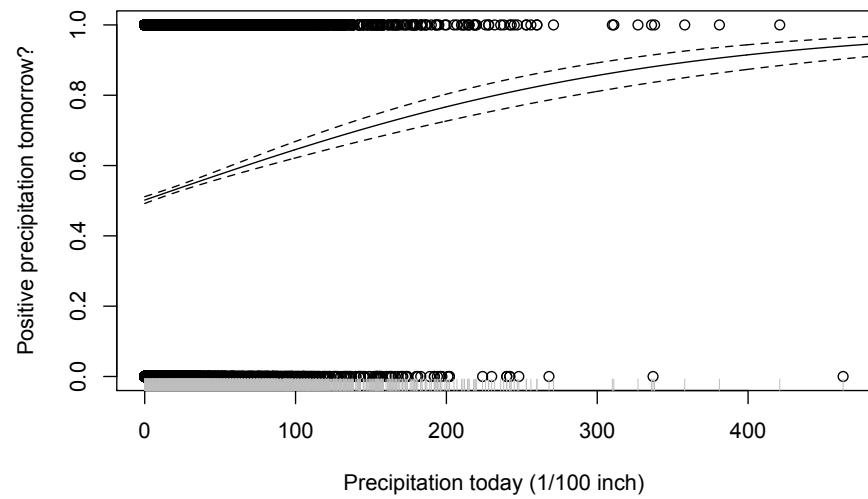
(I have cut off some uninformative bits of the output.) The coefficient on X , the amount of precipitation today, is indeed positive, and (if we can trust R's calculations) highly significant. There is also an intercept term, which is slight positive, but not very significant. We can see what the intercept term means by considering what happens when $X = 0$, i.e., on days of no precipitation. The linear predictor is then $0.0072 + 0 * (0.0059) = 0.0072$, and the predicted probability of precipitation is $e^{0.0072}/(1 + e^{0.0072}) = 0.502$. That is, even when there is no precipitation today, we predict that it is slightly more probable than not that there will be some precipitation tomorrow.³

We can get a more global view of what the model is doing by plotting the data and the predictions (Figure 13.4). This shows a steady increase in the probability of precipitation tomorrow as the precipitation today increases, though with the leveling off characteristic of logistic regression. The (approximate) 95% confidence limits for the predicted probability are (on close inspection) asymmetric, and actually slightly narrower at the far right than at intermediate values of X (Figure 13.3).

How well does this work? We can get a first sense of this by comparing it to a simple nonparametric smoothing of the data. Remembering that when Y is binary, $\Pr[Y = 1|X = x] = E[Y|X = x]$, we can use a smoothing spline to estimate $E[Y|X = x]$ (Figure 13.6). This would not be so great as a model — it ignores the fact that the response is a binary event and we're trying to estimate a probability, the fact that the variance of Y therefore depends on its mean, etc. — but it's at least indicative.

The result is in not-terribly-bad agreement with the logistic regression up to about 1.2 or 1.3 inches of precipitation, after which it runs significantly below the logistic

³For western Washington State, this is plausible — but see below.



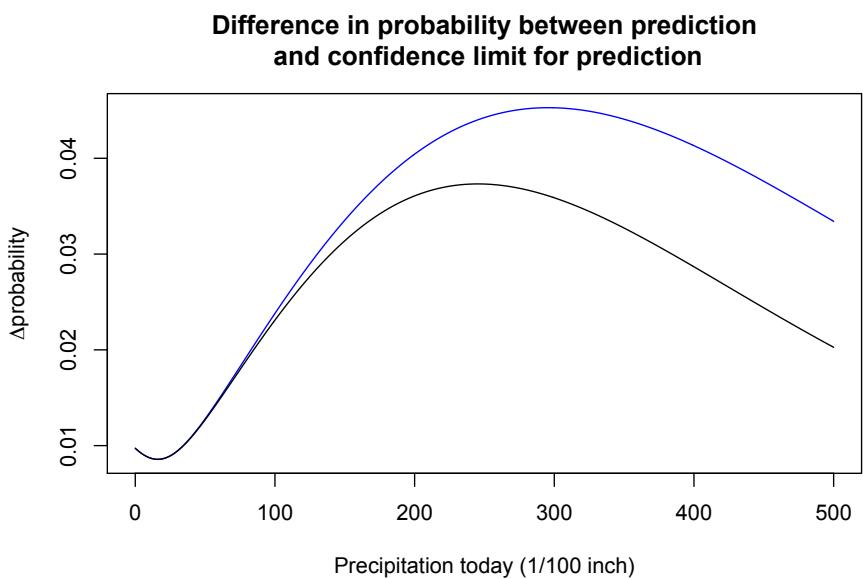
```

plot((tomorrow>0)~today,data=snoq,xlab="Precipitation today (1/100 inch)",
      ylab="Positive precipitation tomorrow?")
rug(snoq$today,side=1,col="grey")

data.plot <- data.frame(today=(0:500))
logistic.predictions <- predict(snoq.logistic,newdata=data.plot,se.fit=TRUE)
lines(0:500,ilogit(logistic.predictions$fit))
lines(0:500,ilogit(logistic.predictions$fit+1.96*logistic.predictions$se.fit),
      lty=2)
lines(0:500,ilogit(logistic.predictions$fit-1.96*logistic.predictions$se.fit),
      lty=2)

```

Figure 13.4: Data (dots), plus predicted probabilities (solid line) and approximate 95% confidence intervals from the logistic regression model (dashed lines). Note that calculating standard errors for predictions on the logit scale, and then transforming, is better practice than getting standard errors directly on the probability scale.

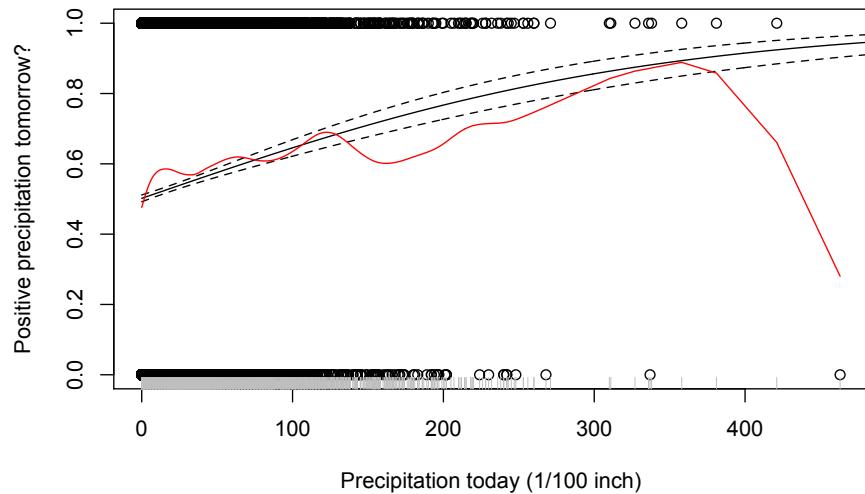


```

plot(0:500,ilogit(logistic.predictions$fit)
 -ilogit(logistic.predictions$fit-1.96*logistic.predictions$se.fit),
 type="l",col="blue",xlab="Precipitation today (1/100 inch)",
 main="Difference in probability between prediction\n
 and confidence limit for prediction",
 ylab = expression(paste(Delta, "probability")))
lines(0:500,ilogit(logistic.predictions$fit+1.96*logistic.predictions$se.fit)
 -ilogit(logistic.predictions$fit))

```

Figure 13.5: Distance from the fitted probability to the upper (black) and lower (blue) confidence limits. Notice that the two are not equal, and somewhat smaller at very large values of X than at intermediate ones. (Why?)



```
snoq.spline <- smooth.spline(x=snoq$today,y=(snoq$tomorrow>0))
lines(snoq.spline,col="red")
```

Figure 13.6: As Figure 13.4, plus a smoothing spline (red).

regression, rejoins it around 3.5 inches of precipitation, and then (as it were) falls off a cliff.

We can do better by fitting a generalized additive model. In this case, with only one predictor variable, this means using non-parametric smoothing to estimate the log odds — we're still using the logistic transformation, but only requiring that the log odds change smoothly with X , not that they be linear in X . The result (Figure 13.7) is actually quite similar to the spline, but a bit better behaved, and has confidence intervals. At the largest values of X , the latter span nearly the whole range from 0 to 1, which is not unreasonable considering the sheer lack of data there.

Visually, the logistic regression curve is usually but not always within the confidence limits of the non-parametric predictor. What can we say about the difference between the two models more quantitatively?

Numerically, the deviance is 18079.69 for the logistic regression, and 18036.77 for the GAM. We can go through the testing procedure outlined in Chapter 12. We need a simulator (which presumes that the logistic regression model is true), and we need to calculate the difference in deviance on simulated data many times.

```
# Simulate from the fitted logistic regression model for Snoqualmie
# Presumes: fitted values of the model are probabilities.
snoq.sim <- function(model=snoq.logistic) {
  fitted.probs <- fitted(model)
  n <- length(fitted.probs)
  new.binary <- rbinom(n, size=1, prob=fitted.probs)
  return(new.binary)
}
```

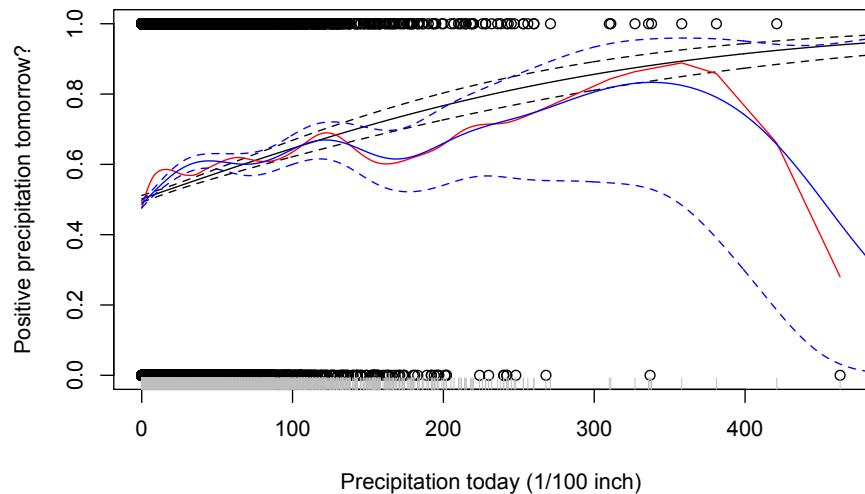
A quick check of the simulator against the observed values:

```
> summary(ifelse(snoq[,1]>0,1,0))
   Min. 1st Qu. Median Mean 3rd Qu. Max.
0.0000 0.0000 1.0000 0.5262 1.0000 1.0000
> summary(snoq.sim())
   Min. 1st Qu. Median Mean 3rd Qu. Max.
0.0000 0.0000 1.0000 0.5264 1.0000 1.0000
```

This suggests that the simulator is not acting crazily.

Now for the difference in deviances:

```
# Simulate from fitted logistic regression, re-fit logistic regression and
# GAM, calculate difference in deviances
diff.dev <- function(model=snoq.logistic,x=snoq[,2]) {
  y.new <- snoq.sim(model)
  GLM.dev <- glm(y.new ~ x,family=binomial)$deviance
  GAM.dev <- gam(y.new ~ s(x),family=binomial)$deviance
  return(GLM.dev-GAM.dev)
}
```



```
library(mgcv)
snoq.gam <- gam((tomorrow>0)~s(today), data=snoq, family=binomial)
gam.predictions <- predict.gam(snoq.gam, newdata=data.plot, se.fit=TRUE)
lines(0:500, ilogit(gam.predictions$fit), col="blue")
lines(0:500, ilogit(gam.predictions$fit+1.96*gam.predictions$se.fit),
      col="blue", lty=2)
lines(0:500, ilogit(gam.predictions$fit-1.96*gam.predictions$se.fit),
      col="blue", lty=2)
```

Figure 13.7: As Figure 13.6, but with the addition of a generalized additive model (blue line) and its confidence limits (dashed blue lines). *Note:* the predict function in the `gam` package does not allow one to calculate standard errors for new data. You may need to un-load the `gam` library first, with `detach(package:gam)`.

A single run of this takes about 1.5 seconds on my computer.

Finally, we calculate the distribution of difference in deviances under the null (that the logistic regression is properly specified), and the corresponding p -value:

```
diff.dev.obs <- snoq.logistic$deviance - snoq.gam$deviance
null.dist.of.diff.dev <- replicate(1000,diff.dev())
p.value <- (1+sum(null.dist.of.diff.dev > diff.dev.obs))/(1+length(null.dist.of.diff.dev))
```

Using a thousand replicates takes about 1500 seconds, or roughly 25 minutes, which is substantial, but not impossible; it gave a p -value of $< 10^{-3}$, and the following sampling distribution:

```
> summary(null.dist.of.diff.dev)
   Min. 1st Qu. Median Mean 3rd Qu. Max.
0.000097 0.002890 0.016770 2.267000 2.897000 29.750000
```

(A preliminary trial run of only 100 replicates, taking a few minutes, gave

```
> summary(null.dist.of.diff.dev)
   Min. 1st Qu. Median Mean 3rd Qu. Max.
0.000291 0.002681 0.013700 2.008000 2.121000 27.820000
```

which implies a p -value of < 0.01 . This would be good enough for many practical purposes.)

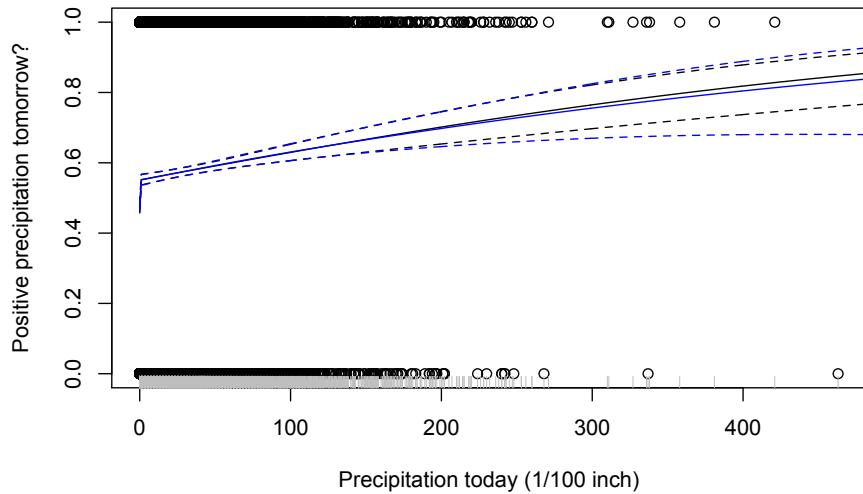
Having detected that there is a problem with the GLM, we can ask where it lies. We could just use the GAM, but it's more interesting to try to diagnose what's going on.

In this respect Figure 13.7 is actually a little misleading, because it leads the eye to emphasize the disagreement between the models at large X , when actually there are very few data points there, and so even large differences in predicted probabilities there contribute little to the over-all likelihood difference. What is actually more important is what happens at $X = 0$, which contains a very large number of observations (about 47% of all observations), and which we have reason to think is a special value anyway.

Let's try introducing a dummy variable for $X = 0$ into the logistic regression, and see what happens. It will be convenient to augment the data frame with an extra column, recording 1 whenever $X = 0$ and 0 otherwise.

```
snoq2 <- data.frame(snoq,dry=ifelse(snoq$today==0,1,0))
snoq2.logistic <- glm((tomorrow > 0) ~ today + dry,data=snoq2,family=binomial)
snoq2.gam <- gam((tomorrow > 0) ~ s(today) + dry,data=snoq2,family=binomial)
```

Notice that I allow the GAM to treat zero as a special value as well, by giving it access to that dummy variable. In principle, with enough data it can decide whether or not that is useful on its own, but since we have guessed that it is, we might as well include it. Figure 13.8 shows the data and the two new models. These are *extremely* close to each other. The new GLM has a deviance of 18015.65, lower than even the GAM before, and the new GAM has a deviance of 18015.21. The p -value is essentially



```

plot((tomorrow>0)^today,data=snoq,xlab="Precipitation today (1/100 inch)",
      ylab="Positive precipitation tomorrow?")
rug(snoq$today,side=1,col="grey")

data.plot=data.frame(data.plot,dry=ifelse(data.plot$today==0,1,0))
logistic.predictions2 <- predict(snoq2.logistic,newdata=data.plot,se.fit=TRUE)
lines(0:500,ilogit(logistic.predictions2$fit))
lines(0:500,ilogit(logistic.predictions2$fit+1.96*logistic.predictions2$se.fit),
      lty=2)
lines(0:500,ilogit(logistic.predictions2$fit-1.96*logistic.predictions2$se.fit),
      lty=2)
gam.predictions2 <- predict.gam(snoq2.gam,newdata=data.plot,se.fit=TRUE)
lines(0:500,ilogit(gam.predictions2$fit),col="blue")
lines(0:500,ilogit(gam.predictions2$fit+1.96*gam.predictions2$se.fit),
      col="blue",lty=2)
lines(0:500,ilogit(gam.predictions2$fit-1.96*gam.predictions2$se.fit),
      col="blue",lty=2)

```

Figure 13.8: As Figure 13.7, but allowing the two models to use a dummy variable indicating when today is completely dry ($X = 0$).

1 — and yet we know that this test *does* have power to detect departures from the parametric model. This is very promising.

Let's turn now to looking at calibration. The actual fraction of no-precipitation days which are followed by precipitation is

```
> mean(snoq$tomorrow[snoq$today==0]>0)
[1] 0.4702199
```

What does the new logistic model predict?

```
> predict(snoq2.logistic,newdata=data.frame(today=0,dry=1),type="response")
[1]
0.4702199
```

This should not be surprising — we've given the model a special parameter dedicated to getting this one probability exactly right! The hope however is that this will change the predictions made on days *with* precipitation so that they are better.

Looking at a histogram of fitted values (`hist(fitted(snoq2.logistic))`) shows a gap in the distribution of predicted probabilities between 0.47 and about 0.55, so we'll look first at days where the predicted probability is between 0.55 and 0.56.

```
> mean(snoq$tomorrow[(fitted(snoq2.logistic) >= 0.55)
& (fitted(snoq2.logistic) < 0.56)] > 0)
[1] 0.5474882
```

Not bad — but a bit painful to write out. Let's write a function:

```
frequency.vs.probability <- function(p.lower,p.upper=p.lower+0.01,
model=snoq2.logistic,events=(snoq$tomorrow>0)) {
  fitted.probs <- fitted(model)
  indices <- (fitted.probs >= p.lower) & (fitted.probs < p.upper)
  ave.prob <- mean(fitted.probs[indices])
  frequency <- mean(events[indices])
  se <- sqrt(ave.prob*(1-ave.prob)/sum(indices))
  out <- list(frequency=frequency,ave.prob=ave.prob,se=se)
  return(out)
}
```

I have added a calculation of the average predicted probability, and a crude estimate of the standard error we should expect if the observations really are binomial with the predicted probabilities⁴. Try the function out before doing anything rash:

```
> frequency.vs.probability(0.55)
$frequency
[1] 0.5474882

$ave.prob
```

⁴This could be improved by averaging predicted variances for each point, but using probability ranges of 0.01 makes it hardly worth the effort.

```
[1] 0.5548081
```

```
$se
[1] 0.00984567
```

This agrees with our previous calculation.

Now we can do this for a lot of probability brackets:

```
f.vs.p <- sapply((55:74)/100,frequency.vs.probability)
```

This comes with some unfortunate R cruft, removable thus

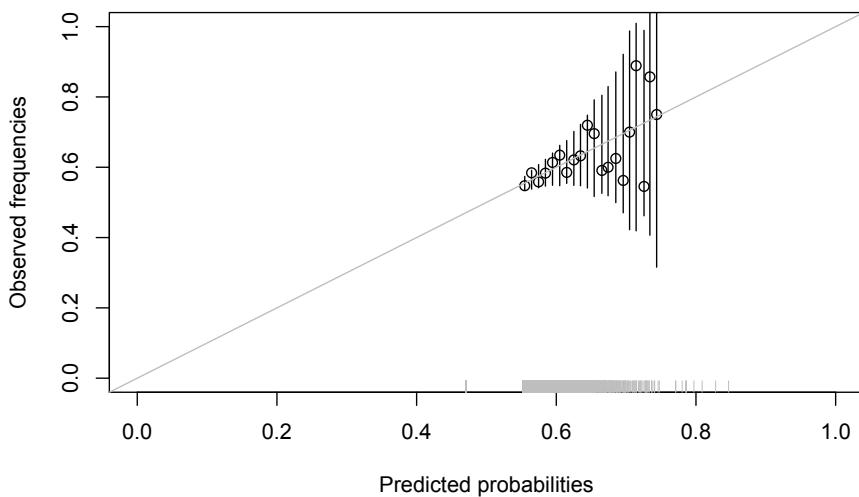
```
f.vs.p <- data.frame(frequency=unlist(f.vs.p["frequency"]),
ave.prob=unlist(f.vs.p["ave.prob"]),se=unlist(f.vs.p["se"]))
```

and we're ready to plot (Figure 13.9). The observed frequencies are generally quite near to the predicted probabilities, especially when the number of observations is large and so the sample frequency *should* be close to the true probability. While I wouldn't want to say this was the last word in weather forecasting⁵, it's surprisingly good for such a simple model.

13.4 Exercises

1. In binomial regression, we have $Y|X = x \sim \text{Binom}(n, p(x))$, where $p(x)$ follows a logistic model. Work out the link function $g(\mu)$, the variance function $V(\mu)$, and the weights w , assuming that n is known and not random.
2. Homework 5, on predicting the death rate in Chicago, is a good candidate for using Poisson regression. Repeat the exercises in that problem set with Poisson-response GAMs. How do the estimated functions change? Why is this any different from just taking the log of the death counts, as we did in the homework?

⁵There is an extensive discussion of this data in chapter 2 of Guttorp's book, including many significant refinements, such as dependence across multiple days.



```
plot(f.vs.p$ave.prob,f.vs.p$frequency,xlim=c(0,1),ylim=c(0,1),
      xlab="Predicted probabilities",ylab="Observed frequencies")
rug(fitted(snoq2.logistic),col="grey")
abline(0,1,col="grey")
segments(x0=f.vs.p$ave.prob,y0=f.vs.p$ave.prob-1.96*f.vs.p$se,
         y1=f.vs.p$ave.prob+1.96*f.vs.p$se)
```

Figure 13.9: Calibration plot for the modified logistic regression model `snoq2.logistic`. Points show the actual frequency of precipitation for each level of predicted probability. Vertical lines are (approximate) 95% sampling intervals for the frequency, given the predicted probability and the number of observations.

14:41 Thursday 25th April, 2013

Part II

**Multivariate Data,
Distributions, and Latent
Structure**

Chapter 14

Multivariate Distributions

14.1 Review of Definitions

Let's review some definitions from basic probability. When we have a random vector \vec{X} with p different components, X_1, X_2, \dots, X_p , the **joint cumulative distribution function** is

$$F(\vec{a}) = F(a_1, a_2, \dots, a_p) = \Pr(X_1 \leq a_1, X_2 \leq a_2, \dots, X_p \leq a_p) \quad (14.1)$$

Thus

$$F(\vec{b}) - F(\vec{a}) = \Pr(a_1 < X_1 \leq b_1, a_2 < X_2 \leq b_2, \dots, a_p < X_p \leq b_p) \quad (14.2)$$

This is the probability that X is in a (hyper-)rectangle, rather than just in an interval.

The **joint probability density function** is

$$p(\vec{x}) = p(x_1, x_2, \dots, x_p) = \frac{\partial^p F(a_1, \dots, a_p)}{\partial a_1 \dots \partial a_p} \Big|_{\vec{a}=\vec{x}} \quad (14.3)$$

Of course,

$$F(\vec{a}) = \int_{-\infty}^{a_1} \int_{-\infty}^{a_2} \dots \int_{-\infty}^{a_p} p(x_1, x_2, \dots, x_p) dx_p \dots dx_2 dx_1 \quad (14.4)$$

(In this case, the order of integration doesn't matter. Why?)

From these, and especially from the joint PDF, we can recover the marginal PDF of any group of variables, say those numbered 1 through q ,

$$p(x_1, x_2, \dots, x_q) = \int p(x_1, x_2, \dots, x_p) dx_{q+1} dx_{q+2} \dots dx_p \quad (14.5)$$

(What are the limits of integration here?) Then the conditional pdf for some variables given the others — say, use variables 1 through q to condition those numbered $q+1$

through p — just comes from division:

$$p(x_{q+1}, x_{q+2}, \dots, x_p | X_1 = x_1, \dots, X_q = x_q) = \frac{p(x_1, x_2, \dots, x_p)}{p(x_1, x_2, \dots, x_q)} \quad (14.6)$$

These two tricks can be iterated, so, for instance,

$$p(x_3 | x_1) = \int p(x_3, x_2 | x_1) dx_2 \quad (14.7)$$

14.2 Multivariate Gaussians

The multivariate Gaussian is just the generalization of the ordinary Gaussian to vectors. Scalar Gaussians are parameterized by a mean μ and a variance σ^2 , so we write $X \sim \mathcal{N}(\mu, \sigma^2)$. Multivariate Gaussians, likewise, are parameterized by a mean vector $\vec{\mu}$, and a variance-covariance matrix Σ , written $\vec{X} \sim \mathcal{MVN}(\vec{\mu}, \Sigma)$. The components of $\vec{\mu}$ are the means of the different components of \vec{X} . The i, j^{th} component of Σ is the covariance between X_i and X_j (so the diagonal of Σ gives the component variances).

Just as the probability density of scalar Gaussian is

$$p(x) = (2\pi\sigma^2)^{-1/2} \exp\left\{-\frac{1}{2}\frac{(x - \mu)^2}{\sigma^2}\right\} \quad (14.8)$$

the probability density of the multivariate Gaussian is

$$p(\vec{x}) = (2\pi \det \Sigma)^{-p/2} \exp\left\{-\frac{1}{2}(\vec{x} - \vec{\mu}) \cdot \Sigma^{-1}(\vec{x} - \vec{\mu})\right\} \quad (14.9)$$

Finally, remember that the parameters of a Gaussian change along with linear transformations

$$X \sim \mathcal{N}(\mu, \sigma^2) \Leftrightarrow aX + b \sim \mathcal{N}(a\mu + b, a^2\sigma^2) \quad (14.10)$$

and we can use this to “standardize” any Gaussian to having mean 0 and variance 1 (by looking at $\frac{X-\mu}{\sigma}$). Likewise, if

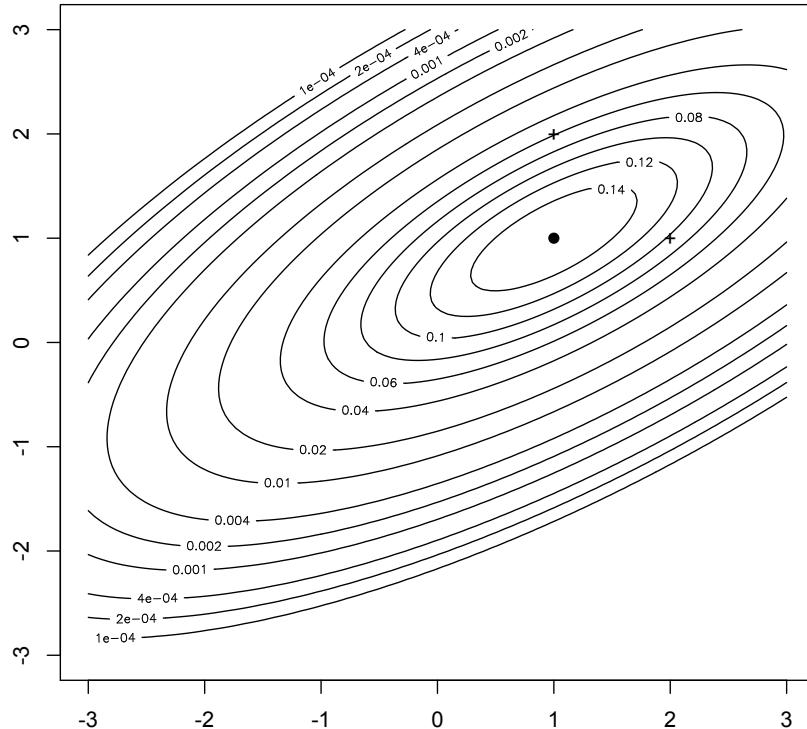
$$\vec{X} \sim \mathcal{MVN}(\vec{\mu}, \Sigma) \quad (14.11)$$

then

$$a\vec{X} + \vec{b} \sim \mathcal{MVN}(a\vec{\mu} + \vec{b}, a\Sigma a^T) \quad (14.12)$$

In fact, the analogy between the ordinary and the multivariate Gaussian is so complete that it is very common to not really distinguish the two, and write \mathcal{N} for both.

The multivariate Gaussian density is most easily visualized when $p = 2$, as in Figure 14.1. The probability contours are ellipses. The density changes comparatively slowly along the major axis, and quickly along the minor axis. The two points marked + in the figure have equal geometric distance from $\vec{\mu}$, but the one to its right lies on a higher probability contour than the one above it, because of the directions of their displacements from the mean.



```

library(mvtnorm)
x.points <- seq(-3,3,length.out=100)
y.points <- x.points
z <- matrix(0,nrow=100,ncol=100)
mu <- c(1,1)
sigma <- matrix(c(2,1,1,1),nrow=2)
for (i in 1:100) {
  for (j in 1:100) {
    z[i,j] <- dmvnorm(c(x.points[i],y.points[j]),mean=mu,sigma=sigma)
  }
}
contour(x.points,y.points,z)

```

Figure 14.1: Probability density contours for a two-dimensional multivariate Gaussian, with mean $\vec{\mu} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ (solid dot), and variance matrix $\Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$. Using `expand.grid`, as in Chapter 4, would be more elegant coding than this double `for` loop.

14:41 Thursday 25th April, 2013

14.2.1 Linear Algebra and the Covariance Matrix

We can use some facts from linear algebra to understand the general pattern here, for arbitrary multivariate Gaussians in an arbitrary number of dimensions. The covariance matrix Σ is symmetric and positive-definite, so we know from matrix algebra that it can be written in terms of its eigenvalues and eigenvectors:

$$\Sigma = \mathbf{v}^T \mathbf{d} \mathbf{v} \quad (14.13)$$

where \mathbf{d} is the diagonal matrix of the eigenvalues of Σ , and \mathbf{v} is the matrix whose columns are the eigenvectors of Σ . (Conventionally, we put the eigenvalues in \mathbf{d} in order of decreasing size, and the eigenvectors in \mathbf{v} likewise, but it doesn't matter so long as we're consistent about the ordering.) Because the eigenvectors are all of length 1, and they are all perpendicular to each other, it is easy to check that $\mathbf{v}^T \mathbf{v} = \mathbf{I}$, so $\mathbf{v}^{-1} = \mathbf{v}^T$ and \mathbf{v} is an orthogonal matrix. What actually shows up in the equation for the multivariate Gaussian density is Σ^{-1} , which is

$$(\mathbf{v}^T \mathbf{d} \mathbf{v})^{-1} = \mathbf{v}^{-1} \mathbf{d}^{-1} (\mathbf{v}^T)^{-1} = \mathbf{v}^T \mathbf{d}^{-1} \mathbf{v} \quad (14.14)$$

Geometrically, orthogonal matrices represent rotations. Multiplying by \mathbf{v} rotates the coordinate axes so that they are parallel to the eigenvectors of Σ . Probabilistically, this tells us that the axes of the probability-contour ellipse are parallel to those eigenvectors. The radii of those axes are proportional to the square roots of the eigenvalues. To see *that*, look carefully at the math. Fix a level for the probability density whose contour we want, say f_0 . Then we have

$$f_0 = (2\pi \det \Sigma)^{-p/2} \exp \left\{ -\frac{1}{2} (\vec{x} - \vec{\mu}) \cdot \Sigma^{-1} (\vec{x} - \vec{\mu}) \right\} \quad (14.15)$$

$$c = (\vec{x} - \vec{\mu}) \cdot \Sigma^{-1} (\vec{x} - \vec{\mu}) \quad (14.16)$$

$$= (\vec{x} - \vec{\mu})^T \mathbf{v}^T \mathbf{d}^{-1} \mathbf{v} (\vec{x} - \vec{\mu}) \quad (14.17)$$

$$= (\vec{x} - \vec{\mu})^T \mathbf{v}^T \mathbf{d}^{-1/2} \mathbf{d}^{-1/2} \mathbf{v} (\vec{x} - \vec{\mu}) \quad (14.18)$$

$$= (\mathbf{d}^{-1/2} \mathbf{v} (\vec{x} - \vec{\mu}))^T (\mathbf{d}^{-1/2} \mathbf{v} (\vec{x} - \vec{\mu})) \quad (14.19)$$

$$= \|\mathbf{d}^{-1/2} \mathbf{v} (\vec{x} - \vec{\mu})\|^2 \quad (14.20)$$

where c combines f_0 and all the other constant factors, and $\mathbf{d}^{-1/2}$ is the diagonal matrix whose entries are one over the square roots of the eigenvalues of Σ . The $\mathbf{v}(\vec{x} - \vec{\mu})$ term takes the displacement of \vec{x} from the mean, $\vec{\mu}$, and replaces the components of that vector with its projection on to the eigenvectors. Multiplying by $\mathbf{d}^{-1/2}$ then scales those projections, and so the radii have to be proportional to the square roots of the eigenvalues.¹

¹If you know about principal components analysis and think that all this manipulation of eigenvectors and eigenvalues of the covariance matrix seems familiar, you're right; this was one of the ways in which PCA was originally discovered. But PCA does not require any distributional assumptions. If you do not know about PCA, wait for Chapter 17.

14.2.2 Conditional Distributions and Least Squares

Suppose that \vec{X} is bivariate, so $p = 2$, with mean vector $\vec{\mu} = (\mu_1, \mu_2)$, and variance matrix $\begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$. One can show (exercise!) that the conditional distribution of X_2 given X_1 is Gaussian, and in fact

$$X_2|X_1 = x_1 \sim \mathcal{N}(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}) \quad (14.21)$$

To understand what is going on here, remember from Chapter 1 that the optimal slope for linearly regressing X_2 on X_1 would be $\text{Cov}[X_2, X_1]/\text{Var}[X_1]$. This is *precisely* the same as $\Sigma_{21}\Sigma_{11}^{-1}$. So in the bivariate Gaussian case, the best linear regression and the optimal regression are exactly the same — there is no need to consider non-linear regressions. Moreover, we get the same conditional variance for each value of x_1 , so the regression of X_2 on X_1 is homoskedastic, with independent Gaussian noise. This is, in short, exactly the situation which all the standard regression formulas aim at.

More generally, if X_1, X_2, \dots, X_p are multivariate Gaussian, then conditioning on X_1, \dots, X_q gives the remaining variables X_{q+1}, \dots, X_p a Gaussian distribution as well.

If we say that $\vec{\mu} = (\vec{\mu}_A, \vec{\mu}_B)$ and $\Sigma = \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix}$, where A stands for the conditioning variables and B for the conditioned, then

$$\vec{X}_B|\vec{X}_A = \vec{x}_a \sim \mathcal{MVN}(\vec{\mu}_B + \Sigma_{BA}\Sigma_{AA}^{-1}(\vec{x}_A - \vec{\mu}_A), \Sigma_{BB} - \Sigma_{BA}\Sigma_{AA}^{-1}\Sigma_{AB}) \quad (14.22)$$

(Remember that here $\Sigma_{BA} = \Sigma_{AB}^T$ [Why?].) This, too, is just doing a linear regression of \vec{X}_B on \vec{X}_A .

14.2.3 Projections of Multivariate Gaussians

A useful fact about multivariate Gaussians is that all their univariate projections are also Gaussian. That is, if $\vec{X} \sim \mathcal{MVN}(\vec{\mu}, \Sigma)$, and we fix any unit vector \vec{w} , then $\vec{w} \cdot \vec{X}$ has a Gaussian distribution. This is easy to see if Σ is diagonal: then $\vec{w} \cdot \vec{X}$ reduces to a sum of independent Gaussians, which we know from basic probability is also Gaussian. But we can use the eigen-decomposition of Σ to check that this holds more generally.

One can also show that the converse is true: if $\vec{w} \cdot \vec{X}$ is a univariate Gaussian for every choice of \vec{w} , then \vec{X} must be multivariate Gaussian. This fact is more useful for probability theory than for data analysis², but it's still worth knowing.

14.2.4 Computing with Multivariate Gaussians

Computationally, it is not hard to write functions to calculate the multivariate Gaussian density, or to generate multivariate Gaussian random vectors. Unfortunately,

²It's a special case of a result called the Cramér-Wold theorem, or the Cramér-Wold device, which asserts that two random vectors \vec{X} and \vec{Y} have the same distribution if and only if $\vec{w} \cdot \vec{X}$ and $\vec{w} \cdot \vec{Y}$ have the same distribution for every \vec{w} .

no one seems to have thought to put a standard set of such functions in the basic set of R packages, so you have to use a different library. The MASS library contains a function, `mvrnorm`, for generating multivariate Gaussian random vectors. The `mvtnorm` contains functions for calculating the density, cumulative distribution and quantiles of the multivariate Gaussian, as well as generating random vectors³. The package `mixtools`, which will use in Chapter 19 for mixture models, includes functions for the multivariate Gaussian density and for random-vector generation.

14.3 Inference with Multivariate Distributions

As with univariate distributions, there are several ways of doing statistical inference for multivariate distributions. Here I will focus on parametric inference, since non-parametric inference is covered in Chapter 15.

14.3.1 Estimation

The oldest method of estimating parametric distributions is **moment-matching** or the **method of moments**. If there are q unknown parameters of the distribution, one picks q expectation values — means, variances, and covariances are popular — and finds algebraic expressions for them in terms of the parameters. One then sets these equal to the sample moments, and solves for the corresponding parameters. This method can fail if you happen to chose algebraically redundant moments, since then you really have fewer equations than unknowns⁴. Perhaps more importantly, it quickly becomes very awkward to set up and solve all the necessary equations, and anyway this neglects a lot of information the data.

The approach which has generally replaced the method of moments is simply the method of maximum likelihood. The likelihood is defined in *exactly* the same way for multivariate distributions as for univariate ones. If the observations \vec{x}_i are assumed to be independent, and θ stands for all the parameters bundled together, then

$$L(\theta) = \prod_{i=1}^n p(\vec{x}_i; \theta) \quad (14.23)$$

and the maximum likelihood estimate (MLE) is

$$\hat{\theta}_{MLE} = \operatorname{argmax}_{\theta} L(\theta) \quad (14.24)$$

Again, as in the univariate case, it is usually simpler and more stable to use the log-likelihood:

$$\ell(\theta) = \sum_{i=1}^n \log p(\vec{x}_i; \theta) \quad (14.25)$$

³It also has such functions for multivariate t distributions, which are to multivariate Gaussians exactly as ordinary t distributions are to univariate Gaussians.

⁴For instance, you can't use variances, covariances *and* correlations, since knowing variances and covariances fixes the correlations.

making use of the fact that

$$\operatorname{argmax}_{\theta} L(\theta) = \operatorname{argmax}_{\theta} \ell(\theta) \quad (14.26)$$

The simplest possible case for this is the multivariate Gaussian, where the MLE is the sample mean vector and the sample covariance matrix. Generally, however, the maximum likelihood estimate and the moment-matching estimate will not coincide.

Of course, for inference, we generally need more than just a point estimate like $\hat{\theta}_{MLE}$, we need some idea of uncertainty. We can get that pretty generically from maximum likelihood. Very informally, since we are maximizing the log-likelihood, the precision with which we estimate the parameter depends on how sharp that maximum is — the bigger the second derivative, the more precise our estimate. In fact, one can show (Wasserman, 2003, §9.7 and 9.10) that

$$\hat{\theta}_{MLE} \rightsquigarrow \mathcal{MVN}(\theta_0, -\mathbf{H}^{-1}(\theta_0)) \quad (14.27)$$

where θ_0 is the true parameter value, and \mathbf{H} is the **Hessian** of the log-likelihood, its matrix of second partial derivatives,

$$H_{jk}(\theta) = \left. \frac{\partial^2 \ell}{\partial \theta_j \partial \theta_k} \right|_{\theta} \quad (14.28)$$

In turn,

$$\frac{1}{n} H_{jk}(\theta_0) \rightarrow \mathbf{E} \left[\frac{\partial^2 \log p(X; \theta)}{\partial \theta_j \partial \theta_k} \right] \equiv -I_{jk}(\theta_0) \quad (14.29)$$

which defines the **Fisher information matrix \mathbf{I}** . One can therefore get (approximate) confidence regions by assuming that $\hat{\theta}_{MLE}$ has a Gaussian distribution with covariance matrix $n^{-1}\mathbf{I}^{-1}(\hat{\theta}_{MLE})$, or, somewhat more accurately, $-\mathbf{H}^{-1}(\hat{\theta}_{MLE})$. We thus get that $\operatorname{Var}[\hat{\theta}_{MLE}] = O(n^{-1})$, and $\hat{\theta}_{MLE} - \theta_0 = O(n^{-1/2})$.

Note that Eq. 14.27 is *only* valid as $n \rightarrow \infty$, and further assumes that (i) the model is well-specified, (ii) the true parameter value θ_0 is in the interior of the parameter space, and (iii) the Hessian matrix is strictly positive. If these conditions fail, then the distribution of the MLE need not be Gaussian, or controlled by the Fisher information matrix, etc.

An alternative to the asymptotic formula, Eq. 14.27, is simply parametric or non-parametric bootstrapping.

14.3.2 Model Comparison

Out of sample, models can be compared on log-likelihood. When a strict out-of-sample comparison is not possible, we can use cross-validation.

In sample, a likelihood ratio test can be used. This has two forms, depending on the relationship between the models. Suppose that there is a large or wide model, with parameter Θ , and a narrow or small model, with parameter θ , which we get

by fixing some of the components of Θ . Thus the dimension of Θ is q and that of θ is $r < q$. Since every distribution we can get from the narrow model we can also get from the wide model, in-sample the likelihood of the wide model must always be larger. Thus

$$\ell(\widehat{\Theta}) - \ell(\widehat{\theta}) \geq 0 \quad (14.30)$$

Here we have a clear null hypothesis, which is that the data comes from the narrower, smaller model. Under this null hypothesis, as $n \rightarrow \infty$,

$$2[\ell(\widehat{\Theta}) - \ell(\widehat{\theta})] \rightsquigarrow \chi^2_{q-r} \quad (14.31)$$

provided that the restriction imposed by the small model doesn't place it on the boundary of the parameter space of Θ . (See Appendix C.)

For instance, suppose that \vec{X} is bivariate, and the larger model is an unrestricted Gaussian, so $\Theta = \{(\mu_1, \mu_2), \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\}$. A possible narrow model might impose the assumption that the components of \vec{X} are uncorrelated, so $\theta = \{(\mu_1, \mu_2), \begin{bmatrix} \Sigma_{11} & 0 \\ 0 & \Sigma_{22} \end{bmatrix}\}$.

This is a restriction on the broader model, but not one which is on the boundary of the parameter space, so the large-sample χ^2 distribution should apply. A restriction which *would* be on the boundary would be to insist that X_2 was constant, so $\Sigma_{22} = 0$. (This would also force $\Sigma_{12} = 0$.)

If, on the other hand, that we have two models, with parameters θ and ψ , and they are completely non-nested, meaning there are no parameter combinations where

$$p(\cdot; \theta) = p(\cdot; \psi) \quad (14.32)$$

then in many ways things become easier. For *fixed* parameter values θ_0, ψ_0 , the mean log-likelihood ratio is just an average of IID terms:

$$\frac{1}{n} [\ell(\theta_0) - \ell(\psi_0)] \equiv \frac{1}{n} \sum_{i=1}^n \Lambda_i \quad (14.33)$$

$$= \frac{1}{n} \sum_{i=1}^n \log \frac{p(x_i; \theta_0)}{p(x_i; \psi_0)} \quad (14.34)$$

By the law of large numbers, then, the mean log-likelihood ratio converges to an expected value $E[\Lambda]$. This is positive if θ_0 has a higher expected log-likelihood than ψ_0 , and negative the other way around. Furthermore, by the central limit theorem, as n grows, the fluctuations around this expected value are Gaussian, with variance σ_Λ^2/n . We can estimate σ_Λ^2 by the sample variance of $\log \frac{p(x_i; \theta_0)}{p(x_i; \psi_0)}$.

Ordinarily, we don't have just a single parameter value for each model, but also ordinarily, $\widehat{\theta}_{MLE}$ and $\widehat{\psi}_{MLE}$ both converge to limits, which we can call θ_0 and ψ_0 . At the cost of some fancy probability theory, one can show that, in the non-nested case,

$$\frac{\sqrt{n}}{n} \frac{\ell(\widehat{\theta}) - \ell(\widehat{\psi})}{\sigma_\Lambda^2} \rightsquigarrow \mathcal{N}(E[\Lambda], 1) \quad (14.35)$$

and that we can consistently estimate $\mathbf{E}[\Lambda]$ and σ_Λ^2 by “plugging in” $\hat{\theta}$ and $\hat{\psi}$ in place of θ_0 and ψ_0 . This gives the **Vuong test** for comparing the two models Vuong (1989). The null hypothesis in the Vuong test is that the two models are equally good (and neither is exactly true). In this case,

$$V = \frac{1}{\sqrt{n}} \frac{\ell(\hat{\theta}) - \ell(\hat{\psi})}{\hat{\sigma}_\Lambda} \rightsquigarrow \mathcal{N}(0, 1) \quad (14.36)$$

If V is significantly positive, we have evidence in favor of the θ model being better (though not necessarily *true*), while if it is significantly negative we have evidence in favor of the ψ model being better.

The cases where two models *partially* overlap is complicated; see Vuong (1989) for the gory details⁵

14.3.3 Goodness-of-Fit

For univariate distributions, we often assess goodness-of-fit through the Kolmogorov-Smirnov (KS) test⁶, where the test statistic is

$$d_{KS} = \max_a |\hat{F}_n(a) - F(a)| \quad (14.37)$$

with \hat{F}_n being the empirical CDF, and F its theoretical counterpart. The null hypothesis here is that the data were drawn IID from F , and what Kolmogorov and Smirnov did was to work out the distribution of d_{KS} under this null hypothesis, and show it was the same for all F (at least for large n). This lets us actually calculate p values.

We could use such a test statistic for multivariate data, where we’d just take the maximum over vectors a , rather than scalars. But the problem is that we do not know its sampling distribution under the null hypothesis in the multivariate case — Kolmogorov and Smirnov’s arguments don’t work there — so we don’t know whether a given value of d_{KS} is large or small or what.

There is however a fairly simple approximate way of turning univariate tests into multivariate ones. Suppose our data consists of vectors $\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n$. Pick a unit vector \vec{w} , and set $z_i = \vec{w} \cdot \vec{x}_i$. Geometrically, this is just the projection of the data along the direction \vec{w} , but these projections are *univariate* random variables. If the \vec{x}_i were drawn from F , then the z_i must have been drawn from the corresponding projection of F , call it $F_{\vec{w}}$. If we can work out the latter distribution, then we can apply our favorite univariate test to the z_i . If the fit is bad, then we know that the \vec{x}_i can’t have come from F . If the fit is good for the z_i , then the fit is also good for the \vec{x}_i — at least

⁵If you are curious about why this central-limit-theorem argument doesn’t work in the nested case, notice that when we have nested models, and the null hypothesis is true, then $\hat{\Theta} \rightarrow \hat{\theta}$, so the numerator in the Vuong test statistic, $[\ell(\hat{\theta}) - \ell(\hat{\psi})]/n$, is converging to zero, but so is the denominator $\hat{\sigma}_\Lambda^2$. Since $0/0$ is undefined, we need to use a stochastic version of L’Hoptial’s rule, which gives us back Eq. 14.31. See, yet again, Vuong (1989).

⁶I discuss the KS test here for concreteness. Much the same ideas apply to the Anderson-Darling test, the Cramér-von Mises test, and others which, not being such good ideas, were only invented by one person.

along the direction \vec{w} . Now, we can either carefully pick \vec{w} to be a direction which we care about for some reason, or we can chose it *randomly*. If the projection of the \vec{x}_i along several random directions matches that of F , it becomes rather unlikely that they fail to match over-all⁷.

To summarize:

1. Chose a random unit vector \vec{W} . (For instance, let $\vec{U} \sim \mathcal{MVN}(0, \mathbf{I}_p)$, and $\vec{W} = \vec{U}/\|\vec{U}\|$.)
2. Calculate $Z_i = \vec{W} \cdot \vec{x}_i$.
3. Calculate the corresponding projection of the theoretical distribution F , call it $F_{\vec{W}}$.
4. Apply your favorite univariate goodness-of-fit test to Z_i and $F_{\vec{W}}$.
5. Repeat (1)–(4) multiple times, with Bonferroni correction for multiple testing.

14.4 Exercises

To think through, not to hand in.

1. Write a function to calculate the density of a multivariate Gaussian with a given mean vector and covariance matrix. Check it against an existing function from one of the packages mentioned in §14.2.4.
2. Write a function to generate multivariate Gaussian random vectors, using `rnorm`.
3. If \vec{X} has mean $\vec{\mu}$ and variance-covariance matrix Σ , and \vec{w} is a fixed, non-random vector, find the mean and variance of $\vec{w} \cdot \vec{X}$.
4. If $\vec{X} \sim \mathcal{MVN}(\vec{\mu}, \Sigma)$, and \mathbf{b} and \mathbf{c} are two non-random matrices, find the covariance matrix of $\mathbf{b}\vec{X}$ and $\mathbf{c}\vec{X}$.

⁷Theoretically, we appeal to the Cramér-Wold device again: the random vectors \vec{X} and \vec{Y} have the same distribution if and only if $\vec{w} \cdot \vec{X}$ and $\vec{w} \cdot \vec{Y}$ have the same distribution for every \vec{w} . Failing to match for any \vec{w} implies that \vec{X} and \vec{Y} have different distributions. Conversely, if \vec{X} and \vec{Y} differ in distribution at all, $\vec{w} \cdot \vec{X}$ must differ in distribution from $\vec{w} \cdot \vec{Y}$ for *some* choice of \vec{w} . Randomizing the choice of \vec{w} gives us power to detect a lot of differences in distribution.

Chapter 15

Estimating Distributions and Densities

We have spent a lot of time looking at how to estimate expectations (which is regression). We have also seen how to estimate variances, by turning it into a problem about expectations. We could extend the same methods to looking at higher moments — if you need to find the conditional skewness or kurtosis functions¹, you can tackle that in the same way as finding the conditional variance. But what if we want to look at the whole distribution?

You've already seen the parametric solution to the problem in earlier statistics courses: posit a parametric model for the density (Gaussian, Student's t, exponential, gamma, beta, Pareto, ...) and estimate the parameters. Maximum likelihood estimates are generally consistent and efficient for such problems. Chapter 14 reminded us of how this machinery can be extended to multivariate data. But suppose you don't have any particular parametric density family in mind, or want to check one — how could we estimate a probability distribution non-parametrically?

15.1 Histograms Revisited

For most of you, making a histogram was probably one of the first things you learned how to do in intro stats (if not before). This is a simple way of estimating a distribution: we split the sample space up into bins, count how many samples fall into each bin, and then divide the counts by the total number of samples. If we hold the bins fixed and take more and more data, then by the law of large numbers we anticipate that the relative frequency for each bin will converge on the bin's probability.

So far so good. But one of the things you learned in intro stats was also to work with probability *density* functions, not just probability *mass* functions. Where do we get pdfs? Well, one thing we could do is to take our histogram estimate, and then say

¹When you find out what the kurtosis is good for, be sure to tell the world.

that the probability density is uniform within each bin. This gives us a piecewise-constant estimate of the density.

Unfortunately, this isn’t going to work — isn’t going to converge on the true pdf — unless we can shrink the bins of the histogram as we get more and more data. To see this, think about estimating the pdf when the data comes from any of the standard distributions, like an exponential or a Gaussian. We can approximate the true pdf $f(x)$ to arbitrary accuracy by a piecewise-constant density (indeed, that’s what happens every time we plot it on our screens), but, for a fixed set of bins, we can only come so close to the true, continuous density.

This reminds us of our old friend the bias-variance trade-off, and rightly so. If we use a large number of very small bins, the minimum bias in our estimate of any density becomes small, but the variance in our estimates grows. (Why does variance increase?) To make some use of this insight, though, there are some things we need to establish first.

- Is learning the whole distribution non-parametrically even feasible?
- How can we measure error so deal with the bias-variance trade-off?

15.2 “The Fundamental Theorem of Statistics”

Let’s deal with the first point first. In principle, something even dumber than shrinking histograms will work to learn the whole distribution. Suppose we have one-dimensional samples x_1, x_2, \dots, x_n with a common cumulative distribution function F . The **empirical cumulative distribution function** on n samples, $\tilde{F}_n(a)$ is

$$\tilde{F}_n(a) \equiv \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{(-\infty, a]}(x_i) \quad (15.1)$$

In words, this is just the fraction of the samples which are $\leq a$. Then the **Glivenko-Cantelli theorem** says

$$\max_a |\tilde{F}_n(a) - F(a)| \rightarrow 0 \quad (15.2)$$

So the empirical CDF converges to the true CDF everywhere; the maximum gap between the two of them goes to zero. Pitman (1979) calls this the “fundamental theorem of statistics”, because it says we can learn distributions just by collecting enough data.² The same kind of result also holds for higher-dimensional vectors.

²Note that for any one, fixed value of a , that $|\tilde{F}_n(a) - F(a)| \rightarrow 0$ is just an application of the law of large numbers. The extra work Glivenko and Cantelli did was to show that this held for *infinitely many* values of a at once, so that even if we focus on the biggest gap between the estimate and the truth, that still shrinks with n . We won’t go into the details, but here’s the basic idea. Fix an $\epsilon > 0$; first show that there is some *finite* set of points on the line, call them b_1, \dots, b_q , such that $|\tilde{F}_n(a) - \tilde{F}_n(b_i)| < \epsilon$ and $|F(a) - F(b_i)| < \epsilon$ for *some* b_i . Next, show that, for large enough n , $|F(b_i) - \tilde{F}_n(b_i)| < \epsilon$ for all the b_i . (This follows from the law of large numbers and the fact that q is finite.) Finally, use the triangle inequality to conclude that, for large enough n , $|\tilde{F}_n(a) - F(a)| < 3\epsilon$. Since ϵ can be made arbitrarily small, the Glivenko-Cantelli theorem follows. (Yes, there are some details I’m glossing over.) This general strategy — combining pointwise

If the Glivenko-Cantelli theorem is so great, why aren't we just content with the empirical CDF? Sometimes we are, but it inconveniently doesn't give us a probability *density*. Suppose that x_1, x_2, \dots, x_n are sorted into increasing order. What probability does the empirical CDF put on the interval (x_i, x_{i+1}) ? Clearly, zero. (Whereas the interval $[x_i, x_{i+1}]$ gets probability $2/n$.) This *could* be right, but we have centuries of experience now with probability distributions, and this tells us that *pretty often* we can expect to find some new samples between our old ones. So we'd like to get a *non-zero* density between our observations.

Using a uniform distribution within each bin of a histogram doesn't have this issue, but it does leave us with the problem of picking where the bins go and how many of them we should use. Of course, there's nothing magic about keeping the bin size the same and letting the number of points in the bins vary; we could equally well pick bins so they had equal counts.³ So what should we do?

15.3 Error for Density Estimates

Our first step is to get clear on what we mean by a "good" density estimate. There are three leading ideas:

1. $\int (f(x) - \hat{f}(x))^2 dx$ should be small: the squared deviation from the true density should be small, averaging evenly over all space.
2. $\int |f(x) - \hat{f}(x)| dx$ should be small: minimize the average *absolute*, rather than squared, deviation.
3. $\int f(x) \log \frac{\hat{f}(x)}{f(x)} dx$ should be small: the average log-likelihood ratio should be kept low.

Option (1) is reminiscent of the MSE criterion we've used in regression. Option (2) looks at what's called the L_1 or **total variation** distance between the true and the estimated density. It has the nice property that $\frac{1}{2} \int |f(x) - \hat{f}(x)| dx$ is exactly the maximum error in our estimate of the probability of *any* set. Unfortunately it's a bit tricky to work with, so we'll skip it here. (But see Devroye and Lugosi (2001)). Finally, minimizing the log-likelihood ratio is intimately connected to maximizing

convergence theorems with approximation arguments — forms the core of what's called **empirical process theory**, which underlies the consistency of basically all the non-parametric procedures we've seen. If this line of thought is at all intriguing, the closest thing to a gentle introduction is Pollard (1989).

³A specific idea for how to do this is sometimes called a $k-d$ tree. We have d random variables and want a joint density for all of them. Fix an ordering of the variables. Start with the first variable, and find the thresholds which divide it into k parts with equal counts. (Usually but not always $k = 2$.) Then sub-divide each part into k equal-count parts on the *second* variable, then sub-divide each of those on the third variable, etc. After splitting on the d^{th} variable, go back to splitting on the first, until no further splits are possible. With n data points, it takes about $\log_k n$ splits before coming down to individual data points. Each of these will occupy a cell of some volume. Estimate the density on that cell as one over that volume. Of course it's not strictly necessary to keep refining all the way down to single points.

the likelihood. We will come back to this (§15.6), but, like most texts on density estimation, we will give more attention to minimizing (1), because it's mathematically tractable.

Notice that

$$\int (f(x) - \hat{f}(x))^2 dx = \int f^2(x) dx - 2 \int \hat{f}(x)f(x)dx + \int \hat{f}^2(x)dx \quad (15.3)$$

The first term on the right hand side doesn't depend on the estimate $\hat{f}(x)$ at all, so we can ignore it for purposes of optimization. The third one only involves \hat{f} , and is just an integral, which we can do numerically. That leaves the middle term, which involves both the true and the estimated density; we can approximate it by

$$-\frac{2}{n} \sum_{i=1}^n \hat{f}(x_i) \quad (15.4)$$

The reason we can do this is that, by the Glivenko-Cantelli theorem, integrals over the true density are approximately equal to sums over the empirical distribution.

So our final error measure is

$$-\frac{2}{n} \sum_{i=1}^n \hat{f}(x_i) + \int \hat{f}^2(x)dx \quad (15.5)$$

In fact, this error measure does *not* depend on having one-dimension data; we can use it in any number of dimensions.⁴ For purposes of cross-validation (you knew that was coming, right?), we can estimate \hat{f} on the training set, and then restrict the sum to points in the testing set.

15.3.1 Error Analysis for Histogram Density Estimates

We now have the tools to do most of the analysis of histogram density estimation. (We'll do it in one dimension for simplicity.) Choose our favorite location x , which lies in a bin whose boundaries are x_0 and $x_0 + h$. We want to estimate the density at x , and this is

$$\hat{f}_n(x) = \frac{1}{h} \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{(x_0, x_0+h]}(x_i) \quad (15.6)$$

Let's call the sum, the number of points in the bin, b . It's a random quantity, $B \sim \text{Binomial}(n, p)$, where p is the true probability of falling into the bin, $p = F(x_0 + h) - F(x_0)$. The mean of B is np , and the variance is $np(1-p)$, so

$$\mathbb{E}[\hat{f}_n(x)] = \frac{1}{nh} \mathbb{E}[B] \quad (15.7)$$

$$= \frac{n[F(x_0 + h) - F(x_0)]}{nh} \quad (15.8)$$

$$= \frac{F(x_0 + h) - F(x_0)}{h} \quad (15.9)$$

⁴Admittedly, in high-dimensional spaces, doing the final integral can become numerically challenging.

and the variance is

$$\text{Var}[\hat{f}_n(x)] = \frac{1}{n^2 b^2} \text{Var}[B] \quad (15.10)$$

$$= \frac{n[F(x_0 + h) - F(x_0)][1 - F(x_0 + h) + F(x_0)]}{n^2 b^2} \quad (15.11)$$

$$= E[\hat{f}_n(x)] \frac{1 - F(x_0 + h) + F(x_0)}{nh} \quad (15.12)$$

If we let $b \rightarrow 0$ as $n \rightarrow \infty$, then

$$E[\hat{f}_b(x)] \rightarrow \lim_{b \rightarrow 0} \frac{F(x_0 + b) - F(x_0)}{b} = f(x_0) \quad (15.13)$$

since the pdf is the derivative of the CDF. But since x is between x_0 and $x_0 + h$, $f(x_0) \rightarrow f(x)$. So if we use smaller and smaller bins as we get more data, the histogram density estimate is unbiased. We'd also like its variance to shrink as the same grows. Since $1 - F(x_0 + h) + F(x_0) \rightarrow 1$ as $h \rightarrow 0$, to get the variance to go away we need $nh \rightarrow \infty$.

To put this together, then, our first conclusion is that histogram density estimates will be consistent when $b \rightarrow 0$ but $nh \rightarrow \infty$ as $n \rightarrow \infty$. The bin-width b needs to shrink, but slower than n^{-1} .

At what rate should it shrink? Small b gives us low bias but (as you can verify from the algebra above) high variance, so we want to find the trade-off between the two. One can calculate the bias at x from our formula for $E[\hat{f}_b(x)]$ through a somewhat lengthy calculus exercise, analogous to what we did for kernel smoothing in Chapter 4⁵; the upshot is that the integrated squared bias is

$$\int (f(x) - E[\hat{f}_b(x)])^2 dx = \frac{b^2}{12} \int (f'(x))^2 dx + o(b^2) \quad (15.14)$$

We already got the variance at x , and when we integrate that over x we find

$$\int \text{Var}[\hat{f}_b(x)] dx = \frac{1}{nb} + o(n^{-1}) \quad (15.15)$$

So the total integrated squared error is

$$\text{ISE} = \frac{b^2}{12} \int (f'(x))^2 dx + \frac{1}{nb} + o(b^2) + o(n^{-1}) \quad (15.16)$$

Differentiating this with respect to b and setting it equal to zero, we get

$$\frac{b_{\text{opt}}}{6} \int (f'(x))^2 dx = \frac{1}{nb_{\text{opt}}^2} \quad (15.17)$$

⁵You need to use the intermediate value theorem multiple times; see for instance Wasserman (2006, sec. 6.8).

$$h_{\text{opt}} = \left(\frac{6}{\int (f'(x))^2 dx} \right)^{1/3} n^{-1/3} = O(n^{-1/3}) \quad (15.18)$$

So we need narrow bins if the density changes rapidly ($\int (f'(x))^2 dx$ is large), and wide bins if the density is relatively flat. No matter how rough the density, the bin width should shrink like $O(n^{-1/3})$. Plugging that rate back into the equation for the ISE, we see that it is $O(n^{-2/3})$.

It turns out that if we pick h by cross-validation, then we attain this optimal rate in the large-sample limit. By contrast, if we *knew* the correct parametric form and just had to estimate the parameters, we'd typically get an error decay of $O(n^{-1})$. This is substantially faster than histograms, so it would be nice if we could make up some of the gap, without having to rely on parametric assumptions.

15.4 Kernel Density Estimates

It turns out that one can improve the convergence rate, as well as getting smoother estimates, but using kernels. The **kernel density estimate** is

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^n \frac{1}{b} K\left(\frac{x - x_i}{b}\right) \quad (15.19)$$

where K is a kernel function such as we encountered when looking at kernel regression. (The factor of $1/b$ inside the sum is so that \hat{f}_h will integrate to 1; we could have included it in both the numerator and denominator of the kernel regression formulae, but then it would've just canceled out.) As before, h is the **bandwidth** of the kernel. We've seen typical kernels in things like the Gaussian. One advantage of using them is that they give us a smooth density everywhere, unlike histograms, and in fact we can even use them to estimate the derivatives of the density, should that be necessary.⁶

15.4.1 Analysis of Kernel Density Estimates

How do we know that kernels will in fact work? Well, let's look at the mean and variance of the kernel density estimate at a particular point x , and use Taylor's theorem

⁶The advantage of histograms is that they're computationally and mathematically simpler.

on the density.

$$\mathbb{E} \left[\hat{f}_b(x) \right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E} \left[\frac{1}{b} K \left(\frac{x - X_i}{b} \right) \right] \quad (15.20)$$

$$= \mathbb{E} \left[\frac{1}{b} K \left(\frac{x - X}{b} \right) \right] \quad (15.21)$$

$$= \int \frac{1}{b} K \left(\frac{x - t}{b} \right) f(t) dt \quad (15.22)$$

$$= \int K(u) f(x - bu) du \quad (15.23)$$

$$= \int K(u) \left[f(x) - bu f'(x) + \frac{b^2 u^2}{2} f''(x) + o(b^2) \right] du \quad (15.24)$$

$$= f(x) + \frac{b^2 f''(x)}{2} \int K(u) u^2 du + o(b^2) \quad (15.25)$$

$$(15.26)$$

because, by definition, $\int K(u) du = 1$ and $\int u K(u) du = 0$. If we call $\int K(u) u^2 du = \sigma_K^2$, then the bias of the kernel density estimate is

$$\mathbb{E} \left[\hat{f}_b(x) \right] - f(x) = \frac{b^2 \sigma_K^2 f''(x)}{2} + o(b^2) \quad (15.27)$$

So the bias will go to zero if the bandwidth b shrinks to zero. What about the variance? Use Taylor's theorem again:

$$\text{Var} \left[\hat{f}_b(x) \right] = \frac{1}{n} \text{Var} \left[\frac{1}{b} K \left(\frac{x - X}{b} \right) \right] \quad (15.28)$$

$$= \frac{1}{n} \left[\mathbb{E} \left[\frac{1}{b^2} K^2 \left(\frac{x - X}{b} \right) \right] - \left(\mathbb{E} \left[\frac{1}{b} K \left(\frac{x - X}{b} \right) \right] \right)^2 \right] \quad (15.29)$$

$$= \frac{1}{n} \left[\int \frac{1}{b^2} K^2 \left(\frac{x - t}{b} \right) dt - [f(x) + O(b^2)]^2 \right] \quad (15.30)$$

$$= \frac{1}{n} \left[\int \frac{1}{b} K^2(u) f(x - bu) du - f^2(x) + O(b^2) \right] \quad (15.31)$$

$$= \frac{1}{n} \left[\int \frac{1}{b} K^2(u) (f(x) - bu f'(x)) du - f^2(x) + O(b) \right] \quad (15.32)$$

$$= \frac{f(x)}{hn} \int K^2(u) du + O(1/n) \quad (15.33)$$

This will go to zero if $nb \rightarrow \infty$ as $n \rightarrow \infty$. So the conclusion is the same as for histograms: b has to go to zero, but slower than $1/n$.

Since the expected squared error at x is the bias squared plus the variance,

$$\frac{b^4 \sigma_K^4 (f''(x))^2}{4} + \frac{f(x)}{hn} \int K^2(u) du + \text{small} \quad (15.34)$$

the expected integrated squared error is

$$\text{ISE} \approx \frac{b^4 \sigma_K^4}{4} \int (f''(x))^2 dx + \frac{\int K^2(u) du}{nb} \quad (15.35)$$

Differentiating with respect to b for the optimal bandwidth b_{opt} , we find

$$b_{\text{opt}}^3 \sigma_K^4 \int (f''(x))^2 dx = \frac{\int K^2(u) du}{nb_{\text{opt}}^2} \quad (15.36)$$

$$b_{\text{opt}} = \left(\frac{\int K^2(u) du}{\sigma_K^4 \int (f''(x))^2 dx} \right)^{1/5} n^{-1/5} = O(n^{-1/5}) \quad (15.37)$$

That is, the best bandwidth goes to zero like one over the fifth root of the number of sample points. Plugging this into Eq. 15.35, the best $\text{ISE} = O(n^{-4/5})$. This is better than the $O(n^{-2/3})$ rate of histograms, but still includes a penalty for having to figure out what kind of distribution we're dealing with. Remarkably enough, using cross-validation to pick the bandwidth gives near-optimal results.⁷

As an alternative to cross-validation, or at least a starting point, one can use Eq. 15.37 to show that the optimal bandwidth for using a Gaussian kernel to estimate a Gaussian distribution is $1.06\sigma n^{-1/5}$, with σ being the standard deviation of the Gaussian. This is sometimes called the **Gaussian reference rule** or the **rule-of-thumb** bandwidth. When you call `density` in R, this is basically what it does.

Yet another technique is the **plug-in method**. Eq. 15.37 calculates the optimal bandwidth from the second derivative of the true density. This doesn't help if we don't know the density, but it becomes useful if we have an initial density estimate which isn't too bad. In the plug-in method, we start with an initial bandwidth (say from the Gaussian reference rule) and use it to get a preliminary estimate of the density. Taking that crude estimate and "plugging it in" to Eq. 15.37 gives us a new bandwidth, and we re-do the kernel estimate with that new bandwidth. Iterating this a few times is optional but not uncommon.

15.4.2 Joint Density Estimates

The discussion and analysis so far has been focused on estimating the distribution of a one-dimensional variable. Just as kernel regression can be done with multiple input variables (§4.3), we can make kernel density estimates of joint distributions. We simply need a kernel for the vector:

$$\hat{f}(\vec{x}) = \frac{1}{n} \sum_{i=1}^n K(\vec{x} - \vec{x}_i) \quad (15.38)$$

⁷Substituting Eq. 15.37 into Eq. 15.35 gives a squared error of $1.25n^{-4/5}\sigma_K^{4/5}(\int (f''(x))^2 dx)^{1/5}(\int K^2(u) du)^{4/5}$. The only two parts of this which depend on the kernel are σ_K and $\int K^2(u) du$. This is the source of the (correct) folklore that the choice of kernel is less important than the choice of bandwidth.

One could use any multivariate distribution as the kernel (provided it is centered and has finite covariance). Typically, however, just as in smoothing, one uses a product kernel, i.e., a product of one-dimensional kernels,

$$K(\vec{x} - \vec{x}_i) = K_1(x^1 - x_i^1)K_2(x^2 - x_i^2)\dots K_d(x^d - x_i^d), \quad (15.39)$$

Doing this requires a bandwidth for each coordinate, so the over-all form of the joint PDF estimate is

$$\hat{f}(\vec{x}) = \frac{1}{n \prod_{j=1}^d b_j} \sum_{i=1}^n \prod_{j=1}^d K_j \left(\frac{x^j - x_i^j}{b_j} \right) \quad (15.40)$$

Going through a similar analysis for d -dimensional data shows that the ISE goes to zero like $O(n^{-4/(4+d)})$, and again, if we use cross-validation to pick the bandwidths, asymptotically we attain this rate. Unfortunately, if d is large, this rate becomes very slow — for instance, if $d = 24$, the rate is $O(n^{-1/7})$. There is simply no universally good way to figure out high-dimensional distributions from scratch; either we make strong parametric assumptions, which could be badly wrong, or we accept a potentially very slow convergence.

15.4.3 Categorical and Ordered Variables

Estimating probability mass functions with discrete variables can be straightforward: there are only a finite number of values, and so one just counts how often they occur and takes the relative frequency. If one has a discrete variable X and a continuous variable Y and one wants a joint distribution, one could just get a separate density for Y for each value of x , and tabulate the probabilities for x .

In principle, this will work, but it can be practically awkward if the number of levels for the discrete variable is large compared to the number of samples. Moreover, for the joint distribution problem, it has us estimating completely separate distributions for Y for every x , without any sharing of information between them. It would seem more plausible to smooth those distributions towards each others. To do this, we need kernels for discrete variables.

Several sets of such kernels have been proposed. The most straightforward, however, are the following. If X is a categorical, unordered variable with c possible values, then, for $0 \leq b < 1$,

$$K(x_1, x_2) = \begin{cases} 1-b & x_1 = x_2 \\ b/c & x \neq x_i \end{cases} \quad (15.41)$$

is a valid kernel. For an ordered x ,

$$K(x_1, x_2) = \binom{c}{|x_1 - x_2|} b^{|x_1 - x_2|} (1-b)^{c - |x_1 - x_2|} \quad (15.42)$$

where $|x_1 - x_2|$ should be understood as just how many levels apart x_1 and x_2 are. As $b \rightarrow 0$, both of these become indicators, and return us to simple relative frequency counting. Both of these are implemented in np.

15.4.4 Practicalities

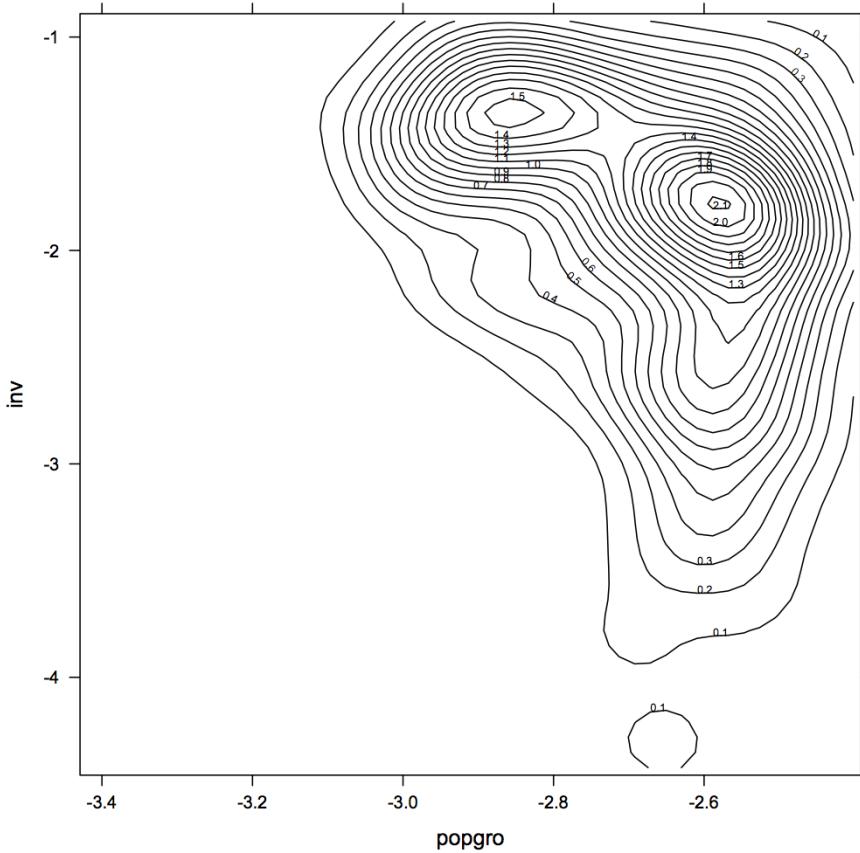
The standard R function `density` implements one-dimensional kernel density estimation, defaulting to Gaussian kernels with the rule-of-thumb bandwidth. There are some options for doing cleverer bandwidth selection, including a plug-in rule. (See the help file.)

For more sophisticated methods, and especially for more dimensions, you'll need to use other packages. The `np` package estimates joint densities using the `npudens` function. (The `u` is for "unconditional".) This has the same sort of automatic bandwidth selection as `npreg`, using cross-validation. Other packages which do kernel density estimation include `KernSmooth` and `sm`.

15.4.5 Kernel Density Estimation in R: An Economic Example

The data set `oecdpanel`, in the `np` library, contains information about much the same sort of variables at the Penn World Tables data you worked with in the homework, over much the same countries and years, but with some of the variables pre-transformed, with identifying country information removed, and slightly different data sources. See `help(oecdpanel)` for details.

Here's an example of using `npudens` with variables from the `oecdpanel` data set [[you saw in the homework]]. We'll look at the joint density of `popgro` (the logarithm of the population growth rate) and `inv` (the logarithm of the investment rate). Figure 15.1 illustrates how to call the command, and a useful trick where we get `np`'s plotting function to do our calculations for us, but then pass the results to a different graphics routine. (See `help(npplot)`.) The distribution we get has two big modes, one at a comparatively low population growth rate (≈ -2.9 — remember this is logged so it's not actually a shrinking population) and high investment (≈ -1.5), and the other at a lower rate of investment (≈ -2) and higher population growth (≈ -2.6). There is a third, much smaller mode at high population growth (≈ -2.7) and very low investment (≈ -4).



```

library(np)
data(oecdpanel)
popinv <- npudens(~popgro+inv, data=oecdpanel)
fhat <- plot(popinv,plot.behavior="data")
fhat <- fhat$d1
library(lattice)
contourplot(fhat$dens~fhat$eval$Var1*fhat$eval$Var2,cuts=20,
            xlab="popgro",ylab="inv",labels=list(cex=0.5))

```

Figure 15.1: Gaussian kernel estimate of the joint distribution of logged population growth rate (popgro) and investment rate (inv). Notice that `npudens` takes a formula, but that there is no dependent variable on the left-hand side of the `~`. With objects produced by the `np` library, one can give the plotting function the argument `plot.behavior` — the default is `plot`, but if it's set to `data` (as here), it calculates all the information needed to plot and returns a separate set of objects, which can be plotted in other functions. (The value `plot-data` does both.) See `help(npplot)` for more.

15.5 Conditional Density Estimation

In addition to estimating marginal and joint densities, we will often want to get conditional densities. The most straightforward way to get the density of Y given X , $f_{Y|X}(y|x)$, is

$$\hat{f}_{Y|X}(y|x) = \frac{\hat{f}_{X,Y}(x,y)}{\hat{f}_X(x)} \quad (15.43)$$

i.e., to estimate the joint and marginal densities and divide one by the other.

To be concrete, let's suppose that we are using a product kernel to estimate the joint density, and that the marginal density is consistent with it:

$$\hat{f}_{X,Y}(x,y) = \frac{1}{nh_X h_Y} \sum_{i=1}^n K_X\left(\frac{x-x_i}{h_X}\right) K_Y\left(\frac{y-y_i}{h_Y}\right) \quad (15.44)$$

$$\hat{f}_X(x) = \frac{1}{nh_X} \sum_{i=1}^n K_X\left(\frac{x-x_i}{h_X}\right) \quad (15.45)$$

Thus we need to pick two bandwidths, h_X and h_Y , one for each variable.

This might seem like a solved problem — we just use cross-validation to find h_X and h_Y so as to minimize the integrated squared error for $\hat{f}_{X,Y}$, and then plug in to Equation 15.43. However, this is a bit hasty, because the optimal bandwidths for the *joint* density are not necessarily the optimal bandwidths for the *conditional* density. An extreme but easy to understand example is when Y is actually independent of X . Since the density of Y given X is just the density of Y , we'd be best off just ignoring X by taking $h_X = \infty$. (In practice, we'd just use a very big bandwidth.) But if we want to find the joint density, we would not want to smooth X away completely like this.

The appropriate integrated squared error measure for the conditional density is

$$\int dx f_X(x) \int dy \left(f_{Y|X}(y|x) - \hat{f}_{Y|X}(y|x) \right)^2 \quad (15.46)$$

and this is what we want to minimize by picking h_X and h_Y . The cross-validation goes as usual.

One nice, and quite remarkable, property of cross-validation for conditional density estimation is that it can detect and exploit conditional independence. Say that $X = (U, V)$, and that Y is independent of U given V — symbolically, $Y \perp\!\!\!\perp U | V$. Then $f_{Y|U,V}(y|u,v) = f_{Y|v}(y|v)$, and we should just ignore U in our estimation of the conditional density. It turns out that when cross-validation is used to pick bandwidths for conditional density estimation, $\hat{h}_U \rightarrow \infty$ when $Y \perp\!\!\!\perp U | V$, but not otherwise (Hall *et al.*, 2004). In other words, cross-validation will automatically detect which variables are irrelevant, and smooth them away.

15.5.1 Practicalities and a Second Example

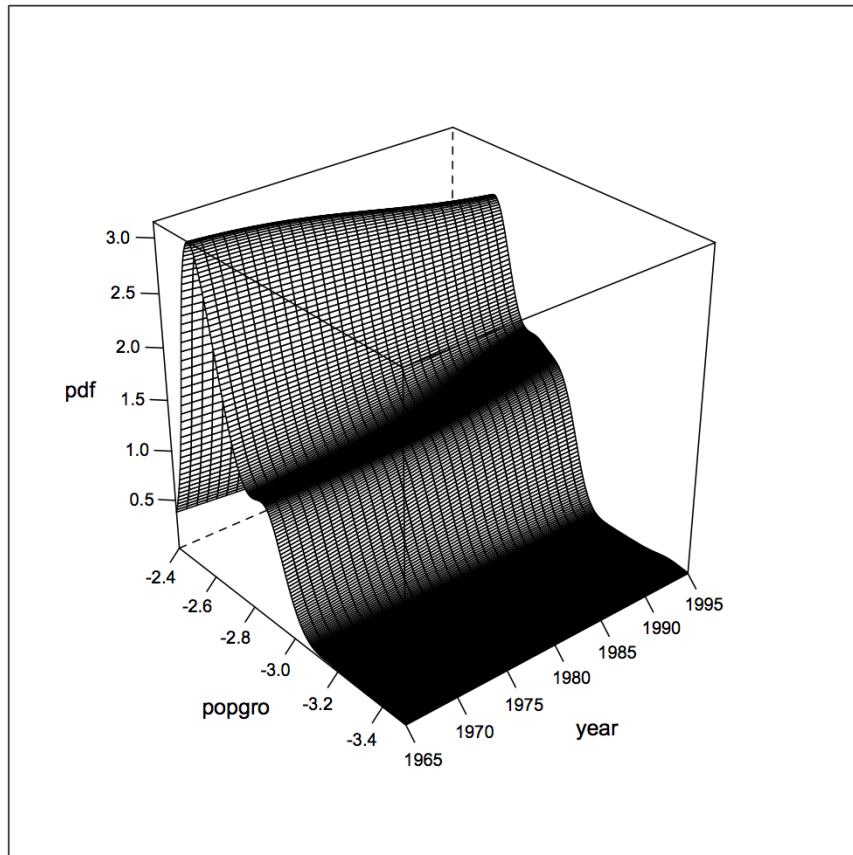
The `np` package implements kernel conditional density estimation through the function `npcdens`. The syntax is pretty much exactly like that of `npreg`, and indeed we can think of estimating the conditional density as a sort of regression, where the dependent variable is actually a distribution.

To give a concrete example, let's look at how the distribution of countries' population growth rates has changed over time, using the `oecdpanel` data (Figure 15.2). The selected bandwidth for `year` is 10, while that for `popgro` is 0.048. (Note that `year` is being treated as a continuous variable.)

You can see from the figure that the mode for population growth rates is towards the high end of observed values, but the mode is shrinking and becoming less pronounced over time. The distribution in fact begins as clearly bimodal, but the smaller mode at the lower growth rate turns into a continuous "shoulder". Overall, Figure 15.2 shows a trend for population growth rates to shrink over time, and for the distribution of growth rates to become less dispersed.

Let's expand on this point. One of the variables in `oecdpanel` is `oeqd`, which is 1 for countries which are members of the Organization for Economic Cooperation and Development, and 0 otherwise. The OECD countries are basically the "developed" ones (stable capitalist democracies). We can include OECD membership as a conditioning variable for population growth (we need to use a categorical-variable kernel), and look at the combined effect of time and development (Figure 15.3).

What the figure shows is that OECD and non-OECD countries both have unimodal distributions of growth rates. The mode for the OECD countries has become sharper, but the value has decreased. The mode for non-OECD countries has also decreased, while the distribution has become more spread out, mostly by having more probability of lower growth rates. (These trends have continued since 1995.) In words, despite the widespread contrary impression, population growth has actually been slowing for decades in both rich and poor countries.

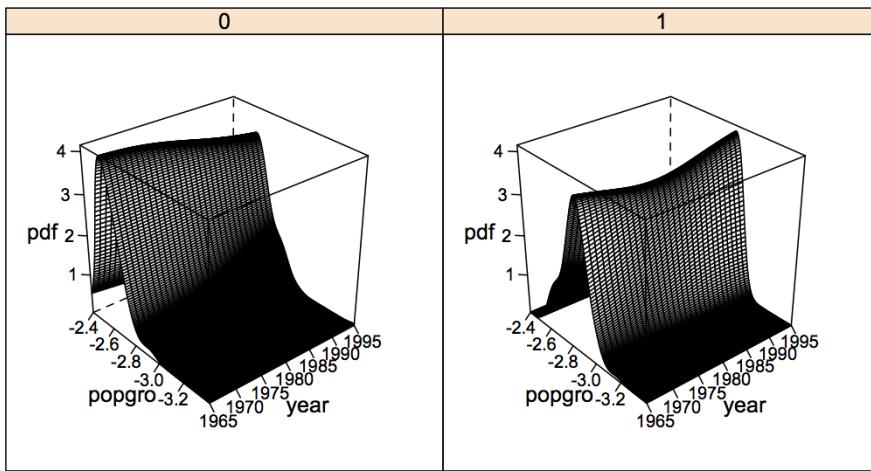


```

pop.cdens <- npcdens(popgro ~ year,data=oecdpanel)
plotting.grid <- expand.grid(year=seq(from=1965,to=1995,by=1) ,
  popgro=seq(from=-3.5,to=-2.4,length.out=300))
fhat <- predict(pop.cdens,newdata=plotting.grid)
wireframe(fhat~plotting.grid$year*plotting.grid$popgro,
  scales=list(arrows=FALSE),xlab="year",ylab="popgro",zlab="pdf")

```

Figure 15.2: Conditional density of logarithmic population growth rates as a function of time.



```

pop.cdens.o <- npcdens(popgro~year+factor(oecd),data=oecdpanel)
oecd.grid <- expand.grid(year=seq(from=1965,to=1995,by=1),
  popgro=seq(from=-3.4,to=-2.4,length.out=300),
  oecd=unique(oecdpanel$oecd))
fhat <- predict(pop.cdens.o,newdata=oecd.grid)
wireframe(fhat~oecd.grid$year*oecd.grid$popgro|oecd.grid$oecd,
  scales=list(arrows=FALSE),xlab="year",ylab="popgro",zlab="pdf")

```

Figure 15.3: Conditional density of population growth rates given year and OECD membership. The left panel is countries not in the OECD, the right is ones which are.

15.6 More on the Expected Log-Likelihood Ratio

I want to say just a bit more about the expected log-likelihood ratio $\int f(x) \log \frac{f(x)}{\hat{f}(x)} dx$. More formally, this is called the **Kullback-Leibler divergence** or **relative entropy** of \hat{f} from f , and is also written $D(f\| \hat{f})$. Let's expand the log ratio:

$$D(f\| \hat{f}) = - \int f(x) \log \hat{f}(x) dx + \int f(x) \log f(x) dx \quad (15.47)$$

The second term does not involve the density estimate, so it's irrelevant for purposes of optimizing over \hat{f} . (In fact, we're just subtracting off the entropy of the true density.) Just as with the squared error, we could try approximating the integral with a sum:

$$\int f(x) \log \hat{f}(x) dx \approx \frac{1}{n} \sum_{i=1}^n \log \hat{f}(x_i) \quad (15.48)$$

which is just the log-likelihood per observation. Since we know and like maximum likelihood methods, why not just use this?

Well, let's think about what's going to happen if we plug in the kernel density estimate:

$$\frac{1}{n} \sum_{i=1}^n \log \left(\frac{1}{nb} \sum_{j=1}^n K\left(\frac{x_j - x_i}{b}\right) \right) = -\log nb + \frac{1}{n} \sum_{i=1}^n \log \left(\sum_{j=1}^n K\left(\frac{x_j - x_i}{b}\right) \right) \quad (15.49)$$

If we take b to be very small, $K(\frac{x_j - x_i}{b}) \approx 0$ unless $x_j = x_i$, so the over-all likelihood becomes

$$\approx -\log nb + \log K(0) \quad (15.50)$$

which goes to $+\infty$ as $b \rightarrow 0$. So if we want to maximize the likelihood of a kernel density estimate, we *always* want to make the bandwidth as small as possible. In fact, the limit is to say that the density is

$$\tilde{f}(x) = \frac{1}{n} \sum_{i=1}^n \delta(x - x_i) \quad (15.51)$$

where δ is the Dirac delta function.⁸ Of course this is just what we'd get if we took the empirical CDF "raw".

What's gone wrong here? Why is maximum likelihood failing us? Well, it's doing exactly what we asked it to: to find the distribution where the observed sample is as probable as possible. Giving any probability to *un*-observed values can only come

⁸Recall that the delta function is defined by how it integrates with other functions: $\int \delta(x)f(x)dx = f(0)$. You can imagine $\delta(x)$ as zero everywhere except at the origin, where it has an infinitely tall, infinitely narrow spike, the area under the spike being one. If you are suspicious that this is really a valid function, you're right; strictly speaking it's just a linear operator on actual functions. We can however approximate it as the limit of well-behaved functions. For instance, take $\delta_b(x) = 1/b$ when $x \in [-b/2, b/2]$ with $\delta_b(x) = 0$ elsewhere, and let b go to zero. This is, of course, where we came in.

at the expense of the probability of observed values, so Eq. 15.51 really is the unrestricted maximum likelihood estimate of the distribution. Anything else imposes some restrictions or constraints which don't, strictly speaking, come from the data. However, those restrictions are what let us generalize to new data, rather than just memorizing the training sample.

One way out of this is to use the *cross-validated* log-likelihood to pick a bandwidth, i.e., to restrict the sum in Eq. 15.48 to running over the testing set only. This way, very small bandwidths don't get an unfair advantage for concentrating around the training set. (If the test points are in fact all very close to the training points, then small bandwidths get a *fair* advantage.) This is in fact the default procedure in the `np` package, through the `bwmETHOD` option ("cv.ml" vs. "cv.ls").

15.7 Simulating from Density Estimates

15.7.1 Simulating from Kernel Density Estimates

There are times when one wants to draw a random sample from the estimated distribution. This is easy with kernel density estimates, because each kernel is itself a probability density, generally a very tractable one. The general pattern goes as follows. Suppose the kernel is Gaussian, that we have scalar observations x_1, x_2, \dots, x_n , and the selected bandwidth is h . Then we pick an integer i uniformly at random from 1 to n , and invoke `rnorm(1, x[i], h)`.⁹ Using a different kernel, we'd just need to use the random number generator function for the corresponding distribution.

We can see that this works, i.e., that it gives the right distribution, with just a little math. A kernel $K(x, x_i, h)$ with bandwidth h and center x_i is a probability density function. The probability the density estimate gives to any set A is just an integral:

$$\hat{F}(A) = \int_A \hat{f}(x) dx \quad (15.52)$$

$$= \int_A \frac{1}{n} \sum_{i=1}^n K(x, x_i, h) dx \quad (15.53)$$

$$= \frac{1}{n} \sum_{i=1}^n \int_A K(x, x_i, h) dx \quad (15.54)$$

$$= \frac{1}{n} \sum_{i=1}^n C(A, x_i, h) \quad (15.55)$$

introducing C to stand for the probability distribution corresponding to the kernel. The simulation procedure works if the probability that the simulated value \tilde{X} falls into A matches this. To generate \tilde{X} , we first pick a random data point, which really

⁹In fact, if we want to draw a sample of size q , `rnorm(q, sample(x, q, replace=TRUE), h)` will work in R — it's important though that sampling be done *with* replacement.

means picking a random integer J , uniformly from 1 to n . Then

$$\Pr(\tilde{X} \in A) = E[\mathbf{1}_A(\tilde{X})] \quad (15.56)$$

$$= E[E[\mathbf{1}_A(\tilde{X})|J]] \quad (15.57)$$

$$= E[C(A, x_J, b)] \quad (15.58)$$

$$= \frac{1}{n} \sum_{i=1}^n C(A, x_i, b) \quad (15.59)$$

The first step uses the fact that a probability is the expectation of an indicator function; the second uses the law of total expectation; the last steps us the definitions of C and J , and the distribution of J .

15.7.1.1 Sampling from a Kernel Joint Density Estimate

The procedure given above works with only trivial modification for sampling from a joint, multivariate distribution. If we're using a product kernel, we pick a random data point, and then sample each coordinate independently. The argument for correctness actually goes exactly as before.

15.7.1.2 Sampling from Kernel Conditional Density Estimates

Sampling from a conditional density estimate with product kernels is again straightforward. The one trick is that one needs to do a *weighted* sample of data points. To see why, look at the conditional distribution (not density) function:

$$\hat{F}(Y \in A | X = x) \quad (15.60)$$

$$= \int_A \hat{f}_{Y|X}(y|x) dy \\ = \int_A \frac{\frac{1}{nb_X b_Y} \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) K_Y\left(\frac{y-y_i}{b_Y}\right)}{\hat{f}_X(x)} dy \quad (15.61)$$

$$= \frac{1}{nb_X b_Y \hat{f}_X(x)} \int_A \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) K_Y\left(\frac{y-y_i}{b_Y}\right) dy \quad (15.62)$$

$$= \frac{1}{nb_X b_Y \hat{f}_X(x)} \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) \int_A K_Y\left(\frac{y-y_i}{b_Y}\right) dy \quad (15.63)$$

$$= \frac{1}{nb_X \hat{f}_X(x)} \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) C_Y(A, y_i, b_Y) \quad (15.64)$$

If we select the data point i with a weight proportional to $K_X\left(\frac{x-x_i}{b_X}\right)$, and then generate \tilde{Y} from the K_Y distribution centered at y_i , then, \tilde{Y} will follow the appropriate probability density function.

15.7.2 Sampling from Histogram Estimates

Sampling from a histogram estimate is also simple, but in a sense goes in the opposite order from kernel simulation. We first randomly pick a bin by drawing from a multinomial distribution, with weights proportional to the bin counts. Once we have a bin, we draw from a uniform distribution over its range.

15.7.3 Examples of Simulating from Kernel Density Estimates

To make all this more concrete, let's continue working with the `oecdpanel` data. Section 15.4.5 shows the joint pdf estimate for the variables `popgro` and `inv` in that data set. These are the logarithms of the population growth rate and investment rate. Undoing the logarithms and taking the density,

```
popinv2 <- npudens(~exp(popgro)+exp(inv), data=oecdpanel)
```

gives Figure 15.4.

Let's abbreviate the actual (not logged) population growth rate as X and the actual (not logged) investment rate as Y in what follows.

Since this is a joint distribution, it implies a certain expected value for Y/X , the ratio of investment rate to population growth rate¹⁰. Extracting this by direct calculation from `popinv2` would not be easy; we'd need to do the integral

$$\int_{x=0}^1 \int_{y=0}^1 \frac{y}{x} \hat{f}_{X,Y}(x,y) dy dx \quad (15.65)$$

To find $E[Y/X]$ by simulation, however, we just need to generate samples from the joint distribution, say $(\tilde{X}_1, \tilde{Y}_1), (\tilde{X}_2, \tilde{Y}_2), \dots, (\tilde{X}_T, \tilde{Y}_T)$, and average:

$$\frac{1}{T} \sum_{i=1}^T \frac{\tilde{Y}_i}{\tilde{X}_i} = \tilde{g}_T \xrightarrow{T \rightarrow \infty} E\left[\frac{Y}{X}\right] \quad (15.66)$$

where the convergence happens because that's the law of large numbers. If the number of simulation points T is big, then $\tilde{g}_T \approx E[Y/X]$. How big do we need to make T ? Use the central limit theorem:

$$\tilde{g}_T \rightsquigarrow \mathcal{N}(E[Y/X], \text{Var}[\tilde{g}_1]/\sqrt{T}) \quad (15.67)$$

How do we find the variance $\text{Var}[\tilde{g}_1]$? We approximate it by simulating.

Code Example 31 is a function which draws a sample from the fitted kernel density estimate. First let's check that it works, by giving it something easy to do, namely reproducing the means, which we *can* work out:

```
> mean(exp(oecdpanel$popgro))
[1] 0.06930789
```

¹⁰Economically, we might want to know this because it would tell us about how quickly the capital stock per person grows.

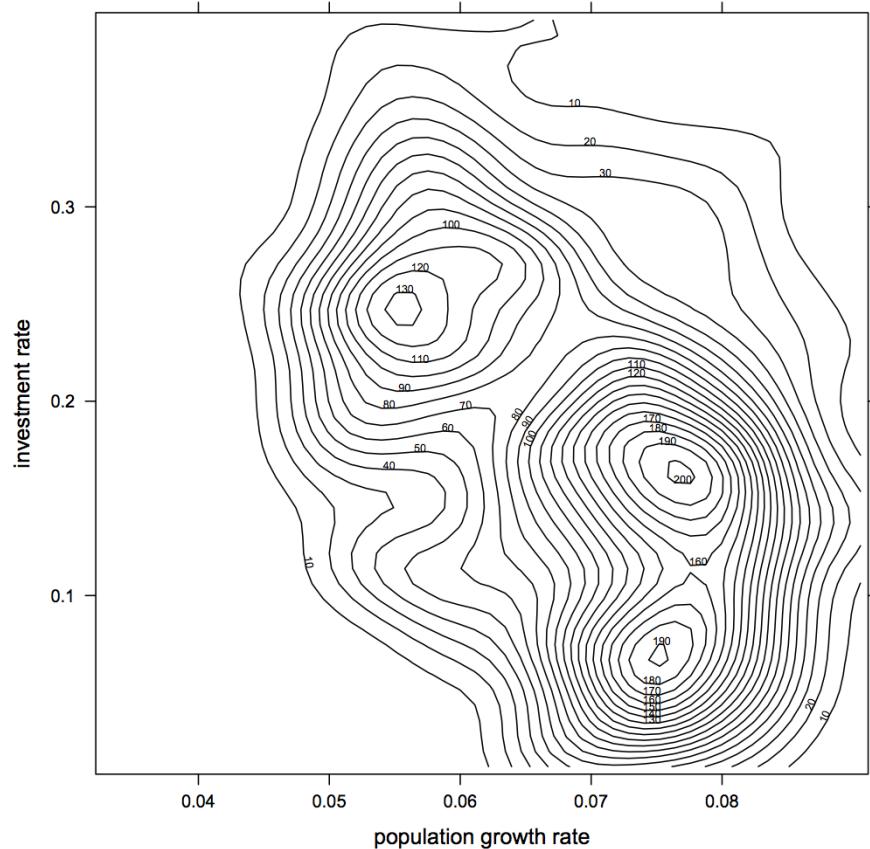


Figure 15.4: Gaussian kernel density estimate for the un-logged population growth rate and investment rate.

```
rpopinv <- function(n) {
  n.train <- length(popinv2$dens)
  ndim <- popinv2$ndim
  points <- sample(1:n.train, size=n, replace=TRUE)
  z <- matrix(0, nrow=n, ncol=ndim)
  for (i in 1:ndim) {
    coordinates <- popinv2$eval[points, i]
    z[, i] <- rnorm(n, coordinates, popinv2$bw[i])
  }
  colnames(z) <- c("pop.growth.rate", "invest.rate")
  return(z)
}
```

Code Example 31: Sampling from the fitted kernel density estimate `popinv2`. Can you see how to modify it to sample from other bivariate density estimates produced by `npudens`? From higher-dimensional distributions? Can you replace the `for` loop with less iterative code?

```
> mean(exp(oecdpanel$inv))
[1] 0.1716247
> colMeans(rpopinv(200))
pop.growth.rate      invest.rate
0.06865678          0.17623612
```

This is pretty satisfactory for only 200 samples, so the simulator seems to be working. Now we just use it:

```
> z <- rpopinv(2000)
> mean(z[, "invest.rate"] / z[, "pop.growth.rate"])
[1] 2.597916
> sd(z[, "invest.rate"] / z[, "pop.growth.rate"]) / sqrt(2000)
[1] 0.0348991
```

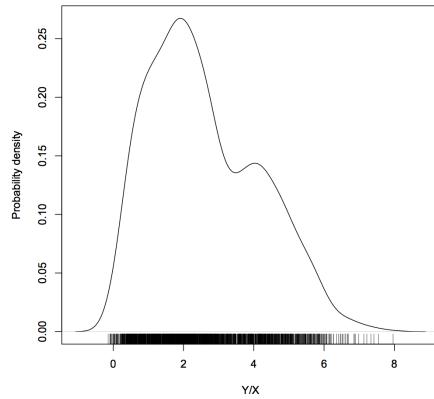
So this tells us that $E[Y/X] \approx 2.59$, with a standard error of ± 0.035 .

Suppose we want not the mean of Y/X but the median?

```
> median(z[, "invest.rate"] / z[, "pop.growth.rate"])
[1] 2.31548
```

Getting the whole distribution of Y/X is not much harder (Figure 15.5). Of course complicated things like distributions converge more slowly than simple things like means or medians, so we want might want to use more than 2000 simulated values for the distribution. Alternately, we could repeat the simulation many times, and look at how much variation there is from one realization to the next (Figure 15.6).

Of course, if we are going to do multiple simulations, we could just average them together. Say that $\hat{g}_T^{(1)}, \hat{g}_T^{(2)}, \dots, \hat{g}_T^{(s)}$ are estimates of our statistic of interest from s



```

YoverX <- z[, "invest.rate"]/z[, "pop.growth.rate"]
plot(density(YoverX), xlab="Y/X", ylab="Probability density", main="")
rug(YoverX, side=1)

```

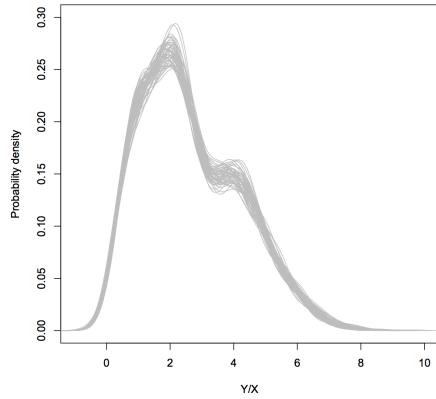
Figure 15.5: Distribution of Y/X implied by the joint density estimate `popinv2`.

independent realizations of the model, each of size T . We can just combine them into one grand average:

$$\tilde{g}_{s,T} = \frac{1}{s} \sum_{i=1}^s \tilde{g}_T^{(i)} \quad (15.68)$$

As an average of IID quantities, the variance of $\tilde{g}_{s,T}$ is $1/s$ times the variance of $\tilde{g}_T^{(1)}$.

By this point, we are getting the sampling distribution of the density of a nonlinear transformation of the variables in our model, with no more effort than calculating a mean.



```

plot(0,xlab="Y/X",ylab="Probability density",type="n",xlim=c(-1,10),
      ylim=c(0,0.3))
one.plot <- function() {
  zprime <- rpopinv(2000)
  YoverXprime <- zprime[, "invest.rate"]/zprime[, "pop.growth.rate"]
  density.prime <- density(YoverXprime)
  lines(density.prime,col="grey")
}
invisible(replicate(50,one.plot()))

```

Figure 15.6: Showing the sampling variability in the distribution of Y/X by “overplotting”. Each line is a distribution from an estimated sample of size 2000, as in Figure 15.5; here 50 of them are plotted on top of each other. The thickness of the bands indicates how much variation there is from simulation to simulation at any given value of Y/X . (Setting the type of the initial `plot` to `n`, for “null”, creates the plotting window, axes, legends, etc., but doesn’t actually plot anything.)

15.8 Exercises

To think through, not to hand in.

1. Reproduce Figure 15.4?
2. Qualitatively, is this compatible with Figure 15.1?
3. How could we use `popinv2` to calculate a joint density for `popgro` and `inv` (not `exp(popgro)` and `exp(inv)`)?
4. Should the density `popinv2` implies for those variables be the same as what we'd get from directly estimating their density with kernels?

Chapter 16

Relative Distributions and Smooth Tests of Goodness-of-Fit

In §5.2.2.3, we saw how to use the quantile function to turn uniformly-distributed random numbers into random numbers with basically arbitrary distributions. In this chapter, we will look at two closely-related data-analysis tools which go the other way, trying to turn data into uniformly-distributed numbers. One of these, the **smooth test**, turns a lot of problems into ones of testing a uniform distribution. Another, the **relative distribution**, gives us a way of comparing whole distributions, rather than specific statistics (like the expectation or the variance).

16.1 Smooth Tests of Goodness of Fit

16.1.1 From Continuous CDFs to Uniform Distributions

Suppose that X has probability density function f , and that f is continuous. The corresponding cumulative distribution function F is then continuous and strictly increasing (on the support of f). Since F is a fixed function, we can ask what the probability distribution of $F(X)$ is. Clearly,

$$\Pr(F(X) \leq 0) = 0 \tag{16.1}$$

$$\Pr(F(X) \leq 1) = 1 \tag{16.2}$$

Since F is continuous and strictly increasing, it has an inverse, the quantile function Q , which is also continuous and strictly increasing. Then, for $0 \leq \alpha \leq 1$,

$$\Pr(F(X) \leq \alpha) = \Pr(Q(F(X)) \leq Q(\alpha)) \tag{16.3}$$

$$= \Pr(X \leq Q(\alpha)) \tag{16.4}$$

$$= F(Q(\alpha)) = \alpha \tag{16.5}$$

Thus, when F is continuous and strictly-increasing, $F(X)$ is uniformly distributed on the unit interval,

$$F(X) \sim \text{Unif}(0, 1) \quad (16.6)$$

If the distribution of X is F , but we guess that it has some other distribution, with CDF F_0 , then this trick will not work. $F_0(X)$ will still be in the unit interval, but it won't be uniformly distributed:

This only works if X really is distributed according to F . If instead X were distributed according, say, F_0 , then $F(X)$ will still be in the unit interval, but it will not be uniformly distributed:

$$\Pr(F_0(X) \leq a) = \Pr(X \leq Q_0(a)) \quad (16.7)$$

$$= F(Q_0(a)) \neq a \quad (16.8)$$

because $F_0 \neq Q^{-1}$.

Putting this together, we see that when X has a continuous distribution, $F(X) \sim \text{Unif}(0, 1)$ if and only if F is the cumulative distribution function for X . This means that we can reduce the problem of testing whether $X \sim F$ to that of testing whether $F(X)$ is uniform. We need to work out *one* testing problem, rather than many different testing problems for many different distributions.

16.1.2 Testing Uniformity

Now we have a random variable, say Y , which lives on the unit interval $[0, 1]$, and we want to test whether it is uniformly distributed. There are several different ways we could do this. One frequently-used strategy is to use the Kolmogorov-Smirnov test: calculate the K-S distance,

$$d_{KS} = \max_{a \in [0, 1]} |\hat{F}_{n,Y}(a) - a| \quad (16.9)$$

where $\hat{F}_{n,Y}(a)$ is the empirical CDF of Y , and look up the appropriate p -value for the K-S test. One could use any other one-sample non-parametric test here, like Cramér-von Mises or Anderson-Darling¹. All of these tests can work quite well in the right circumstances, and they have the advantage of requiring little additional work over and above typing `ks.test` or the like.

16.1.3 Neyman's Smooth Test

There are however two disadvantages of just applying off-the-shelf tests to check uniformity. One is that it turns out that they often do not have very high power. The other, which is in some ways even more serious, is that rejecting the null hypothesis of uniformity doesn't tell you *how* uniformity fails — it doesn't suggest any sort of natural alternative.

¹You could even use a χ^2 test, but this would be dumb. Because the χ^2 test requires discrete data, using it means binning continuous values, thereby destroying information, to no good purpose.

As you can guess from my having brought up these points, there is a test which avoids both difficulties, called **Neyman's smooth test**. It works by embedding the uniform distribution on the unit interval in a larger class of alternatives, and then testing the null of uniformity against those alternatives.

The alternatives all have pdfs of the form

$$g(y; \theta) \equiv \begin{cases} \frac{e^{\sum_{j=1}^d \theta_j b_j(y)}}{z(\theta)} & 0 \leq y \leq 1 \\ 0 & \text{elsewhere} \end{cases} \quad (16.10)$$

where the b_j are carefully chosen functions (see below), and the **normalizing factor** or **partition function** $z(\theta)$ just makes sure the density integrates to 1:

$$z(\theta) \equiv \int_0^1 e^{\sum_{j=1}^d \theta_j b_j(y)} dy \quad (16.11)$$

No matter what functions we pick for the b_j , uniformity corresponds to the choice $\theta = 0$, since then the density is just 1. As we move θ slightly away from 0, the density departs *smoothly* from uniformity; hence the name of the test.

To ensure that everything works out, we need to put some requirements on the functions b_j : they need to be **orthogonal** to each other and to the constant function,

$$\int_0^1 b_j(y) dy = 0 \quad (16.12)$$

$$\int_0^1 b_j(y) b_k(y) dy = 0 \quad (16.13)$$

and **normalized** in magnitude,

$$\int_0^1 b_j^2(y) dy = 1 \quad (16.14)$$

Further details, while practically important, do not matter for the general idea of the test, so I'll put them off to §16.1.3.1.

We can estimate θ by maximum likelihood. Because uniformity corresponds to $\theta = 0$, we can test the hypothesis that $\theta = 0$ against the alternative that $\theta \neq 0$ with a likelihood ratio test. Writing $\ell(\hat{\theta})$ for the log-likelihood under the MLE, and $\ell(0)$ for the log-likelihood under the null, by general results on the likelihood-ratio (Appendix C), under the null, as $n \rightarrow \infty$,

$$2(\ell(\hat{\theta}) - \ell(0)) \rightsquigarrow \chi_d^2 \quad (16.15)$$

In fact, $\ell(0) = 0$ (why?), so we only need to calculate the log-likelihood under the alternative, and reject uniformity when, and only when, that log-likelihood is large.

Alternatively, and this was Neyman's original recommendation and what is usually meant by his "smooth test", we can calculate the sample mean of each of the b_j ,

$$\bar{b}_j = \frac{1}{n} \sum_{i=1}^n b_j(y_i) \quad (16.16)$$

and form the test statistic

$$\Psi^2 = n \sum_{j=1}^d \bar{b}_j^2 \quad (16.17)$$

which also has a χ_d^2 distribution under the null.²

It can be shown that Neyman's smooth test has, in a certain sense, optimal power against smooth alternatives like this — see Rayner and Best (1989) or Bera and Ghosh (2002) for the gory details. More importantly, for data analysis, when we reject the null hypothesis of uniformity, we have a ready-made alternative to fall back on, namely $g(y; \hat{\theta})$.

To make all this work, we have to pick some "basis functions" b_j , and we need to decide how many of them we want to use, d .

16.1.3.1 Choice of Function Basis

Neyman's original proposal was to use **orthonormal polynomials** for basis functions: b_j would be a polynomial of degree j , which was orthogonal to all the ones before it,

$$\int_0^1 b_j(y) b_k(y) dy = 0 \quad \forall k < j \quad (16.18)$$

including the constant "polynomial" $b_0(y) = 1$, and normalized to size 1,

$$\int_0^1 b_j^2(y) dy = 1 \quad (16.19)$$

Since there are $j+1$ coefficients in a polynomial of degree j , and this gives $j+1$ equations, the polynomial is uniquely determined. In fact, there are recursively formulas which let you find the coefficients of b_j from those of the previous polynomials³. Figure 16.1 shows the first few of these polynomials, and their exponentiated versions (which are what appear in Eq. 16.10).

²To appreciate what's going on, notice that $\bar{b}_j \rightarrow 0$ under the null, by the law of large numbers. (This is where being orthogonal to the constant function $b_0(y) = 1$ comes in.) Multiplying \bar{b}_j^2 by n corresponds to looking at $\sqrt{n}\bar{b}_j$, which should, by the central limit theorem, be a Gaussian; the variance of this Gaussian is 1. (This is where normalizing each b_j comes in.) Finally, $\sqrt{n}\bar{b}_j$ and $\sqrt{n}\bar{b}_k$ are uncorrelated. (This is where the mutual orthogonality of the b_j comes in.) Thus, the Ψ^2 statistic is a sum of d uncorrelated standard Gaussians, which has a χ_d^2 distribution.

³In fact, the polynomials Neyman proposed to use are, as he knew, the "Legendre polynomials", though many math books (and Wikipedia) give the version of those defined on $[-1, 1]$, rather than on $[0, 1]$. If l_j is the polynomial on $[-1, 1]$, then $b_j(y) = l_j(2(y - 0.5))$.

Experience has shown that the specific choice of basis functions doesn't matter as much as ensuring that they are orthonormal. One could, for instance, use $b_j(y) = c_j \cos 2\pi jy$, where c_j is a normalizing constant⁴.

16.1.3.2 Choice of Number of Basis Functions

As we make d in Eq. 16.10, we include more and more distributions in the alternative to the null hypothesis of uniformity. In fact, since any smooth function on $[0, 1]$ can be approximated arbitrarily closely by sufficiently-high order polynomials⁵, as we let $d \rightarrow \infty$ we eventually get *all* continuous distributions, other than uniformity, as part of the alternative. However, using a large value of d means estimating a lot of parameters, which means we are at risk of over-fitting. What to do?

Neyman's original advice was to guess a particular value of d before looking at the data and stick to it. (He thought $d = 4$ would usually be enough.) More modern approaches try to adaptively pick a good value of d . We could attempt this through cross-validation based on the log-likelihood, but what's usually done, in implemented software, is to pick d to maximize Schwarz's information criterion:

$$d^* = \operatorname{argmax}_d \frac{1}{n} \ell(\hat{\theta}^{(d)}) - \frac{d}{2} \frac{\log n}{n} \quad (16.20)$$

which imposes an extra penalty for each parameter (d), with the size of the penalty depending on how much data we have, and getting relatively harsher as n grows⁶. So in a **data-driven smooth test** (Kallenberg and Ledwina, 1997), we pick d^* using Eq. 16.20, and then compute the test statistic using d^* .

Unfortunately, since d^* is random (through the data), the nice asymptotic theory which says that the test statistic is χ_d^2 under the null hypothesis no longer applies. However, this is why we have bootstrapping: by simulating from the null hypothesis, which remember is just $\text{Unif}(0, 1)$, and treating the simulation output like real data we can work out the sampling distribution as accurately as we need. This sampling distribution then gives us our p -values.

16.1.3.3 Application: Combining p -Values

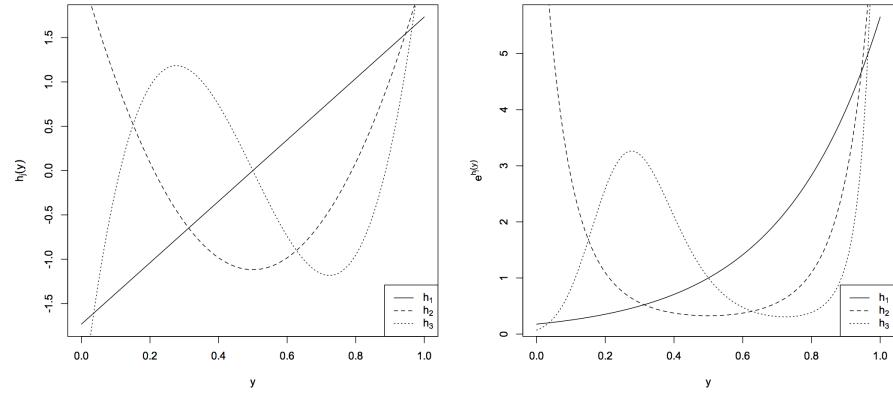
One useful property of p -values is that they are always uniformly distributed on $[0, 1]$ under the null hypothesis⁷. Suppose we have conducted a bunch of tests of the same null hypothesis — these might be different clinical trials of the same drug, or

⁴If this makes you think of Fourier analysis, you're right.

⁵This may be obvious, but making it precise (what do we mean by "smooth" and "arbitrarily close"??) is the "Stone-Weierstrauss theorem". There is nothing magic about polynomials here; we could also use sines and cosines, or many other function bases.

⁶It is common in the literature to see the criterion written out multiplied through by n , or even by $2n$. Also, it is often called the "Bayesian information criterion", or BIC. This is an unfortunate name, because, despite what Schwarz (1978) thought, it really has nothing at all to do with Bayes's rule or even Bayesian statistics. It's best thought of as a fast, but very crude and not always very accurate, approximation to cross-validation. If you want to know more, Claeskens and Hjort (2008) is probably the best reference.

⁷Unless someone has messed up a calculation, that is.

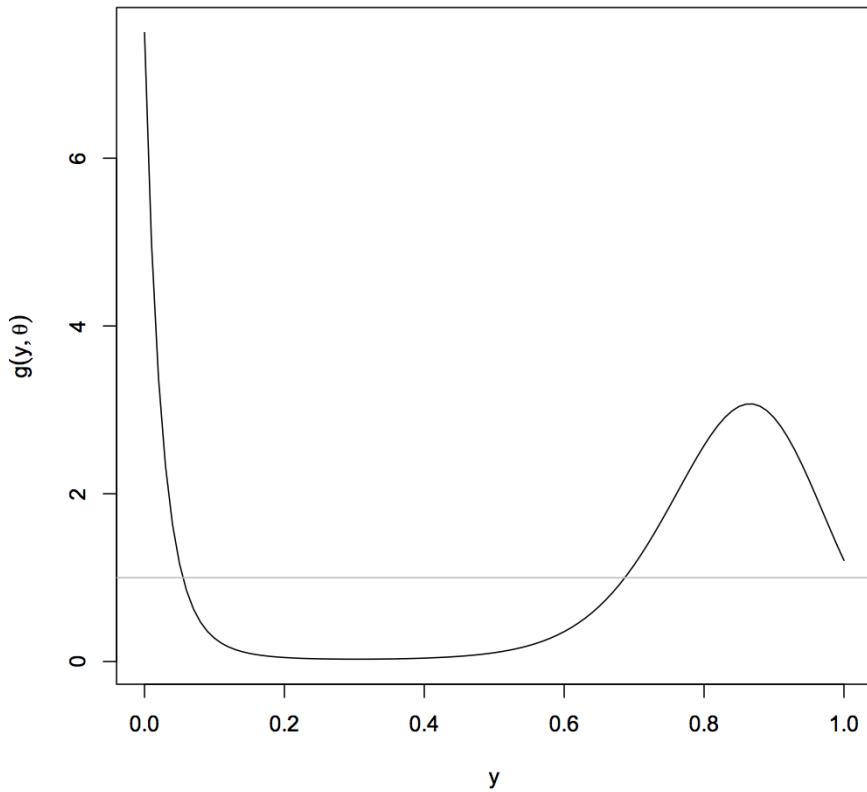


```

h1 <- function(y) { sqrt(12)*(y-0.5) }
h2 <- function(y) { sqrt(5)*(6*(y-0.5)^2-0.5) }
h3 <- function(y) { sqrt(7)*(20*(y-0.5)^3 - 3*(y-0.5)) }
curve(h1(x),ylab=expression(h[j](y)),xlab="y")
curve(h2(x),add=TRUE,lty="dashed")
curve(h3(x),add=TRUE,lty="dotted")
legend(legend=c(expression(h[1]),expression(h[2]),expression(h[3])),
lty=c("solid","dashed","dotted"),x="bottomright")
curve(exp(h1(x)),ylab=expression(e^h[j](y)),xlab="y")
curve(exp(h2(x)),add=TRUE,lty="dashed")
curve(exp(h3(x)),add=TRUE,lty="dotted")
legend(legend=c(expression(h[1]),expression(h[2]),expression(h[3])),
lty=c("solid","dashed","dotted"),x="bottomright")

```

Figure 16.1: Left panel: the first three of the basis functions for Neyman's smooth tests, h_1 , h_2 and h_3 . Each h_j is a polynomial of order j which is orthogonal to the others, in the sense that $\int_0^1 h_j(y)h_k(y)dy = 0$ when $j \neq k$, but normalized in size, $\int_0^1 h_j^2(y)dy = 1$. The right panel shows $e^{h_j(y)}$, to give an indication of how the functions contribute to the probability density in Eq. 16.10.



```

x <- (1:1e6)/1e6
z <- sum(exp(h1(x)+h2(x)-h3(x)))/1e6
curve(exp(h1(x)+h2(x)-h3(x))/z,xlab="y",ylab=expression(g(y,theta)))
abline(h=1,col="grey")

```

Figure 16.2: Illustration of a smooth alternative density: using the same basis functions as before, with $\theta_1 = 1$, $\theta_2 = 1$, $\theta_3 = -1$. The first two lines of the R calculate the normalizing constant $z(\theta)$ by a simple numerical integral. The grey line shows the uniform density.

attempts to replicate some surprising effect in separate laboratories⁸. If the tests are independent, then the p -values should be IID and uniform. It would seem like we should be able to combine these into some over-all p -value. This is *precisely* what Neyman's smooth test of uniformity lets us do.

16.1.3.4 Density Estimation by Series Expansion

As an aside, notice what we have done. By using a large enough d , as I said, densities which look like Eq. 16.10 can come as close as we like to any smooth density on $[0, 1]$. And now we have at least two ways of picking d : by cross-validation, or by the Schwarz information criterion (Eq. 16.20). If we let $d \rightarrow \infty$ as $n \rightarrow \infty$, then we have a way of approximating any density on the unit interval, without knowing what it was to start with, or assuming a particular parametric form for it. That is, we have a way of doing non-parametric density estimation, at least on $[0, 1]$, without using kernels.

If you want to estimate a density on $(-\infty, \infty)$ instead of on $[0, 1]$, you can do so by using a transformation, e.g., the inverse logit. This is the opposite of what you did in the homework, where you used a transformation to take $[0, 1]$ to $(-\infty, \infty)$ so you could use kernel density estimation.

16.1.4 Smooth Tests of Non-Uniform Parametric Families

Remember that we went into all these details about testing uniformity because we want to test whether X is distributed according to some continuous distribution with CDF F . From §16.1.1, if we define $Y = F(X)$, then $X \sim F$ is equivalent to $Y \sim \text{Unif}(0, 1)$, so we have a two-step procedure for testing whether $X \sim F$:

1. Use the CDF F to transform the data, $y_i = F(x_i)$
2. Test whether the transformed data y_i are uniform

Let's think about what the alternatives considered in the test look like. For y , the alternative densities are (to repeat Eq. 16.10)

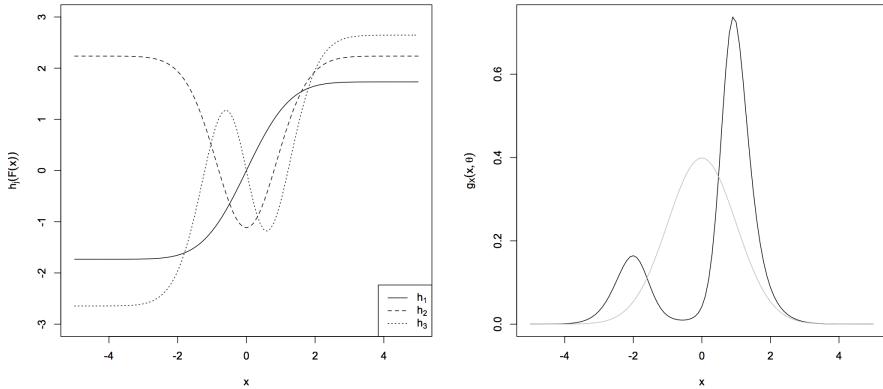
$$g(y; \theta) \equiv \begin{cases} \frac{e^{\sum_{j=1}^d \theta_j b_j(y)}}{z(\theta)} & 0 \leq y \leq 1 \\ 0 & \text{elsewhere} \end{cases} \quad (16.21)$$

Since $X = F^{-1}(Y)$, this implies a density for X :

$$g_X(x; \theta) = \frac{e^{\sum_{j=1}^d \theta_j b_j(F(x))}}{z(\theta)} \frac{dF}{dx} \quad (16.22)$$

$$= \frac{e^{\sum_{j=1}^d \theta_j b_j(F(x))}}{z(\theta)} f(x) \quad (16.23)$$

⁸These are typical examples of **meta-analysis**, trying to combine the results of many different data analyses (without just going back to the original data).



```

curve(h1(pnorm(x)),xlab="x",ylab=expression(h[j](F(x))),from=-5,to=5,
      ylim=c(-3,3))
curve(h2(pnorm(x)),add=TRUE,lty="dashed")
curve(h3(pnorm(x)),add=TRUE,lty="dotted")
legend(legend=c(expression(h[1]),expression(h[2]),expression(h[3])),
       lty=c("solid","dashed","dotted"),x="bottomright")
curve(dnorm(x)*exp(h1(pnorm(x))+h2(pnorm(x))-h3(pnorm(x))/z,xlab="x",
             ylab=expression(g[X](x,theta)),from=-5,to=5)
curve(dnorm(x),add=TRUE,col="grey")

```

Figure 16.3: Left panel: the basis functions from Figure 16.1 composed with the standard Gaussian CDF. Right panel: the alternative to the standard Gaussian corresponding to the alternative to the uniform distribution plotted in Figure 16.2, i.e., $\theta_1 = \theta_2 = 1$, $\theta_3 = -1$. The grey curve is the standard Gaussian density, corresponding to the flat line in Figure 16.2.

where f is the pdf corresponding to the CDF F . (Why do we not have to worry about setting this to zero outside some range?) Just like $g(\cdot; \theta)$ is a modulation or distortion of the uniform density, $g_X(\cdot; \theta)$ is a modulation or distortion of $f(\cdot)$. If and when we reject the density f , $g_X(\cdot; \theta)$ is available to us as an alternative.

Even if $h_j(y)$ is a polynomial in y , $h_j(F(x))$ will not (in general) be a polynomial in x , but it remains true that

$$\int_{-\infty}^{\infty} h_j(F(x))h_k(F(x))f(x)dx = \delta_{jk} \quad (16.24)$$

Figure 16.3 illustrates what happens to the basis functions, and to particular alternatives.

When it comes to the actual smooth test, we can either use the likelihood ratio, or we can calculate

$$\bar{h}_j = \frac{1}{n} \sum_{i=1}^n h_j(y_i) = \frac{1}{n} \sum_{i=1}^n h_j(F(x_i)) \quad (16.25)$$

leading as before to the test statistic

$$\Psi^2 = n \sum_{j=1}^n \bar{h}_j^2 \quad (16.26)$$

The distribution of the test statistics is unchanged under the null hypothesis, i.e., still χ_d^2 if d is fixed. (There are still d degrees of freedom, because we are still fixing d parameters from distributions of the form Eq. 16.23.) If d is chosen from the data, we still need to bootstrap, but can do so just as before.

16.1.4.1 Estimated Parameters

So far, the discussion has assumed that F is fixed and won't change with the data. This is often not very realistic. Rather, F comes from some parametrized family of distributions, with parameter (say) β , i.e., $F(\cdot; \beta)$ is a different CDF for each value of β . For Gaussians, for instance, β is a vector consisting of the mean and variance (or mean and standard deviation). Let's assume that there are always corresponding densities, $f(\cdot; \beta)$, and these are always continuous.

We don't know β so we have to estimate it. After estimating, we'd like to test whether the model really matches the data. It would be convenient if we could do the following:

1. Get estimate $\hat{\beta}$ from x_1, x_2, \dots, x_n
2. Calculate $y_i = F(x_i; \hat{\beta})$
3. Apply a smooth test of uniformity to y_1, y_2, \dots, y_n

That is, it would be convenient if we could just *ignore* the fact that we had to estimate β .

We can do this if $\hat{\beta}$ is the maximum likelihood estimate. To understand this, think about the family of alternative distributions we're now considering in the test. Substituting into Eq. 16.23, they are

$$g_X(x; \beta, \theta) = \frac{e^{\sum_{j=1}^d \theta_j h_j(F(x; \beta))}}{z(\theta)} f(x; \beta) \quad (16.27)$$

The null hypothesis that $X \sim F(\cdot; \beta)$ for some β is thus corresponds to $X \sim G_X(\cdot; \beta, 0)$ — we are still fixing d parameters in the larger family. And, generally speaking, when we fix d parameters in a parametric model, we get a χ_d^2 distribution in the log-likelihood ratio test (Appendix C). If d is not fixed but data-driven, then, again, we need to bootstrap.

16.1.5 Implementation in R

The main implementation of smooth tests available in R is the `ddst` package (Biecek and Ledwina, 2010), standing for “data-driven smooth tests”. It provides a `ddst.uniform.test`, which we could use for any family where we can calculate the CDF. But it also provides functions for directly testing several families of distributions, notably Gaussians (`ddst.norm.test`) and exponentials (`ddst.exp.test`).

16.1.5.1 Some Examples

Let’s give `ddst.norm.test` some Gaussian data and see what happens.

```
> r <- rnorm(100)
> ddst.norm.test(r)
```

Data Driven Smooth Test for Normality

```
data: r, base: ddst.base.legendre, c: 100
WT* = 0.6183, n. coord = 1
```

This reminds us what the data was, tells us that the test used Legendre polynomials (as opposed to cosines), that d was selected to be 1, and that the value of the test statistic was 0.6183. (The c setting has to do with the order-selection penalty, and is basically ignorable for most users.) These numbers are all attributes of the returned object.

What is missing is the p -value, because this is computationally expensive to calculate. (You can control how many bootstraps it uses, but the default is 1000.)

```
> ddst.norm.test(r, compute.p=TRUE)
```

Data Driven Smooth Test for Normality

```
data: r, base: ddst.base.legendre, c: 100
WT* = 0.6183, n. coord = 1, p-value = 0.476
```

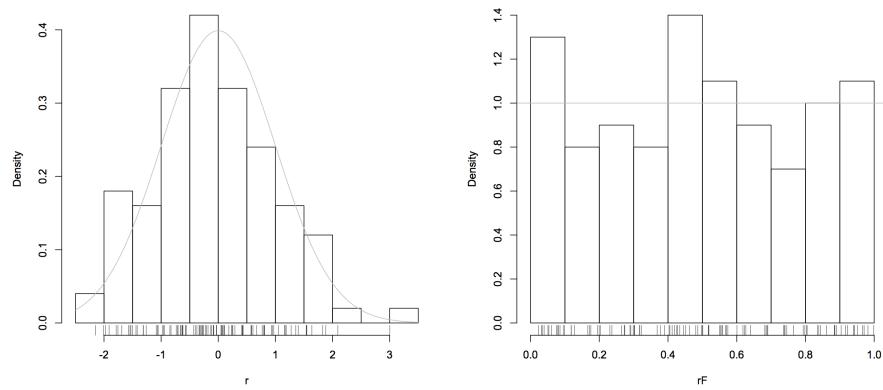
So the p -value is 0.476, giving us no reason to reject a Gaussian distribution when we’re looking at numbers from the standard Gaussian. If we ignored the fact that d was selected from the data and plugged into the corresponding χ_d^2 distribution, we’d get a p -value of

```
> pchisq(0.6183, df=1, lower.tail=FALSE)
[1] 0.4316797
```

which to say a relative error of about 10%.

What if we give the procedure some non-Gaussian data? Say, the same amount of data from a t distribution with 5 degrees of freedom?

```
> ng <- rt(100, df=5)
> ddst.norm.test(ng, compute.p=TRUE)
```

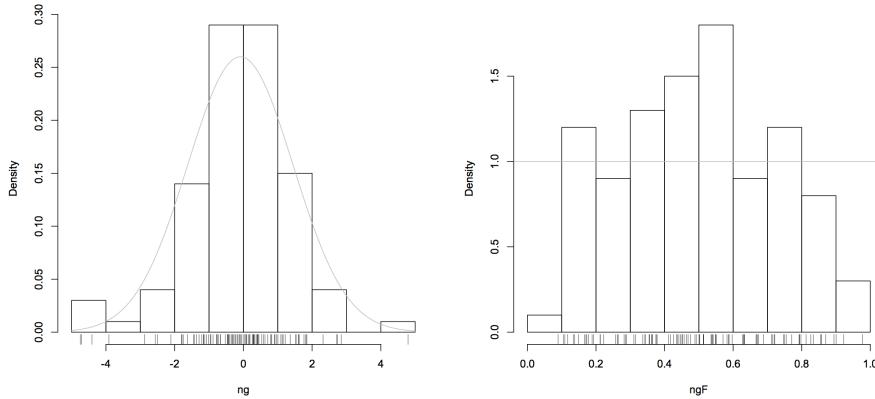


```

plot(hist(r),freq=FALSE,main="")
rug(r)
curve(dnorm(x),add=TRUE,col="grey")
rF <- pnorm(r,mean=mean(r),sd=sd(r))
plot(hist(rF),freq=FALSE,main="")
rug(rF)
abline(h=1,col="grey")

```

Figure 16.4: Left panel: histogram of the 100 random values from the standard Gaussian used in the text (exact values marked along the horizontal axis), plus the true density in grey. Right panel: transforming the data according to the Gaussian *fitted to the data* by maximum likelihood.



```
plot(hist(ng), freq=FALSE, main="")
rug(ng)
curve(dnorm(x, mean=mean(ng), sd=sd(ng)), add=TRUE, col="grey")
ngF <- pnorm(r, mean=mean(ng), sd=sd(ng))
plot(hist(ngF), freq=FALSE, main="")
rug(ngF)
abline(h=1, col="grey")
```

Figure 16.5: Treating the draw from the t distribution discussed in the text the same as the Gaussian sample in Figure 16.4.

Data Driven Smooth Test for Normality

```
data: ng, base: ddst.base.legendre, c: 100
WT* = 16.5623, n. coord = 2, p-value = 0.007
```

Of course, it won't *always* reject, because we're only looking at 100 samples, and the t distribution isn't *that* different from a Gaussian. Still, when I repeat this experiment many times, we get quite respectable power at the standard 5% size:

```
> mean(replicate(100,
+ ddst.norm.test(rt(100, df=5), compute.p=TRUE)$p.value)<0.05)
[1] 0.51
```

See Exercise 3 for a small project of `ddst.exp.test` to check a Pareto distribution.

16.1.6 Conditional Distributions and Calibration

Suppose that we are not interested in the marginal distribution of X , but rather its conditional distribution given some other variable or variables C (for “covariates”). If the conditional density $f(x|C = c)$ is continuous in x for every c , then it is easy to argue, in parallel with §16.1.1, that $F(X|C = c)$, the conditional CDF, should $\sim \text{Unif}(0, 1)$. So, as long as we use the conditional CDF to transform X , we can apply smooth tests as before.

One important use of this is regression residuals. Suppose X is the target variable of a regression, with C being the predictor variables⁹, and we have some parametric distribution in mind for the noise (Gaussian, say), with the noise ϵ being independent of C . Then the model is $X = r(C) + \epsilon$, so looking at the conditional CDF of X given Z is equivalent to looking at the unconditional CDF of the residuals. We can then actually *test* whether the residuals are Gaussian, rather than just squinting at a Q-Q plot. We could also do this by applying a K-S test to the transformed residuals, but everything that was said above in favor of smooth tests would still apply.

Notice, by the way, that by applying the CDF transformation to the residuals, we are checking whether the model is properly calibrated, i.e., whether events it says happen with probability p actually have a frequency close to p . We do need to impose assumptions about the distribution of the noise to check calibration for a regression model, since if we *just* predict expected values, we say nothing about how often any particular range of values should happen.

Later, when we look at graphical models and at time series, we will see several other important situations where a statistical model is really about conditional distributions, and so can be checked by looking at conditional CDF transformations. It seems to be somewhat more common to apply K-S tests than smooth tests after the conditional CDF transformation (e.g., Bai 2003), but I think this is just because smooth tests are not as widely known and used as they should be.

⁹I know you’re used to X being the predictor and Y being the target.

16.2 Relative Distributions

So far, I have been talking about how we can test whether our data follows some hypothesized distribution, or family of distributions, by using the fact that $F(X)$ is uniform if and only if X has CDF F . If the values of $F(x_i)$ are close enough to being uniform, the true CDF has to be pretty close to F (with high confidence); if they are far from uniform, the true CDF has to be far from F (again with high confidence).

In many situations, however, we already know (or are at least pretty sure) that X doesn't have some distribution, say F_0 , and what we are interested in is *how* X fails to follow it; we want, in other words, to compare the distribution of X to some reference distribution F_0 . For instance:

1. We are trying a new medical procedure, and we want to compare the distribution of outcomes for patients who got the treatment to those who did not.
2. We want to compare the distribution of some social outcome across two categories at the same time. (For instance, we might compare income, or lifespan, for men and for women.)
3. We might want to compare members of the same category at different times, or in different locations. (We might compare the income distribution of American men in 1980 to that of 2010, or the lifespans of American men in 2010 to those of Canadian men.)
4. We might compare our actual population to the distribution predicted by a model we know to be too simple (or just approximate) to try to learn what it is missing.

You learned how to do comparisons of simple summaries of distributions in baby stats. (For instance, you learned how to compare group means by doing t -tests.) While these certainly have their places, they can miss an awful lot. For example, a few years ago now an anesthesiologist came to the CMU statistics department for help evaluating a new pain-management procedure, which was supposed to reduce how many pain-killers patients recovering from surgery needed. Under both the old procedure and the new one, the distribution was strongly bimodal, with some patients needing very little by way of pain-killers, many needing much more, and a few needing an awful lot of drugs. Simply looking at the change in the mean amount of drugs taken, or even the changes in the mean and the variance, would have told us very little about whether things were any better¹⁰.

In this example, the **reference distribution**, F_0 , is given by the distribution of drug demand for patients on the old pain-management protocol. The new or **comparison** sample, x_1, \dots, x_n , are realizations of a random variable X , representing the demand for pain-killers under the new protocol. X follows the **comparison distribution** F , which is presumably not the same as F_0 ; how does it differ?

¹⁰I am omitting some details, and not providing a reference because the study is still, so far as I know, unpublished.

The idea of the **relative distribution** is to characterize the change in distributions by using F_0 to transform X into $[0, 1]$, and then looking at how it departs from uniformity. The **relative data**, or **grades**, are

$$r_i = F_0(x_i) \quad (16.28)$$

Simply put, we take the comparison data points and see where they fall in the reference distribution.

What is the cumulative distribution function of the relative data? Let's look at this first at the population level, where we have F_0 (the reference distribution) and F (the comparison distribution), rather than just samples. Let's call the CDF of the relative data G :

$$G(a) \equiv \Pr(R \leq a) \quad (16.29)$$

$$= \Pr(F_0(X) \leq a) \quad (16.30)$$

$$= \Pr(X \leq Q_0(a)) \quad (16.31)$$

$$= F(Q_0(a)) \quad (16.32)$$

where remember $Q_0 = F_0^{-1}$ is the quantile function of the reference distribution. This in turn implies a probability density function of the relative data:

$$g(y) \equiv \frac{dG}{da} \Big|_{a=y} \quad (16.33)$$

$$= \frac{dF}{du} \Big|_{u=Q_0(y)} \frac{dF_0^{-1}}{da} \Big|_{a=y} \quad (16.34)$$

$$= f(Q_0(y)) \frac{1}{f_0(Q_0(y))} = \frac{f(Q_0(y))}{f_0(Q_0(y))} \quad (16.35)$$

This only applies when $y \in [0, 1]$; elsewhere, $g(y)$ is straightforwardly 0.

When $g(y) > 1$, we have $f(Q_0(y)) > f_0(Q_0(y))$ — that is, values around $Q_0(y)$ are relatively more probable in the comparison distribution than in the reference distribution. Likewise, when $g(y) < 1$, the comparison distribution puts less weight on values around $Q_0(y)$ than does the reference distribution. If the comparison distribution was exactly the same as the reference distribution, we would, of course, get $g(y) = 1$ everywhere.

One very important property of the relative distribution is that it is invariant under monotone transformations. That is, suppose instead of looking at X , we looked at $h(X)$ for some monotonic function h . (An obvious example would be change of units, but we might also take logs or powers.) Summary statistics like differences in means are generally not even *equi*-variant¹¹. But it is easy to check (Exercise 4) that

¹¹Remember that a statistic, say δ , is a function of the data, $\delta(x_1, x_2, \dots, x_n)$. The statistic is *invariant* under a transformation h if $\delta(h(x_1), h(x_2), \dots, h(x_n)) = \delta(x_1, x_2, \dots, x_n)$ — the transformation does not change the statistic. The statistic is *equivariant* if it “changes along with” the transformation, $\delta(h(x_1), h(x_2), \dots, h(x_n)) = h(\delta(x_1, x_2, \dots, x_n))$. Maximum likelihood estimates are equivariant. Statistics like the mean are equivariant under linear and affine transformations (but not others).

the relative distribution of $h(X)$ is the same as the relative distribution of X . This expresses the idea that the difference between the reference and comparison distributions is independent of our choice of a coordinate system for X .

16.2.1 Estimating the Relative Distribution

In some situations, the reference distribution can come from a theoretical model, but the comparison distribution is unknown, though we have samples. Estimating the relative density g is then extremely similar to what we had to do in the last section for hypothesis testing. Non-parametric estimation of g can thus proceed either through fitting series expansions like Eq. 16.10 (with a data-driven choice of d , as above), or through using a fixed, data-independent transformation to map $[0, 1]$ to $(-\infty, \infty)$ and using kernel density estimation¹².

If, on the other hand, neither the reference nor the comparison distribution is fully known, but we have samples from both, estimating the relative distribution involves estimating Q_0 , the quantile function of the reference distribution. This is typically estimated as just the empirical quantile function, but in principle one could use, say, kernel smoothing to get at Q_0 . Once we have an estimate for it, though, we have reduced the problem of estimating g to the case considered in the previous paragraph.

Uncertainty in the estimate of the relative density g is, as usual, most easily assessed through the bootstrap. Be careful to include the uncertainty in estimates of Q_0 as well, if the reference quantiles have to be estimated. One can, however, also use asymptotic approximations (Handcock and Morris, 1999, §9.6).

16.2.2 R Implementation and Examples

Relative distribution methods were introduced by Handcock and Morris (1998, 1999), who also wrote an R package, `reldist`, which is by far the easiest way to work with relative distributions. Rather than explain abstractly how this works, we'll turn immediately to examples.

16.2.2.1 Example: Conservative versus Liberal Brains

In the homework, we have looked at the data from Kanai *et al.* (2011), which record the volumes of two parts of the brain, the amygdala and the anterior cingulate cortex (ACC), adjusted for body size, sex, etc., and political orientation on a five-point ordinal scale, with 1 being the most conservative and 5 the most liberal¹³. The subjects being British university students, the lowest score for political orientation recorded was 2, and so we will look at relative distributions between those students and the rest of the sample. That is, we take the conservatives as the comparison sample, and the rest as the reference sample¹⁴.

¹²We saw how to do this in the homework

¹³I am grateful to Dr. Kanai for graciously sharing the data.

¹⁴This implies no value judgment about conservatives being “weird”, but rather reflects the fact that there are many fewer of them than of non-conservatives in this data.

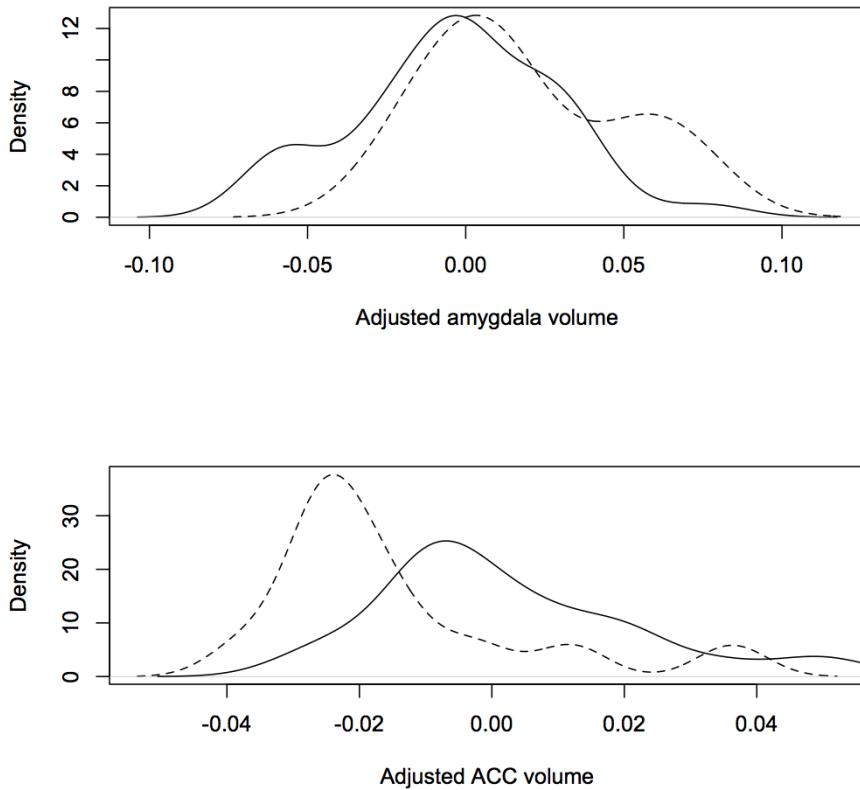
Having loaded the data into the data frame `n90`, we can look at simple density estimates for the two classes and the two variables (Figure 16.6). This indicates that conservative subjects tend to have relatively larger amygdalas and relatively smaller ACCs, though with very considerable overlap. (We are not looking at the uncertainty here at all.)

Enough preliminaries; let's find the relative distribution. Figure 16.7).

```
library(reldist)
acc.rel <- reldist(y=n90$acc[n90$orientation<3],
                     yo=n90$acc[n90$orientation>2], ci=TRUE,
                     ylabs=pretty(n90$acc[n90$orientation>2]),
                     main="Relative density of adjusted ACC volume")
```

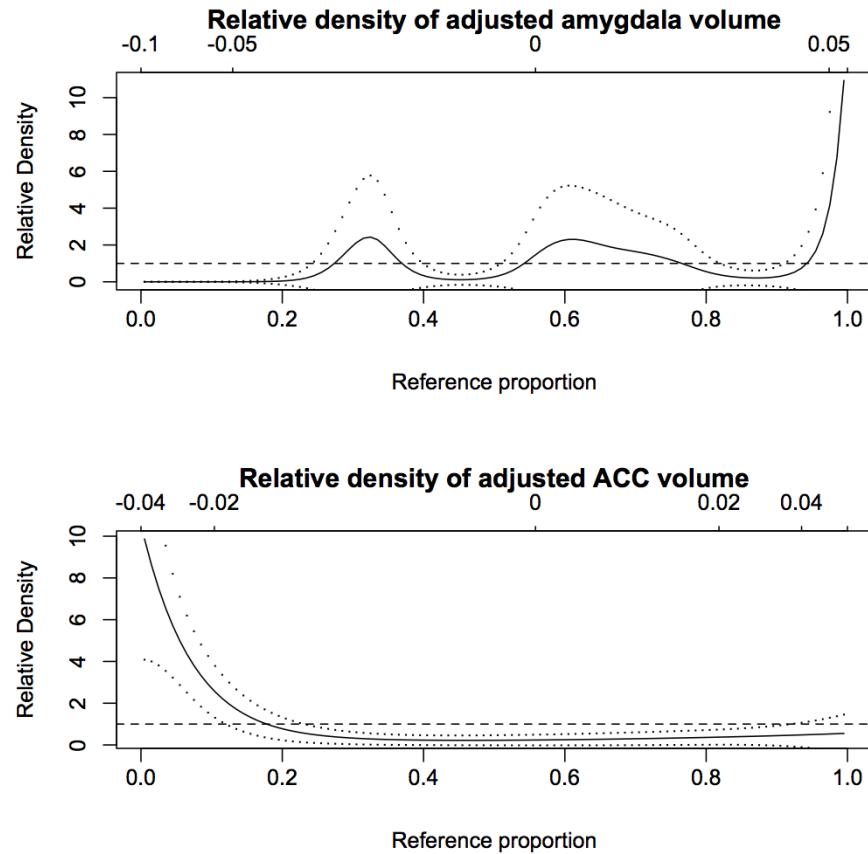
The first argument is the comparison sample; the second is the reference sample. The labeling of the horizontal axis is in terms of the quantiles of the reference distribution; I convert this back to the original units with the optional `yolabs` argument¹⁵. The dots show a pointwise 95%-confidence band, but based on asymptotic approximations which should not be taken seriously when there are only 77 reference samples and just 13 comparison samples.

¹⁵The function `pretty()` is a built-in routine for coming up with reasonable axis tick-marks from a vector. See `help(pretty)`.



```
par(mfrow=c(2,1))
plot(density(n90$amygdala[n90$orientation>2]),main="",
      xlab="Adjusted amygdala volume")
lines(density(n90$amygdala[n90$orientation<3]),lty="dashed")
plot(density(n90$acc[n90$orientation<3]),lty="dashed",main="",
      xlab="Adjusted ACC volume")
lines(density(n90$acc[n90$orientation>2]))
```

Figure 16.6: Estimated densities for the (adjusted) volume of the amygdala (upper panel) and ACC (lower panel) in non-conservative (solid lines) and conservative (dashed) students.



```
par(mfrow=c(2,1))
reldist(y=n90$amygdala[n90$orientation<3],
        yo=n90$amygdala[n90$orientation>2],ci=TRUE,
        ylabs=pretty(n90$amygdala[n90$orientation>2]),
        main="Relative density of adjusted amygdala volume")
reldist(y=n90$acc[n90$orientation<3],
        yo=n90$acc[n90$orientation>2],ci=TRUE,
        ylabs=pretty(n90$acc[n90$orientation>2]),
        main="Relative density of adjusted ACC volume")
```

Figure 16.7: Relative distribution of adjusted brain-region volumes, contrasting conservative subjects (comparison samples) to non-conservative subjects (reference samples). Dots indicate 95% confidence limits, but these are based on asymptotic approximations which don't work here. (The supposed lower limit for the relative density of the amygdala is almost always negative!) The dashed lines mark a relative density of 1, which would be

```
library(np)
data(oecdpanel)
in.oecd <- oecdpanel$oecd==1
reldist(y=oecdpanel$growth[in.oecd],
        yo=oecdpanel$growth[!in.oecd],
        ylabs=pretty(oecdpanel$growth[!in.oecd]),
        ci=TRUE, ylim=c(0,3))
```

Figure 16.8: Relative distribution of the per-capita GDP growth rates of OECD-member countries compared to those of non-OECD countries.

16.2.2.2 Example: Economic Growth Rates

For a second example, let's return to the OECD data on economic growth featured in Chapter 15. We want to know how the economic growth rates of countries which are already economically developed compares to the growth rates of developing and undeveloped countries. I approximate “is a developed country” by “is a membership of the OECD”, as in §15.5.1. I will take the non-developed countries as the reference distribution and the OECD members as the comparison group, mostly because there are more of the former and they are more diverse.

The basic commands now go as before (aside from loading the data from a different library):

```
library(np)
data(oecdpanel)
in.oecd <- oecdpanel$oecd==1
reldist(y=oecdpanel$growth[in.oecd],
        yo=oecdpanel$growth[!in.oecd],
        ylabs=pretty(oecdpanel$growth[!in.oecd]))
```

Examining the resulting plot (Figure 16.8), the relative distribution is unimodal, peaking around the 60th percentile of the reference distribution, a growth rate of about 2.5% per year. The relative distribution drops below 1 at both low (negative) or high ($> 0.05\%$) growth rates — developed countries, at least over the period of this data, tend to grow steadily and within a fairly narrow band, without so much of both the positive and negative extremes of non-developed countries¹⁶

It's also worth illustrating how to use `reldist` for comparison to a theoretical CDF. A *very* primitive, or better yet nihilistic, model of economic growth would say that the factors causing economies to grow or shrink are so many, and so various, and so complicated that there is no hope of tracking them systematic, but rather that we should regard them as effectively random. As we know from introduction probability, the average of many small independent forces has a Gaussian distribution; so

¹⁶It's easy to tell a story for why the distribution of growth rates for poor countries is so wide. Some poor countries grow very slowly or even shrink because they suffer from poor institutions, corruption, war, lack of resources, technological backwardness, etc.; some poor countries grow very quickly if they over-come or escape these obstacles and can quickly make use of technologies developed elsewhere. Nobody has a particular good story for why the growth rates of all developed countries are so similar.

we'll just assume that each country grows (or shrinks) by some independent Gaussian amount every year.

Doing this just means applying the cumulative distribution function of the model's distribution to the values from our comparison sample, as in Figure 16.9. The result does not look too different from Figure 16.8. (This does not mean that the nihilistic model of economic growth is right.)

16.2.3 Adjusting for Covariates

Another nice use of relative distributions is in adjusting for covariates or predictors more flexibly than is easy to do with regression. Suppose that we have measurements of *two* variables, X and Z . In general, when we move from the reference population to the comparison population, both variables will change their marginal distributions. If the marginal distribution of Z changes, and the conditional distribution of X given Z did not, then the marginal distribution of X would change. It is often informative to know how the change in the distribution of X compares to what would be anticipated just from the change in Z :

- The two populations might be male and female workers in the same industry, with X income and Z (say) education, or some measure of qualifications.
- The two populations might be students at two different schools, or taught in two different ways, with X their test scores at the end of the year, and Z some measure of prior knowledge.

Write the conditional density of X given Z in the reference population as $f_0(x|z)$. Then, just from the definitions of conditional and marginal probability,

$$f_0(x) = \int f_0(x|z)f_0(z)dz \quad (16.36)$$

If the distribution of the covariate Z is instead taken from the comparison population, we get a *different* distribution for x ,

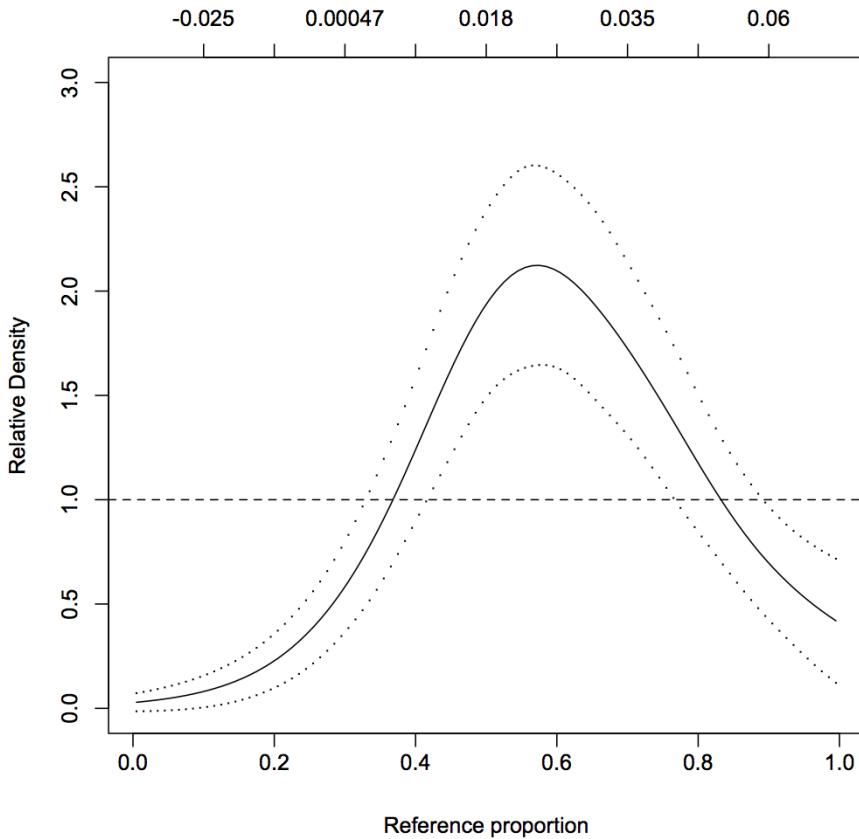
$$f_{0C}(x) = \int f_0(x|z)f_C(z)dz \quad (16.37)$$

with the C standing for “covariate” or “compensated”, depending on who you talk to. This is the distribution we would have seen for X if the distribution of X shifted but the relation between X and Z did not.

Before, we looked at the relative distribution of the comparison distribution F to the reference distribution F_0 , which had the density (Eq. 16.35) $g(y) = f(Q_0(y))/f_0(Q_0(y))$. Notice that

$$\frac{f(Q_0(y))}{f_0(Q_0(y))} = \frac{f_{0C}(Q_0(y))}{f_0(Q_0(y))} \frac{f(Q_0(y))}{f_{0C}(Q_0(y))} \quad (16.38)$$

The first ratio on the right-hand side the relative density of F_{0C} compared to f_0 ; the second ratio is the relative density of F compared to F_{0C} .



```

growth.mean <- mean(oecdpanel$growth[!in.oecd])
growth.sd <- sd(oecdpanel$growth[!in.oecd])
r = rnorm(oecdpanel$growth[in.oecd], growth.mean, growth.sd)
reldist(y=r, ci=TRUE, ylim=c(0,3))
top.ticks <- (1:9)/10
top.tick.values <- signif(qnorm(top.ticks, growth.mean, growth.sd), 2)
axis(side=3, at=top.ticks, labels=top.tick.values)

```

Figure 16.9: Distribution of the growth rates of developed countries, relative to a Gaussian fitted to all growth rates.

I have written everything as though Z were just a scalar, but it could be a vector, so we can adjust for multiple covariates at once. Also, it is important to emphasize that there is no implication that Z is in any sense the cause of X here (though such adjustments are often more *interesting* when that's true).

16.2.3.1 Example: Adjusting Growth Rates

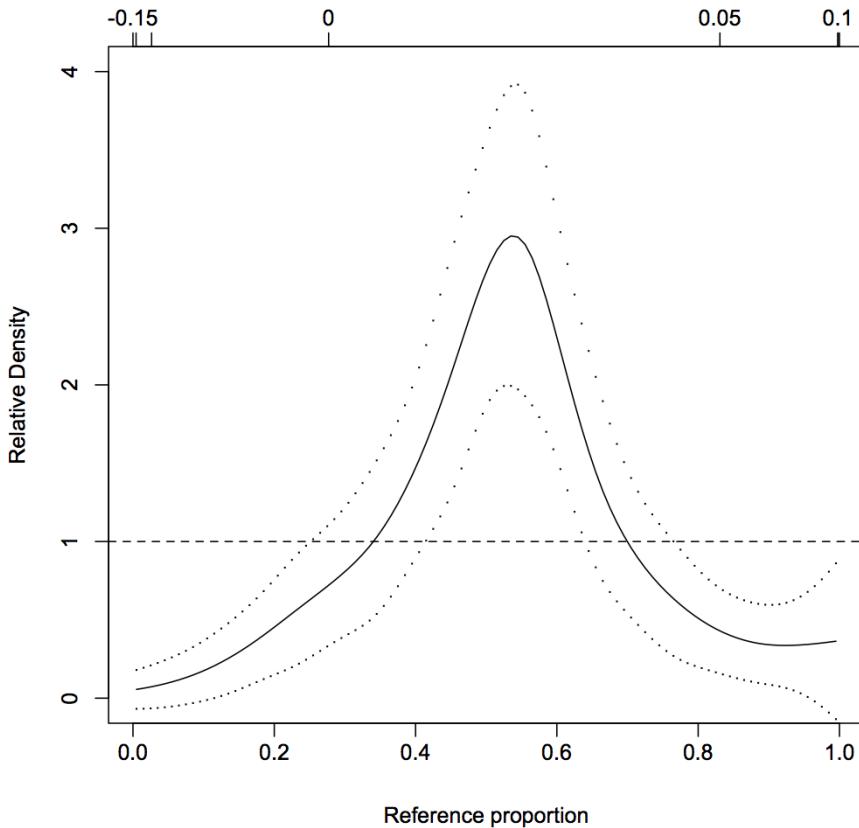
It will be easier to see how this works with an example. The `oecdpanel` data set also includes a variable called `humancap`, which is the log of the average number of years of education of people over the age of fifteen¹⁷. How do the growth rates of developed countries compare to those of undeveloped countries once we adjust for education?

As Figure 16.10 shows, after adjusting for education levels, the relative density shifts somewhat to the left, with its peak peaked closer to the median of the reference distribution. That is, some of the higher-than-usual growth of the developed countries can be explained away by their (unusually high: Figure 16.11) levels of education. But the relative density is now even *more* sharply peaked than it was before.

Again, it would be rash to read too much causality into this. It could be that education promotes economic growth¹⁸, or it could be that education is a luxury of rich societies, which grow faster than average for other reasons.

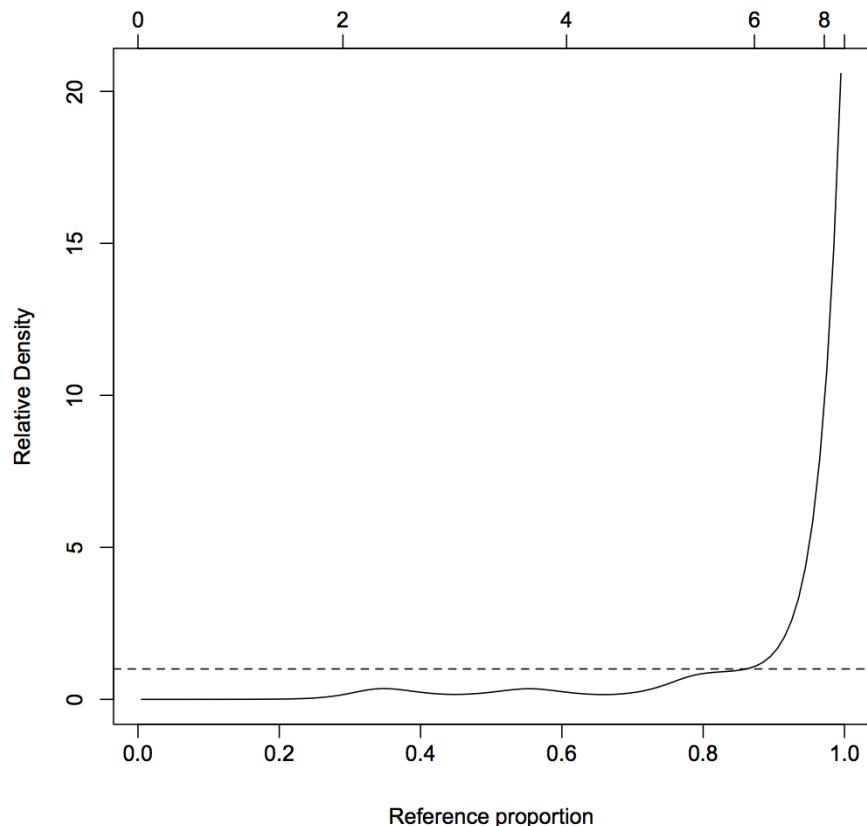
¹⁷If you look at `help(oecdpanel)`, it calls this variable “average secondary school enrollment rate”, but that’s clearly wrong, and examining the original papers referenced there shows the correct meaning of the variable. I am not sure why it was logged. (Incidentally, `humancap` stands for “human capital”. Whether education is best thought of in this way, or indeed whether years of schooling are a good measure of human capital, are hard questions which we fortunately do not have to answer.)

¹⁸Certainly it’s convenient for a teacher to think so.



```
reldist(y=oecdpanel$growth[in.oecd],
        yo=oecdpanel$growth[!in.oecd],
        ylabs=pretty(oecdpanel$growth[!in.oecd]),
        z=oecdpanel$humancap[in.oecd],
        zo=oecdpanel$humancap[!in.oecd],
        decomp="covariate",
        ci=TRUE, ylim=c(0,4))
```

Figure 16.10: Relative distribution of per-capita GDP growth rates after adjusting for education (humancap).



```
reldist(y=exp(oecdpanel$humancap[in.oecd]),
        yo=exp(oecdpanel$humancap[!in.oecd]),
        yolabs=pretty(exp(oecdpanel$humancap[!in.oecd])))
```

Figure 16.11: Relative distribution of years of education, comparing OECD countries to non-OECD countries.

16.3 Further Reading

On smooth tests of goodness of fit, see Bera and Ghosh (2002) (a pleasantly *enthusiastic* paper) and Rayner and Best (1989). The `ddst` package is ultimately based on Kallenberg and Ledwina (1997). On relative distributions, see Handcock and Morris (1998) (an expository paper aimed at social scientists) and Handcock and Morris (1999) (a more comprehensive book with technical details).

16.4 Exercises

To think through, not to hand in.

1. §16.1.3.1 asserts that one could use cosines orthonormal basis functions in a Neyman test, with $b_j(x) = c_j \cos 2\pi j x$. Find an expression for the normalizing constant c_j such that these functions satisfy Eq. 16.18 and Eq. 16.19.
2. Prove Eq. 16.24. *Hint:* change of variables. Also, prove that

$$\int_{-\infty}^{\infty} f(x) \exp^{\sum_{j=1}^d \theta_j b_j(F(x))} dx = \int_0^1 \exp^{\sum_{j=1}^d \theta_j b_j(y)} dy = z(\theta) \quad (16.39)$$

3. If $X \sim \text{Pareto}(\alpha, x_0)$, then $\log X / x_0 \sim \text{Exp}(\alpha)$ — the log of a power-law distributed variable has an exponential distribution. Using the `wealth.dat` data from Chapter 6 and `ddst.exp.test`, test whether net worths over $\$3 \times 10^8$ follow a Pareto distribution.
4. Let $T = h(X)$ for some fixed and strictly monotonic function h . Prove that the relative density of T is the same as the relative density of X . *Hint:* find the density of T under both the reference and comparison distribution in terms of f_0, f and h .

Chapter 17

Principal Components Analysis

Principal components analysis (PCA) is one of a family of techniques for taking high-dimensional data, and using the dependencies between the variables to represent it in a more tractable, lower-dimensional form, without losing too much information. PCA is one of the simplest and most robust ways of doing such **dimensionality reduction**. It is also one of the oldest, and has been rediscovered many times in many fields, so it is also known as the Karhunen-Loëve transformation, the Hotelling transformation, the method of empirical orthogonal functions, and singular value decomposition¹. We will call it PCA.

17.1 Mathematics of Principal Components

We start with p -dimensional vectors, and want to summarize them by projecting down into a q -dimensional subspace. Our summary will be the projection of the original vectors on to q directions, the **principal components**, which span the subspace.

There are several equivalent ways of deriving the principal components mathematically. The simplest one is by finding the projections which maximize the variance. The first principal component is the direction in space along which projections have the largest variance. The second principal component is the direction which maximizes variance among all directions orthogonal to the first. The k^{th} component is the variance-maximizing direction orthogonal to the previous $k - 1$ components. There are p principal components in all.

Rather than maximizing variance, it might sound more plausible to look for the projection with the smallest average (mean-squared) distance between the original vectors and their projections on to the principal components; this turns out to be equivalent to maximizing the variance.

Throughout, assume that the data have been “centered”, so that every variable has mean 0. If we write the centered data in a matrix \mathbf{x} , where rows are objects and

¹Strictly speaking, singular value decomposition is a matrix algebra trick which is used in the most common algorithm for PCA.

columns are variables, then $\mathbf{x}^T \mathbf{x} = n\mathbf{v}$, where \mathbf{v} is the covariance matrix of the data. (You should check that last statement!)

17.1.1 Minimizing Projection Residuals

We'll start by looking for a one-dimensional projection. That is, we have p -dimensional vectors, and we want to project them on to a line through the origin. We can specify the line by a unit vector along it, \vec{w} , and then the projection of a data vector \vec{x}_i on to the line is $\vec{x}_i \cdot \vec{w}$, which is a scalar. (Sanity check: this gives us the right answer when we project on to one of the coordinate axes.) This is the distance of the projection from the origin; the actual coordinate in p -dimensional space is $(\vec{x}_i \cdot \vec{w})\vec{w}$. The mean of the projections will be zero, because the mean of the vectors \vec{x}_i is zero:

$$\frac{1}{n} \sum_{i=1}^n (\vec{x}_i \cdot \vec{w})\vec{w} = \left(\left(\frac{1}{n} \sum_{i=1}^n \vec{x}_i \right) \cdot \vec{w} \right) \vec{w} \quad (17.1)$$

If we try to use our projected or **image** vectors instead of our original vectors, there will be some error, because (in general) the images do not coincide with the original vectors. (When do they coincide?) The difference is the error or **residual** of the projection. How big is it? For any one vector, say \vec{x}_i , it's

$$\|\vec{x}_i - (\vec{w} \cdot \vec{x}_i)\vec{w}\|^2 = (\vec{x}_i - (\vec{w} \cdot \vec{x}_i)\vec{w}) \cdot (\vec{x}_i - (\vec{w} \cdot \vec{x}_i)\vec{w}) \quad (17.2)$$

$$= \vec{x}_i \cdot \vec{x}_i - \vec{x}_i \cdot (\vec{w} \cdot \vec{x}_i)\vec{w} \quad (17.3)$$

$$- (\vec{w} \cdot \vec{x}_i)\vec{w} \cdot \vec{x}_i + (\vec{w} \cdot \vec{x}_i)\vec{w} \cdot (\vec{w} \cdot \vec{x}_i)\vec{w} \quad (17.4)$$

$$= \|\vec{x}_i\|^2 - 2(\vec{w} \cdot \vec{x}_i)^2 + (\vec{w} \cdot \vec{x}_i)^2 \vec{w} \cdot \vec{w} \quad (17.4)$$

$$= \vec{x}_i \cdot \vec{x}_i - (\vec{w} \cdot \vec{x}_i)^2 \quad (17.5)$$

since $\vec{w} \cdot \vec{w} = \|\vec{w}\|^2 = 1$. Add those residuals up across all the vectors:

$$MSE(\vec{w}) = \frac{1}{n} \sum_{i=1}^n \|\vec{x}_i\|^2 - (\vec{w} \cdot \vec{x}_i)^2 \quad (17.6)$$

$$= \frac{1}{n} \left(\sum_{i=1}^n \|\vec{x}_i\|^2 - \sum_{i=1}^n (\vec{w} \cdot \vec{x}_i)^2 \right) \quad (17.7)$$

The first summation doesn't depend on \vec{w} , so it doesn't matter for trying to minimize the mean squared residual. To make the MSE small, what we must do is make the second sum big, i.e., we want to maximize

$$\frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \vec{x}_i)^2 \quad (17.8)$$

which we can see is the sample mean of $(\vec{w} \cdot \vec{x}_i)^2$. The mean of a square is always equal to the square of the mean plus the variance:

$$\frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \vec{x}_i)^2 = \left(\frac{1}{n} \sum_{i=1}^n \vec{x}_i \cdot \vec{w} \right)^2 + \text{Var} [\vec{w} \cdot \vec{x}_i] \quad (17.9)$$

Since we've just seen that the mean of the projections is zero, minimizing the residual sum of squares turns out to be equivalent to maximizing the variance of the projections.

(Of course in general we don't want to project on to just one vector, but on to multiple principal components. If those components are orthogonal and have the unit vectors $\vec{w}_1, \vec{w}_2, \dots, \vec{w}_k$, then the image of x_i is its projection into the space spanned by these vectors,

$$\sum_{j=1}^k (\vec{x}_i \cdot \vec{w}_j) \vec{w}_j \quad (17.10)$$

The mean of the projection on to each component is still zero. If we go through the same algebra for the mean squared error, it turns [Exercise 2] out that the cross-terms between different components all cancel out, and we are left with trying to maximize the sum of the variances of the projections on to the components.)

17.1.2 Maximizing Variance

Accordingly, let's maximize the variance! Writing out all the summations grows tedious, so let's do our algebra in matrix form. If we stack our n data vectors into an $n \times p$ matrix, \mathbf{x} , then the projections are given by $\mathbf{x}\mathbf{w}$, which is an $n \times 1$ matrix. The variance is

$$\sigma_{\vec{w}}^2 = \frac{1}{n} \sum_i (\vec{x}_i \cdot \vec{w})^2 \quad (17.11)$$

$$= \frac{1}{n} (\mathbf{x}\mathbf{w})^T (\mathbf{x}\mathbf{w}) \quad (17.12)$$

$$= \frac{1}{n} \mathbf{w}^T \mathbf{x}^T \mathbf{x}\mathbf{w} \quad (17.13)$$

$$= \mathbf{w}^T \frac{\mathbf{x}^T \mathbf{x}}{n} \mathbf{w} \quad (17.14)$$

$$= \mathbf{w}^T \mathbf{v}\mathbf{w} \quad (17.15)$$

We want to chose a unit vector \vec{w} so as to maximize $\sigma_{\vec{w}}^2$. To do this, we need to make sure that we only look at unit vectors — we need to constrain the maximization. The constraint is that $\vec{w} \cdot \vec{w} = 1$, or $\mathbf{w}^T \mathbf{w} = 1$. To enforce this constraint, we introduce a Lagrange multiplier λ (Appendix E) and do a larger unconstrained optimization:

$$\mathcal{L}(\mathbf{w}, \lambda) \equiv \sigma_{\mathbf{w}}^2 - \lambda(\mathbf{w}^T \mathbf{w} - 1) \quad (17.16)$$

$$\frac{\partial L}{\partial \lambda} = \mathbf{w}^T \mathbf{w} - 1 \quad (17.17)$$

$$\frac{\partial L}{\partial \mathbf{w}} = 2\mathbf{v}\mathbf{w} - 2\lambda\mathbf{w} \quad (17.18)$$

Setting the derivatives to zero at the optimum, we get

$$\mathbf{w}^T \mathbf{w} = 1 \quad (17.19)$$

$$\mathbf{v}\mathbf{w} = \lambda\mathbf{w} \quad (17.20)$$

Thus, desired vector \mathbf{w} is an **eigenvector** of the covariance matrix \mathbf{v} , and the maximizing vector will be the one associated with the largest **eigenvalue** λ . This is good news, because finding eigenvectors is something which can be done comparatively rapidly, and because eigenvectors have many nice mathematical properties, which we can use as follows.

We know that \mathbf{v} is a $p \times p$ matrix, so it will have p different eigenvectors.² We know that \mathbf{v} is a covariance matrix, so it is symmetric, and then linear algebra tells us that the eigenvectors must be orthogonal to one another. Again because \mathbf{v} is a covariance matrix, it is a **positive matrix**, in the sense that $\vec{x} \cdot \mathbf{v}\vec{x} \geq 0$ for any \vec{x} . This tells us that the eigenvalues of \mathbf{v} must all be ≥ 0 .

The eigenvectors of \mathbf{v} are the **principal components** of the data. We know that they are all orthogonal to each other from the previous paragraph, so together they span the whole p -dimensional space. The first principal component, i.e. the eigenvector which goes the largest value of λ , is the direction along which the data have the most variance. The second principal component, i.e. the second eigenvector, is the direction orthogonal to the first component with the most variance. Because it is orthogonal to the first eigenvector, their projections will be uncorrelated. In fact, projections on to all the principal components are uncorrelated with each other. If we use q principal components, our weight matrix \mathbf{w} will be a $p \times q$ matrix, where each column will be a different eigenvector of the covariance matrix \mathbf{v} . The eigenvalues will give the total variance described by each component. The variance of the projections on to the first q principal components is then $\sum_{i=1}^q \lambda_i$.

17.1.3 More Geometry; Back to the Residuals

Suppose that the data really are q -dimensional. Then \mathbf{v} will have only q positive eigenvalues, and $p - q$ zero eigenvalues. If the data fall near a q -dimensional subspace, then $p - q$ of the eigenvalues will be nearly zero.

If we pick the top q components, we can define a projection operator \mathbf{P}_q . The images of the data are then $\mathbf{x}\mathbf{P}_q$. The **projection residuals** are $\mathbf{x} - \mathbf{x}\mathbf{P}_q$ or $\mathbf{x}(\mathbf{I} - \mathbf{P}_q)$. (Notice that the residuals here are vectors, not just magnitudes.) If the data really are q -dimensional, then the residuals will be zero. If the data are *approximately* q -dimensional, then the residuals will be small. In any case, we can define the R^2 of the projection as the fraction of the original variance kept by the image vectors,

$$R^2 \equiv \frac{\sum_{i=1}^q \lambda_i}{\sum_{j=1}^p \lambda_j} \quad (17.21)$$

just as the R^2 of a linear regression is the fraction of the original variance of the dependent variable kept by the fitted values.

²Exception: if $n < p$, there are only n distinct eigenvectors and eigenvalues.

The $q = 1$ case is especially instructive. We know that the residual vectors are all orthogonal to the projections. Suppose we ask for the first principal component of the residuals. This will be the direction of largest variance which is perpendicular to the first principal component. In other words, it will be the second principal component of the data. This suggests a recursive algorithm for finding all the principal components: the k^{th} principal component is the leading component of the residuals after subtracting off the first $k - 1$ components. In practice, it is faster to use eigenvector-solvers to get all the components at once from \mathbf{v} , but this idea is correct in principle.

This is a good place to remark that if the data really fall in a q -dimensional subspace, then \mathbf{v} will have only q positive eigenvalues, because after subtracting off those components there will be no residuals. The other $p - q$ eigenvectors will all have eigenvalue 0. If the data cluster around a q -dimensional subspace, then $p - q$ of the eigenvalues will be very small, though how small they need to be before we can neglect them is a tricky question.³

Projections on to the first two or three principal components can be visualized; however they may not be enough to really give a good summary of the data. Usually, to get an R^2 of 1, you need to use all p principal components.⁴ How many principal components you should use depends on your data, and how big an R^2 you need. In some fields, you can get better than 80% of the variance described with just two or three components. A sometimes-useful device is to plot $1 - R^2$ versus the number of components, and keep extending the curve until it flattens out.

17.1.4 Statistical Inference, or Not

You may have noticed, and even been troubled by, the fact that I have said nothing at all yet like “assume the data are drawn at random from some distribution”, or “assume the different rows of the data frame are statistically independent”. This is because no such assumption is required for principal components. All it does is say “these data can be summarized using projections along these directions”. It says nothing about the larger population or stochastic process the data came from; it doesn’t even suppose the latter exist.

However, we could *add* a statistical assumption and see how PCA behaves under those conditions. The simplest one is to suppose that the data are IID draws from a distribution with covariance matrix \mathbf{V}_0 . Then the sample covariance matrix $\mathbf{V} \equiv \frac{1}{n} \mathbf{X}^T \mathbf{X}$ will converge on \mathbf{V}_0 as $n \rightarrow \infty$. Since the principal components are smooth functions of \mathbf{V} (namely its eigenvectors), they will tend to converge as n grows⁵. So,

³Be careful when $n < p$. Any two points define a line, and three points define a plane, etc., so if there are fewer data points than variables, it is *necessarily* true that the fall on a low-dimensional subspace. In §17.3.1, we represent stories in the *New York Times* as vectors with $p \approx 440$, but $n = 102$. Finding that only 102 principal components keep all the variance is not an empirical discovery but a mathematical artifact.

⁴The exceptions are when some of your variables are linear combinations of the others, so that you don’t really have p *different* variables, or when $n < p$.

⁵There is a wrinkle if \mathbf{V}_0 has “degenerate” eigenvalues, i.e., two or more eigenvectors with the same eigenvalue. Then any linear combination of those vectors is also an eigenvector, with the same eigenvalue (Exercise 3.) For instance, if \mathbf{V}_0 is the identity matrix, then every vector is an eigenvector, and PCA

Variable	Meaning
Sports	Binary indicator for being a sports car
SUV	Indicator for sports utility vehicle
Wagon	Indicator
Minivan	Indicator
Pickup	Indicator
AWD	Indicator for all-wheel drive
RWD	Indicator for rear-wheel drive
Retail	Suggested retail price (US\$)
Dealer	Price to dealer (US\$)
Engine	Engine size (liters)
Cylinders	Number of engine cylinders
Horsepower	Engine horsepower
CityMPG	City gas mileage
HighwayMPG	Highway gas mileage
Weight	Weight (pounds)
Wheelbase	Wheelbase (inches)
Length	Length (inches)
Width	Width (inches)

Table 17.1: Features for the 2004 cars data.

along with that additional assumption about the data-generating process, PCA does make a prediction: in the future, the principal components will look like they do now.

17.2 Example: Cars

Let's work an example. The data⁶ consists of 388 cars from the 2004 model year, with 18 features. Eight features are binary indicators; the other 11 features are numerical (Table 17.1). All of the features except Type are numerical. Table 17.2 shows the first few lines from the data set. PCA only works with numerical variables, so we have ten of them to play with.

There are *two* R functions for doing PCA, `princomp` and `prcomp`, which differ in how they do the actual calculation.⁷ The latter is generally more robust, so we'll just use it.

```
cars04 = read.csv("cars-fixed04.dat")
cars04.pca = prcomp(cars04[,8:18], scale.=TRUE)
```

routines will return an essentially arbitrary collection of mutually perpendicular vectors. Generically, however, any arbitrarily small tweak to V_0 will break the degeneracy.

⁶On the course website; from <http://www.amstat.org/publications/jse/datasets/04cars.txt>, with incomplete records removed.

⁷`princomp` actually calculates the covariance matrix and takes its eigenvalues. `prcomp` uses a different technique called “singular value decomposition”.

```

Sports, SUV, Wagon, Minivan, Pickup, AWD, RWD, Retail, Dealer, Engine,
Cylinders, Horsepower, CityMPG, HighwayMPG, Weight, Wheelbase, Length, Width
Acura 3.5 RL, 0, 0, 0, 0, 0, 0, 0, 43755, 39014, 3.5, 6, 225, 18, 24, 3880, 115, 197, 72
Acura MDX, 0, 1, 0, 0, 0, 1, 0, 36945, 33337, 3.5, 6, 265, 17, 23, 4451, 106, 189, 77
Acura NSX S, 1, 0, 0, 0, 0, 0, 1, 89765, 79978, 3.2, 6, 290, 17, 24, 3153, 100, 174, 71

```

Table 17.2: The first few lines of the 2004 cars data set.

The second argument to `prcomp` tells it to first scale all the variables to have variance 1, i.e., to standardize them. You should experiment with what happens with this data when we don't standardize.

We can now extract the loadings or weight matrix from the `cars04.pca` object. For comprehensibility I'll just show the first two components.

```

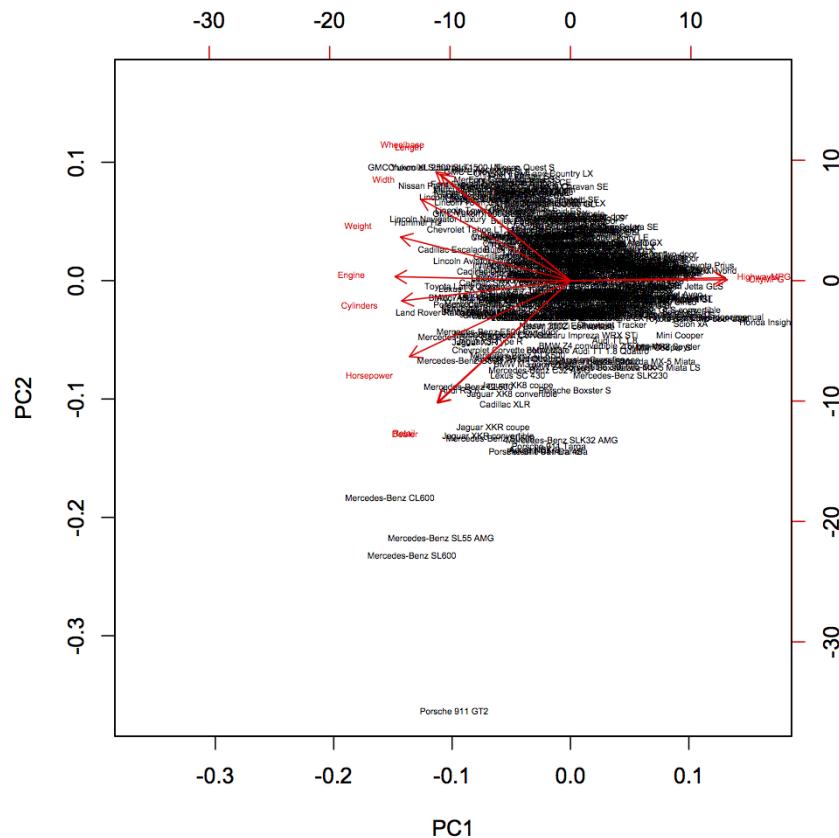
> round(cars04.pca$rotation[,1:2], 2)
      PC1    PC2
Retail   -0.26 -0.47
Dealer   -0.26 -0.47
Engine   -0.35  0.02
Cylinders -0.33 -0.08
Horsepower -0.32 -0.29
CityMPG    0.31  0.00
HighwayMPG 0.31  0.01
Weight     -0.34  0.17
Wheelbase   -0.27  0.42
Length     -0.26  0.41
Width      -0.30  0.31

```

This says that all the variables *except* the gas-mileages have a negative projection on to the first component. This means that there is a negative correlation between mileage and everything else. The first principal component tells us about whether we are getting a big, expensive gas-guzzling car with a powerful engine, or whether we are getting a small, cheap, fuel-efficient car with a wimpy engine.

The second component is a little more interesting. Engine size and gas mileage hardly project on to it at all. Instead we have a contrast between the physical size of the car (positive projection) and the price and horsepower. Basically, this axis separates mini-vans, trucks and SUVs (big, not so expensive, not so much horsepower) from sports-cars (small, expensive, lots of horse-power).

To check this interpretation, we can use a useful tool called a **biplot**, which plots the data, along with the projections of the original variables, on to the first two components (Figure 17.1). Notice that the car with the lowest value of the second component is a Porsche 911, with pick-up trucks and mini-vans at the other end of the scale. Similarly, the highest values of the first component all belong to hybrids.



```
biplot(cars04.pca, cex=0.4)
```

Figure 17.1: “Biplot” of the 2004 cars data. The horizontal axis shows projections on to the first principal component, the vertical axis the second component. Car names are written at their projections on to the components (using the coordinate scales on the top and the right). Red arrows show the projections of the original variables on to the principal components (using the coordinate scales on the bottom and on the left).

17.3 Latent Semantic Analysis

Information retrieval systems (like search engines) and people doing computational text analysis often represent documents as what are called **bags of words**: documents are represented as vectors, where each component counts how many times each word in the dictionary appears in the text. This throws away information about word order, but gives us something we can work with mathematically. Part of the representation of one document might look like:

a	abandoned	abc	ability	able	about	above	abroad	absorbed	absorbing	abstract
43	0	0	0	0	10	0	0	0	0	1

and so on through to “zebra”, “zoology”, “zygote”, etc. to the end of the dictionary. These vectors are very, very large! At least in English and similar languages, these bag-of-word vectors have three outstanding properties:

1. Most words do not appear in most documents; the bag-of-words vectors are very **sparse** (most entries are zero).
2. A small number of words appear many times in almost all documents; these words tell us almost nothing about what the document is about. (Examples: “the”, “is”, “of”, “for”, “at”, “a”, “and”, “here”, “was”, etc.)
3. Apart from those hyper-common words, most words’ counts are correlated with some but not all other words; words tend to come in bunches which appear together.

Taken together, this suggests that we do not really get a lot of value from keeping around *all* the words. We would be better off if we could project down a smaller number of new variables, which we can think of as combinations of words that tend to appear together in the documents, or not at all. But this tendency needn’t be absolute — it can be partial because the words mean slightly different things, or because of stylistic differences, etc. This is *exactly* what principal components analysis does.

To see how this can be useful, imagine we have a collection of documents (a **corpus**), which we want to search for documents about agriculture. It’s entirely possible that many documents on this topic don’t actually contain the *word* “agriculture”, just closely related words like “farming”. A simple search on “agriculture” will miss them. But it’s very likely that the occurrence of these related words is well-correlated with the occurrence of “agriculture”. This means that all these words will have similar projections on to the principal components, and will be easy to find documents whose principal components projection is like that for a query about agriculture. This is called **latent semantic indexing**.

To see why this is *indexing*, think about what goes into coming up with an index for a book by hand. Someone draws up a list of topics and then goes through the book noting all the passages which refer to the topic, and maybe a little bit of what they say there. For example, here’s the start of the entry for “Agriculture” in the index to Adam Smith’s *The Wealth of Nations*:

AGRICULTURE, the labour of, does not admit of such subdivisions as manufactures, 6; this impossibility of separation, prevents agriculture from improving equally with manufactures, 6; natural state of, in a new colony, 92; requires more knowledge and experience than most mechanical professions, and yet is carried on without any restrictions, 127; the terms of rent, how adjusted between landlord and tenant, 144; is extended by good roads and navigable canals, 147; under what circumstances pasture land is more valuable than arable, 149; gardening not a very gainful employment, 152–3; vines the most profitable article of culture, 154; estimates of profit from projects, very fallacious, *ib.*; cattle and tillage mutually improve each other, 220; ...

and so on. (Agriculture is an important topic in *The Wealth of Nations*.) It's asking a lot to hope for a computer to be able to do something like this, but we could at least hope for a list of pages like "6, 92, 126, 144, 147, 152 – –3, 154, 220, ...". One could imagine doing this by treating each page as its own document, forming its bag-of-words vector, and then returning the list of pages with a non-zero entry for "agriculture". This will fail: only two of those nine pages actually contains that word, and this is pretty typical. On the other hand, they are full of words strongly correlated with "agriculture", so asking for the pages which are most similar in their principal components projection to that word will work great.⁸

At first glance, and maybe even second, this seems like a wonderful trick for extracting meaning, or **semantics**, from pure correlations. Of course there are also all sorts of ways it can fail, not least from spurious correlations. If our training corpus happens to contain lots of documents which mention "farming" and "Kansas", as well as "farming" and "agriculture", latent semantic indexing will not make a big distinction between the relationship between "agriculture" and "farming" (which is genuinely semantic) and that between "Kansas" and "farming" (which is accidental, and probably wouldn't show up in, say, a corpus collected from Europe).

Despite this susceptibility to spurious correlations, latent semantic indexing is an *extremely* useful technique in practice, and the foundational papers (Deerwester *et al.*, 1990; Landauer and Dumais, 1997) are worth reading.

17.3.1 Principal Components of the New York *Times*

To get a more concrete sense of how latent semantic analysis works, and how it reveals semantic information, let's apply it to some data. The accompanying R file and R workspace contains some news stories taken from the New York *Times* Annotated Corpus (Sandhaus, 2008), which consists of about 1.8 million stories from the *Times*, from 1987 to 2007, which have been hand-annotated by actual human beings with standardized machine-readable information about their contents. From this corpus, I have randomly selected 57 stories about art and 45 stories about music, and turned them into a bag-of-words data frame, one row per story, one column per word; plus an indicator in the first column of whether the story is one about art or one about

⁸Or it should anyway; I haven't actually done the experiment with this book.

music.⁹ The original data frame thus has 102 rows, and 4432 columns: the categorical label, and 4431 columns with counts for every distinct word that appears in at least one of the stories.¹⁰

The PCA is done as it would be for any other data:

```
nyt.pca <- prcomp(nyt.frame[,-1])
nyt.latent.sem <- nyt.pca$rotation
```

We need to omit the first column in the first command because it contains categorical variables, and PCA doesn't apply to them. The second command just picks out the matrix of projections of the variables on to the components — this is called `rotation` because it can be thought of as rotating the coordinate axes in feature-vector space.

Now that we've done this, let's look at what the leading components are.

```
> signif(sort(nyt.latent.sem[,1],decreasing=TRUE)[1:30],2)
   music      trio     theater    orchestra   composers      opera
   0.110      0.084     0.083      0.067      0.059      0.058
  theaters      m     festival      east      program      y
   0.055      0.054     0.051      0.049      0.048      0.048
 jersey     players committee     sunday      june      concert
   0.047      0.047     0.046      0.045      0.045      0.045
 symphony     organ matinee misstated instruments      p
   0.044      0.044     0.043      0.042      0.041      0.041
 X.d        april   samuel      jazz      pianist society
   0.041      0.040     0.040      0.039      0.038      0.038
> signif(sort(nyt.latent.sem[,1],decreasing=FALSE)[1:30],2)
   she      her      ms       i      said      mother      cooper
  -0.260    -0.240    -0.200    -0.150    -0.130    -0.110    -0.100
   my painting process paintings      im      he      mrs
  -0.094    -0.088    -0.071    -0.070    -0.068    -0.065    -0.065
   me gagosian      was      picasso      image sculpture baby
  -0.063    -0.062    -0.058    -0.057    -0.056    -0.056    -0.055
 artists     work      photos      you      nature      studio      out
  -0.055    -0.054    -0.051    -0.051    -0.050    -0.050    -0.050
   says      like
  -0.050    -0.049
```

These are the thirty words with the largest positive and negative projections on to the first component.¹¹ The words with positive projections are mostly associated with music, those with negative components with the visual arts. The letters "m" and "p"

⁹Actually, following standard practice in language processing, I've normalized the bag-of-word vectors so that documents of different lengths are comparable, and used "inverse document-frequency weighting" to de-emphasize hyper-common words like "the" and emphasize more informative words. See the lecture notes for data mining if you're interested.

¹⁰If we were trying to work with the complete corpus, we should expect at least 50000 words, and perhaps more.

¹¹Which direction is positive and which negative is of course arbitrary; basically it depends on internal choices in the algorithm.

show up with msuic because of the combination “p.m”, which our parsing breaks into two single-letter words, and because stories about music give show-times more often than do stories about art. Personal pronouns appear with art stories because more of those quote people, such as artists or collectors.¹²

What about the second component?

```
> signif(sort(nyt.latent.sem[,2],decreasing=TRUE)[1:30],2)
      art      museum     images    artists   donations   museums
  0.150      0.120     0.095     0.092     0.075     0.073
  painting      tax  paintings  sculpture  gallery  sculptures
  0.073      0.070     0.065     0.060     0.055     0.051
  painted      white  patterns   artist   nature   service
  0.050      0.050     0.047     0.047     0.046     0.046
decorative      feet   digital   statue   color   computer
  0.043      0.043     0.043     0.042     0.042     0.041
  paris       war collections   diamond   stone   dealers
  0.041      0.041     0.041     0.041     0.041     0.040
> signif(sort(nyt.latent.sem[,2],decreasing=FALSE)[1:30],2)
      her      she theater     opera      ms
 -0.220     -0.220    -0.160    -0.130    -0.130
      i      hour production   sang   festival
 -0.083     -0.081    -0.075    -0.075    -0.074
  music      musical    songs   vocal orchestra
 -0.070     -0.070    -0.068    -0.067    -0.067
      la      singing matinee performance   band
 -0.065     -0.065    -0.061    -0.061    -0.060
 awards      composers   says      my      im
 -0.058     -0.058    -0.058    -0.056    -0.056
  play      broadway   singer   cooper performances
 -0.056     -0.055    -0.052    -0.051    -0.051
```

Here the positive words are about art, but more focused on acquiring and trading (“collections”, “dealers”, “donations”, “dealers”) than on talking with artists or about them. The negative words are musical, specifically about musical theater and vocal performances.

I could go on, but by this point you get the idea.

17.4 PCA for Visualization

Let’s try displaying the *Times* stories using the principal components. (Assume that the objects from just before are still in memory.)

```
plot(nyt.pca$x[,1:2],type="n")
points(nyt.pca$x[nyt.frame[,"class.labels"]=="music",1:2],pch="m",col="blue")
points(nyt.pca$x[nyt.frame[,"class.labels"]=="art",1:2],pch="a",col="red")
```

¹²You should check out these explanations for yourself. The raw stories are part of the R workspace.

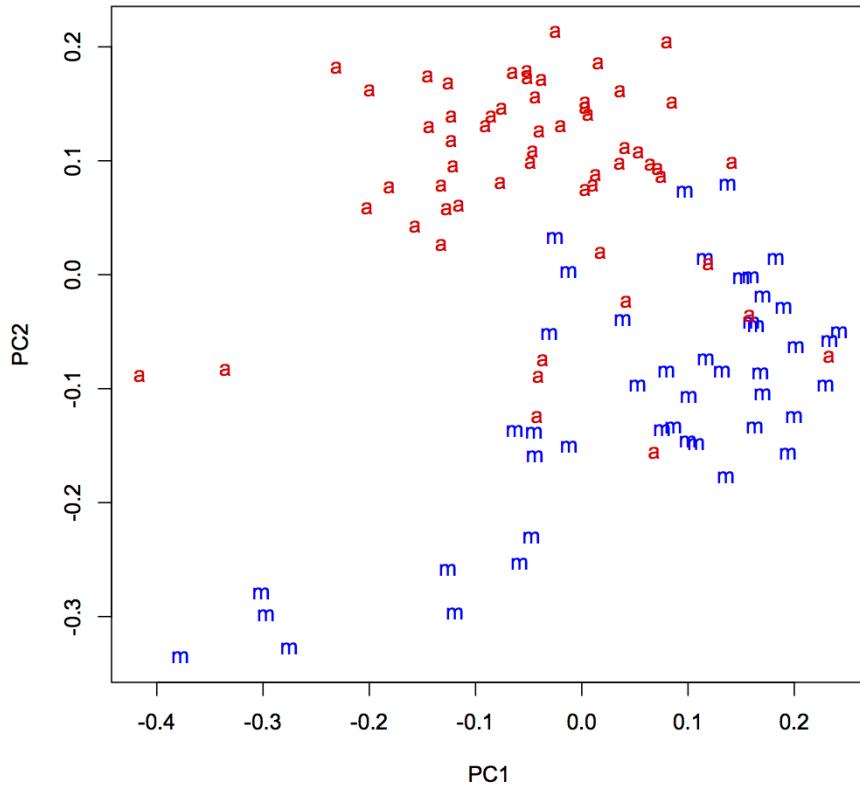


Figure 17.2: Projection of the *Times* stories on to the first two principal components. Labels: “a” for art stories, “m” for music.

The first command makes an empty plot — I do this just to set up the axes nicely for the data which will actually be displayed. The second and third commands plot a blue “m” at the location of each music story, and a red “a” at the location of each art story. The result is Figure 17.2.

Notice that even though we have gone from 4431 dimensions to 2, and so thrown away a lot of information, we could draw a line across this plot and have most of the art stories on one side of it and all the music stories on the other. If we let ourselves use the first four or five principal components, we’d still have a thousand-fold savings in dimensions, but we’d be able to get almost-perfect separation between the two classes. This is a sign that PCA is really doing a good job at summarizing the information in the word-count vectors, and in turn that the bags of words give us a lot of information about the meaning of the stories.

The figure also illustrates the idea of **multidimensional scaling**, which means finding low-dimensional points to represent high-dimensional data by preserving the distances between the points. If we write the original vectors as $\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n$, and their images as $\vec{y}_1, \vec{y}_2, \dots, \vec{y}_n$, then the MDS problem is to pick the images to minimize the difference in distances:

$$\sum_i \sum_{j \neq i} (\|\vec{y}_i - \vec{y}_j\| - \|\vec{x}_i - \vec{x}_j\|)^2 \quad (17.22)$$

This will be small if distances between the image points are all close to the distances between the original points. PCA accomplishes this precisely because \vec{y}_i is itself close to \vec{x}_i (on average).

17.5 PCA Cautions

Trying to guess at what the components might mean is a good idea, but like many good ideas it's easy to go overboard. Specifically, once you attach an idea in your mind to a component, and especially once you attach a *name* to it, it's very easy to forget that those are names and ideas you made up; to **reify** them, as you might reify clusters. Sometimes the components actually do measure real variables, but sometimes they just reflect patterns of covariance which have many different causes. If I did a PCA of the same variables but for, say, European cars, I might well get a similar first component, but the second component would probably be rather different, since SUVs are much less common there than here.

A more important example comes from population genetics. Starting in the late 1960s, L. L. Cavalli-Sforza and collaborators began a huge project of mapping human genetic variation — of determining the frequencies of different genes in different populations throughout the world. (Cavalli-Sforza *et al.* (1994) is the main summary; Cavalli-Sforza has also written several excellent popularizations.) For each point in space, there are a very large number of variables, which are the frequencies of the various genes among the people living there. Plotted over space, this gives a map of that gene's frequency. What they noticed (unsurprisingly) is that many genes had similar, but not identical, maps. This led them to use PCA, reducing the huge number of variables (genes) to a few components. Results look like Figure 17.3. They interpreted these components, very reasonably, as signs of large population movements. The first principal component for Europe and the Near East, for example, was supposed to show the expansion of agriculture out of the Fertile Crescent. The third, centered in steppes just north of the Caucasus, was supposed to reflect the expansion of Indo-European speakers towards the end of the Bronze Age. Similar stories were told of other components elsewhere.

Unfortunately, as Novembre and Stephens (2008) showed, spatial patterns like this are what one should expect to get when doing PCA of any kind of spatial data with local correlations, because that essentially amounts to taking a Fourier transform, and picking out the low-frequency components.¹³ They simulated genetic diffusion processes, without any migration or population expansion, and got results that

¹³Remember that PCA re-writes the original vectors as a weighted sum of new, orthogonal vectors, just

looked very like the real maps (Figure 17.4). This doesn't mean that the stories of the maps *must be* wrong, but it does undercut the principal components as evidence for those stories.

17.6 Exercises

1. Step through the `pca.R` file on the class website. Then replicate the analysis of the cars data given above.
2. Suppose that we use q directions, given by q orthogonal length-one vectors $\vec{w}_1, \dots, \vec{w}_q$. We want to show that minimizing the mean squared error is equivalent to maximizing the sum of the variances of the scores along these directions.
 - (a) Write \mathbf{w} for the matrix forms by stacking the \vec{w}_i . Prove that $\mathbf{w}^T \mathbf{w} = \mathbf{I}_q$.
 - (b) Find the matrix of q -dimensional scores in terms of \mathbf{x} and \mathbf{w} . *Hint:* your answer should reduce to $\vec{x}_i \cdot \vec{w}_1$ when $q = 1$.
 - (c) Find the matrix of p -dimensional approximations based on these scores in terms of \mathbf{x} and \mathbf{w} . *Hint:* your answer should reduce to $(\vec{x}_i \cdot \vec{w}_1)\vec{w}_1$ when $q = 1$.
 - (d) Show that the MSE of using the vectors $\vec{w}_1, \dots, \vec{w}_q$ is the sum of two terms, one of which depends only on \mathbf{x} and not \mathbf{w} , and the other depends only on the scores along those directions (and not otherwise on what those directions are). *Hint:* look at the derivation of Eq. 17.5, and use Exercise 2a.
 - (e) Explain in what sense minimizing projection residuals is equivalent to maximizing the sum of variances along the different directions.
3. Suppose that \mathbf{u} has two eigenvectors, \vec{w}_1 and \vec{w}_2 , with the same eigenvalue a . Prove that any linear combination of \vec{w}_1 and \vec{w}_2 is also an eigenvector of \mathbf{u} , and also has eigenvalue a .

as Fourier transforms do. When there is a lot of spatial correlation, values at nearby points are similar, so the low-frequency modes will have a lot of amplitude, i.e., carry a lot of the variance. So first principal components will tend to be similar to the low-frequency Fourier modes.

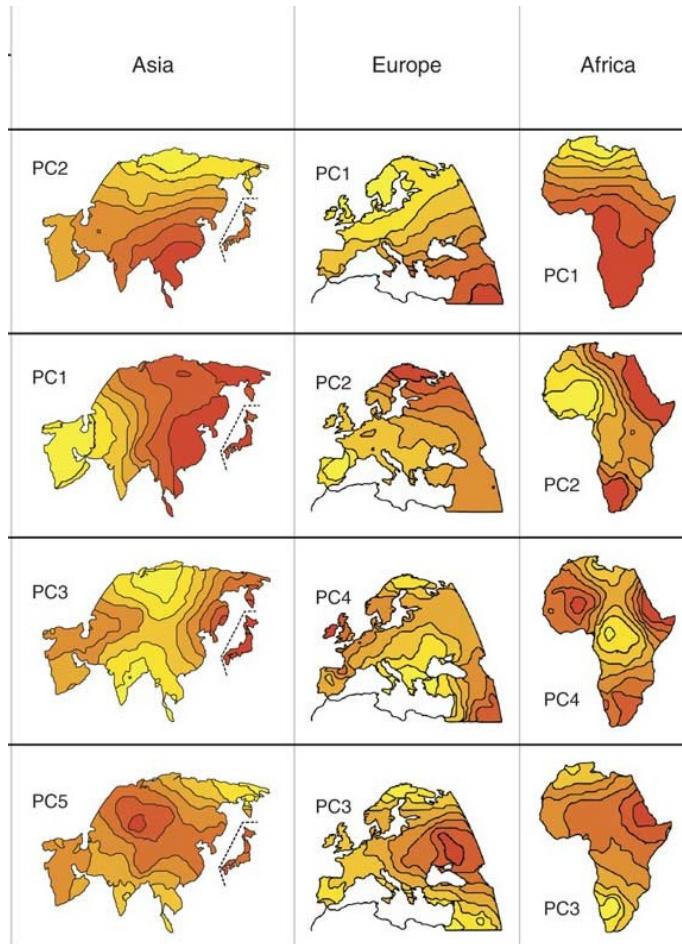


Figure 17.3: Principal components of genetic variation in the old world, according to Cavalli-Sforza *et al.* (1994), as re-drawn by Novembre and Stephens (2008).

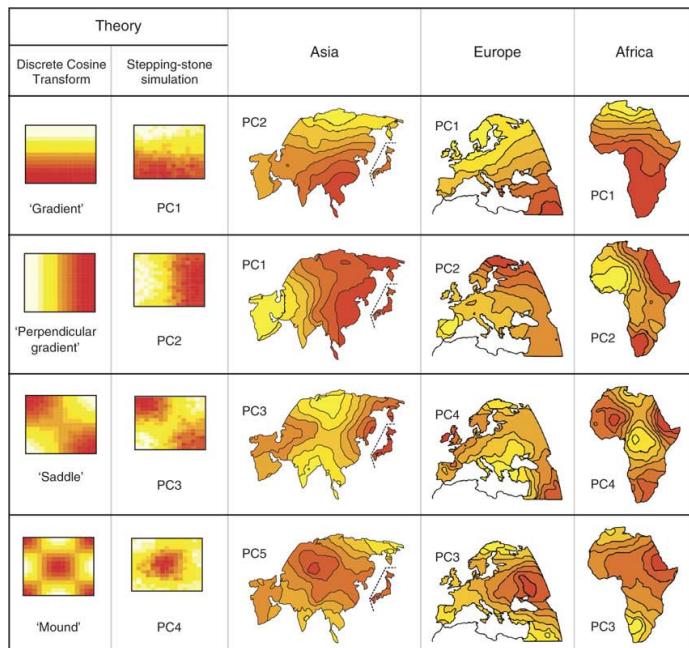


Figure 17.4: How the PCA patterns can arise as numerical artifacts (far left column) or through simple genetic diffusion (next column). From Novembre and Stephens (2008).

Chapter 18

Factor Analysis

18.1 From PCA to Factor Analysis

Let's sum up PCA. We start with n different p -dimensional vectors as our data, i.e., each observation as p numerical variables. We want to reduce the number of dimensions to something more manageable, say q . The principal components of the data are the q orthogonal directions of greatest variance in the original p -dimensional space; they can be found by taking the top q eigenvectors of the sample covariance matrix. Principal components analysis summarizes the data vectors by projecting them on to the principal components.

All of this is purely an algebraic undertaking; it involves no probabilistic assumptions whatsoever. It also supports no statistical inferences — saying nothing about the population or stochastic process which made the data, it just summarizes the data. How can we add some probability, and so some statistics? And what does that let us do?

Start with some notation. \mathbf{X} is our data matrix, with n rows for the different observations and p columns for the different variables, so X_{ij} is the value of variable j in observation i . Each principal component is a vector of length p , and there are p of them, so we can stack them together into a $p \times p$ matrix, say \mathbf{w} . Finally, each data vector has a projection on to each principal component, which we collect into an $n \times p$ matrix \mathbf{F} . Then

$$\begin{aligned} \mathbf{X} &= \mathbf{F}\mathbf{w} \\ [n \times p] &= [n \times p][p \times p] \end{aligned} \tag{18.1}$$

where I've checked the dimensions of the matrices underneath. This is an exact equation involving no noise, approximation or error, but it's kind of useless; we've replaced p -dimensional vectors in \mathbf{X} with p -dimensional vectors in \mathbf{F} . If we keep only to $q < p$ largest principal components, that corresponds to dropping columns from

\mathbf{F} and rows from \mathbf{w} . Let's say that the truncated matrices are \mathbf{F}_q and \mathbf{w}_q . Then

$$\begin{aligned} \mathbf{X} &\approx \mathbf{F}_q \mathbf{w}_q \\ [n \times p] &= [n \times q][q \times p] \end{aligned} \tag{18.2}$$

The error of approximation — the difference between the left- and right-hand-sides of Eq. 18.2 — will get smaller as we increase q . (The line below the equation is a sanity-check that the matrices are the right size, which they are. Also, at this point the subscript qs get too annoying, so I'll drop them.) We can of course make the two sides match exactly by adding an error or residual term on the right:

$$\mathbf{X} = \mathbf{F}\mathbf{w} + \epsilon \tag{18.3}$$

where ϵ has to be an $n \times p$ matrix.

Now, Eq. 18.3 should look more or less familiar to you from regression. On the left-hand side we have a measured outcome variable (\mathbf{X}), and on the right-hand side we have a systematic prediction term ($\mathbf{F}\mathbf{w}$) plus a residual (ϵ). Let's run with this analogy, and start treating ϵ as *noise*, as a random variable which has got some distribution, rather than whatever arithmetic says is needed to balance the two sides. (This move is the difference between just drawing a straight line through a scatter plot, and inferring a linear regression.) Then \mathbf{X} will also be a random variable. When we want to talk about the random variable which goes in the i^{th} column of \mathbf{X} , we'll call it X_i .

What about \mathbf{F} ? Well, in the analogy it corresponds to the independent variables in the regression, which ordinarily we treat as fixed rather than random, but that's because we actually get to observe them; here we don't, so it will make sense to treat \mathbf{F} , too, as random. Now that they are random variables, we say that we have q **factors**, rather than components, that \mathbf{F} is the matrix of **factor scores** and \mathbf{w} is the matrix of **factor loadings**. The variables in \mathbf{X} are called **observable** or **manifest** variables, those in \mathbf{F} are **hidden** or **latent**. (Technically ϵ is also latent.)

Before we can actually do much with this model, we need to say more about the distributions of these random variables. The traditional choices are as follows.

1. All of the observable random variables X_i have mean zero and variance 1.
2. All of the latent factors have mean zero and variance 1.
3. The noise terms ϵ all have mean zero.
4. The factors are uncorrelated across individuals (rows of \mathbf{F}) and across variables (columns).
5. The noise terms are uncorrelated across individuals, and across observable variables.
6. The noise terms are uncorrelated with the factor variables.

Item (1) isn't restrictive, because we can always center and standardize our data. Item (2) isn't restrictive either — we could always center and standardize the factor variables without really changing anything. Item (3) actually follows from (1) and (2). The substantive assumptions — the ones which will give us predictive power but could also go wrong, and so really define the factor model — are the others, about lack of correlation. Where do they come from?

Remember what the model looks like:

$$\mathbf{X} = \mathbf{F}\mathbf{w} + \boldsymbol{\epsilon} \quad (18.4)$$

All of the systematic patterns in the observations \mathbf{X} should come from the first term on the right-hand side. The residual term $\boldsymbol{\epsilon}$ should, if the model is working, be unpredictable noise. Items (3) through (5) express a very strong form of this idea. In particular it's vital that the noise be uncorrelated with the factor scores.

18.1.1 Preserving correlations

There is another route from PCA to the factor model, which many people like but which I find less compelling; it starts by changing the objectives.

PCA aims to minimize the mean-squared distance from the data to their projects, or what comes to the same thing, to preserve variance. But it doesn't preserve correlations. That is, the correlations of the features of the image vectors are not the same as the correlations among the features of the original vectors (unless $q = p$, and we're not really doing any data reduction). We might value those correlations, however, and want to preserve them, rather than trying to approximate the actual data.¹ That is, we might ask for a set of vectors whose image in the feature space will have the same correlation matrix as the original vectors, or as close to the same correlation matrix as possible while still reducing the number of dimensions. This leads to the factor model we've already reached, as we'll see.

18.2 The Graphical Model

It's common to represent factor models visually, as in Figure 18.1. This is an example of a **graphical model**, in which the nodes or vertices of the graph represent random variables, and the edges of the graph represent direct statistical dependencies between the variables. The figure shows the observables or features in square boxes, to indicate that they are manifest variables we can actual measure; above them are the factors, drawn in round bubbles to show that we don't get to see them. The fact that there are no direct linkages between the factors shows that they are independent of one another. From below we have the noise terms, one to an observable.

¹Why? Well, originally the answer was that the correlation coefficient had just been invented, and was about the only way people had of measuring relationships between variables. Since then it's been propagated by statistics courses where it is the only way people are *taught* to measure relationships. The great statistician John Tukey once wrote “Does anyone know when the correlation coefficient is useful, as opposed to when it is used? If so, why not tell us?” (Tukey, 1954, p. 721).

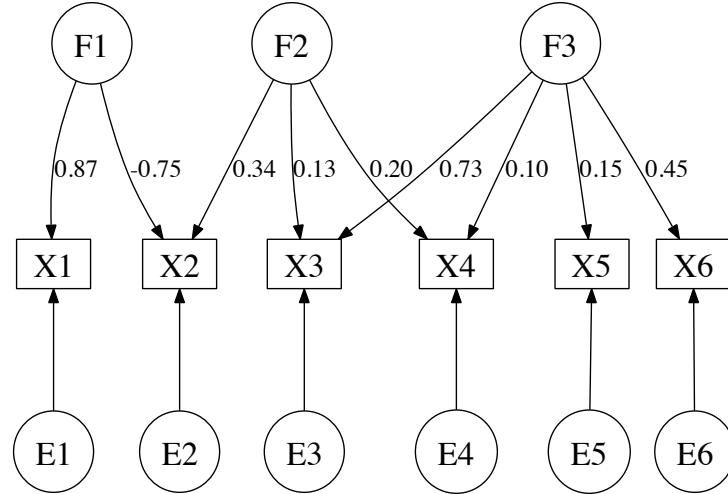


Figure 18.1: Graphical model form of a factor model. Circles stand for the unobserved variables (factors above, noises below), boxes for the observed features. Edges indicate non-zero coefficients — entries in the factor loading matrix \mathbf{w} , or specific variances ψ_i . Arrows representing entries in \mathbf{w} are decorated with those entries. Note that it is common to omit the noise variables in such diagrams, with the implicit understanding that every variable with an incoming arrow also has an incoming noise term.

Notice that not every observable is connected to every factor: this depicts the fact that some entries in \mathbf{w} are zero. In the figure, for instance, X_1 has an arrow only from F_1 and not the other factors; this means that while $w_{11} = 0.87$, $w_{21} = w_{31} = 0$.

Drawn this way, one sees how the factor model is **generative** — how it gives us a recipe for producing new data. In this case, it's: draw new, independent values for the factor scores F_1, F_2, \dots, F_q ; add these up with weights from \mathbf{w} ; and then add on the final noises $\epsilon_1, \epsilon_2, \dots, \epsilon_p$. If the model is right, this is a procedure for generating new, synthetic data with the same characteristics as the real data. In fact, it's a story about how the real data came to be — that there really are some latent variables (the factor scores) which linearly cause the observables to have the values they do.

18.2.1 Observables Are Correlated Through the Factors

One of the most important consequences of the factor model is that observable variables are correlated with each other *solely* because they are correlated with the hidden factors. To see how this works, take X_1 and X_2 from the diagram, and let's calculate their covariance. (Since they both have variance 1, this is the same as their correlation.)

$$\text{Cov}[X_1, X_2] = \mathbf{E}[X_1 X_2] - \mathbf{E}[X_1] \mathbf{E}[X_2] \quad (18.5)$$

$$= \mathbf{E}[X_1 X_2] \quad (18.6)$$

$$= \mathbf{E}[(F_1 w_{11} + F_2 w_{21} + \epsilon_1)(F_1 w_{12} + F_2 w_{22} + \epsilon_2)] \quad (18.7)$$

$$\begin{aligned} &= \mathbf{E}\left[F_1^2 w_{11} w_{12} + F_1 F_2 (w_{11} w_{22} + w_{21} w_{12}) + F_2^2 w_{21} w_{22}\right] \\ &\quad + \mathbf{E}[\epsilon_1 \epsilon_2] + \mathbf{E}[\epsilon_1 (F_1 w_{12} + F_2 w_{22})] \\ &\quad + \mathbf{E}[\epsilon_2 (F_1 w_{11} + F_2 w_{21})] \end{aligned} \quad (18.8)$$

Since the noise terms are uncorrelated with the factor scores, and the noise terms for different variables are uncorrelated with each other, *all* the terms containing ϵ s have expectation zero. Also, F_1 and F_2 are uncorrelated, so

$$\text{Cov}[X_1, X_2] = \mathbf{E}[F_1^2] w_{11} w_{12} + \mathbf{E}[F_2^2] w_{21} w_{22} \quad (18.9)$$

$$= w_{11} w_{12} + w_{21} w_{22} \quad (18.10)$$

using the fact that the factors are scaled to have variance 1. This says that the covariance between X_1 and X_2 is what they have from both correlating with F_1 , plus what they have from both correlating with F_2 ; if we had more factors we would add on $w_{31} w_{32} + w_{41} w_{42} + \dots$ out to $w_{q1} w_{q2}$. And of course this would apply as well to any other pair of observable variables. So the general form is

$$\text{Cov}[X_i, X_j] = \sum_{k=1}^q w_{ki} w_{kj} \quad (18.11)$$

so long as $i \neq j$.

The jargon says that observable i **loads on** factor k when $w_{ki} \neq 0$. If two observables do not load on to any of the same factors, if they do not share any common factors, then they will be independent. If we could condition on ("control for") the factors, all of the observables would be conditionally independent.

Graphically, we draw an arrow from a factor node to an observable node if and only if the observable loads on the factor. So then we can just *see* that two observables are correlated if they both have in-coming arrows from the same factors. (To find the actual correlation, we multiply the weights on all the edges connecting the two observable nodes to the common factors; that's Eq. 18.11.) Conversely, even though the factors are marginally independent of each other, if two factors both send arrows to the same observable, then they are dependent conditional on that observable.²

²To see that this makes sense, suppose that $X_1 = F_1 w_{11} + F_2 w_{21} + \epsilon_1$. If we know the value of X_1 , we know what F_1 , F_2 and ϵ_1 have to add up to, so they are conditionally dependent.

18.2.2 Geometry: Approximation by Hyper-planes

Each observation we take is a vector in a p -dimensional space; the factor model says that these vectors have certain geometric relations to each other — that the data has a certain shape. To see what that is, pretend for right now that we can turn off the noise terms ϵ . The loading matrix \mathbf{w} is a $q \times p$ matrix, so each row of \mathbf{w} is a vector in p -dimensional space; call these vectors $\vec{w}_1, \vec{w}_2, \dots, \vec{w}_q$. Without the noise, our observable vectors would be linear combinations of these vectors (with the factor scores saying how much each vector contributes to the combination). Since the factors are orthogonal to each other, we know that they span a q -dimensional sub-space of the p -dimensional space — a line if $q = 1$, a plane if $q = 2$, in general a hyper-plane. If the factor model is true and we turn off noise, we would find all the data lying *exactly* on this hyper-plane. Of course, *with* noise we expect that the data vectors will be scattered around the hyper-plane; how close depends on the variance of the noise. But this is still a rather specific prediction about the shape of the data.

A weaker prediction than “the data lie on a low-dimensional plane in the high-dimensional space” is “the data lie on some low-dimensional surface, possibly curved, in the high-dimensional space”; there are techniques for trying to recover such surfaces, which can work even when factor analysis fails. But they are more complicated than factor analysis and outside the scope of this class. (Take data mining.)

18.3 Roots of Factor Analysis in Causal Discovery

The roots of factor analysis go back to work by Charles Spearman just over a century ago (Spearman, 1904); he was trying to discover the hidden structure of human intelligence. His observation was that schoolchildren’s grades in different subjects were all correlated with each other. He went beyond this to observe a particular *pattern* of correlations, which he thought he could explain as follows: the reason grades in math, English, history, etc., are all correlated is performance in these subjects is all correlated with *something else*, a general or **common** factor, which he named “general intelligence”, for which the natural symbol was of course g or G .

Put in a form like Eq. 18.4, Spearman’s model becomes

$$\mathbf{X} = \epsilon + \mathbf{G}\mathbf{w} \quad (18.12)$$

where \mathbf{G} is an $n \times 1$ matrix (i.e., a row vector) and \mathbf{w} is a $1 \times p$ matrix (i.e., a column vector). The correlation between feature i and G is just $w_i \equiv w_{1i}$, and, if $i \neq j$,

$$v_{ij} \equiv \text{Cov}[X_i, X_j] = w_i w_j \quad (18.13)$$

where I have introduced v_{ij} as a short-hand for the covariance.

Up to this point, this is all so much positing and assertion and hypothesis. What Spearman did next, though, was to observe that this hypothesis carried a very strong implication about the *ratios* of correlation coefficients. Pick any four distinct features,

i, j, k, l . Then, if the model (18.12) is true,

$$\frac{\mathbf{v}_{ij}/\mathbf{v}_{kj}}{\mathbf{v}_{il}/\mathbf{v}_{kl}} = \frac{w_i w_j / w_k w_j}{w_i w_l / w_k w_l} \quad (18.14)$$

$$= \frac{w_i / w_k}{w_i / w_k} \quad (18.15)$$

$$= 1 \quad (18.16)$$

The relationship

$$\mathbf{v}_{ij}\mathbf{v}_{kl} = \mathbf{v}_{il}\mathbf{v}_{kj} \quad (18.17)$$

is called the “tetrad equation”, and we will meet it again later when we consider methods for causal discovery in Part III. In Spearman’s model, this is one tetrad equation for every set of four distinct variables.

Spearman found that the tetrad equations held in his data on school grades (to a good approximation), and concluded that a single general factor of intelligence must exist³. This was, of course, logically fallacious.

Later work, using large batteries of different kinds of intelligence tests, showed that the tetrad equations do not hold in general, or more exactly that departures from them are too big to explain away as sampling noise. (Recall that the equations are about the true correlations between the variables, but we only get to see sample correlations, which are always a little off.) The response, done in an *ad hoc* way by Spearman and his followers, and then more systematically by Thurstone, was to introduce *multiple* factors. This breaks the tetrad equation, but still accounts for the correlations among features by saying that features are really directly correlated with factors, and uncorrelated conditional on the factor scores.⁴ Thurstone’s form of factor analysis is basically the one people still use — there have been refinements, of course, but it’s mostly still his method.

18.4 Estimation

The factor model introduces a whole bunch of new variables to explain the observables: the factor scores \mathbf{F} , the factor loadings or weights \mathbf{w} , and the observable-specific variances ψ_i . The factor scores are specific to each individual, and individuals by assumption are independent, so we can’t expect them to really generalize. But the loadings \mathbf{w} are, supposedly, characteristic of the population. So it would be nice if we could separate estimating the population parameters from estimating the attributes of individuals; here’s how.

Since the variables are centered, we can write the covariance matrix in terms of the data frames:

$$\mathbf{v} = \mathbf{E} \left[\frac{1}{n} \mathbf{X}^T \mathbf{X} \right] \quad (18.18)$$

³Actually, the equations didn’t hold when music was one of the grades, so Spearman argued musical ability did not load on general intelligence.

⁴You can (and should!) read the classic “The Vectors of Mind” paper (Thurstone, 1934) online.

(This is the true, population covariance matrix on the left.) But the factor model tells us that

$$\mathbf{X} = \mathbf{F}\mathbf{w} + \boldsymbol{\epsilon} \quad (18.19)$$

This involves the factor scores \mathbf{F} , but remember that when we looked at the correlations between individual variables, those went away, so let's substitute Eq. 18.19 into Eq. 18.18 and see what happens:

$$\mathbb{E} \left[\frac{1}{n} \mathbf{X}^T \mathbf{X} \right] \quad (18.20)$$

$$= \frac{1}{n} \mathbb{E} [(\boldsymbol{\epsilon}^T + \mathbf{w}^T \mathbf{F}^T)(\mathbf{F}\mathbf{w} + \boldsymbol{\epsilon})] \quad (18.21)$$

$$= \frac{1}{n} (\mathbb{E} [\boldsymbol{\epsilon}^T \boldsymbol{\epsilon}] + \mathbf{w}^T \mathbb{E} [\mathbf{F}^T \boldsymbol{\epsilon}] + \mathbb{E} [\boldsymbol{\epsilon}^T \mathbf{F}] \mathbf{w} + \mathbf{w}^T \mathbb{E} [\mathbf{F}^T \mathbf{F}] \mathbf{w}) \quad (18.22)$$

$$= \psi + 0 + 0 + \frac{1}{n} \mathbf{w}^T n \mathbf{I} \mathbf{w} \quad (18.23)$$

$$= \psi + \mathbf{w}^T \mathbf{w} \quad (18.24)$$

Behold:

$$\mathbf{v} = \psi + \mathbf{w}^T \mathbf{w} \quad (18.25)$$

The individual-specific variables \mathbf{F} have gone away, leaving only population parameters on both sides of the equation.

18.4.1 Degrees of Freedom

It only takes a bit of playing with Eq. 18.25 to realize that we are in trouble. Like any matrix equation, it represents a system of equations. How many equations in how many unknowns? Naively, we'd say that we have p^2 equations (one for each element of the matrix \mathbf{v}), and $p + pq$ unknowns (one for each diagonal element of ψ , plus one for each element of \mathbf{w}). If there are more equations than unknowns, then there is generally no solution; if there are fewer equations than unknowns, then there are generally infinitely many solutions. Either way, solving for \mathbf{w} seems hopeless (unless $q = p - 1$, in which case it's not very helpful). What to do?

Well, first let's do the book-keeping for degrees of freedom more carefully. The observables variables are scaled to have standard deviation one, so the diagonal entries of \mathbf{v} are all 1. Moreover, any covariance matrix is symmetric, so we are left with only $p(p - 1)/2$ degrees of freedom in \mathbf{v} — only that many equations. On the other side, scaling to standard deviation 1 means we don't *really* need to solve separately for ψ — it's fixed as soon as we know what $\mathbf{w}^T \mathbf{w}$ is — which saves us p unknowns. Also, the entries in \mathbf{w} are not completely free to vary independently of each other, because each row has to be orthogonal to every other row. (Look back at the notes on PCA.) Since there are q rows, this gives $q(q - 1)/2$ constraints on \mathbf{w} — we can think of these as either extra equations, or as reductions in the number of free parameters (unknowns).⁵

⁵Notice that $\psi + \mathbf{w}^T \mathbf{w}$ is automatically symmetric, since ψ is diagonal, so we don't need to impose any extra constraints to get symmetry.

Summarizing, we really have $p(p - 1)/2$ degrees of freedom in \mathbf{v} , and $pq - q(q - 1)/2$ degrees of freedom in \mathbf{w} . If these two match, then there is (in general) a unique solution which will give us \mathbf{w} . But in general they will *not* be equal; then what? Let us consider the two cases.

18.4.1.0.1 More unknowns (free parameters) than equations (constraints) This is fairly straightforward: there is no unique solution to Eq. 18.25; instead there are *infinitely many* solutions. It's true that the loading matrix \mathbf{w} does have to satisfy some constraints, that not just any \mathbf{w} will work, so the data does give us some information, but there is a continuum of different parameter settings which are all match the covariance matrix perfectly. (Notice that we are working with the population parameters here, so this isn't an issue of having only a limited sample.) There is just no way to use data to decide between these different parameters, to identify which one is right, so we say the model is **unidentifiable**. Most software for factor analysis, include R's `factanal` function, will check for this and just refuse to fit a model with too many factors relative to the number of observables.

18.4.1.0.2 More equations (constraints) than unknowns (free parameters) This is more interesting. In general, systems of equations like this are **overdetermined**, meaning that there is no way to satisfy all the constraints at once, and there isn't even a single solution. It's just not possible to write an arbitrary covariance matrix \mathbf{v} among, say, seven variables in terms of, say, a one-factor model (as $p(p - 1)/2 = 7(7 - 1)/2 = 21 > 7(1) - 1(1 - 1)/2 = 7 = pq - q(q - 1)/2$). But it *is* possible for special covariance matrices. In these situations, the factor model actually has testable implications for the data — it says that only certain covariance matrices are possible and not others. For example, we saw above that the one-factor model implies the tetrad equations must hold among the observable covariances; the constraints on \mathbf{v} for multiple-factor models are similar in kind but more complicated algebraically. By testing these implications, we can check whether or not our favorite factor model is right.⁶

Now we don't know the true, population covariance matrix \mathbf{v} , but we can estimate it from data, getting an estimate $\hat{\mathbf{v}}$. The natural thing to do then is to equate this with the parameters and try to solve for the latter:

$$\hat{\mathbf{v}} = \hat{\psi} + \hat{\mathbf{w}}^T \hat{\mathbf{w}} \quad (18.26)$$

The book-keeping for degrees of freedom here is the same as for Eq. 18.25. If q is too large relative to p , the model is unidentifiable; if it is too small, the matrix equation can only be solved if $\hat{\mathbf{v}}$ is of the right, restricted form, i.e., if the model is right. Of course even if the model is right, the sample covariances are the true covariances plus noise, so we shouldn't expect to get an *exact* match, but we can try in various ways to minimize the discrepancy between the two sides of the equation.

⁶Actually, we need to be a little careful here. If we find that the tetrad equations don't hold, we know a one-factor model must be wrong. We could only conclude that the one-factor model must be right if we found that the tetrad equations held, *and* that there were no other models which implied those equations; but, as we'll see, there are.

18.4.2 A Clue from Spearman's One-Factor Model

Remember that in Spearman's model with a single general factor, the covariance between observables i and j in that model is the product of their factor weightings:

$$\mathbf{v}_{ij} = w_i w_j \quad (18.27)$$

The exception is that $\mathbf{v}_{ii} = w_i^2 + \psi_i$, rather than w_i^2 . However, if we look at $\mathbf{u} = \mathbf{v} - \boldsymbol{\psi}$, that's the same as \mathbf{v} off the diagonal, and a little algebra shows that its diagonal entries are, in fact, just w_i^2 . So if we look at any two rows of \mathbf{U} , they're proportional to each other:

$$\mathbf{u}_{ij} = \frac{w_i}{w_k} \mathbf{u}_{kj} \quad (18.28)$$

This means that, when Spearman's model holds true, there is actually only *one* linearly-independent row in \mathbf{u} .

Recall from linear algebra that the **rank** of a matrix is how many linearly independent rows it has.⁷ Ordinarily, the matrix is of **full rank**, meaning all the rows are linearly independent. What we have just seen is that when Spearman's model holds, the matrix \mathbf{u} is *not* of full rank, but rather of rank 1. More generally, when the factor model holds with q factors, the matrix $\mathbf{u} = \mathbf{w}^T \mathbf{w}$ has rank q . The diagonal entries of \mathbf{u} , called the **common variances** or **commonalities**, are no longer automatically 1, but rather show how much of the variance in each observable is associated with the variances of the latent factors. Like \mathbf{v} , \mathbf{u} is a positive symmetric matrix.

Because \mathbf{u} is a positive symmetric matrix, we know from linear algebra that it can be written as

$$\mathbf{u} = \mathbf{c} \mathbf{d} \mathbf{c}^T \quad (18.29)$$

where \mathbf{c} is the matrix whose columns are the eigenvectors of \mathbf{u} , and \mathbf{d} is the diagonal matrix whose entries are the eigenvalues. That is, if we use all p eigenvectors, we can reproduce the covariance matrix exactly. Suppose we instead use \mathbf{c}_q , the $p \times q$ matrix whose columns are the eigenvectors going with the q largest eigenvalues, and likewise make \mathbf{d}_q the diagonal matrix of those eigenvalues. Then $\mathbf{c}_q \mathbf{d}_q \mathbf{c}_q^T$ will be a symmetric positive $p \times p$ matrix. This is a matrix of rank q , and so can only equal \mathbf{u} if the latter also has rank q . Otherwise, it's an approximation which grows more accurate as we let q grow towards p , and, at any given q , it's a better approximation to \mathbf{u} than any other rank- q matrix. This, finally, is the precise sense in which factor analysis tries to preserve correlations, as opposed to principal components trying to preserve variance.

To resume our algebra, define $\mathbf{d}_q^{1/2}$ as the $q \times q$ diagonal matrix of the square roots of the eigenvalues. Clearly $\mathbf{d}_q = \mathbf{d}_q^{1/2} \mathbf{d}_q^{1/2}$. So

$$\mathbf{c}_q \mathbf{d}_q \mathbf{c}_q^T = \mathbf{c}_q \mathbf{d}_q^{1/2} \mathbf{d}_q^{1/2} \mathbf{c}_q^T = (\mathbf{c}_q \mathbf{d}_q^{1/2}) (\mathbf{c}_q \mathbf{d}_q^{1/2})^T \quad (18.30)$$

So we have

$$\mathbf{u} \approx (\mathbf{c}_q \mathbf{d}_q^{1/2}) (\mathbf{c}_q \mathbf{d}_q^{1/2})^T \quad (18.31)$$

⁷We could also talk about the columns; it wouldn't make any difference.

but at the same time we know that $\mathbf{u} = \mathbf{w}^T \mathbf{w}$. So we just identify \mathbf{w} with $(\mathbf{c}_q \mathbf{d}_q^{1/2})^T$:

$$\mathbf{w} = (\mathbf{c}_q \mathbf{d}_q^{1/2})^T \quad (18.32)$$

and we are done with our algebra.

Let's think a bit more about how well we're approximating \mathbf{v} . The approximation will always be exact when $q = p$, so that there is one factor for each feature (in which case $\psi = 0$ always). Then all factor analysis does for us is to rotate the coordinate axes in feature space, so that the new coordinates are uncorrelated. (This is the same was what PCA does with p components.) The approximation can *also* be exact with fewer factors than features if the reduced covariance matrix is of less than full rank, and we use at least as many factors as the rank.

18.4.3 Estimating Factor Loadings and Specific Variances

The classical method for estimating the factor model is now simply to do this eigenvector approximation on the *sample* correlation matrix. Define the **reduced** or **adjusted** sample correlation matrix as

$$\hat{\mathbf{u}} = \hat{\mathbf{v}} - \hat{\psi} \quad (18.33)$$

We can't actually calculate $\hat{\mathbf{u}}$ until we know, or have a guess as to, $\hat{\psi}$. A reasonable and common starting-point is to do a linear regression of each feature j on all the other features, and then set $\hat{\psi}_j$ to the mean squared error for that regression. (We'll come back to this guess later.)

Once we have the reduced correlation matrix, find its top q eigenvalues and eigenvectors, getting matrices $\hat{\mathbf{c}}_q$ and $\hat{\mathbf{d}}_q$ as above. Set the factor loadings accordingly, and re-calculate the specific variances:

$$\hat{\mathbf{w}} = (\mathbf{c}_q \mathbf{d}_q^{1/2})^T \quad (18.34)$$

$$\hat{\psi}_j = 1 - \sum_{r=1}^k w_{rj}^2 \quad (18.35)$$

$$\tilde{\mathbf{v}} \equiv \hat{\psi} + \hat{\mathbf{w}}^T \hat{\mathbf{w}} \quad (18.36)$$

The “predicted” covariance matrix $\tilde{\mathbf{v}}$ in the last line is exactly right on the diagonal (by construction), and should be closer off-diagonal than anything else we could do with the same number of factors. However, our guess as to \mathbf{u} depended on our initial guess about ψ , which has in general changed, so we can try iterating this (i.e., re-calculating \mathbf{c}_q and \mathbf{d}_q), until we converge.

18.5 Maximum Likelihood Estimation

It has probably not escaped your notice that the estimation procedure above requires a starting guess as to ψ . This makes its consistency somewhat shaky. (If we continually put in ridiculous values for ψ , there's no reason to expect that $\hat{\mathbf{w}} \rightarrow \mathbf{w}$, even

with immensely large samples.) On the other hand, we know from our elementary statistics courses that maximum likelihood estimates are generally consistent, unless we choose a spectacularly bad model. Can we use that here?

We can, but at a cost. We have so far got away with just making assumptions about the means and covariances of the factor scores F . To get an actual likelihood, we need to assume something about their distribution as well.

The usual assumption is that $F_{ik} \sim \mathcal{N}(0, 1)$, and that the factor scores are independent across factors $k = 1, \dots, q$ and individuals $i = 1, \dots, n$. With this assumption, the features have a multivariate normal distribution $\vec{X}_i \sim \mathcal{N}(0, \psi + \mathbf{w}^T \mathbf{w})$. This means that the log-likelihood is

$$L = -\frac{np}{2} \log 2\pi - \frac{n}{2} \log |\psi + \mathbf{w}^T \mathbf{w}| - \frac{n}{2} \text{tr}((\psi + \mathbf{w}^T \mathbf{w})^{-1} \hat{\mathbf{v}}) \quad (18.37)$$

where $\text{tr } \mathbf{a}$ is the **trace** of the matrix \mathbf{a} , the sum of its diagonal elements. Notice that the likelihood only involves the data through the sample covariance matrix $\hat{\mathbf{v}}$ — the actual factor scores F are not needed for the likelihood.

One can either try direct numerical maximization, or use a two-stage procedure. Starting, once again, with a guess as to ψ , one finds that the optimal choice of $\psi^{1/2} \mathbf{w}^T$ is given by the matrix whose columns are the q leading eigenvectors of $\psi^{1/2} \hat{\mathbf{v}} \psi^{1/2}$. Starting from a guess as to \mathbf{w} , the optimal choice of ψ is given by the diagonal entries of $\hat{\mathbf{v}} - \mathbf{w}^T \mathbf{w}$. So again one starts with a guess about the unique variances (e.g., the residuals of the regressions) and iterates to convergence.⁸

The differences between the maximum likelihood estimates and the “principal factors” approach can be substantial. If the data appear to be normally distributed (as shown by the usual tests), then the additional efficiency of maximum likelihood estimation is highly worthwhile. Also, as we’ll see below, it is a lot easier to test the model assumptions if one uses the MLE.

18.5.1 Alternative Approaches

Factor analysis is an example of trying to approximate a full-rank matrix, here the covariance matrix, with a low-rank matrix, or a low-rank matrix plus some corrections, here $\psi + \mathbf{w}^T \mathbf{w}$. Such matrix-approximation problems are currently the subject of very intense interest in statistics and machine learning, with many new methods being proposed and refined, and it is very plausible that some of these will prove to work better than older approaches to factor analysis.

In particular, Kao and Van Roy (2011) have recently used these ideas to propose a new factor-analysis algorithm, which simultaneously estimates the number of factors and the factor loadings, and does so through a modification of PCA, distinct from the old “principal factors” method. In their examples, it works better than conventional approaches, but whether this will hold true generally is not clear. They do not, unfortunately, provide code.

⁸The algebra is tedious. See section 3.2 in Bartholomew (1987) if you really want it. (Note that Bartholomew has a sign error in his equation 3.16.)

18.5.2 Estimating Factor Scores

The probably the best method for estimating factor scores is the “regression” or “Thomson” method, which says

$$\hat{F}_{ir} = \sum_j X_{ij} b_{ij} \quad (18.38)$$

and seeks the weights b_{ij} which will minimize the mean squared error, $E[(\hat{F}_{ir} - F_{ir})^2]$. You can work out the b_{ij} as an exercise, assuming you know \mathbf{w} .

18.6 The Rotation Problem

Recall from linear algebra that a matrix \mathbf{o} is **orthogonal** if its inverse is the same as its transpose, $\mathbf{o}^T \mathbf{o} = \mathbf{I}$. The classic examples are rotation matrices. For instance, to rotate a two-dimensional vector through an angle α , we multiply it by

$$\mathbf{r}_\alpha = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} \quad (18.39)$$

The inverse to this matrix must be the one which rotates through the angle $-\alpha$, $\mathbf{r}_\alpha^{-1} = \mathbf{r}_{-\alpha}$, but trigonometry tells us that $\mathbf{r}_{-\alpha} = \mathbf{r}_\alpha^T$.

To see why this matters to us, go back to the matrix form of the factor model, and insert an orthogonal $q \times q$ matrix and its transpose:

$$\mathbf{X} = \epsilon + \mathbf{F}\mathbf{w} \quad (18.40)$$

$$= \epsilon + \mathbf{F}\mathbf{o}\mathbf{o}^T\mathbf{w} \quad (18.41)$$

$$= \epsilon + \mathbf{H}\mathbf{y} \quad (18.42)$$

We've changed the factor scores to $\mathbf{H} \equiv \mathbf{H}\mathbf{o}$, and we've changed the factor loadings to $\mathbf{y} \equiv \mathbf{o}^T\mathbf{w}$, but nothing about the features has changed *at all*. We can do as many orthogonal transformations of the factors as we like, with no observable consequences whatsoever.⁹

Statistically, the fact that different parameter settings give us the same observational consequences means that the parameters of the factor model are **unidentifiable**. The rotation problem is, as it were, the revenant of having an ill-posed problem: we thought we'd slain it through heroic feats of linear algebra, but it's still around and determined to have its revenge.¹⁰

⁹Notice that the log-likelihood only involves $\mathbf{w}^T\mathbf{w}$, which is equal to $\mathbf{w}^T\mathbf{o}\mathbf{o}^T\mathbf{w} = \mathbf{y}^T\mathbf{y}$, so even assuming Gaussian distributions doesn't let us tell the difference between the original and the transformed variables. In fact, if $\vec{\mathbf{F}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, then $\vec{\mathbf{F}}\mathbf{o} \sim \mathcal{N}(\mathbf{0}\mathbf{o}, \mathbf{o}^T\mathbf{I}\mathbf{o}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ — in other words, the rotated factor scores still satisfy our distributional assumptions.

¹⁰Remember that we obtained the loading matrix \mathbf{w} as a solution to $\mathbf{w}^T\mathbf{w} = \mathbf{u}$, that is to we got \mathbf{w} as a kind of matrix square root of the reduced correlation matrix. For a real number u there are two square roots, i.e., two numbers w such that $w \times w = u$, namely the usual $w = \sqrt{u}$ and $w = -\sqrt{u}$, because $(-1) \times (-1) = 1$. Similarly, whenever we find one solution to $\mathbf{w}^T\mathbf{w} = \mathbf{u}$, $\mathbf{o}^T\mathbf{w}$ is another solution, because $\mathbf{o}\mathbf{o}^T = \mathbf{I}$. So while the usual “square root” of \mathbf{u} is $\mathbf{w} = \mathbf{d}_q^{1/2}\mathbf{c}$, for any orthogonal matrix $\mathbf{o}^T\mathbf{d}_q^{1/2}\mathbf{c}$ will always work just as well.

Mathematically, this should not be surprising at all. The factor live in a q -dimensional vector space of their own. We should be free to set up any coordinate system we feel like on that space. Changing coordinates in factor space will just require a compensating change in how factor space coordinates relate to feature space (the factor loadings matrix \mathbf{w}). That's all we've done here with our orthogonal transformation.

Substantively, this should be rather troubling. If we can rotate the factors as much as we like without consequences, how on Earth can we interpret them?

18.7 Factor Analysis as a Predictive Model

Unlike principal components analysis, factor analysis really does give us a predictive model. Its prediction is that if we draw a new member of the population and look at the vector of observables we get from them,

$$\vec{X} \sim \mathcal{N}(0, \mathbf{w}^T \mathbf{w} + \psi) \quad (18.43)$$

if we make the usual distributional assumptions. Of course it might seem like it makes a more refined, conditional prediction,

$$\vec{X} | \vec{F} \sim \mathcal{N}(F\mathbf{w}, \psi) \quad (18.44)$$

but the problem is that there is no way to guess at or estimate the factor scores \vec{F} until after we've seen \vec{X} , at which point anyone can predict X perfectly. So the actual forecast is given by Eq. 18.43.¹¹

Now, without going through the trouble of factor analysis, one could always just postulate that

$$\vec{X} \sim \mathcal{N}(0, \mathbf{v}) \quad (18.45)$$

and estimate \mathbf{v} ; the maximum likelihood estimate of it is the observed covariance matrix, but really we could use any estimator of the covariance matrix. The closer our is to the true \mathbf{v} , the better our predictions. One way to think of factor analysis is that it looks for the maximum likelihood estimate, but constrained to matrices of the form $\mathbf{w}^T \mathbf{w} + \psi$.

On the plus side, the constrained estimate has a faster rate of convergence. That is, both the constrained and unconstrained estimates are consistent and will converge on their optimal, population values as we feed in more and more data, but for the same amount of data the constrained estimate is probably closer to its limiting value. In other words, the constrained estimate $\hat{\mathbf{w}}^T \hat{\mathbf{w}} + \hat{\psi}$ has less variance than the unconstrained estimate $\hat{\mathbf{v}}$.

On the minus side, maybe the true, population \mathbf{v} just can't be written in the form $\mathbf{w}^T \mathbf{w} + \psi$. Then we're getting biased estimates of the covariance and the bias will *not*

¹¹A subtlety is that we might get to see some but not all of \vec{X} , and use that to predict the rest. Say $\vec{X} = (X_1, X_2)$, and we see X_1 . Then we could, in principle, compute the conditional distribution of the factors, $p(\vec{F}|X_1)$, and use that to predict X_2 . Of course one could do the same thing using the correlation matrix, factor model or no factor model.

go away, even with infinitely many samples. Using factor analysis rather than just fitting a multivariate Gaussian means betting that either this bias is really zero, or that, with the amount of data on hand, the reduction in variance outweighs the bias.

(I haven't talked about estimated errors in the parameters of a factor model. With large samples and maximum-likelihood estimation, one could use the usual asymptotic theory. For small samples, one bootstraps as usual.)

18.7.1 How Many Factors?

How many factors should we use? All the tricks people use for the how-many-principal-components question can be tried here, too, with the obvious modifications. However, some other answers can also be given, using the fact that the factor model does make predictions, unlike PCA.

1. *Log-likelihood ratio tests* Sample covariances will almost never be exactly equal to population covariances. So even if the data comes from a model with q factors, we can't expect the tetrad equations (or their multi-factor analogs) to hold exactly. The question then becomes whether the observed covariances are compatible with sampling fluctuations in a q -factor model, or are too big for that.

We can tackle this question by using log likelihood ratio tests. The crucial observations are that a model with q factors is a special case of a model with $q+1$ factors (just set a row of the weight matrix to zero), and that in the most general case, $q = p$, we can get *any* covariance matrix \mathbf{v} into the form $\mathbf{w}^T \mathbf{w}$. (Set $\psi = 0$ and proceed as in the "principal factors" estimation method.)

As explained in Appendix C, if $\hat{\theta}$ is the maximum likelihood estimate in a restricted model with u parameters, and $\hat{\Theta}$ is the MLE in a more general model with $r > s$ parameters, containing the former as a special case, and finally ℓ is the log-likelihood function

$$2[\ell(\hat{\Theta}) - \ell(\hat{\theta})] \rightsquigarrow \chi^2_{r-s} \quad (18.46)$$

when the data came from the small model. The general regularity conditions needed for this to hold apply to Gaussian factor models, so we can test whether one factor is enough, two, etc.

(Said another way, adding another factor never reduces the likelihood, but the equation tells us how much to *expect* the log-likelihood to go up when the new factor really adds nothing and is just over-fitting the noise.)

Determining q by getting the smallest one without a significant result in a likelihood ratio test is fairly traditional, but statistically messy.¹² To raise a subject we'll return to, if the true $q > 1$ and all goes well, we'll be doing lots of hypothesis tests, and making sure this compound procedure works reliably is harder

¹²Suppose q is really 1, but by chance that gets rejected. Whether $q = 2$ gets rejected in term is not independent of this!

than controlling any one test. Perhaps more worrisomely, calculating the likelihood relies on distributional assumptions for the factor scores and the noises, which are hard to check for latent variables.

2. If you are comfortable with the distributional assumptions, use Eq. 18.43 to predict new data, and see which q gives the best predictions — for comparability, the predictions should be compared in terms of the log-likelihood they assign to the testing data. If genuinely new data is not available, use cross-validation.

Comparative prediction, and especially cross-validation, seems to be somewhat rare with factor analysis. There is no good reason why this should be so.

18.7.1.1 R^2 and Goodness of Fit

For PCA, we saw that R^2 depends on the sum of the eigenvalues. For factor models, the natural notion of R^2 is the sum of squared factor loadings:

$$R^2 = \frac{\sum_{j=1}^q \sum_{k=1}^p w_{jk}^2}{p} \quad (18.47)$$

(Remember that the factors are, by design, uncorrelated with each other, and that the entries of \mathbf{w} are the correlations between factors and observables.) If we write \mathbf{w} in terms of eigenvalues and eigenvectors as in §18.4.2, $\mathbf{w} = (\mathbf{c}_q \mathbf{d}_q^{1/2})^T$, then you can show that the numerator in R^2 is, again, a sum of eigenvalues.

People sometimes select the number of factors by looking at how much variance they “explain” — really, how much variance is kept after smoothing on to the plane. As usual with model selection by R^2 , there is little good to be said for this, except that it is fast and simple.

In particular, R^2 should not be used to assess the goodness-of-fit of a factor model. The bluntest way to see this is to simulate data which does *not* come from a factor model, fit a small number of factors, and see what R^2 one gets. This was done by Peterson (2000), who found that it was easy to get R^2 of 0.4 or 0.5, and sometimes even higher¹³ The same paper surveyed values of R^2 from the published literature on factor models, and found that the typical value was also somewhere around 0.5; no doubt this was just a coincidence¹⁴.

Instead of looking at R^2 , it is much better to check goodness-of-fit by actually goodness-of-fit tests. We looked at some tests of multivariate goodness-of-fit in Chapter 14. In the particular case of factor models with the Gaussian assumption, we can use a log-likelihood ratio test, checking the null hypothesis that the number of factors = q against the alternative of an arbitrary multivariate Gaussian (which is the same as p factors). This test is automatically performed by `factanal` in R.

If the Gaussian assumption is dubious but we want a factor model and goodness-of-fit anyway, we can look at the difference between the empirical covariance matrix

¹³See also <http://bactra.org/weblog/523.html> for a similar experiment, with R code.

¹⁴Peterson (2000) also claims that reported values of R^2 for PCA are roughly equal to those of factor analysis, but by this point I hope that none of you take that as an argument in favor of PCA.

\mathbf{v} and the one estimated by the factor model, $\hat{\psi} + \hat{\mathbf{w}}^T \hat{\mathbf{w}}$. There are several notions of distance between matrices (matrix norms) which could be used as test statistics; one could also use the sum of squared differences between the entries of \mathbf{v} and those of $\hat{\psi} + \hat{\mathbf{w}}^T \hat{\mathbf{w}}$. Sampling distributions would have to come from bootstrapping, where we would want to simulate from the factor model.

18.8 Reification, and Alternatives to Factor Models

A natural impulse, when looking at something like Figure 18.1, is to **reify** the factors, and to treat the arrows **causally**: that is, to say that there really is *some* variable corresponding to each factor, and that changing the value of that variable will change the features. For instance, one might want to say that there is a real, physical variable corresponding to the factor F_1 , and that increasing this by one standard deviation will, on average, increase X_1 by 0.87 standard deviations, decrease X_2 by 0.75 standard deviations, and do nothing to the other features. Moreover, changing any of the other factors has no effect on X_1 .

Sometimes all this is even right. How can we tell when it's right?

18.8.1 The Rotation Problem Again

Consider the following matrix, call it r :

$$\begin{bmatrix} \cos 30 & -\sin 30 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (18.48)$$

Applied to a three-dimensional vector, this rotates it thirty degrees counter-clockwise around the vertical axis. If we apply r to the factor loading matrix of the model in the figure, we get the model in Figure 18.2. Now instead of X_1 being correlated with the other variables only through one factor, it's correlated through two factors, and X_4 has incoming arrows from three factors.

Because the transformation is orthogonal, the distribution of the observations is unchanged. In particular, the fit of the new factor model to the data will be *exactly* as good as the fit of the old model. If we try to take this causally, however, we come up with a very different interpretation. The quality of the fit to the data does not, therefore, let us distinguish between these two models, and so these two stories about the causal structure of the data.

The rotation problem does not rule out the idea that checking the fit of a factor model would let us discover *how many* hidden causal variables there are.

18.8.2 Factors or Mixtures?

Suppose we have two distributions with probability densities $f_0(x)$ and $f_1(x)$. Then we can define a new distribution which is a **mixture** of them, with density $f_\alpha(x) = (1 - \alpha)f_0(x) + \alpha f_1(x)$, $0 \leq \alpha \leq 1$. The same idea works if we combine more than two distributions, so long as the sum of the **mixing weights** sum to one (as do α

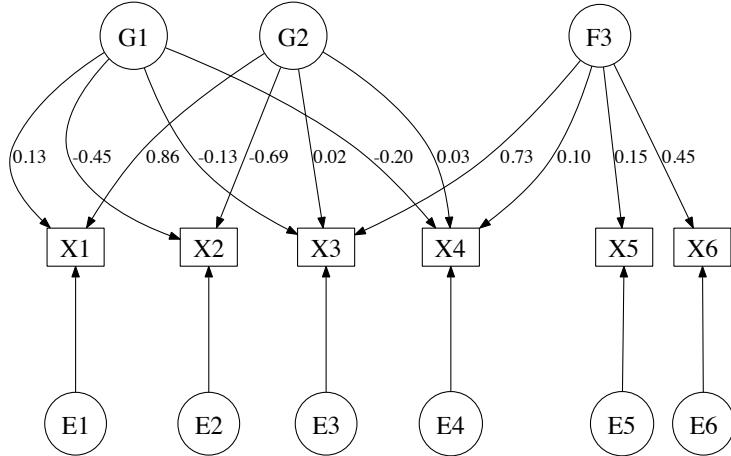


Figure 18.2: The model from Figure 18.1, after rotating the first two factors by 30 degrees around the third factor's axis. The new factor loadings are rounded to two decimal places.

and $1 - \alpha$). We will look more later at mixture models, which provide a very flexible and useful way of representing complicated probability distributions. They are also a probabilistic, predictive alternative to the kind of clustering techniques we've seen before this: each distribution in the mixture is basically a cluster, and the mixing weights are the probabilities of drawing a new sample from the different clusters.¹⁵

I bring up mixture models here because there is a very remarkable result: any linear, Gaussian factor model with k factors is equivalent to some mixture model with $k + 1$ clusters, in the sense that the two models have the same means and covariances (Bartholomew, 1987, pp. 36–38). Recall from above that the likelihood of a factor model depends on the data only through the correlation matrix. If the data really were generated by sampling from $k + 1$ clusters, then a model with k factors can match the covariance matrix very well, and so get a very high likelihood. This means it will, by the usual test, seem like a very good fit. Needless to say, however, the causal interpretations of the mixture model and the factor model are very different. The two *may* be distinguishable if the clusters are well-separated (by looking to see whether the data are unimodal or not), but that's not exactly guaranteed.

¹⁵We will get into mixtures in considerable detail in the next lecture.

All of which suggests that factor analysis can't really tell us whether we have k continuous hidden causal variables, or one discrete hidden variable taking $k+1$ values.

18.8.3 The Thomson Sampling Model

We have been working with fewer factors than we have features. Suppose that's not true. Suppose that each of our features is actually a linear combination of a *lot* of variables we don't measure:

$$X_{ij} = \eta_{ij} + \sum_{k=1}^q A_{ik} T_{kj} = \eta_{ij} + \vec{A}_i \cdot \vec{T}_j \quad (18.49)$$

where $q \gg p$. Suppose further that the latent variables A_{ik} are totally independent of one another, but they all have mean 0 and variance 1; and that the noises η_{ij} are independent of each other and of the A_{ik} , with variance ϕ_j ; and the T_{kj} are independent of everything. What then is the covariance between X_{ia} and X_{ib} ? Well, because $E[X_{ia}] = E[X_{ib}] = 0$, it will just be the expectation of the product of the features:

$$E[X_{ia} X_{ib}] \quad (18.50)$$

$$= E[(\eta_{ia} + \vec{A}_i \cdot \vec{T}_a)(\eta_{ib} + \vec{A}_i \cdot \vec{T}_b)] \quad (18.51)$$

$$= E[\eta_{ia} \eta_{ib}] + E[\eta_{ia} \vec{A}_i \cdot \vec{T}_b] + E[\eta_{ib} \vec{A}_i \cdot \vec{T}_a] + E[(\vec{A}_i \cdot \vec{T}_a)(\vec{A}_i \cdot \vec{T}_b)] \quad (18.52)$$

$$= 0 + 0 + 0 + E\left[\left(\sum_{k=1}^q A_{ik} T_{ka}\right)\left(\sum_{l=1}^q A_{il} T_{lb}\right)\right] \quad (18.53)$$

$$= E\left[\sum_{k,l} A_{ik} A_{il} T_{ka} T_{lb}\right] \quad (18.54)$$

$$= \sum_{k,l} E[A_{ik} A_{il}] T_{ka} T_{lb} \quad (18.55)$$

$$= \sum_{k,l} E[A_{ik} A_{il}] E[T_{ka} T_{lb}] \quad (18.56)$$

$$= \sum_{k=1}^q E[T_{ka} T_{kb}] \quad (18.57)$$

where to get the last line I use the fact that $E[A_{ik} A_{il}] = 1$ if $k = l$ and = 0 otherwise. If the coefficients T are fixed, then the last expectation goes away and we merely have the same kind of sum we've seen before, in the factor model.

Instead, however, let's say that the coefficients T are themselves random (but independent of A and η). For each feature X_{ia} , we fix a proportion z_a between 0 and 1. We then set $T_{ka} \sim \text{Bernoulli}(z_a)$, with $T_{ka} \perp\!\!\!\perp T_{lb}$ unless $k = l$ and $a = b$. Then

$$E[T_{ka} T_{kb}] = E[T_{ka}] E[T_{kb}] = z_a z_b \quad (18.58)$$

and

$$\mathbb{E}[X_{ia}X_{ib}] = qz_az_b \quad (18.59)$$

Of course, in the one-factor model,

$$\mathbb{E}[X_{ia}X_{ib}] = w_aw_b \quad (18.60)$$

So this random-sampling model looks *exactly* like the one-factor model with factor loadings proportional to z_a . The tetrad equation, in particular, will hold.

Now, it doesn't make a lot of sense to imagine that every time we make an observation we change the coefficients T randomly. Instead, let's suppose that they are first generated randomly, giving values T_{kj} , and then we generate feature values according to Eq. 18.49. The covariance between X_{ia} and X_{ib} will be $\sum_{k=1}^q T_{ka}T_{kb}$. But this is a sum of IID random values, so by the law of large numbers as q gets large this will become very close to qz_az_b . Thus, for nearly all choices of the coefficients, the feature covariance matrix should come very close to satisfying the tetrad equations and looking like there's a single general factor.

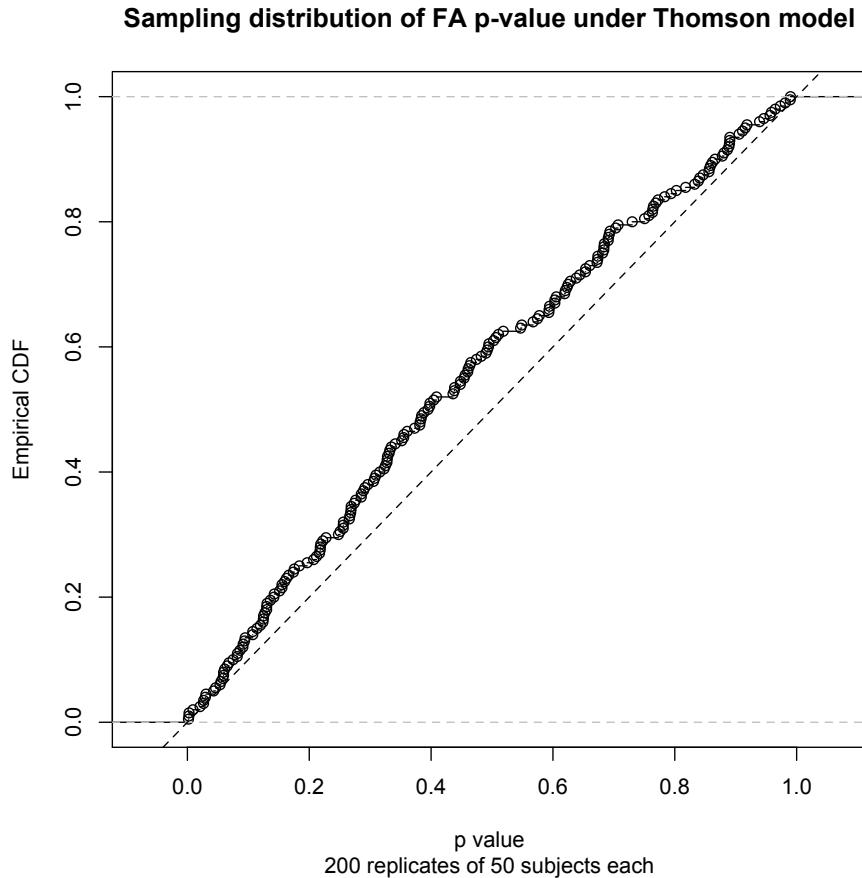
In this model, each feature is a linear combination of a *random sample* of a huge pool of completely independent features, plus some extra noise specific to the feature.¹⁶ Precisely *because* of this, the features are correlated, and the pattern of correlations is that of a factor model with one factor. The appearance of a single common cause actually arises from the fact that the number of causes is immense, and there is no particular pattern to their influence on the features.

The file `thomson-model.R` (on the class website) simulates the Thomson model.

```
> tm = rthomson(50, 11, 500, 50)
> factanal(tm$data, 1)
```

The first command generates data from $n = 50$ items with $p = 11$ features and $q = 500$ latent variables. (The last argument controls the average size of the specific variances ϕ_j .) The result of the factor analysis is of course variable, depending on the random draws; my first attempt gave the proportion of variance associated with the factor as 0.391, and the p -value as 0.527. Repeating the simulation many times, one sees that the p -value is pretty close to uniformly distributed, which is what it should be if the null hypothesis is true (Figure 18.3). For fixed n , the distribution becomes closer to uniform the larger we make q . In other words, the goodness-of-fit test has little or no power against the alternative of the Thomson model.

¹⁶When Godfrey Thomson introduced this model in 1914, he used a slightly different procedure to generate the coefficient T_{kj} . For each feature he drew a uniform integer between 1 and q , call it q_j , and then sampled the integers from 1 to q *without replacement* until he had q_j random numbers; these were the values of k where $T_{kj} = 1$. This is basically similar to what I describe, setting $z_j = q_j/q$, but a bit harder to analyze in an elementary way. — Thomson (1916), the original paper, includes what we would now call a simulation study of the model, where Thomson stepped through the procedure to produce simulated data, calculate the empirical correlation matrix of the features, and check the fit to the tetrad equations. Not having a computer, Thomson generated the values of T_{kj} with a deck of cards, and of the A_{ik} and η_{ij} by rolling 5220 dice.



```
> plot(ecdf(replicate(200,factanal(rthomson(50,11,500,50)$data,1)$PVAL)),  
      xlab="p value",ylab="Empirical CDF",  
      main="Sampling distribution of FA p-value under Thomson model",  
      sub="200 replicates of 50 subjects each")  
> abline(0,1,lty=2)
```

Figure 18.3: Mimcry of the one-factor model by the Thomson model. The Thomson model was simulated 200 times with the parameters given above; each time, the simulated data was then fit to a factor model with one factor, and the p -value of the goodness-of-fit test extracted. The plot shows the empirical cumulative distribution function of the p -values. If the null hypothesis were exactly true, then $p \sim \text{Unif}(0, 1)$, and the theoretical CDF would be the diagonal line (dashed).

Modifying the Thomson model to look like multiple factors grows notationally cumbersome; the basic idea however is to use multiple pools of independently-sampled latent variables, and sum them:

$$X_{ij} = \eta_{ij} + \sum_{k=1}^{q_1} A_{ik} T_{kj} + \sum_{k=1}^{q_2} B_{ik} R_{kj} + \dots \quad (18.61)$$

where the T_{kj} coefficients are uncorrelated with the R_{kj} , and so forth. In expectation, if there are r such pools, this *exactly* matches the factor model with r factors, and any particular realization is overwhelmingly likely to match if the q_1, q_2, \dots, q_r are large enough.¹⁷

It's not feasible to estimate the T of the Thomson model in the same way that we estimate factor loadings, because $q > p$. This is not the point of considering the model, which is rather to make it clear that we actually learn very little about where the data come from when we learn that a factor model fits well. It could mean that the features arise from combining a small number of factors, or on the contrary from combining a huge number of factors in a random fashion. A lot of the time the latter is a more plausible-sounding story.¹⁸

For example, a common application of factor analysis is in marketing: you survey consumers and ask them to rate a bunch of products on a range of features, and then do factor analysis to find attributes which summarize the features. That's fine, but it may well be that each of the features is influenced by lots of aspects of the product you don't include in your survey, and the correlations are really explained by different features being affected by many of the same small aspects of the product. Similarly for psychological testing: answering any question is really a pretty complicated process involving lots of small processes and skills (of perception, several kinds of memory, problem-solving, attention, etc.), which overlap partially from question to question.

Exercises

1. Prove Eq. 18.13.
2. Why is it fallacious to go from “the data have the kind of correlations predicted by a one-factor model” to “the data were generated by a one-factor model”?
3. Show that the correlation between the j^{th} feature and G , in the one-factor model, is w_j .
4. Check that Eq. 18.11 and Eq. 18.25 are compatible.

¹⁷A recent paper on the Thomson model (Bartholomew *et al.*, 2009) proposes just this modification to multiple factors and to Bernoulli sampling. However, I propose this independently, in the fall 2008 version of these notes, about a year before their paper.

¹⁸Thomson (1939) remains one of the most insightful books on factor analysis, though obviously there have been a lot of technical refinements since he wrote. It's strongly recommended for anyone who plans to make much use of factor analysis. While out of print, used copies are reasonably plentiful and cheap, and at least one edition is free online (URL in the bibliography).

5. Find the weights b_{ij} for the Thomson estimator, assuming you know \mathbf{w} . Do you need to assume a Gaussian distribution?
6. Step through the examples in the accompanying R code on the class website.

Chapter 19

Mixture Models

19.1 Two Routes to Mixture Models

19.1.1 From Factor Analysis to Mixture Models

In factor analysis, the origin myth is that we have a fairly small number, q of real variables which happen to be unobserved (“latent”), and the much larger number p of variables we do observe arise as linear combinations of these factors, plus noise. The mythology is that it’s possible for us (or for Someone) to *continuously* adjust the latent variables, and the distribution of observables likewise changes continuously. What if the latent variables are not continuous but ordinal, or even categorical? The natural idea would be that each value of the latent variable would give a different distribution of the observables.

19.1.2 From Kernel Density Estimates to Mixture Models

We have also previously looked at kernel density estimation, where we approximate the true distribution by sticking a small ($\frac{1}{n}$ weight) copy of a kernel pdf at each observed data point and adding them up. With enough data, this comes arbitrarily close to any (reasonable) probability density, but it does have some drawbacks. Statistically, it labors under the curse of dimensionality. Computationally, we have to remember *all* of the data points, which is a lot. We saw similar problems when we looked at fully non-parametric regression, and then saw that both could be ameliorated by using things like additive models, which impose more constraints than, say, unrestricted kernel smoothing. Can we do something like that with density estimation?

Additive modeling for densities is not as common as it is for regression — it’s harder to think of times when it would be natural and well-defined¹ — but we can

¹Remember that the integral of a probability density over all space must be 1, while the integral of a regression function doesn’t have to be anything in particular. If we had an additive density, $f(\mathbf{x}) = \sum_j f_j(x_j)$, ensuring normalization is going to be very tricky; we’d need $\sum_j \int f_j(x_j) dx_1 dx_2 \dots dx_p = 1$. It would be easier to ensure normalization while making the *log*-density additive, but that assumes the features are

do things to restrict density estimation. For instance, instead of putting a copy of the kernel at *every* point, we might pick a small number $K \ll n$ of points, which we feel are somehow typical or representative of the data, and put a copy of the kernel at each one (with weight $\frac{1}{K}$). This uses less memory, but it ignores the other data points, and lots of them are probably very similar to those points we're taking as prototypes. The differences between prototypes and many of their neighbors are just matters of chance or noise. Rather than remembering all of those noisy details, why not collapse those data points, and just remember their common distribution? Different regions of the data space will have different shared distributions, but we can just combine them.

19.1.3 Mixture Models

More formally, we say that a distribution f is a **mixture** of K **component** distributions f_1, f_2, \dots, f_K if

$$f(x) = \sum_{k=1}^K \lambda_k f_k(x) \quad (19.1)$$

with the λ_k being the **mixing weights**, $\lambda_k > 0$, $\sum_k \lambda_k = 1$. Eq. 19.1 is a complete stochastic model, so it gives us a recipe for generating new data points: first pick a distribution, with probabilities given by the mixing weights, and then generate one observation according to that distribution. Symbolically,

$$Z \sim \text{Mult}(\lambda_1, \lambda_2, \dots, \lambda_K) \quad (19.2)$$

$$X|Z \sim f_Z \quad (19.3)$$

where I've introduced the discrete random variable Z which says which component X is drawn from.

I haven't said what kind of distribution the f_k s are. In principle, we could make these completely arbitrary, and we'd still have a perfectly good mixture model. In practice, a lot of effort is given over to **parametric mixture** models, where the f_k are all from the same parametric family, but with different parameters — for instance they might all be Gaussians with different centers and variances, or all Poisson distributions with different means, or all power laws with different exponents. (It's not necessary, just customary, that they all be of the same kind.) We'll write the parameter, or parameter vector, of the k^{th} component as θ_k , so the model becomes

$$f(x) = \sum_{k=1}^K \lambda_k f(x; \theta_k) \quad (19.4)$$

The over-all parameter vector of the mixture model is thus $\theta = (\lambda_1, \lambda_2, \dots, \lambda_K, \theta_1, \theta_2, \dots, \theta_K)$.

Let's consider two extremes. When $K = 1$, we have a simple parametric distribution, of the usual sort, and density estimation reduces to estimating the parameters, by maximum likelihood or whatever else we feel like. On the other hand when

independent of each other.

$K = n$, the number of observations, we have gone back towards kernel density estimation. If K is fixed as n grows, we still have a parametric model, and avoid the curse of dimensionality, but a mixture of (say) ten Gaussians is more flexible than a single Gaussian — thought it may still be the case that the true distribution just can't be written as a ten-Gaussian mixture. So we have our usual bias-variance or accuracy-precision trade-off — using many components in the mixture lets us fit many distributions very accurately, with low approximation error or bias, but means we have more parameters and so we can't fit any one of them as precisely, and there's more variance in our estimates.

19.1.4 Geometry

In Chapter 17, we looked at principal components analysis, which finds linear structures with q space (lines, planes, hyper-planes, ...) which are good approximations to our p -dimensional data, $q \ll p$. In Chapter 18, we looked at factor analysis, where which imposes a statistical model for the distribution of the data around this q -dimensional plane (Gaussian noise), and a statistical model of the distribution of representative points on the plane (also Gaussian). This set-up is implied by the mythology of linear continuous latent variables, but can arise in other ways.

Now, we know from geometry that it takes $q+1$ points to define a q -dimensional plane, and that in general any $q+1$ points on the plane will do. This means that if we use a mixture model with $q+1$ components, we will also get data which clusters around a q -dimensional plane. Furthermore, by adjusting the mean of each component, and their relative weights, we can make the global mean of the mixture whatever we like. And we can even match the covariance matrix of any q -factor model by using a mixture with $q+1$ components². Now, this mixture distribution will hardly ever be exactly the same as the factor model's distribution — mixtures of Gaussians aren't Gaussian, the mixture will usually (but not always) be multimodal while the factor distribution is always unimodal — but it will have the same geometry, the same mean and the same covariances, so we will have to look beyond those to tell them apart. Which, frankly, people hardly ever do.

19.1.5 Identifiability

Before we set about trying to estimate our probability models, we need to make sure that they are identifiable — that if we have distinct representations of the model, they make distinct observational claims. It is easy to let there be too many parameters, or the wrong choice of parameters, and lose identifiability. If there *are* distinct representations which are observationally equivalent, we either need to change our model, change our representation, or fix on a *unique* representation by some convention.

- With additive regression, $E[Y|X = x] = \alpha + \sum_j f_j(x_j)$, we can add arbitrary constants so long as they cancel out. That is, we get the same predictions from $\alpha + c_0 + \sum_j f_j(x_j) + c_j$ when $c_0 = -\sum_j c_j$. This is another model of the same form, $\alpha' + \sum_j f'_j(x_j)$, so it's not identifiable. We dealt with this by imposing

²See Bartholomew (1987, pp. 36–38). The proof is tedious algebraically.

the convention that $\alpha = \mathbb{E}[Y]$ and $\mathbb{E}[f_j(X_j)] = 0$ — we picked out a favorite, convenient representation from the infinite collection of equivalent representations.

- Linear regression becomes unidentifiable with collinear features. Collinearity is a good reason to not use linear regression (i.e., we change the model.)
- Factor analysis is unidentifiable because of the rotation problem. Some people respond by trying to fix on a particular representation, others just ignore it.

Two kinds of identification problems are common for mixture models; one is trivial and the other is fundamental. The trivial one is that we can always swap the labels of any two components with no effect on anything observable at all — if we decide that component number 1 is now component number 7 and vice versa, that doesn't change the distribution of X at all. This **label degeneracy** can be annoying, especially for some estimation algorithms, but that's the worst of it.

A more fundamental lack of identifiability happens when mixing two distributions from a parametric family just gives us a third distribution from the same family. For example, suppose we have a single binary feature, say an indicator for whether someone will pay back a credit card. We might think there are two kinds of customers, with high- and low- risk of not paying, and try to represent this as a mixture of Bernoulli distribution. If we try this, we'll see that we've gotten a *single* Bernoulli distribution with an intermediate risk of repayment. A mixture of Bernoulli is always just another Bernoulli. More generally, a mixture of discrete distributions over any finite number of categories is just another distribution over those categories³

19.1.6 Probabilistic Clustering

Yet another way to view mixture models, which I hinted at when I talked about how they are a way of putting similar data points together into “clusters”, where clusters are represented by, precisely, the component distributions. The idea is that all data points of the same type, belonging to the same cluster, are more or less equivalent and all come from the same distribution, and any differences between them are matters of chance. This view *exactly* corresponds to mixture models like Eq. 19.1; the hidden variable Z I introduced above is just the cluster label.

One of the very nice things about probabilistic clustering is that Eq. 19.1 actually *claims* something about what the data looks like; it says that it follows a certain distribution. We can check whether it does, and we can check whether *new* data follows this distribution. If it does, great; if not, if the predictions systematically fail, then

³That is, a mixture of any two $n = 1$ multinomials is another $n = 1$ multinomial. This is not generally true when $n > 1$; for instance, a mixture of a $\text{Binom}(2, 0.75)$ and a $\text{Binom}(2, 0.25)$ is *not* a $\text{Binom}(2, p)$ for any p . (EXERCISE: show this.) However, both of those binomials is a distribution on $\{0, 1, 2\}$, and so is their mixture. This apparently trivial point actually leads into very deep topics, since it turns out that which models can be written as mixtures of others is strongly related to what properties of the data-generating process can actually be learned from data: see Lauritzen (1984). (Thanks to Bob Carpenter for pointing out an error in an earlier draft.)

the model is wrong. We can compare different probabilistic clusterings by how well they predict (say under cross-validation).⁴

In particular, probabilistic clustering gives us a sensible way of answering the question “how many clusters?” The best number of clusters to use is the number which will best generalize to future data. If we don’t want to wait around to get new data, we can approximate generalization performance by cross-validation, or by any other adaptive model selection procedure.

19.1.7 Simulation

Simulating from a mixture model works rather like simulating from a kernel density estimate. To draw a new value \tilde{X} , first draw a random integer Z from 1 to k , with probabilities λ_k , then draw from the Z^{th} mixture component. (That is, $\tilde{X}|Z \sim f_Z$.) Note that if we want multiple draws, $\tilde{X}_1, \tilde{X}_2, \dots, \tilde{X}_b$, each of them needs an independent Z ,

19.2 Estimating Parametric Mixture Models

From intro stats., we remember that it’s generally a good idea to estimate distributions using maximum likelihood, when we can. How could we do that here?

Remember that the likelihood is the probability (or probability density) of observing our data, as a function of the parameters. Assuming independent samples, that would be

$$\prod_{i=1}^n f(x_i; \theta) \quad (19.5)$$

for observations x_1, x_2, \dots, x_n . As always, we’ll use the logarithm to turn multiplication into addition:

$$\ell(\theta) = \sum_{i=1}^n \log f(x_i; \theta) \quad (19.6)$$

$$= \sum_{i=1}^n \log \sum_{k=1}^K \lambda_k f(x_i; \theta_k) \quad (19.7)$$

⁴Contrast this with k -means or hierarchical clustering, which you may have seen in other classes: they make no predictions, and so we have no way of telling if they are right or wrong. Consequently, comparing different non-probabilistic clusterings is a lot harder!

Let's try taking the derivative of this with respect to one parameter, say θ_j .

$$\frac{\partial \ell}{\partial \theta_j} = \sum_{i=1}^n \frac{1}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} \lambda_j \frac{\partial f(x_i; \theta_j)}{\partial \theta_j} \quad (19.8)$$

$$= \sum_{i=1}^n \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} \frac{1}{f(x_i; \theta_j)} \frac{\partial f(x_i; \theta_j)}{\partial \theta_j} \quad (19.9)$$

$$= \sum_{i=1}^n \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} \frac{\partial \log f(x_i; \theta_j)}{\partial \theta_j} \quad (19.10)$$

If we just had an ordinary parametric model, on the other hand, the derivative of the log-likelihood would be

$$\sum_{i=1}^n \frac{\partial \log f(x_i; \theta_j)}{\partial \theta_j} \quad (19.11)$$

So maximizing the likelihood for a mixture model is like doing a *weighted* likelihood maximization, where the weight of x_i depends on cluster, being

$$w_{ij} = \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} \quad (19.12)$$

The problem is that these weights depend on the parameters we are trying to estimate!

Let's look at these weights w_{ij} a bit more. Remember that λ_j is the probability that the hidden class variable Z is j , so the numerator in the weights is the joint probability of getting $Z = j$ and $X = x_i$. The denominator is the marginal probability of getting $X = x_i$, so the ratio is the conditional probability of $Z = j$ given $X = x_i$,

$$w_{ij} = \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} = p(Z = j | X = x_i; \theta) \quad (19.13)$$

If we try to estimate the mixture model, then, we're doing weighted maximum likelihood, with weights given by the posterior cluster probabilities. These, to repeat, depend on the parameters we are trying to estimate, so there seems to be a vicious circle.

But, as the saying goes, one man's vicious circle is another man's successive approximation procedure. A crude way of doing this⁵ would start with an initial guess about the component distributions; find out which component each point is most likely to have come from; re-estimate the components using only the points assigned to it, etc., until things converge. This corresponds to taking all the weights w_{ij} to be either 0 or 1. However, it does not maximize the likelihood, since we've seen that to do so we need fractional weights.

What's called the EM algorithm is simply the obvious refinement of this "hard" assignment strategy.

⁵Related to what's called " k -means" clustering.

1. Start with guesses about the mixture components $\theta_1, \theta_2, \dots, \theta_K$ and the mixing weights $\lambda_1, \dots, \lambda_K$.
2. Until nothing changes very much:
 - (a) Using the current parameter guesses, calculate the weights w_i , (E-step)
 - (b) Using the current weights, maximize the weighted likelihood to get new parameter estimates (M-step)
3. Return the final parameter estimates (including mixing proportions) and cluster probabilities

The M in “M-step” and “EM” stands for “maximization”, which is pretty transparent. The E stands for “expectation”, because it gives us the conditional probabilities of different values of Z , and probabilities are expectations of indicator functions. (In fact in some early applications, Z was binary, so one really was computing the expectation of Z .) The whole thing is also called the “expectation-maximization” algorithm.

19.2.1 More about the EM Algorithm

The EM algorithm turns out to be a general way of maximizing the likelihood when some variables are unobserved, and hence useful for other things besides mixture models. So in this section, where I try to explain why it works, I am going to be a bit more general abstract. (Also, it will actually cut down on notation.) I'll pack the whole sequence of observations x_1, x_2, \dots, x_n into a single variable d (for “data”), and likewise the whole sequence of z_1, z_2, \dots, z_n into h (for “hidden”). What we want to do is maximize

$$\ell(\theta) = \log p(d; \theta) = \log \sum_b p(d, h; \theta) \quad (19.14)$$

This is generally hard, because even if $p(d, h; \theta)$ has a nice parametric form, that is lost when we sum up over all possible values of h (as we saw above). The essential trick of the EM algorithm is to maximize not the log likelihood, but a *lower bound* on the log-likelihood, which is more tractable; we'll see that this lower bound is sometimes tight, i.e., coincides with the actual log-likelihood, and in particular does so at the global optimum.

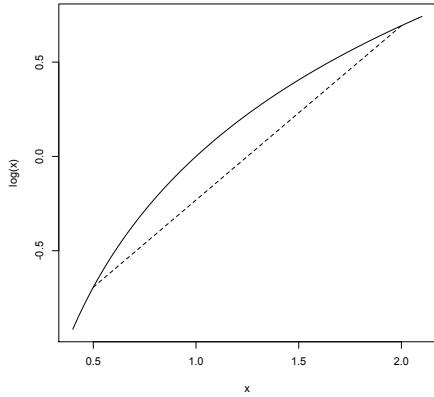
We can introduce an arbitrary⁶ distribution on h , call it $q(h)$, and we'll

$$\ell(\theta) = \log \sum_h p(d, h; \theta) \quad (19.15)$$

$$= \log \sum_h \frac{q(h)}{q(h)} p(d, h; \theta) \quad (19.16)$$

$$= \log \sum_h q(h) \frac{p(d, h; \theta)}{q(h)} \quad (19.17)$$

⁶Well, almost arbitrary; it shouldn't give probability zero to value of h which has positive probability for all θ .



```
curve(log(x),from=0.4,to=2.1)
segments(0.5,log(0.5),2,log(2),lty=2)
```

Figure 19.1: The logarithm is a concave function, i.e., the curve connecting any two points lies above the straight line doing so. Thus the average of logarithms is less than the logarithm of the average.

So far so trivial.

Now we need a geometric fact about the logarithm function, which is that its curve is concave: if we take any two points on the curve and connect them by a straight line, the curve lies above the line (Figure 19.1). Algebraically, this means that

$$w \log t_1 + (1-w) \log t_2 \leq \log w t_1 + (1-w)t_2 \quad (19.18)$$

for any $0 \leq w \leq 1$, and any points $t_1, t_2 > 0$. Nor does this just hold for two points: for any r points $t_1, t_2, \dots, t_r > 0$, and any set of non-negative weights $\sum_{i=1}^r w_i = 1$,

$$\sum_{i=1}^r w_i \log t_i \leq \log \sum_{i=1}^r w_i t_i \quad (19.19)$$

In words: the log of the average is at least the average of the logs. This is called **Jensen's inequality**. So

$$\log \sum_b q(b) \frac{p(d, b; \theta)}{q(b)} \geq \sum_b q(b) \log \frac{p(d, b; \theta)}{q(b)} \quad (19.20)$$

$$\equiv J(q, \theta) \quad (19.21)$$

We are bothering with this because we hope that it will be easier to maximize this lower bound on the likelihood than the actual likelihood, and the lower bound

is reasonably tight. As to tightness, suppose that $q(h) = p(h|d; \theta)$. Then

$$\frac{p(d, h; \theta)}{q(h)} = \frac{p(d, h; \theta)}{p(h|d; \theta)} = \frac{p(d, h; \theta)}{p(h, d; \theta)/p(d; \theta)} = p(d; \theta) \quad (19.22)$$

no matter what h is. So with that choice of q , $J(q, \theta) = \ell(\theta)$ and the lower bound is tight. Also, since $J(q, \theta) \leq \ell(\theta)$, this choice of q maximizes J for fixed θ .

Here's how the EM algorithm goes in this formulation.

1. Start with an initial guess $\theta^{(0)}$ about the components and mixing weights.
2. Until nothing changes very much
 - (a) E-step: $q^{(t)} = \operatorname{argmax}_q J(q, \theta^{(t)})$
 - (b) M-step: $\theta^{(t+1)} = \operatorname{argmax}_{\theta} J(q^{(t)}, \theta)$
3. Return final estimates of θ and q

The E and M steps are now nice and symmetric; both are about maximizing J . It's easy to see that, after the E step,

$$J(q^{(t)}, \theta^{(t)}) \geq J(q^{(t-1)}, \theta^{(t)}) \quad (19.23)$$

and that, after the M step,

$$J(q^{(t)}, \theta^{(t+1)}) \geq J(q^{(t)}, \theta^{(t)}) \quad (19.24)$$

Putting these two inequalities together,

$$J(q^{(t+1)}, \theta^{(t+1)}) \geq J(q^{(t)}, \theta^{(t)}) \quad (19.25)$$

$$\ell(\theta^{(t+1)}) \geq \ell(\theta^{(t)}) \quad (19.26)$$

So each EM iteration can only improve the likelihood, guaranteeing convergence to a local maximum. Since it only guarantees a local maximum, it's a good idea to try a few different initial values of $\theta^{(0)}$ and take the best.

We saw above that the maximization in the E step is just computing the posterior probability $p(h|d; \theta)$. What about the maximization in the M step?

$$\sum_b q(h) \log \frac{p(d, h; \theta)}{q(h)} = \sum_b q(h) \log p(d, h; \theta) - \sum_b q(h) \log q(h) \quad (19.27)$$

The second sum doesn't depend on θ at all, so it's irrelevant for maximizing, giving us back the optimization problem from the last section. This confirms that using the lower bound from Jensen's inequality hasn't yielded a different algorithm!

19.2.2 Further Reading on and Applications of EM

My presentation of the EM algorithm draws *heavily* on Neal and Hinton (1998).

Because it's so general, the EM algorithm is applied to *lots* of problems with missing data or latent variables. Traditional estimation methods for factor analysis, for example, can be replaced with EM. (Arguably, some of the older methods *were* versions of EM.) A common problem in time-series analysis and signal processing is that of “filtering” or “state estimation”: there's an unknown signal S_t , which we want to know, but all we get to observe is some noisy, corrupted measurement, $X_t = h(S_t) + \eta_t$. (A historically important example of a “state” to be estimated from noisy measurements is “Where is our rocket and which way is it headed?” — see McGee and Schmidt, 1985.) This is solved by the EM algorithm, with the signal as the hidden variable; Fraser (2008) gives a really good introduction to such models and how they use EM.

Instead of just doing mixtures of densities, one can also do mixtures of predictive models, say mixtures of regressions, or mixtures of classifiers. The hidden variable Z here controls which regression function to use. A general form of this is what's known as a **mixture-of-experts** model (Jordan and Jacobs, 1994; Jacobs, 1997) — each predictive model is an “expert”, and there can be a quite complicated set of hidden variables determining which expert to use when.

The EM algorithm is so useful and general that it has in fact been re-invented multiple times. The name “EM algorithm” comes from the statistics of mixture models in the late 1970s; in the time series literature it's been known since the 1960s as the “Baum-Welch” algorithm.

19.2.3 Topic Models and Probabilistic LSA

Mixture models over words provide an alternative to latent semantic indexing for document analysis. Instead of finding the principal components of the bag-of-words vectors, the idea is as follows. There are a certain number of **topics** which documents in the corpus can be about; each topic corresponds to a distribution over words. The distribution of words in a document is a mixture of the topic distributions. That is, one can generate a bag of words by first picking a topic according to a multinomial distribution (topic i occurs with probability λ_i), and then picking a word from that topic's distribution. The distribution of topics varies from document to document, and this is what's used, rather than projections on to the principal components, to summarize the document. This idea was, so far as I can tell, introduced by Hofmann (1999), who estimated everything by EM. **Latent Dirichlet allocation**, due to Blei and collaborators (Blei *et al.*, 2003) is an important variation which smoothes the topic distributions; there is a CRAN package called `lda`. Blei and Lafferty (2009) is a good recent review paper of the area.

19.3 Non-parametric Mixture Modeling

We could replace the M step of EM by some other way of estimating the distribution of each mixture component. This could be a fast-but-crude estimate of parameters

(say a method-of-moments estimator if that's simpler than the MLE), or it could even be a non-parametric density estimator of the type we talked about in Chapter 15. (Similarly for mixtures of regressions, etc.) Issues of dimensionality re-surface now, as well as convergence: because we're not, in general, increasing J at each step, it's harder to be sure that the algorithm will in fact converge. This is an active area of research.

19.4 Worked Computing Example: Snoqualmie Falls Revisited

19.4.1 Mixture Models in R

There are several R packages which implement mixture models. The `mclust` package (<http://www.stat.washington.edu/mclust/>) is pretty much standard for Gaussian mixtures. One of the most recent and powerful is `mixtools` (Benaglia *et al.*, 2009), which, in addition to classic mixtures of parametric densities, handles mixtures of regressions and some kinds of non-parametric mixtures. The `FlexMix` package (Leisch, 2004) is (as the name implies) very good at flexibly handling complicated situations, though you have to do some programming to take advantage of this.

19.4.2 Fitting a Mixture of Gaussians to Real Data

Let's go back to the Snoqualmie Falls data set, last used in §13.3. There we built a system to forecast whether there would be precipitation on day t , on the basis of how much precipitation there was on day $t - 1$. Let's look at the distribution of the amount of precipitation on the wet days.

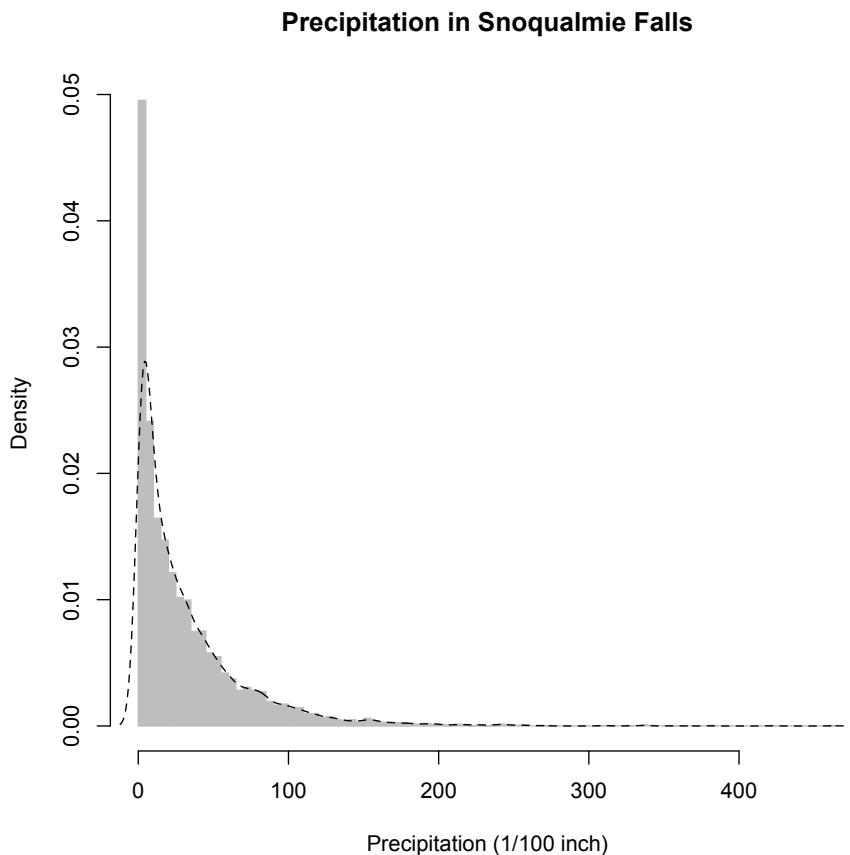
```
snoqualmie <- read.csv("snoqualmie.csv", header=FALSE)
snoqualmie.vector <- na.omit(unlist(snoqualmie))
snoq <- snoqualmie.vector[snoqualmie.vector > 0]
```

Figure 19.2 shows a histogram (with a fairly large number of bins), together with a simple kernel density estimate. This suggests that the distribution is rather skewed to the right, which is reinforced by the simple summary statistics

```
> summary(snoq)
   Min. 1st Qu. Median    Mean 3rd Qu.    Max.
1.00     6.00   19.00  32.28   44.00 463.00
```

Notice that the mean is larger than the median, and that the distance from the first quartile to the median is much smaller (13/100 of an inch of precipitation) than that from the median to the third quartile (25/100 of an inch). One way this could arise, of course, is if there are multiple types of wet days, each with a different characteristic distribution of precipitation.

We'll look at this by trying to fit Gaussian mixture models with varying numbers of components. We'll start by using a mixture of two Gaussians. We *could* code up



```
plot(hist(snoq,breaks=101),col="grey",border="grey",freq=FALSE,
      xlab="Precipitation (1/100 inch)",main="Precipitation in Snoqualmie Falls")
lines(density(snoq),lty=2)
```

Figure 19.2: Histogram (grey) for precipitation on wet days in Snoqualmie Falls. The dashed line is a kernel density estimate, which is not completely satisfactory. (It gives non-trivial probability to negative precipitation, for instance.)

the EM algorithm for fitting this mixture model from scratch, but instead we'll use the `mixtools` package.

```
library(mixtools)
snoq.k2 <- normalmixEM(snoq,k=2,maxit=100,epsilon=0.01)
```

The EM algorithm “runs until convergence”, i.e., until things change so little that we don't care any more. For the implementation in `mixtools`, this means running until the log-likelihood changes by less than `epsilon`. The default tolerance for convergence is not 10^{-2} , as here, but 10^{-8} , which can take a very long time indeed. The algorithm also stops if we go over a maximum number of iterations, even if it has not converged, which by default is 1000; here I have dialed it down to 100 for safety's sake. What happens?

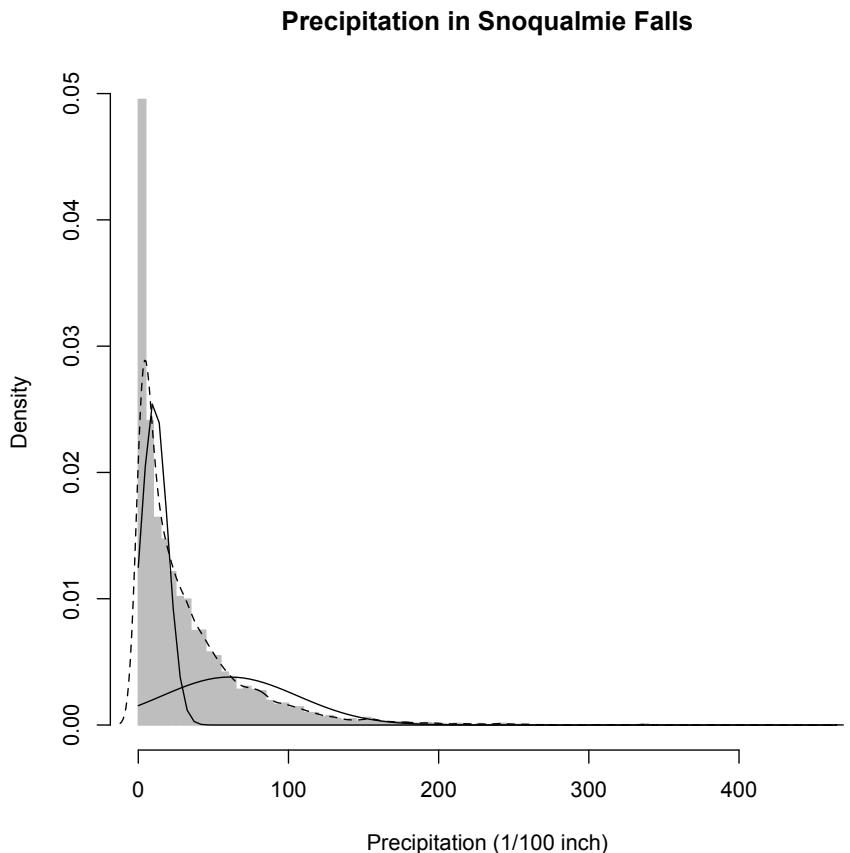
```
> snoq.k2 <- normalmixEM(snoq,k=2,maxit=100,epsilon=0.01)
number of iterations= 59
> summary(snoq.k2)
summary of normalmixEM object:
      comp 1     comp 2
lambda 0.557564 0.442436
mu     10.267390 60.012594
sigma   8.511383 44.998102
loglik at estimate: -32681.21
```

There are two components, with weights (`lambda`) of about 0.56 and 0.44, two means (`mu`) and two standard deviations (`sigma`). The over-all log-likelihood, obtained after 59 iterations, is -32681.21 . (Demanding convergence to $\pm 10^{-8}$ would thus have required the log-likelihood to change by less than one part in a trillion, which is quite excessive when we only have 6920 observations.)

We can plot this along with the histogram of the data and the non-parametric density estimate. I'll write a little function for it.

```
plot.normal.components <- function(mixture,component.number,...) {
  curve(mixture$lambda[component.number] *
    dnorm(x,mean=mixture$mu[component.number],
          sd=mixture$sigma[component.number]), add=TRUE, ...)
}
```

This adds the density of a given component to the current plot, but scaled by the share it has in the mixture, so that it is visually comparable to the over-all density.



```
plot(hist(snoq,breaks=101),col="grey",border="grey",freq=FALSE,
      xlab="Precipitation (1/100 inch)",main="Precipitation in Snoqualmie Falls")
lines(density(snoq),lty=2)
sapply(1:2,plot.normal.components,mixture=snoq.k2)
```

Figure 19.3: As in the previous figure, plus the components of a mixture of two Gaussians, fitted to the data by the EM algorithm (dashed lines). These are scaled by the mixing weights of the components.

19.4.3 Calibration-checking for the Mixture

Examining the two-component mixture, it does not look altogether satisfactory — it seems to consistently give too much probability to days with about 1 inch of precipitation. Let's think about how we could check things like this.

When we looked at logistic regression, we saw how to check probability forecasts by checking calibration — events predicted to happen with probability p should in fact happen with frequency $\approx p$. Here we don't have a binary event, but we do have lots of probabilities. In particular, we have a cumulative distribution function $F(x)$, which tells us the probability that the precipitation is $\leq x$ on any given day. When x is continuous and has a continuous distribution, $F(x)$ should be uniformly distributed.⁷ The CDF of a two-component mixture is

$$F(x) = \lambda_1 F_1(x) + \lambda_2 F_2(x) \quad (19.28)$$

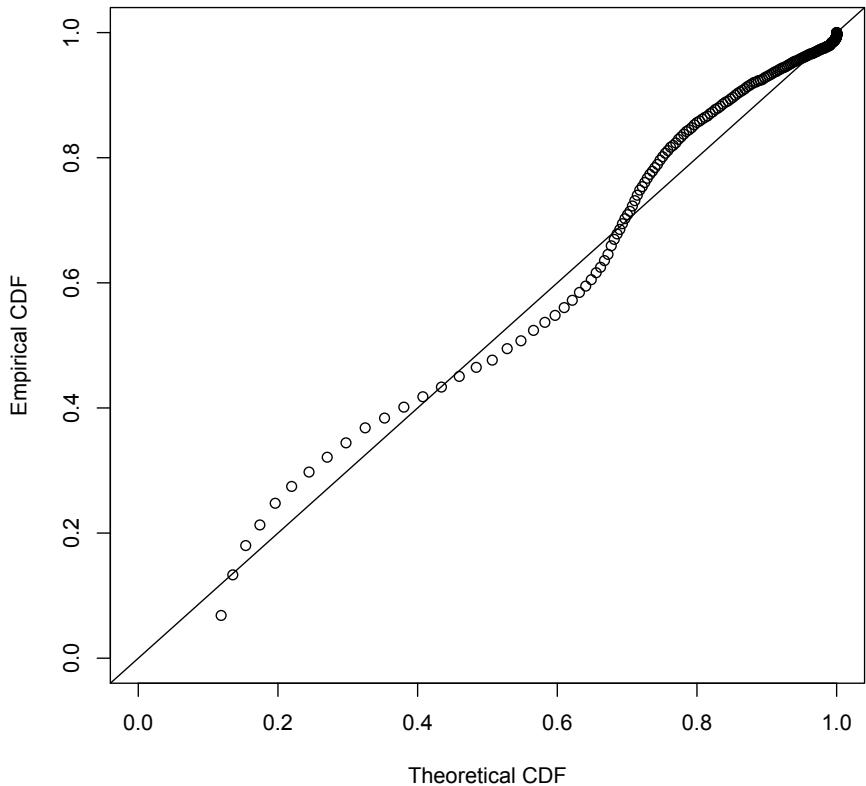
and similarly for more components. A little R experimentation gives a function for computing the CDF of a Gaussian mixture:

```
pnormmmix <- function(x,mixture) {
  lambda <- mixture$lambda
  k <- length(lambda)
  pnorm.from.mix <- function(x,component) {
    lambda[component]*pnorm(x,mean=mixture$mu[component],
                           sd=mixture$sigma[component])
  }
  pnorms <- sapply(1:k,pnorm.from.mix,x=x)
  return(rowSums(pnorms))
}
```

and so produce a plot like Figure 19.4.3. We do not have the tools to assess whether the *size* of the departure from the main diagonal is significant⁸, but the fact that the errors are so very structured is rather suspicious.

⁷We saw this principle when we looked at generating random variables in Chapter 5.

⁸Though we could: the most straight-forward thing to do would be to simulate from the mixture, and repeat this with simulation output.



```

distinct.snoq <- sort(unique(snoq))
tcdfs <- pnormmmix(distinct.snoq,mixture=snoq.k2)
ecdfs <- ecdf(snoq)(distinct.snoq)
plot(tcdfs,ecdfs,xlab="Theoretical CDF",ylab="Empirical CDF",xlim=c(0,1),
      ylim=c(0,1))
abline(0,1)

```

Figure 19.4: Calibration plot for the two-component Gaussian mixture. For each distinct value of precipitation x , we plot the fraction of days predicted by the mixture model to have $\leq x$ precipitation on the horizontal axis, versus the actual fraction of days $\leq x$.

19.4.4 Selecting the Number of Components by Cross-Validation

Since a two-component mixture seems iffy, we could consider using more components. By going to three, four, etc. components, we improve our in-sample likelihood, but of course expose ourselves to the danger of over-fitting. Some sort of model selection is called for. We could do cross-validation, or we could do hypothesis testing. Let's try cross-validation first.

We can already do fitting, but we need to calculate the log-likelihood on the held-out data. As usual, let's write a function; in fact, let's write two.

```
dnormalmix <- function(x,mixture,log=FALSE) {
  lambda <- mixture$lambda
  k <- length(lambda)
  # Calculate share of likelihood for all data for one component
  like.component <- function(x,component) {
    lambda[component]*dnorm(x,mean=mixture$mu[component],
                            sd=mixture$sigma[component])
  }
  # Create array with likelihood shares from all components over all data
  likes <- sapply(1:k,like.component,x=x)
  # Add up contributions from components
  d <- rowSums(likes)
  if (log) {
    d <- log(d)
  }
  return(d)
}

loglike.normalmix <- function(x,mixture) {
  loglike <- dnormalmix(x,mixture,log=TRUE)
  return(sum(loglike))
}
```

To check that we haven't made a big mistake in the coding:

```
> loglike.normalmix(snoq,mixture=snoq.k2)
[1] -32681.2
```

which matches the log-likelihood reported by `summary(snoq.k2)`. But our function can be used on different data!

We *could* do five-fold or ten-fold CV, but just to illustrate the approach we'll do simple data-set splitting, where a randomly-selected half of the data is used to fit the model, and half to test.

```
n <- length(snoq)
data.points <- 1:n
data.points <- sample(data.points) # Permute randomly
train <- data.points[1:floor(n/2)] # First random half is training
```

```

test <- data.points[-(1:floor(n/2))] # 2nd random half is testing
candidate.component.numbers <- 2:10
loglikes <- vector(length=1+length(candidate.component.numbers))
# k=1 needs special handling
mu<-mean(snoq[train]) # MLE of mean
sigma <- sd(snoq[train])*sqrt((n-1)/n) # MLE of standard deviation
loglikes[1] <- sum(dnorm(snoq[test],mu,sigma,log=TRUE))
for (k in candidate.component.numbers) {
  mixture <- normalmixEM(snoq[train],k=k,maxit=400,epsilon=1e-2)
  loglikes[k] <- loglike.normalmix(snoq[test],mixture=mixture)
}

```

When you run this, you will probably see a lot of warning messages saying “One of the variances is going to zero; trying new starting values.” The issue is that we can give any one value of x arbitrarily high likelihood by centering a Gaussian there and letting its variance shrink towards zero. This is however generally considered unhelpful — it leads towards the pathologies that keep us from doing pure maximum likelihood estimation in non-parametric problems (Chapter 15) — so when that happens the code recognizes it and starts over.

If we look at the log-likelihoods, we see that there is a dramatic improvement with the first few components, and then things slow down a lot⁹:

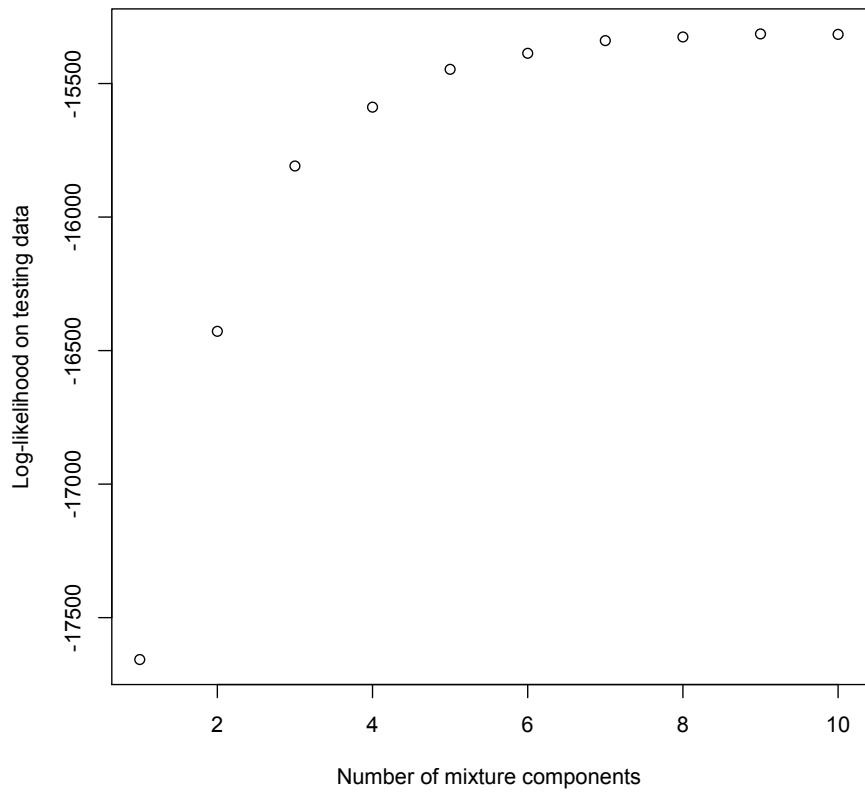
```

> loglikes
[1] -17656.86 -16427.83 -15808.77 -15588.44 -15446.77 -15386.74
[7] -15339.25 -15325.63 -15314.22 -15315.88

```

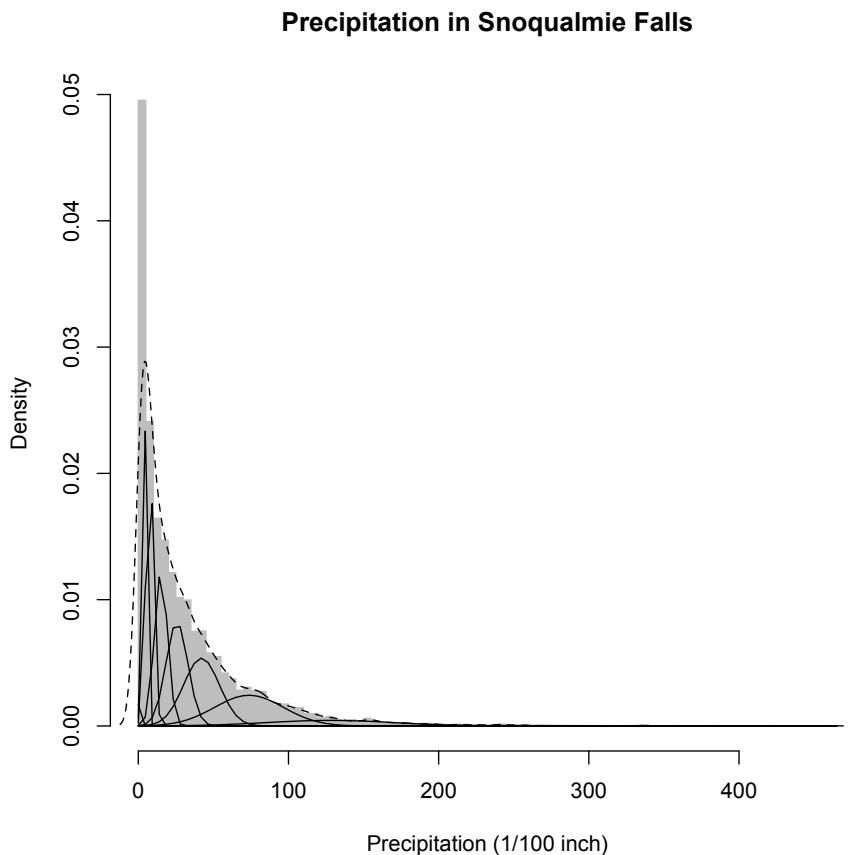
(See also Figure 19.5). This favors nine components to the mixture. It looks like Figure 19.6. The calibration is now nearly perfect, at least on the training data (Figure 19.4.4).

⁹Notice that the numbers here are about half of the log-likelihood we calculated for the two-component mixture on the complete data. This is as it should be, because log-likelihood is proportional to the number of observations. (Why?) It’s more like the *sum* of squared errors than the *mean* squared error. If we want something which is directly comparable across data sets of different size, we should use the log-likelihood per observation.



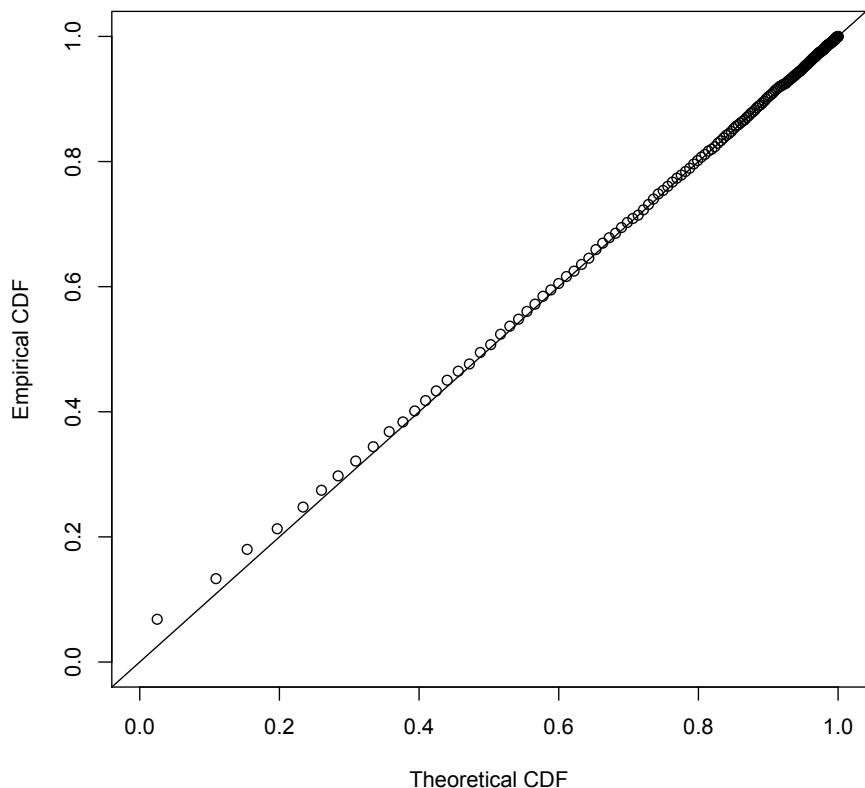
```
plot(x=1:10, y=loglikes, xlab="Number of mixture components",
      ylab="Log-likelihood on testing data")
```

Figure 19.5: Log-likelihoods of different sizes of mixture models, fit to a random half of the data for training, and evaluated on the other half of the data for testing.



```
snoq.k9 <- normalmixEM(snoq,k=9,maxit=400,epsilon=1e-2)
plot(hist(snoq,breaks=101),col="grey",border="grey",freq=FALSE,
     xlab="Precipitation (1/100 inch)",main="Precipitation in Snoqualmie Falls")
lines(density(snoq),lty=2)
sapply(1:9,plot.normal.components,mixture=snoq.k9)
```

Figure 19.6: As in Figure 19.3, but using the nine-component Gaussian mixture.



```
distinct.snoq <- sort(unique(snoq))
tcdfs <- pnormmmix(distinct.snoq,mixture=snoq.k9)
ecdfs <- ecdf(snoq)(distinct.snoq)
plot(tcdfs,ecdfs,xlab="Theoretical CDF",ylab="Empirical CDF",xlim=c(0,1),
     ylim=c(0,1))
abline(0,1)
```

Figure 19.7: Calibration plot for the nine-component Gaussian mixture.

19.4.5 Interpreting the Mixture Components, or Not

The components of the mixture are far from arbitrary. It appears from Figure 19.6 that as the mean increases, so does the variance. This impression is confirmed from Figure 19.8. Now it *could* be that there really are nine types of rainy days in Snoqualmie Falls which just so happen to have this pattern of distributions, but this seems a bit suspicious — as though the mixture is trying to use Gaussians systematically to approximate a fundamentally different distribution, rather than get at something which really is composed of nine distinct Gaussians. This judgment relies on our scientific understanding of the weather, which makes us surprised by seeing a pattern like this in the parameters. (Calling this “scientific knowledge” is a bit excessive, but you get the idea.) Of course we are sometimes wrong about things like this, so it is certainly not conclusive. Maybe there really *are* nine types of days, each with a Gaussian distribution, and some subtle meteorological reason why their means and variances should be linked like this. For that matter, maybe our understanding of meteorology is wrong.

There are two directions to take this: the purely statistical one, and the substantive one.

On the purely statistical side, if all we care about is being able to describe the distribution of the data and to predict future precipitation, then it doesn’t really matter whether the nine-component Gaussian mixture is true in any ultimate sense. Cross-validation picked nine components not because there really are nine types of days, but because a nine-component model had the best trade-off between approximation bias and estimation variance. The selected mixture gives a pretty good account of itself, nearly the same as the kernel density estimate (Figure 19.9). It requires 26 parameters¹⁰, which may seem like a lot, but the kernel density estimate requires keeping around all 6920 data points plus a bandwidth. On sheer economy, the mixture then has a lot to recommend it.

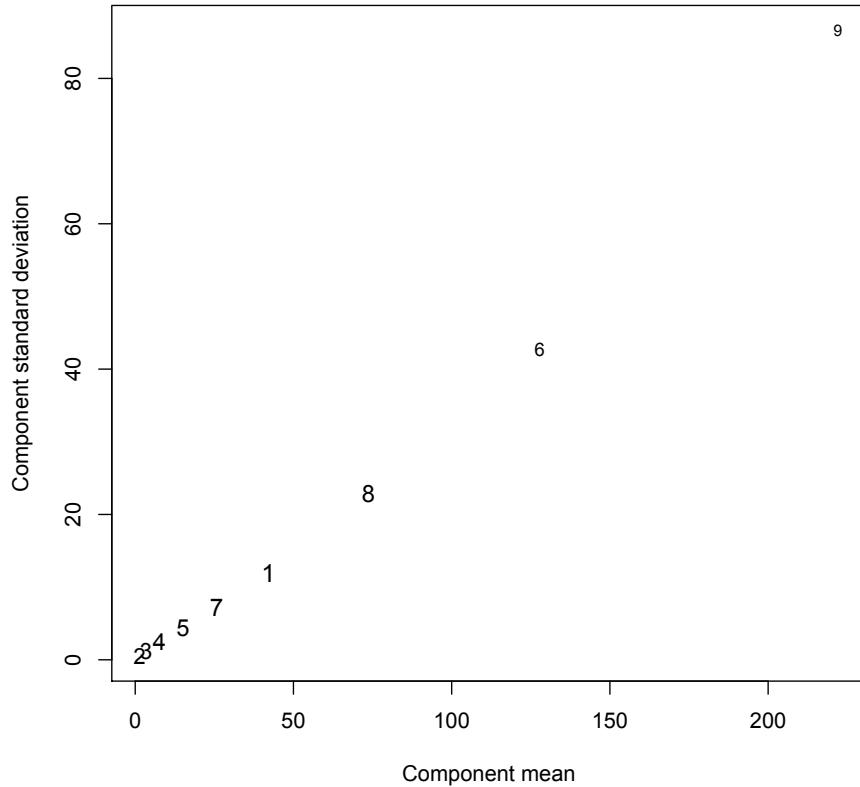
On the substantive side, there are various things we could do to check the idea that wet days really do divide into nine types. These are going to be informed by our background knowledge about the weather. One of the things we know, for example, is that weather patterns more or less repeat in an annual cycle, and that different types of weather are more common in some parts of the year than in others. If, for example, we consistently find type 6 days in August, that suggests that is at least compatible with these being real, meteorological patterns, and not just approximation artifacts.

Let’s try to look into this visually. `snoq.k9$posterior` is a 6920×9 array which gives the probability for each day to belong to each class. I’ll boil this down to assigning each day to its most probable class:

```
day.classes <- apply(snoq.k9$posterior, 1, which.max)
```

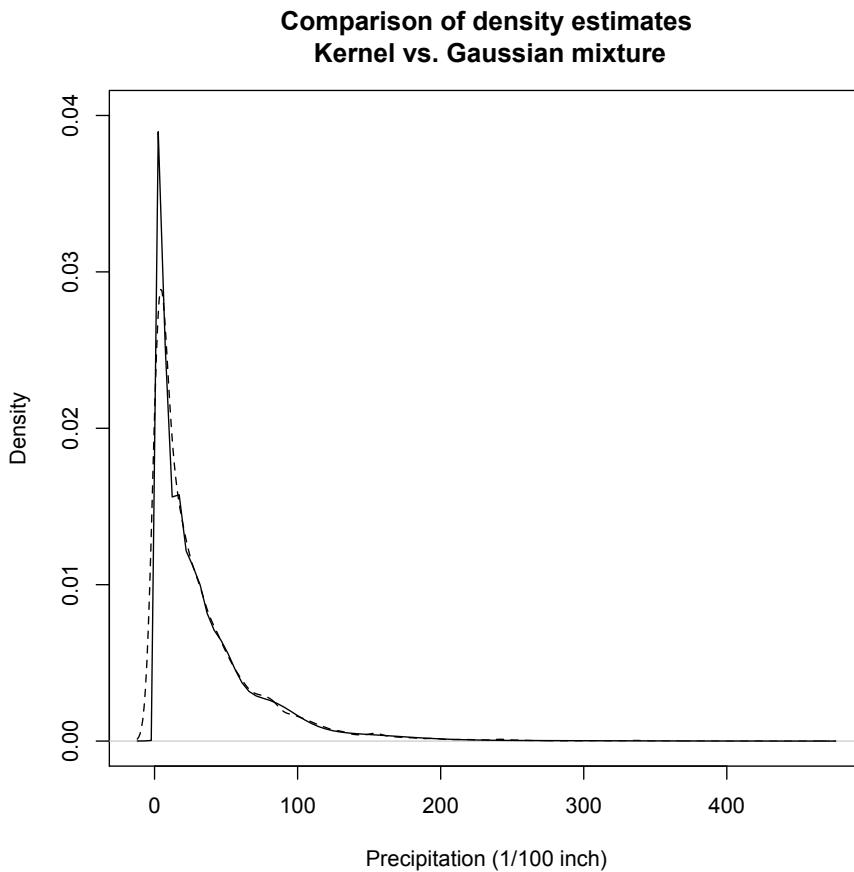
We can’t just plot this and hope to see any useful patterns, because we want to see stuff recurring every year, and we’ve stripped out the dry days, the division into years, the padding to handle leap-days, etc. Fortunately, `snoqualmie` has all that, so we’ll make a copy of that and edit `day.classes` into it.

¹⁰A mean and a standard deviation for each of nine components (=18 parameters), plus mixing weights (nine of them, but they have to add up to one).



```
plot(0,xlim=range(snoq.k9$mu),ylim=range(snoq.k9$sigma),type="n",
      xlab="Component mean", ylab="Component standard deviation")
points(x=snoq.k9$mu,y=snoq.k9$sigma,pch=as.character(1:9),
       cex=sqrt(0.5+5*snoq.k9$lambda))
```

Figure 19.8: Characteristics of the components of the 9-mode Gaussian mixture. The horizontal axis gives the component mean, the vertical axis its standard deviation. The area of the number representing each component is proportional to the component's mixing weight.



```
plot(density(snoq),lty=2,ylim=c(0,0.04),
  main=paste("Comparison of density estimates\n",
             "Kernel vs. Gaussian mixture"),
  xlab="Precipitation (1/100 inch)")
curve(dnormalmix(x,snoq.k9),add=TRUE)
```

Figure 19.9: Dashed line: kernel density estimate. Solid line: the nine-Gaussian mixture. Notice that the mixture, unlike the KDE, gives negligible probability to negative precipitation.

```
snoqualmie.classes <- snoqualmie
wet.days <- (snoqualmie > 0) & !(is.na(snoqualmie))
snoqualmie.classes[wet.days] <- day.classes
```

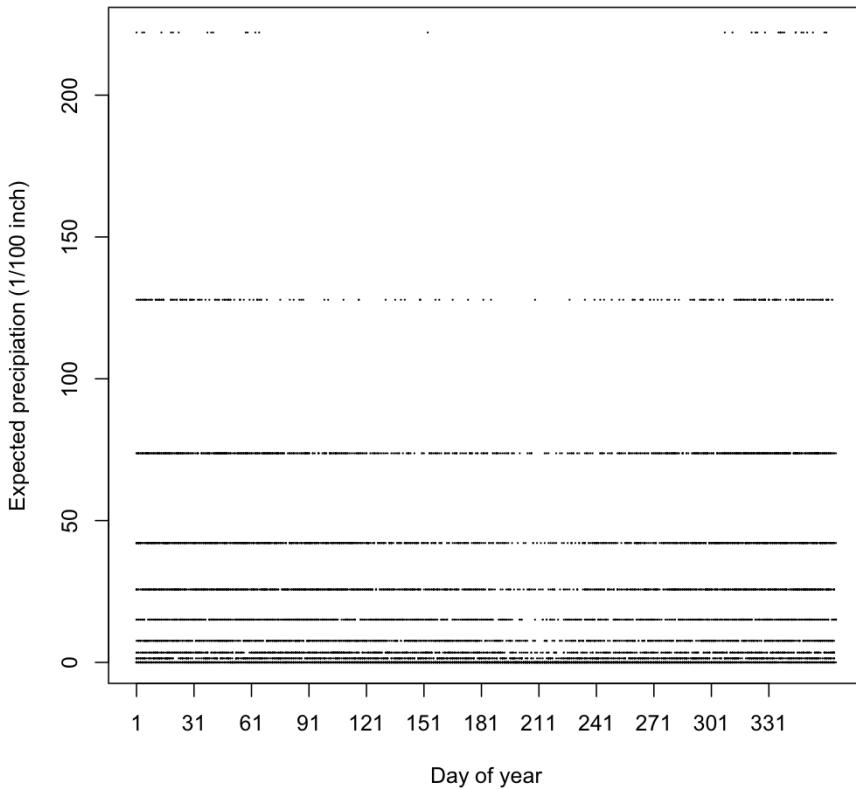
(Note that `wet.days` is a 36×366 logical array.) Now, it's somewhat inconvenient that the index numbers of the components do not perfectly correspond to the mean amount of precipitation — class 9 really is more similar to class 6 than to class 8. (See Figure 19.8.) Let's try replacing the numerical labels in `snoqualmie.classes` by those means.

```
snoqualmie.classes[wet.days] <- snoq.k9$mu[day.classes]
```

This leaves alone dry days (still zero) and NA days (still NA). Now we can plot (Figure 19.10).

The result is discouraging if we want to read any deeper meaning into the classes. The class with the heaviest amounts of precipitation is most common in the winter, but so is the classes with the second-heaviest amount of precipitation, the etc. It looks like the weather changes smoothly, rather than really having discrete classes. In this case, the mixture model seems to be merely a predictive device, and not a revelation of hidden structure.¹¹

¹¹A distribution called a “type II generalized Pareto”, where $p(x) \propto (1 + x/\sigma)^{-\theta-1}$, provides a decent fit here. (See Shalizi 2007; Arnold 1983 on this distribution and its estimation.) With only two parameters, rather than 26, its log-likelihood is only 1% higher than that of the nine-component mixture, and it is almost but not quite as calibrated. One origin of the type II Pareto is as a mixture of exponentials (Maguire *et al.*, 1952). If $X|Z \sim \text{Exp}(\sigma/Z)$, and Z itself has a Gamma distribution, $Z \sim \Gamma(\theta, 1)$, then the unconditional distribution of X is type II Pareto with scale σ and shape θ . We might therefore investigate fitting a finite mixture of exponentials, rather than of Gaussians, for the Snoqualmie Falls data. We might of course still end up concluding that there is a continuum of different sorts of days, rather than a finite set of discrete types.



```
plot(0,xlim=c(1,366),ylim=range(snoq.k9$mu),type="n",xaxt="n",
      xlab="Day of year",ylab="Expected precipitation (1/100 inch)")
axis(1,at=1+(0:11)*30)
for (year in 1:nrow(snoqualmie.classes)) {
  points(1:366,snoqualmie.classes[year,],pch=16,cex=0.2)
}
```

Figure 19.10: Plot of days classified according to the nine-component mixture. Horizontal axis: day of the year, numbered from 1 to 366 (to handle leap-years). Vertical axis: expected amount of precipitation on that day, according to the most probable class for the day.

19.4.6 Hypothesis Testing for Mixture-Model Selection

An alternative to using cross-validation to select the number of mixtures is to use hypothesis testing. The k -component Gaussian mixture model is nested within the $(k+1)$ -component model, so the latter must have a strictly higher likelihood on the training data. If the data really comes from a k -component mixture (the null hypothesis), then this extra increment of likelihood will follow one distribution, but if the data come from a larger model (the alternative), the distribution will be different, and stochastically larger.

Based on general likelihood theory, we might expect that the null distribution is, for large sample sizes,

$$2(\log L_{k+1} - \log L_k) \sim \chi^2_{\dim(k+1) - \dim(k)} \quad (19.29)$$

where L_k is the likelihood under the k -component mixture model, and $\dim(k)$ is the number of parameters in that model. (See Appendix C.) There are however several reasons to distrust such an approximation, including the fact that we are approximating the likelihood through the EM algorithm. We can instead just find the null distribution by simulating from the smaller model, which is to say we can do a parametric bootstrap.

While it is not too hard to program this by hand (Exercise 4), the `mixtools` package contains a function to do this for us, called `boot.comp`, for “bootstrap comparison”. Let’s try it out¹².

```
# See footnote regarding this next command
source("http://www.stat.cmu.edu/~cshalizi/402/lectures/20-mixture-examples/bootcomp.R")
snoq.boot <- boot.comp(snoq, max.comp=10, mix.type="normalmix",
                        maxit=400, epsilon=1e-2)
```

This tells `boot.comp()` to consider mixtures of up to 10 components (just as we did with cross-validation), increasing the size of the mixture it uses when the difference between k and $k+1$ is significant. (The default is “significant at the 5% level”, as assessed by 100 bootstrap replicates, but that’s controllable.) The command also tells it what kind of mixture to use, and passes along control settings to the EM algorithm which does the fitting. Each individual fit is fairly time-consuming, and we are requiring 100 at each value of k . This took about five minutes to run on my laptop.

This selected three components (rather than nine), and accompanied this decision with a rather nice trio of histograms explaining why (Figure 19.11). Remember that `boot.comp()` stops expanding the model when there’s even a 5% chance of that the apparent improvement could be due to mere over-fitting. This is actually pretty conservative, and so ends up with rather fewer components than cross-validation.

Let’s explore the output of `boot.comp()`, conveniently stored in the object `snoq.boot`.

¹²As of this writing (5 April 2011), there is a subtle, only-sporadically-appearing bug in the version of this function which is part of the released package. The `bootcomp.R` file on the class website contains a fix, kindly provided by Dr. Derek Young, and should be sourced after loading the package, as in the code example following. Dr. Young informs me that the fix will be incorporated in the next release of the `mixtools` package, scheduled for later this month.

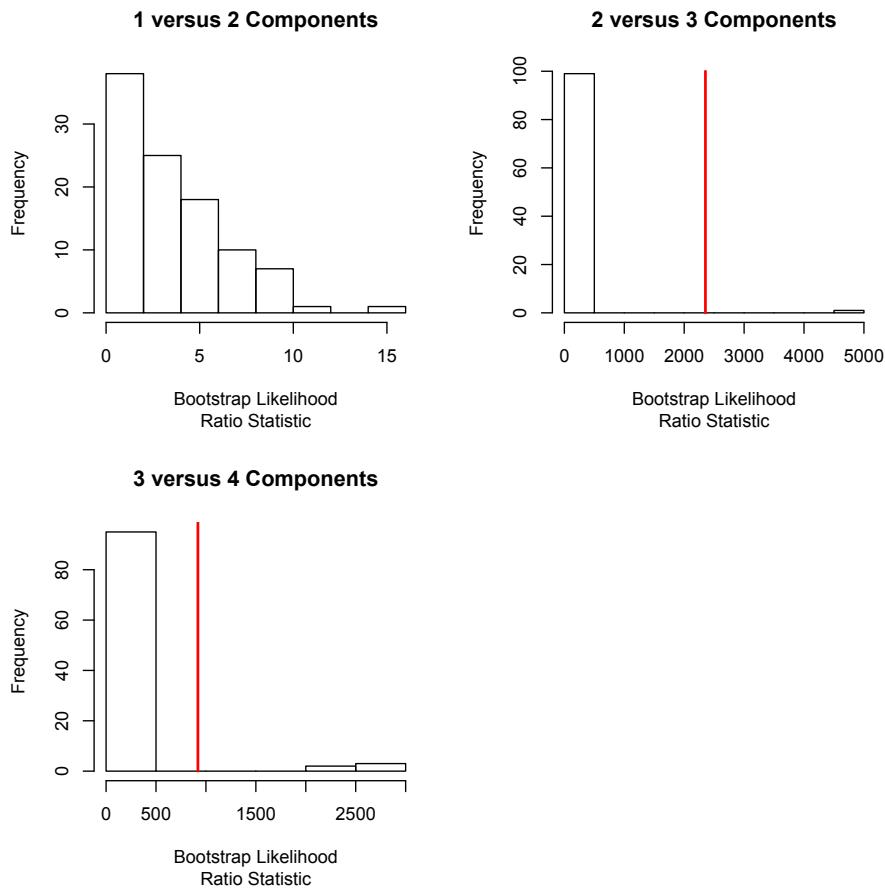


Figure 19.11: Histograms produced by `boot.comp()`. The vertical red lines mark the observed difference in log-likelihoods.

```
> str(snoq.boot)
List of 3
$ p.values   : num [1:3] 0 0.01 0.05
$ log.lik    :List of 3
..$ : num [1:100] 5.889 1.682 9.174 0.934 4.682 ...
..$ : num [1:100] 2.434 0.813 3.745 6.043 1.208 ...
..$ : num [1:100] 0.693 1.418 2.372 1.668 4.084 ...
$ obs.log.lik: num [1:3] 5096 2354 920
```

This tells us that `snoq.boot` is a list with three elements, called `p.values`, `log.lik` and `obs.log.lik`, and tells us a bit about each of them. `p.values` contains the *p*-values for testing H_1 (one component) against H_2 (two components), testing H_2 against H_3 , and H_3 against H_4 . Since we set a threshold *p*-value of 0.05, it stopped at the last test, accepting H_3 . (Under these circumstances, if the difference between $k = 3$ and $k = 4$ was really important to us, it would probably be wise to increase the number of bootstrap replicates, to get more accurate *p*-values.) `log.lik` is itself a list containing the bootstrapped log-likelihood ratios for the three hypothesis tests; `obs.log.lik` is the vector of corresponding observed values of the test statistic.

Looking back to Figure 19.5, there is indeed a dramatic improvement in the generalization ability of the model going from one component to two, and from two to three, and diminishing returns to complexity thereafter. Stopping at $k = 3$ produces pretty reasonable results, though repeating the exercise of Figure 19.10 is no more encouraging for the reality of the latent classes.

19.5 Exercises

To think through, not to hand in.

1. Write a function to simulate from a Gaussian mixture model. Check that it works by comparing a density estimated on its output to the theoretical density.
2. Work through the E-step and M-step for a mixture of two Poisson distributions.
3. Code up the EM algorithm for a mixture of K Gaussians. Simulate data from $K = 3$ Gaussians. How well does your code assign data-points to components if you give it the actual Gaussian parameters as your initial guess? If you give it other initial parameters?
4. Write a function to find the distribution of the log-likelihood ratio for testing the hypothesis that the mixture has k Gaussian components against the alternative that it has $k + 1$, by simulating from the k -component model. Compare the output to the `boot.comp` function in `mixtools`.
5. Write a function to fit a mixture of exponential distributions using the EM algorithm. Does it do any better at discovering sensible structure in the Snoqualmie Falls data?
6. Explain how to use relative distribution plots to check calibration, along the lines of Figure 19.4.3.

Chapter 20

Graphical Models

We have spent a lot of time looking at ways of figuring out how one variable (or set of variables) depends on another variable (or set of variables) — this is the core idea in regression and in conditional density estimation. We have also looked at how to estimate the joint distribution of variables, both with kernel density estimation and with models like factor and mixture models. The later two show an example of how to get the joint distribution by combining a conditional distribution (observables given factors; mixture components) with a marginal distribution (Gaussian distribution of factors; the component weights). When dealing with complex sets of dependent variables, it would be nice to have a general way of composing conditional distributions together to get joint distributions, and especially nice if this gave us a way of reasoning about what we could ignore, of seeing which variables are irrelevant to which other variables. This is what **graphical models** let us do.

20.1 Conditional Independence and Factor Models

The easiest way into this may be to start with the diagrams we drew for factor analysis. There, we had observables and we had factors, and each observable depended on, or loaded on, some of the factors. We drew a diagram where we had nodes, standing for the variables, and arrows running from the factors to the observables which depended on them. In the factor model, all the observables were conditionally independent of each other, given all the factors:

$$p(X_1, X_2, \dots, X_p | F_1, F_2, \dots, F_q) = \prod_{i=1}^p p(X_i | F_1, \dots, F_q) \quad (20.1)$$

But in fact observables are also independent of the factors they do not load on, so this is still too complicated. Let's write $\text{loads}(i)$ for the set of factors on which the observable X_i loads. Then

$$p(X_1, X_2, \dots, X_p | F_1, F_2, \dots, F_q) = \prod_{i=1}^p p(X_i | F_{\text{loads}(i)}) \quad (20.2)$$

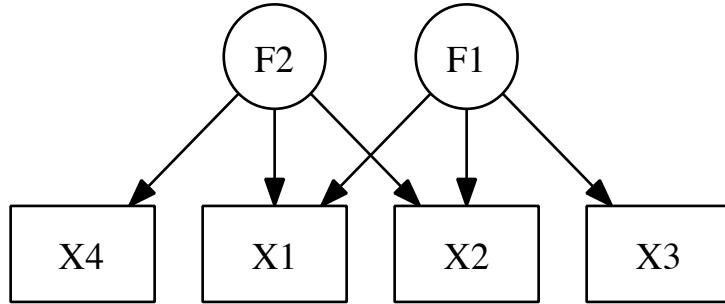


Figure 20.1: Illustration of a typical model with two latent factors (F_1 and F_2 , in circles) and four observables (X_1 through X_4).

Consider Figure 20.1. The conditional distribution of observables given factors is

$$p(X_1, X_2, X_3, X_4 | F_1, F_2) = p(X_1 | F_1, F_2)p(X_2 | F_1, F_2)p(X_3 | F_1)p(X_4 | F_2) \quad (20.3)$$

X_1 loads on F_1 and F_2 , so it is independent of everything else, given those two variables. X_1 is unconditionally dependent on X_2 , because they load on common factors, F_1 and F_2 ; and X_1 and X_3 are also dependent, because they both load on F_1 . In fact, X_1 and X_2 are still dependent given F_1 , because X_2 still gives information about F_2 . But X_1 and X_3 are independent given F_1 , because they have no other factors in common. Finally, X_3 and X_4 are unconditionally independent because they have no factors in common. But they become dependent given X_1 , which provides information about both the common factors.

None of these assertions rely on the detailed assumptions of the factor model, like Gaussian distributions for the factors, or linear dependence between factors and observables. What they rely on is that X_i is independent of *everything else*, given the factors it loads on. The idea of graphical models is to generalize this, by focusing on relations of direct dependence, and the conditional independence relations implied by them.

20.2 Directed Acyclic Graph (DAG) Models

We have a collection of variables, which to be generic I'll write X_1, X_2, \dots, X_p . These may be discrete, continuous, or even vectors; it doesn't matter. We represent these visually as nodes in a graph. There are arrows connecting some of these nodes. If an

arrow runs from X_i to X_j , then X_i is a **parent** of X_j . This is, as the name “parent” suggests, an anti-symmetric relationship, i.e., X_j cannot also be the parent of X_i . This is why we use an arrow, and why the graph is **directed**¹. We write the set of all parents of X_j as $\text{parents}(j)$; this generalizes the notion of the factors which an observable loads on to. The joint distribution “decomposes according to the graph”:

$$p(X_1, X_2, \dots, X_p) = \prod_{i=1}^p p(X_i | X_{\text{parents}(i)}) \quad (20.4)$$

If X_i has no parents, because it has no incoming arrows, take $p(X_i | X_{\text{parents}(i)})$ just to be the marginal distribution $p(X_i)$. Such variables are called **exogenous**; the others, with parents, are **endogenous**. An unfortunate situation could arise where X_1 is the parent of X_2 , which is the parent of X_3 , which is the parent of X_1 . Perhaps, under some circumstances, we could make sense of this and actually calculate with Eq. 20.4, but the general practice is to rule it out by assuming the graph is **acyclic**, i.e., that it has no cycles, i.e., that we cannot, by following a series of arrows in the graph, go from one node to other nodes and ultimately back to our starting point. Altogether we say that we have a **directed acyclic graph**, or **DAG**, which represents the direct dependencies between variables.²

What good is this? The primary virtue is that if we are dealing with a DAG model, the graph tells us all the dependencies we need to know; those are the conditional distributions of variables on their parents, appearing in the product on the right hand side of Eq. 20.4. (This includes the distribution of the exogeneous variables.) This fact has two powerful sets of implications, for probabilistic reasoning and for statistical inference.

Let’s take inference first, because it’s more obvious: all that we have to estimate are the conditional distributions $p(X_i | X_{\text{parents}(i)})$. We do not have to estimate the distribution of X_i given *all* of the other variables, unless of course they are all parents of X_i . Since estimating distributions, or even just regressions, conditional on many variables is hard, it is extremely helpful to be able to read off from the graph which variables we can *ignore*. Indeed, if the graph tells us that X_i is exogeneous, we don’t have to estimate it conditional on anything, we just have to estimate its marginal distribution.

20.2.1 Conditional Independence and the Markov Property

The probabilistic implication of Eq. 20.4 is perhaps even more important, and that has to do with conditional independence. Pick any two variables X_i and X_j , where X_j is not a parent of X_i . Consider the distribution of X_i conditional on its parents *and* X_j . There are two possibilities. (i) X_j is not a descendant of X_i . Then we can see that X_i and X_j are conditionally independent. This is true *no matter what* the actual conditional distribution functions involved are; it’s just implied by the joint

¹See Appendix F for a brief review of the ideas and jargon of graph theory.

²See §20.4 for remarks on undirected graphical models, and graphs with cycles.

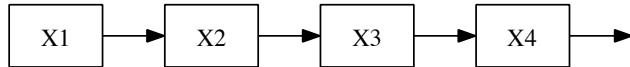


Figure 20.2: DAG for a discrete-time Markov process. At each time t , X_t is the child of X_{t-1} and the parent of X_{t+1} .

distribution respecting the graph. (ii) Alternatively, X_j is a descendant of X_i . Then in general they are not independent, even conditional on the parents of X_i . So the graph implies that certain conditional independence relations will hold, but that others in general will *not* hold.

As you know from your probability courses, a sequence of random variables X_1, X_2, X_3, \dots forms a **Markov process**³ when “the past is independent of the future given the present”: that is,

$$X_{t+1} \perp\!\!\!\perp (X_{t-1}, X_{t-2}, \dots, X_1) | X_t \quad (20.5)$$

from which it follows that

$$(X_{t+1}, X_{t+2}, X_{t+3}, \dots) \perp\!\!\!\perp (X_{t-1}, X_{t-2}, \dots, X_1) | X_t \quad (20.6)$$

which is called the **Markov property**. DAG models have a similar property: if we take any collection of nodes I , it is independent of its non-descendants, given its parents:

$$X_I \perp\!\!\!\perp X_{\text{non-descendants}(I)} | X_{\text{parents}(I)} \quad (20.7)$$

This is the **directed graph Markov property**. The ordinary Markov property is in fact a special case of this, when the graph looks like Figure 20.2⁴.

20.3 Examples of DAG Models and Their Uses

Factor models are examples of DAG models (as we’ve seen). So are mixture models (Figure 20.3) and Markov chains (see above). DAG models are considerably more flexible, however, and can combine observed and unobserved variables in many ways.

Consider, for instance, Figure 20.4. Here there are two exogenous variables, labeled “Smoking” and “Asbestos”. Everything else is endogenous. Notice that “Yellow teeth” is a child of “Smoking” alone. This does not mean that (in the model)

³After the Russian mathematician A. A. Markov, who introduced the theory of Markov processes in the course of a mathematical dispute with his arch-nemesis, to show that probability and statistics could apply to dependent events, and hence that Christianity was not *necessarily* true (I am not making this up: Basharin *et al.*, 2004).

⁴To see this, take the “future” nodes, indexed by $t + 1$ and up, as the set I . Their parent consists just of X_t , and all their non-descendants are the even earlier nodes at times $t - 1, t - 2$, etc.

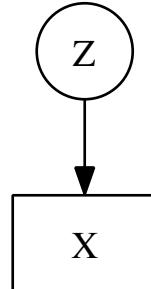


Figure 20.3: DAG for a mixture model. The latent class Z is exogenous, and the parent of the observable random vector X . (If the components of X are conditionally independent given Z , they could be represented as separate boxes on the lower level.)

whether someone's teeth get yellowed (and, if so, how much) is a function of smoking alone; it means that whatever other influences go into that are independent of the rest of the model, and so unsystematic that we can think about those influences, taken together, as noise.

Continuing, the idea is that how much someone smokes influences how yellow their teeth become, and also how much tar builds up in their lungs. Tar in the lungs, in turn, leads to cancer, as does by exposure to asbestos.

Now notice that, in this model, teeth-yellowing will be unconditionally dependent on, i.e., associated with, the level of tar in the lungs, because they share a common parent, namely smoking. Yellow teeth and tarry lungs will however be conditionally independent given that parent, so if we control for smoking we should not be able to predict the state of someone's teeth from the state of their lungs or vice versa.

On the other hand, smoking and exposure to asbestos are independent, at least in this model, as they are both exogenous⁵. Conditional on whether someone has cancer, however, smoking and asbestos will become *dependent*.

To understand the logic of this, suppose (what is in fact true) that both how much someone smokes and how much they are exposed to asbestos raises the risk of cancer. Conditional on not having cancer, then, one was probably exposed to little of either tobacco smoke or asbestos. Conditional on both not having cancer and having

⁵If we had two variables which in some physical sense were exogenous but dependent on each other, we would represent them in a DAG model by either a single *vector-valued* random variable (which would get only one node), or as children of a latent unobserved variable, which was truly exogenous.

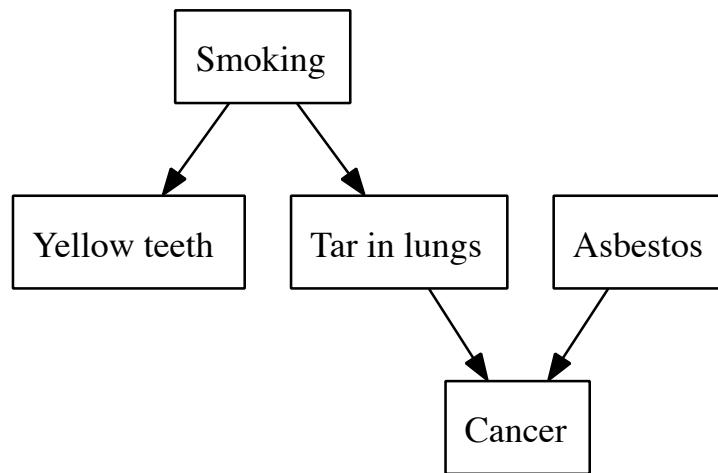


Figure 20.4: DAG model indicating (hypothetical) relationships between smoking, asbestos, cancer, and covariates.

been exposed to a high level of asbestos, one probably was exposed to an unusually low level of tobacco smoke. Vice versa, no cancer plus high levels of tobacco tend to imply especially little exposure to asbestos. We thus have created a negative association between smoking and asbestos by conditioning on cancer. Naively, a regression where we “controlled for” cancer would in fact tell us that exposure to asbestos keeps tar from building up in the lungs, prevents smoking, and whitens teeth.

More generally, conditioning on a third variable can *create* dependence between otherwise independent variables, when what we are conditioning on is a common *descendant* of the variables in question.⁶ This conditional dependence is *not* some kind of finite-sample artifact or error — it really is there in the joint probability distribution. If all we care about is prediction, then it is perfectly legitimate to use it. In the world of Figure 20.4, it really is true that you can predict the color of someone’s teeth from whether they have cancer and how much asbestos they’ve been exposed to, so if that’s what you want to predict⁷, why not use that information? But if you want to do more than just make predictions without understanding, if you want to understand the structure tying together these variables, if you want to do *science*, if you don’t want to go around telling yourself that asbestos whitens teeth, you really do need to know the graph.⁸

20.3.1 Missing Variables

Suppose that we do cannot observe one of the variables, such as the quantity of tar in the lungs, but we somehow know all of the conditional distributions required by the graph. (Tar build-up in the lungs might indeed be hard to measure for living people.) Because we have a joint distribution for *all* the variables, we could estimate the conditional distribution of one of them given the rest, using the definition of conditional probability and of integration:

$$p(X_i|X_1, X_2, X_{i-1}, X_{i+1}, X_p) = \frac{p(X_1, X_2, X_{i-1}, X_i, X_{i+1}, X_p)}{\int p(X_1, X_2, X_{i-1}, x_i, X_{i+1}, X_p) dx_i} \quad (20.8)$$

We could in principle do this for *any* joint distribution. When the joint distribution comes from a DAG model, however, we can simplify this considerably. Recall that, from Eq. 20.7, X_i conditioning on its parents makes X_i independent of all its non-descendants. We can therefore drop from the conditioning everything which isn’t either a parent of X_i , or a descendant. In fact, it’s not hard to see that given the children of X_i , its more remote descendants are also redundant. Actually *doing* the calculation then boils down to a version of the EM algorithm.⁹

⁶Economists, psychologists, and other non-statisticians often repeat the advice that if you want to know the effect of X on Y , you should not condition on Z when Z is endogenous. This is bit of folklore is an incorrect relic of the days of ignorance, though it shows a sound indistinct groping towards a truth those people were unable to grasp. If we want to know whether asbestos is associated with tar in the lungs, conditioning on the yellowness of teeth is fine, even though that is an endogenous variable.

⁷Maybe you want to guess who’d be interested in buying whitening toothpaste.

⁸We return to this example in §21.2.2.

⁹Graphical models, especially directed ones, are often called “Bayes nets” or “Bayesian networks”, because this equation is, or can be seen as, a version of Bayes’s rule. Since of course it follows directly from the definition of conditional probability, there is really nothing Bayesian about them.

If we observe only a subset of the other variables, we can still use the DAG to determine which ones actually matter to estimating X_i , and which ones are superfluous. The calculations then however become much more intricate.¹⁰

20.4 Non-DAG Graphical Models: Undirected Graphs and Directed Graphs with Cycles

This section is optional, as, for various reasons, we will not use these models in this course.

20.4.1 Undirected Graphs

There is a lot of work on probability models which are based on *undirected* graphs, in which the relationship between random variables linked by edges is completely symmetric, unlike the case of DAGs¹¹. Since the relationship is symmetric, the preferred metaphor is not “parent and child”, but “neighbors”. The models are sometimes called **Markov networks** or **Markov random fields**, but since DAG models have a Markov property of their own, this is not a happy choice of name, and I’ll just call them “undirected graphical models”.

The key Markov property for undirected graphical models is that any set of nodes I is independent of the rest of the graph given its neighbors:

$$X_I \perp\!\!\!\perp X_{\text{non-neighbors}(I)} | X_{\text{neighbors}(I)} \quad (20.9)$$

This corresponds to a factorization of the joint distribution, but a more complex one than that of Eq. 20.4, because a symmetric neighbor-of relation gives us no way of *ordering* the variables, and conditioning the later ones on the earlier ones. The trick turns out to go as follows. First, as a bit of graph theory, a **clique** is a set of nodes which are all neighbors of each other, and which cannot be expanded without losing that property. We write the collection of all cliques in a graph G as $\text{cliques}(G)$. Second, we introduce **potential functions** ψ_c which take clique configurations and return non-negative numbers. Third, we say that a joint distribution is a **Gibbs distribution**¹² when

$$p(X_1, X_2, \dots, X_p) \propto \prod_{c \in \text{cliques}(G)} \psi_c(X_{i \in c}) \quad (20.10)$$

That is, the joint distribution is a product of factors, one factor for each clique. Frequently, one introduces what are called **potential functions**, $U_c = \log \psi_c$, and then one has

$$p(X_1, X_2, \dots, X_p) \propto e^{-\sum_{c \in \text{cliques}(G)} U_c(X_{i \in c})} \quad (20.11)$$

¹⁰There is an extensive discussion of relevant methods in Jordan (1998).

¹¹I am told that this is more like the idea of causation in Buddhism, as something like “co-dependent origination”, than the asymmetric one which Europe and the Islamic world inherited from the Greeks (especially Aristotle), but you would really have to ask a philosopher about that.

¹²After the American physicist and chemist J. W. Gibbs, who introduced such distributions as part of **statistical mechanics**, the theory of the large-scale patterns produced by huge numbers of small-scale interactions.

The key correspondence is what is sometimes called the **Gibbs-Markov theorem**: a distribution is a Gibbs distribution with respect to a graph G if, and only if, it obeys the Markov property with neighbors defined according to G .¹³

In many practical situations, one combines the assumption of an undirected graphical model with the further assumption that the joint distribution of all the random variables is a multivariate Gaussian, giving a **Gaussian graphical model**. An important consequence of this assumption is that the graph can be “read off” from the inverse of the covariance matrix Σ , sometimes called the **precision matrix**. Specifically, there is an edge linking X_i to X_j if and only if $(\Sigma^{-1})_{ij} \neq 0$. (See Lauritzen (1996) for an extensive discussion.) These ideas sometimes still work for non-Gaussian distributions, when there is a natural way of transforming them to be Gaussian (Liu *et al.*, 2009), though it is unclear just how far that goes.

20.4.1.0.1 Further reading Markov random fields where the graph is a regular lattice are used extensively in spatial statistics. Good introductory-level treatments are provided by Kindermann and Snell (1980) (the full text of which is free online), and by Guttorp (1995), which also covers the associated statistical methods. Winkler (1995) is also good, but presumes more background in statistical theory. (I would recommend reading it after Guttorp.) Guyon (1995) is at a similar level of sophistication to Winkler, but, unlike the previous references, emphasizes the situations where the graph is *not* a regular lattice. Griffeath (1976), while presuming more probability theory on the part of the reader, is extremely clear and insightful, including what is simultaneously one of the deepest and most transparent proofs of the Gibbs-Markov theorem. Lauritzen (1996) is a mathematically rigorous treatment of graphical models from the viewpoint of theoretical statistics, covering both the directed and undirected cases.

If you are curious about Gibbs distributions in, so to speak, their natural habitat, the book by Sethna (2006), also free online, is the best introduction to statistical mechanics I have seen, and presumes very little knowledge of actual physics on the part of the reader. Honerkamp (2002) is less friendly, but tries harder to make connections to statistics. If you already know what an exponential family is, then Eq. 20.11 is probably extremely suggestive, and you should read Mandelbrot (1962).

20.4.2 Directed but Cyclic Graphs

Much less work has been done on directed graphs with cycles. It is very hard to give these a causal interpretation, in the fashion described in the next chapter. Feedback processes are of course very common in nature and technology, and one might think

¹³This theorem was proved, in slightly different versions, under slightly different conditions, and by very different methods, more or less simultaneously by (alphabetically) Dobrushin, Griffeath, Grimmett, and Hammersley and Clifford, and almost proven by Ruelle. In the statistics literature, it has come to be called the “Hammersley-Clifford” theorem, for no particularly good reason. In my opinion, the clearest and most interesting version of the theorem is that of Griffeath (1976), an elementary exposition of which is given by Pollard (<http://www.stat.yale.edu/~pollard/Courses/251.spring04/Handouts/Hammersley-Clifford.pdf>). (On the other hand, Griffeath was one of my teachers, so discount accordingly.) Calling it the “Gibbs-Markov theorem” says more about the content, and is fairer to all concerned.

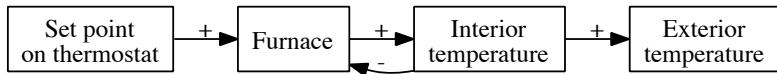


Figure 20.5: Directed but cyclic graphical model of a feedback loop. Signs (+, – on arrows are “guides to the mind”. Cf. Figure 20.6.

to represent these as cycles in a graph. A model of a thermostat, for instance, might have variables for the set-point temperature, the temperature outside, how much the furnace runs, and the actual temperature inside, with a cycle between the latter two (Figure 20.5).

Thinking in this way is however simply sloppy. It always takes *some* time to traverse a feedback loop, and so the cycle really “unrolls” into an acyclic graph linking similar variables at *different* times (Figure 20.6). Sometimes¹⁴, it is clear that when people draw a diagram like Figure 20.5, the incoming arrows really refer to the change, or rate of change, of the variable in question, so it is merely a visual short-hand for something like Figure 20.6.

Directed graphs with cycles are thus primarily useful when measurements are so slow or otherwise imprecise that feedback loops cannot be unrolled into the actual dynamical processes which implement them, and one is forced to hope that one can reason about equilibria instead¹⁵. If you insist on dealing with cyclic directed graphical models, see Richardson (1996); Lacerda *et al.* (2008) and references therein.

20.5 Further Reading

The paper collection Jordan (1998) is actually extremely good, unlike most collections of edited papers; Jordan and Sejnowski (2001) is also useful. Lauritzen (1996) is thorough but more mathematically demanding. The books by Spirtes *et al.* (1993, 2001) and by Pearl (1988, 2000, 2009b) are classics, especially for their treatment of causality, of which much more soon. Glymour (2001) discusses applications to psychology.

While I have presented DAG models as an outgrowth of factor analysis, their historical ancestry is actually closer to the “path analysis” models introduced by the great mathematical biologist Sewall Wright in the 1920s to analyze processes of development and genetics. These proved extremely influential in psychology. Loehlin (1992) is user-friendly, though aimed at psychologists with less mathematical sophistication than students taking this course. Li (1975), while older, is very enthusiastic and has many interesting applications.

¹⁴As in Puccia and Levins (1985), and the LoopAnalyst package based on it (Dinno, 2009).

¹⁵Economists are fond of doing so, generally without providing any rationale, based in economic theory, for supposing that equilibrium *is* a good approximation.

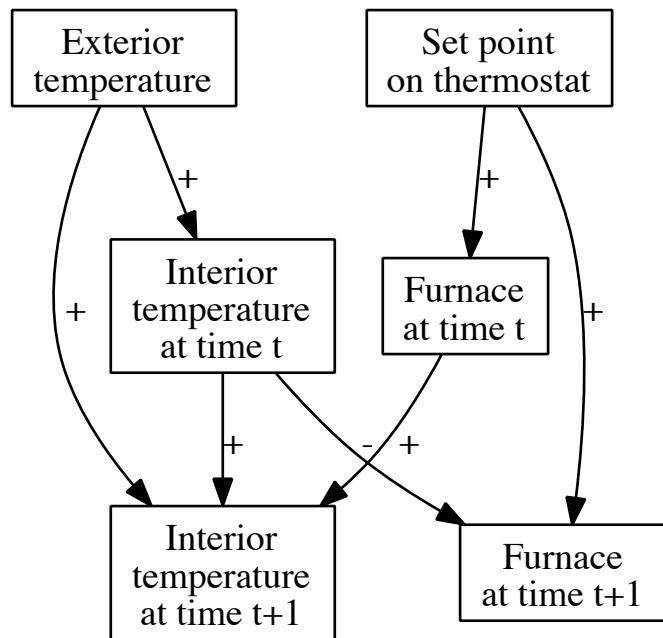


Figure 20.6: Directed, acyclic graph for the situation in Figure 20.5, taking into account the fact that it takes time to traverse a feedback loop. One should imagine this repeating to times $t + 2$, $t + 3$, etc., and extending backwards to times $t - 1$, $t - 2$, etc., as well. Notice that there are no longer any cycles.

14:41 Thursday 25th April, 2013

Part III

Causal Inference

Chapter 21

Graphical Causal Models

21.1 Causation and Counterfactuals

Take a piece of cotton, say an old rag. Apply flame to it; the cotton burns. We say the fire *caused* the cotton to burn. The flame is certainly *correlated* with the cotton burning, but, as we all know, correlation is not causation (Figure 21.1). Perhaps every time we set rags on fire we handle them with heavy protective gloves; the gloves don't make the cotton burn, but the statistical dependence is strong. So what is causation?

We do not have to settle 2500 years (or more) of argument among philosophers and scientists. For our purposes, it's enough to realize that the concept has a **counterfactual** component: if, contrary to fact, the flame had not been applied to the rag, then the rag would not have burned¹. On the other hand, the fire makes the cotton burn whether we are wearing protective gloves or not.

To say it a somewhat different way, the distributions we observe in the world are the outcome of complicated stochastic processes. The mechanisms which set the value of one variable inter-lock with those which set other variables. When we make a probabilistic prediction by conditioning — whether we predict $E[Y|X = x]$ or $\Pr(Y|X = x)$ or something more complicated — we are just filtering the output of those mechanisms, picking out the cases where they happen to have set X to the value x , and looking at what goes along with that.

When we make a *causal* prediction, we want to know what would happen if the usual mechanisms controlling X were suspended and it was *set* to x . How would this change propagate to the other variables? What distribution would result for Y ? This is often, perhaps even usually, what people really want to know from a data analysis, and they settle for statistical prediction either because they think it *is* causal prediction, or for lack of a better alternative.

Causal inference is the undertaking of trying to answer causal questions from empirical data. Its fundamental difficulty is that we are trying to derive counterfactual conclusions with only factual premises. As a matter of habit, we come to

¹If you immediately start thinking about quibbles, like "What if we hadn't applied the flame, but the rag was struck by lightning?", then you may have what it takes to be a philosopher.

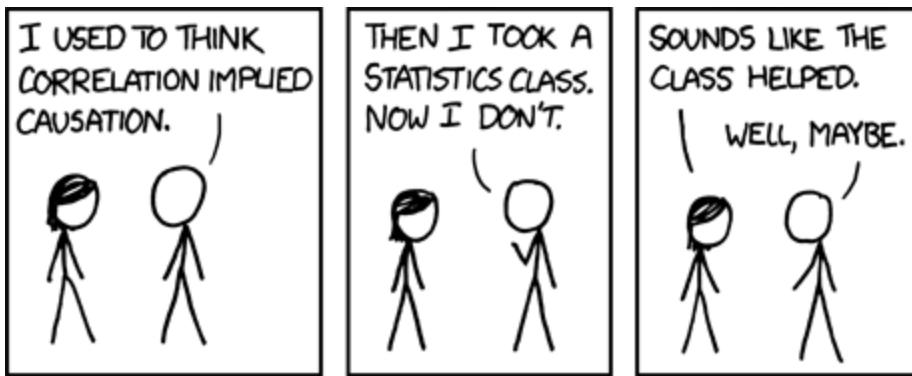


Figure 21.1: “Correlation doesn’t imply causation, but it does waggle its eyebrows suggestively and gesture furtively while mouthing ‘look over there’” (Image and text copyright by Randall Munroe, used here under a Creative Commons attribution-noncommercial license; see <http://xkcd.com/552/>.)

expect cotton to burn when we apply flames. We might even say, on the basis of purely statistical evidence, that the world has this habit. But as a matter of pure logic, no amount of evidence about what *did* happen can compel beliefs about what *would* have happened under non-existent circumstances². (For all my *data* shows, all the rags I burn just so happened to be on the verge of spontaneously bursting into flames anyway.) We must supply some counter-factual or causal premise, linking what we see to what we could have seen, to derive causal conclusions.

One of our goals, then, in causal inference will be to make the causal premises as weak and general as possible, so as to limit what we take on faith.

21.2 Causal Graphical Models

We will need a formalism for representing causal relations. It will not surprise you by now to learn that these will be graphical models. We will in fact use DAG models from last time, with “parent” interpreted to mean “directly causes”. These will be **causal graphical models**, or **graphical causal models**.³

We make the following assumptions.

1. There is some directed acyclic graph G representing the relations of causation among the our variables.

²The first person to really recognize this seems to have been the medieval Muslim theologian and anti-philosopher al Ghazali (1100/1997). (See Kogan (1985) for some of the history.) Very similar arguments were revived centuries later by Hume (1739); whether there was some line of intellectual descent linking them — that is, any causal connection — I don’t know.

³Because DAG models have joint distributions which factor according to the graph, we can always write them in the form of a set of equations, as $X_i = f_i(X_{\text{parents}(i)}) + \epsilon_i$, with the catch that the noise ϵ_i is not necessarily independent of X_i ’s parents. This is what is known, in many of the social sciences, as a **structural equation model**. So those are, strictly, a sub-class of DAG models. They are also often used to represent causal structure.

2. **The Causal Markov condition:** The joint distribution of the variables obeys the Markov property on G .
3. **Faithfulness:** The joint distribution has all of the conditional independence relations implied by the causal Markov property, and *only* those conditional independence relations.

The point of the faithfulness condition is to rule out “conspiracies among the parameters”, where, say, two causes of a common effect, which would typically be dependent conditional on that effect, have their impact on the joint effect and their own distributions matched just so exactly that they remain conditionally independent.

21.2.1 Calculating the “effects of causes”

Let's fix two sub-sets of variables in the graph, X_C and X_E . (Assume they don't overlap, and call everything else X_N .) If we want to make a *probabilistic* prediction for X_E 's value when X_c takes a particular value, x_c , that's the conditional distribution, $\Pr(X_E|X_c = x_c)$, and we saw last time how to calculate that using the graph. Conceptually, this amounts to selecting, out of the whole population or ensemble, the sub-population or sub-ensemble where $X_c = x_c$, and accepting whatever other behavior may go along with that.

Now suppose we want to ask what the effect would be, causally, of setting X_C to a particular value x_c . We represent this by “doing surgery on the graph”: we (i) eliminate any arrows coming in to nodes in X_c , (ii) fix their values to x_c , and (iii) calculate the resulting distribution for X_E in the new graph. By steps (i) and (ii), we imagine suspending or switching off the mechanisms which ordinarily set X_c . The other mechanisms in the assemblage are left alone, however, and so step (iii) propagates the fixed values of X_c through them. We are not *selecting* a sub-population, but producing a new one.

If setting X_c to different values, say x_c and x'_c , leads to different distributions for X_E , then we say that X_c **has an effect** on X_E — or, slightly redundantly, **has a causal effect** on X_E . Sometimes⁴ “the effect of switching from x_c to x'_c ” specifically refers to a change in the expected value of X_E , but since profoundly different distributions can have the same mean, this seems needlessly restrictive.⁵ If one is interested in average effects of this sort, they are computed by the same procedure.

It is convenient to have a short-hand notation for this procedure of causal conditioning. One more-or-less standard idea, introduced by Judea Pearl, is to introduce a *do* operator which encloses the conditioning variable and its value. That is,

$$\Pr(X_E|X_c = x_c) \tag{21.1}$$

is probabilistic conditioning, or selecting a sub-ensemble from the old mechanisms; but

$$\Pr(X_E|do(X_c = x_c)) \tag{21.2}$$

⁴Especially in economics.

⁵Economists are also fond of the horribly misleading usage of talking about “an X effect” or “the effect of X ” when they mean the regression coefficient of X . Don't do this.

is causal conditioning, or producing a new ensemble. Sometimes one sees this written as $\Pr(X_E|X_c \hat{=} x_c)$, or even $\Pr(X_E|\hat{x}_c)$. I am actually fond of the do notation and will use it.

Suppose that $\Pr(X_E|X_c = x_c) = \Pr(X_E|do(X_c = x_c))$. This would be extremely convenient for causal inference. The conditional distribution on the right is the causal, counter-factual distribution which tells us what would happen if x_c was imposed. The distribution on the left is the ordinary probabilistic distribution we have spent years learning how to estimate from data. When do they coincide?

One time when they would is if X_c contains all the parents of X_E , and none of its descendants. Then, by the Markov property, X_E is independent of all other variables given X_C , and removing the arrows *into* X_C will not change that, or the conditional distribution of X_E given its parents. Doing causal inference for other choices of X_C will demand other conditional independence relations implied by the Markov property. This is the subject of Chapter 22.

21.2.2 Back to Teeth

Let us return to the example of Figure 20.4, and consider the relationship between exposure to asbestos and the staining of teeth. In the model depicted by that figure, the joint distribution factors as⁶

$$\begin{aligned} p(\text{Yellow teeth, Smoking, Asbestos, Tar in lungs, Cancer}) \\ = & p(\text{Smoking})p(\text{Asbestos}) \\ & \times p(\text{Tar in lungs}|\text{Smoking}) \\ & \times p(\text{Yellow teeth}|\text{Smoking}) \\ & \times p(\text{Cancer}|\text{Asbestos, Tar in lungs}) \end{aligned} \tag{21.3}$$

As we saw, whether or not someone's teeth are yellow (in this model) is unconditionally independent of asbestos exposure, but conditionally *dependent* on asbestos, given whether or not they have cancer. A logistic regression of tooth color on asbestos would show a non-zero coefficient, after "controlling for" cancer. This coefficient would become significant with enough data. The usual interpretation of this coefficient would be to say that the log-odds of yellow teeth increase by so much for each one unit increase in exposure to asbestos, "other variables being held equal".⁷ But to see the actual causal effect of increasing exposure to asbestos by one unit, we'd want to compare $p(\text{Yellow teeth}|do(\text{Asbestos} = a))$ to $p(\text{Yellow teeth}|do(\text{Asbestos} = a + 1))$, and it's easy to check (Exercise 1) that these two distributions have to be the same. In this case, because asbestos is exogenous, one will in fact get the same result for $p(\text{Yellow teeth}|do(\text{Asbestos} = a))$ and for $p(\text{Yellow teeth}|\text{Asbestos} = a)$.

For a more substantial example, consider Figure 21.2⁸ The question of interest

⁶I am grateful to Janet E. Rosenbaum for pointing out an error in an earlier version of this example.

⁷Nothing hinges on this being a logistic regression, similar interpretations are given to all the other standard models.

⁸Based on de Oliveira *et al.* (2010), and the discussion of this paper by Chris Blattman (<http://chrisblattman.com/2010/06/01/does-brushing-your-teeth-lower-cardiovascular-disease/>).

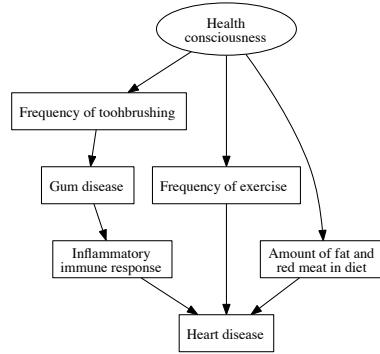


Figure 21.2: Graphical model illustrating hypothetical pathways linking brushing your teeth to not getting heart disease.

here is whether regular brushing and flossing actually prevents heart-disease. The mechanism by which it might do so is as follows: brushing is known to make it less likely for people to get gum disease. When you have gum disease, your gums are subject to constant, low-level inflammation. This inflammation (which can be measured through various messenger chemicals of the immune system) is thought to increase the risk of heart disease. Against this, people who are generally health-conscious are likely to brush regularly, and to take other actions, like regularly exercising and controlling their diets, which also make them less likely to get heart disease. In this case, if we were to manipulate whether people brush their teeth⁹, we would shift the graph from Figure 21.2 to Figure 21.3, and we would have

$$p(\text{Heart disease}|\text{Brushing} = b) \neq p(\text{Heart disease}|do(\text{Brushing} = b)) \quad (21.4)$$

⁹Hopefully, by ensuring that everyone brushes, rather than keeping people from brushing.

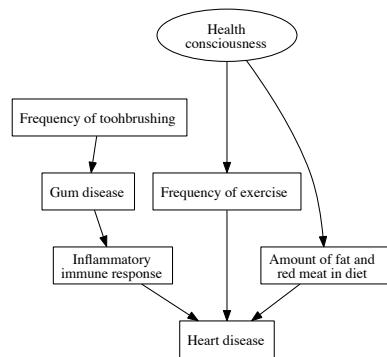


Figure 21.3: The previous graphical model, “surgically” altered to reflect a manipulation (do) of brushing.

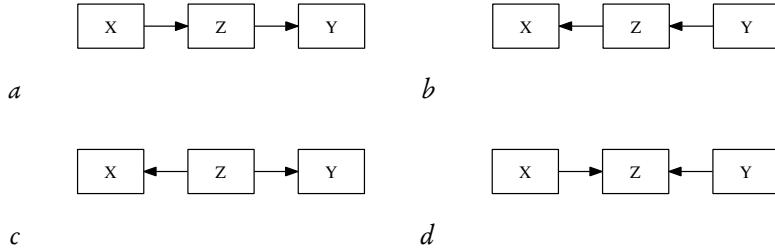


Figure 21.4: Four DAGs for three linked variables. The first two (*a* and *b*) are called **chains**; *c* is a **fork**; *d* is a **collider**. If these were the whole of the graph, we would have $X \not\perp\!\!\!\perp Y$ and $X \perp\!\!\!\perp Y|Z$. For the collider, however, we would have $X \perp\!\!\!\perp Y$ while $X \not\perp\!\!\!\perp Y|Z$.

21.3 Conditional Independence and d -Separation

It is clearly very important to us to be able to deduce when two sets of variables are conditionally independent of each other given a third. One of the great uses of DAGs is that they give us a fairly simple criterion for this, in terms of the graph itself. *All* distributions which conform to a given DAG share a common set of conditional independence relations, implied by the Markov property, no matter what their parameters or the form of the distributions. Faithful distributions have *no other* conditional independence relations. Let us think this through.

Our starting point is that while *causal influence* flows one way through the graph, along the directions of arrows from parents to children, *statistical information* can flow in either direction. We can certainly make inferences about an effect from its causes, but we can equally make inferences about causes from their effects. It might be harder to actually do the calculations¹⁰, and we might be left with more uncertainty, but we could do it.

While we can do inference in either direction across any one edge, we do have to worry about whether we can propagate this information further. Consider the four graphs in Figure 21.4. In every case, we condition on X , which acts as the source of information. In the first three cases, we can (in general) propagate the information from X to Z to Y — the Markov property tells us that Y is independent of its non-descendants given its parents, but in none of those cases does that make X and Y independent. In the last graph, however, what's called a **collider**¹¹, we cannot propagate the information, because Y has no parents, and X is not its descendant, hence they are independent. We learn about Z from X , but this doesn't tell us anything about Z 's other cause, Y .

¹⁰Janzing (2007) makes the very interesting suggestion that the direction of causality can be discovered by using this — roughly speaking, that if $X|Y$ is much harder to compute than is $Y|X$, we should presume that $X \rightarrow Y$ rather than the other way around.

¹¹Because two incoming arrows “collide” there.

All of this flips around when we condition on the intermediate variable (Z in Figure 21.4). The chains (Figures 21.4a and b), conditioning on the intermediate variable blocks the flow of information from X to Y — we learn nothing more about Y from X and Z than from Z alone, at least not along this path. This is also true of the **fork** (Figure 21.4c) — conditional on their common cause, the two effects are uninformative about each other. But in a collider, conditioning on the common effect Z makes X and Y dependent on each other, as we've seen before. In fact, if we don't condition on Z , but do condition on a descendant of Z , we also create dependence between Z 's parents.

We are now in a position to work out conditional independence relations. We pick our two favorite variables, X and Y , and condition them both on some third set of variables S . If S **blocks** every undirected path¹² from X to Y , then they must be conditionally independent given S . An unblocked path is also called **active**. A path is active when every variable along the path is active; if even one variable is blocked by S , the whole path is blocked. A variable Z along a path is active, conditioning on S , if

1. Z is a collider along the path, and in S ; or,
2. Z is a descendant of a collider, and in S ; or
3. Z is not a collider, and not in S .

Turned around, Z is blocked or de-activated by conditioning on S if

1. Z is a non-collider and in S ; or
2. Z is collider, and neither Z nor any of its descendants is in S

In words, S blocks a path when it blocks the flow of information by conditioning on the middle node in a chain or fork, and doesn't create dependence by conditioning on the middle node in a collider (or the descendant of a collider). Only *one* node in a path must be blocked to block the whole path. When S blocks *all* the paths between X and Y , we say it **d-separates** them¹³. A collection of variables U is d-separated from another collection V by S if every $X \in U$ and $Y \in V$ are d-separated.

In every distribution which obeys the Markov property, d-separation implies conditional independence. If the distribution is also faithful to the graph, then conditional independence also implies d-separation¹⁴. In a faithful causal graphical model, then, conditional independence is exactly the same as blocking the flow of information across the graph. This turns out to be the single most important fact enabling causal inference; we will see how that works next time.

¹²Whenever I talk about undirected paths, I mean paths without cycles.

¹³The “d” stands for “directed”

¹⁴We will not prove this, though I hope I have made it plausible. You can find demonstrations in Spirtes *et al.* (2001); Pearl (2000); Lauritzen (1996).

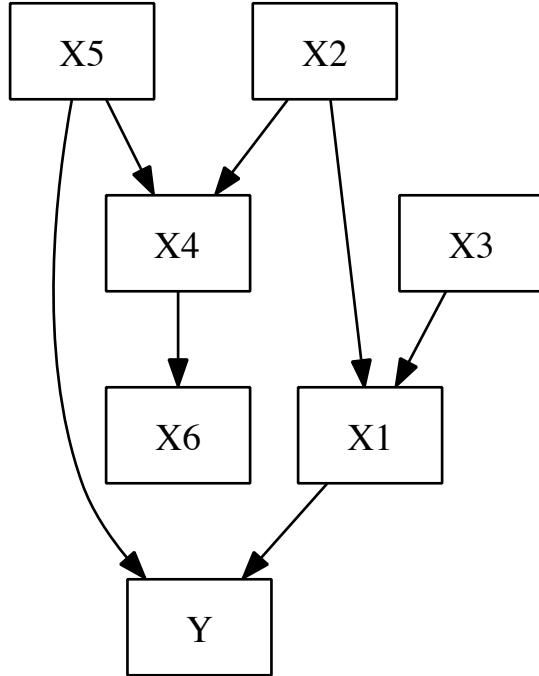


Figure 21.5: Example DAG used to illustrate d-separation.

21.3.1 D-Separation Illustrated

The discussion of d-separation has been rather abstract, and perhaps confusing for that reason. Figure 21.5 shows a DAG which might make this clearer and more concrete¹⁵.

If we make the conditioning set S the empty set, that is, we condition on nothing, we “block” paths which pass through colliders. For instance, there are three exogenous variables in the graph, X_2, X_3 and X_5 . Because they have no parents, any path from one to another must go over a collider (see exercises). If we do not condition on anything, therefore, we find that the exogenous variables are d-separated and thus independent. Since X_3 is not on any path linking X_2 and X_5 , or descended from a node on any such path, if we condition only on X_3 , then X_2 and X_5 are still d-separated, so $X_2 \perp\!\!\!\perp X_5 | X_3$. There are two paths linking X_3 to X_5 : $X_3 \rightarrow X_1 \leftarrow X_2 \rightarrow X_4 \leftarrow X_5$, and

¹⁵I am grateful to Donald Schoolmaster, Jr., for pointing out errors in an earlier version of this example.

$X_3 \rightarrow X_1 \rightarrow Y \leftarrow X_5$. Conditioning on X_2 blocks the first path (since X_2 is part of it, but is a fork), and also blocks the second path (since X_2 is not part of it, and Y is a blocked collider). Thus, $X_3 \perp\!\!\!\perp X_5 | X_2$. Similarly, $X_3 \perp\!\!\!\perp X_2 | X_5$ (Exercise 4).

For a slightly more interesting example, let's look at the relation between X_3 and Y . There are two paths here: $X_3 \rightarrow X_1 \rightarrow Y$, and $X_3 \rightarrow X_1 \leftarrow X_2 \rightarrow X_4 \leftarrow X_5 \rightarrow Y$. If we condition on nothing, the first path, which is a simple chain, is open, so X_3 and Y are d-connected and dependent. If we condition on X_1 , we block the first path. X_1 is a collider on the second path, so conditioning on it makes it active. However, there is a second collider, X_4 , along this path, and just conditioning on X_1 does not activate the second collider, so the path as a whole remains blocked.

$$Y \not\perp\!\!\!\perp X_3 \quad (21.5)$$

$$Y \perp\!\!\!\perp X_3 | X_1 \quad (21.6)$$

To activate the second path, we can condition on X_1 and either X_4 (a collider along that path) or on X_6 (a descendant of a collider) or on both:

$$Y \not\perp\!\!\!\perp X_3 | X_1, X_4 \quad (21.7)$$

$$Y \not\perp\!\!\!\perp X_3 | X_1, X_6 \quad (21.8)$$

$$Y \not\perp\!\!\!\perp X_3 | X_1, X_4, X_6 \quad (21.9)$$

Conditioning on X_4 and/or X_6 does not activate the $X_3 \rightarrow X_1 \rightarrow Y$ path, but it's enough for there to be one active path to create dependence.

To block the second path again, after having opened it in one of these ways, we can condition on X_2 (since it is a fork along that path, and conditioning on a fork blocks it), or on X_5 (also a fork), or on both X_2 and X_5 . So

$$Y \perp\!\!\!\perp X_3 | X_1, X_2 \quad (21.10)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_5 \quad (21.11)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_2, X_5 \quad (21.12)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_2, X_4 \quad (21.13)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_2, X_6 \quad (21.14)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_2, X_5, X_6 \quad (21.15)$$

etc., etc.

Let's look at the relationship between X_4 and Y . X_4 is not an ancestor of Y , or a descendant of it, but they do share common ancestors, X_5 and X_2 . Unconditionally, Y and X_4 are dependent, both through the path going $X_4 \leftarrow X_5 \rightarrow Y$, and through that going $X_4 \leftarrow X_2 \rightarrow X_1 \rightarrow Y$. Along both paths, the exogenous variables are forks, so *not* conditioning on them leaves the path unblocked. X_4 and Y become d-separated when we condition on X_5 and X_2 .

X_6 and X_3 have no common ancestors. Unconditionally, they should be independent, and indeed they are: the two paths are $X_6 \leftarrow X_4 \leftarrow X_2 \rightarrow X_1 \leftarrow X_3$, and $X_6 \leftarrow X_4 \leftarrow X_5 \rightarrow Y \leftarrow X_1 \leftarrow X_3$. Both paths contain a single collider (X_1 and Y , respectively), so if we do not condition on them the paths are blocked and X_6 and X_3

are independent. If we condition on either Y or X_1 (or both), however, we unblock the paths, and X_6 and X_3 become d-connected, hence dependent. To get back to d-separation while conditioning on Y , we must also condition on X_4 or X_5 , or both. To get d-separation while conditioning on X_1 , we must also condition on X_4 , or on X_2 , or on X_4 and X_2 . If we condition on both X_1 and Y and want d-separation, we could just add conditioning on X_4 , or we could condition on X_2 and X_5 , or all three.

If the abstract variables are insufficiently concrete, consider reading them as follows:

$$\begin{aligned} Y &\Leftrightarrow \text{Grade in 402} \\ X_1 &\Leftrightarrow \text{Effort spent on 402} \\ X_2 &\Leftrightarrow \text{Enjoyment of statistics} \\ X_3 &\Leftrightarrow \text{Workload this semester} \\ X_4 &\Leftrightarrow \text{Quality of work in 401} \\ X_5 &\Leftrightarrow \text{Amount learned in 401} \\ X_6 &\Leftrightarrow \text{Grade in 401} \end{aligned}$$

Pretending, for the sake of illustration, that this is accurate, how heavy your workload is this semester (X_3) would predict, or rather retrodict, your grade in modern regression last semester (X_6), once we control for how much effort you put into data analysis this semester (X_1). Changing your workload this semester would not, however, reach backwards in time to raise or lower your grade in regression.

21.3.2 Linear Graphical Models and Path Coefficients

We began our discussion of graphical models with factor analysis as our starting point. Factor models are a special case of linear (directed) graphical models, a.k.a. path models¹⁶ As with factor models, in the larger class we typically center all the variables (so they have expectation zero) and scale them (so they have variance 1). In factor models, the variables were split into two sets, the factors and the observables, and all the arrows went from factors to observables. In the more general case, we do not necessarily have this distinction, but we still assume the arrows from a directed acyclic graph. The conditional expectation of each variable is a linear combination of the values of its parents:

$$E[X_i | X_{\text{parents}(i)}] = \sum_{j \in \text{parents}(i)} w_{ji} X_j \quad (21.16)$$

just as in a factor model. In a factor model, the coefficients w_{ji} were the factor loadings. More generally, they are called **path coefficients**.

The path coefficients determine all of the correlations between variables in the model. To find the correlation between X_i and X_j , we proceed as follows:

- Find all of the undirected paths between X_i and X_j .

¹⁶Some people use the phrase “structural equation models” for such models exclusively.

- Discard all of the paths which go through colliders.
- For each remaining path, multiply all the path coefficients along the path.
- Sum up these products over paths.

These rules were introduced by the great geneticist and mathematical biologist Sewall Wright in the early 20th century, in a series of papers culminating in Wright (1934)¹⁷. These “Wright path rules” often seem mysterious, particularly the bit where paths with colliders are thrown out. But from our perspective, we can see that what Wright is doing is finding all of the *unblocked* paths between X_i and X_j . Each path is a channel along which information (here, correlation) can flow, and so we add across channels.

It is frequent, and customary, to assume that all of the variables are Gaussian. (We saw this in factor models as well.) With this extra assumption, the joint distribution of all the variables is a multivariate Gaussian, and the correlation matrix (which we find from the path coefficients) gives us the joint distribution.

If we want to find conditional correlations, $\text{corr}(X_i, X_j | X_k, X_l, \dots)$, we still sum up over the unblocked paths. If we have avoided conditioning on colliders, then this is just a matter of dropping the now-blocked paths from the sum. If on the other hand we have conditioned on a collider, that path *does* become active (unless blocked elsewhere), and we in fact need to modify the path weights. Specifically, we need to work out the correlation induced between the two parents of the collider, by conditioning on that collider. This can be calculated from the path weights, and some fairly tedious algebra¹⁸. The important thing is to remember that the rule of d-separation still applies, and that conditioning on a collider can create correlations.

21.3.3 Positive and Negative Associations

We say that variables X and Y are **positively associated** if increasing X predicts, on average, an increase in Y , and vice versa¹⁹; if increasing X predicts a decrease in Y , then they are **negatively associated**. If this holds when conditioning out other variables, we talk about positive and negative partial associations. Heuristically, positive association means positive correlation in the neighborhood of any given x , though the magnitude of the positive correlation need not be constant. Note that not all dependent variables have to have a definite sign for their association.

We can multiply together the signs of positive and negative partial associations along a path in a graphical model, the same we can multiply together path coefficients in a linear graphical model. Paths which contain (inactive!) colliders should be neglected. If all the paths connecting X and Y have the same sign, then we know that over-all association between X and Y must have that sign. If different paths have different signs, however, then signs alone are not enough to tell us about the over-all association.

¹⁷That paper is now freely available online, and worth reading. See also http://www.ssc.wisc.edu/soc/class/soc952/Wright/wright_biblio.htm for references to, and in some cases copies of, related papers by Wright.

¹⁸See for instance Li *et al.* (1975).

¹⁹I.e., if $\frac{d\mathbb{E}[Y|X=x]}{dx} \geq 0$

If we are interested in conditional associations, we have to consider whether our conditioning variables block paths or not. Paths which are blocked by conditioning should be dropped from consideration. If a path contains an activated collider, we need to include it, but we reverse the sign of one arrow into the collider. That is, if $X \xrightarrow{+} Z \xleftarrow{-} Y$, and we condition on Z , we need to replace one of the plus signs with a minus sign, because the two parents now have an over-all negative association.²⁰ If on the other hand one of the incoming arrows had a positive association and the other was negative, we need to flip one of them so they are both positive or both negative; it doesn't matter which, since it creates a positive association between the parents²¹.

21.4 Independence, Conditional Independence, and Information Theory

Take two random variables, X and Y . They have some joint distribution, which we can write $p(x,y)$. (If they are both discrete, this is the joint probability mass function; if they are both continuous, this is the joint probability density function; if one is discrete and the other is continuous, there's still a distribution, but it needs more advanced tools.) X and Y each have marginal distributions as well, $p(x)$ and $p(y)$. $X \perp\!\!\!\perp Y$ if and only if the joint distribution is the product of the marginals:

$$X \perp\!\!\!\perp Y \Leftrightarrow p(x,y) = p(x)p(y) \quad (21.17)$$

We can use this observation to measure how dependent X and Y are. Let's start with the log-likelihood ratio between the joint distribution and the product of marginals:

$$\log \frac{p(x,y)}{p(x)p(y)} \quad (21.18)$$

This will always be exactly 0 when $X \perp\!\!\!\perp Y$. We use its average value as our measure of dependence:

$$I[X;Y] \equiv \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \quad (21.19)$$

(If the variables are continuous, replace the sum with an integral.) Clearly, if $X \perp\!\!\!\perp Y$, then $I[X;Y] = 0$. One can show²² that $I[X;Y] \geq 0$, and that $I[X;Y] = 0$ implies $X \perp\!\!\!\perp Y$. The quantity $I[X;Y]$ is clearly symmetric between X and Y . Less obviously, $I[X;Y] = I[f(X);g(Y)]$ whenever f and g are invertible functions. This

²⁰If both smoking and asbestos are positively associated with lung cancer, and we know the patient does not have lung cancer, then high levels of smoking must be compensated for by low levels of asbestos, and vice versa.

²¹If yellow teeth are positively associated with smoking and negatively associated with dental insurance, and we know the patient does not have yellow teeth, then high levels of smoking must be compensated for by excellent dental care, and conversely poor dental care must be compensated for by low levels of smoking.

²²Using the same type of convexity argument (“Jensen’s inequality”) we used in Lecture 19 for understanding the details of the EM algorithm.

coordinate-freedom means that $I[X; Y]$ measures *all forms* of dependence, not just linear relationships, like the ordinary (Pearson) correlation coefficient, or monotone dependence, like the rank (Spearman) correlation coefficient. In information theory, $I[X; Y]$ is called the **mutual information**, or **Shannon information**, between X and Y . So we have the very natural statement that random variables are independent just when they have no information about each other.

There are (at least) two ways of giving an operational meaning to $I[X; Y]$. One, the original use of the notion, has to do with using knowledge of Y to improve the efficiency with which X can be encoded into bits (Shannon, 1948; Cover and Thomas, 2006). While this is very important — it's literally transformed the world since 1945 — it's not very statistical. For statisticians, what matters is that if we test the hypothesis that X and Y are independent, with joint distribution $p(x)p(y)$, against the hypothesis that they dependent, with joint distribution $p(x,y)$, then our power to detect dependence grows exponentially with the number of samples, and the exponential rate at which it grows is $I[X; Y]$. More exactly, if we take independence to be the null hypothesis, and β_n is the error probability with n samples,

$$-\frac{1}{n} \log \beta_n \rightarrow I[X; Y] \quad (21.20)$$

(See Cover and Thomas (2006) again, or Kullback (1968).) So positive mutual information means dependence, and the magnitude of mutual information tells us about how detectable the dependence is.

Suppose we conditioned X and Y on a third variable (or variables) Z . For each realization z , we can calculate the mutual information,

$$I[X; Y|Z = z] \equiv \sum_{x,y} p(x,y|z) \log \frac{p(x,y|z)}{p(x|z)p(y|z)} \quad (21.21)$$

And we can average over z ,

$$I[X; Y|Z] \equiv \sum_z p(z) I[X; Y|Z = z] \quad (21.22)$$

This is the **conditional mutual information**. It will not surprise you at this point to learn that $X \perp\!\!\!\perp Y|Z$ if and only if $I[X; Y|Z] = 0$. The magnitude of the conditional mutual information tells us how easy it is to detect conditional dependence.

21.5 Further Reading

The two foundational books on graphical causal models are Spirtes *et al.* (2001) and Pearl (2009b). Both are excellent and recommended in the strongest possible terms; but if you had to read just one, I would recommend Spirtes *et al.* (2001). If on the other hand you do not feel up to reading a book at all, then Pearl (2009a) is much shorter, and covers most of the high points. (Also, it's free online.) The textbook by Morgan and Winship (2007) is much less demanding mathematically, which also

means it is less complete conceptually, but it does explain the crucial ideas clearly, simply, and with abundant examples.²³ Lauritzen (1996) has a mathematically rigorous treatment of d-separation (among many other things), but de-emphasizes causality.

Linear path models have a very large literature, going back to the early 20th century; see references in the previous chapter. Many software packages for linear structural equation models and path analysis offer options to search for models; these are not, in general, reliable (Spirtes *et al.*, 2001).

On information theory (§21.4), the best book is Cover and Thomas (2006) by a large margin. Raginsky (2011) provides a fascinating information-theoretic account of graphical causal models and $do()$, in terms of the notion of directed (rather than mutual) information.

21.6 Exercises

1. Show, for the graphical model in Figure 20.4, that $p(\text{Yellow teeth} | do(\text{Asbestos} = a))$ is always the same as $p(\text{Yellow teeth} | do(\text{Asbestos} = a + 1))$.
2. Find all the paths between the exogenous variables in Figure 21.5, and verify that every such path goes through at least one collider.
3. Is it true that in any DAG, every path between exogenous variables must go through at least one collider, or descendant of a collider? Either prove it or construct a counter-example in which it is not true. Does the answer change if we say “go through at least one collider”, rather than “collider or descendant of a collider”?
4. Prove that $X_2 \perp\!\!\!\perp X_3 | X_5$ in Figure 21.5.

²³This textbook also discusses an alternative formalism for counterfactuals, due to Donald Rubin. While Rubin has done very distinguished work in causal inference, his formalism is vastly harder to manipulate than are graphical models, but has no more expressive power. (Pearl (2009a) has a convincing discussion of this point.) I have accordingly skipped the Rubin formalism here, but good accounts are available in Morgan and Winship (2007, ch. 2), and in Rubin’s collected papers (Rubin, 2006).

Chapter 22

Identifying Causal Effects from Observations

There are two problems which are both known as “causal inference”:

1. Given the causal structure of a system, estimate the effects the variables have on each other.
2. Given data about a system, find its causal structure.

The first problem is easier, so we’ll begin with it.

22.1 Causal Effects, Interventions and Experiments

As a reminder, when I talk about the causal effect of X on Y , which I write

$$\Pr(Y|do(X = x)) \tag{22.1}$$

I mean the distribution of Y which would be generated, counterfactually, were X to be set to the particular value x . This is not, in general, the same as the ordinary conditional distribution

$$\Pr(Y|X = x) \tag{22.2}$$

The reason these are different is that the latter represents taking the original population, as it is, and just filtering it to get the sub-population where $X = x$. The processes which set X to that value may also have influenced Y through other channels, and so this distribution will not, typically, really tell us what would happen if we reached in and manipulated X . We can sum up the contrast in a little table (Table 22.1). As we saw in Chapter 20, if we have the full graph for a directed acyclic graphical model, it tells us how to calculate the joint distribution of all the variables, from which of course the conditional distribution of any one variable given another follows. As we saw in Chapter 21, calculations of $\Pr(Y|do(X = x))$ use a “surgically” altered graph, in which all arrows into X are removed, and its value is pinned at x ,

Probabilistic conditioning	Causal conditioning
$\Pr(Y X = x)$	$\Pr(Y do(X = x))$
Factual	Counter-factual
Select a sub-population	Generate a new population
Predicts passive observation	Predicts active manipulation
Calculate from full DAG	Calculate from surgically-altered DAG
Always identifiable when X and Y are observable	Not always identifiable even when X and Y are observable

Table 22.1: Contrasts between ordinary probabilistic conditioning and causal conditioning. (See below on identifiability.)

but the rest of the graph is as before. If we know the DAG, and we know the distribution of each variable given its parents, we can calculate any causal effect we want, by graph-surgery.

22.1.1 The Special Role of Experiment

If we want to estimate $\Pr(Y|do(X = x))$, the most reliable procedure is also the simplest: actually manipulate X to the value x , and see what happens to Y . (As my mother says, “Why think, when you can just do the experiment?”) A causal or counter-factual assumption is still required here, which is that the *next* time we repeat the manipulation, the system will respond similarly, but this is pretty weak as such assumptions go.

While this seems like obvious common sense to us now, it is worth taking a moment to reflect on the fact that systematic experimentation is a very recent thing; it only goes back to around 1600. Since then, the knowledge we have acquired by combining experiments with mathematical theories have totally transformed human life, but for the first four or five thousand years of civilization, philosophers and sages much smarter than (almost?) any scientist now alive would have dismissed experiment as something fit only for cooks, potters and blacksmiths, who didn’t *really* know what they were doing.

The major obstacle the experimentalist must navigate around is to make sure they the experiment they are doing is the one they *think* they are doing. Symbolically, when we want to know $\Pr(Y|do(X = x))$, we need to make sure that we are *only* manipulating X , and not accidentally doing $\Pr(Y|do(X = x), Z = z)$ (because we are only experimenting on a sub-population), or $\Pr(Y|do(X = x, Z = z))$ (because we are also, inadvertently, manipulating Z). There are two big main divisions about how to avoid these confusions.

1. The older strategy is to *deliberately* control or manipulate as many other variables as possible. If we find $\Pr(Y|do(X = x, Z = z))$ and $\Pr(Y|do(X = x', Z = z))$ then we know the differences between them are indeed just due to changing X . This strategy, of actually controlling or manipulating whatever we can, is the

traditional one in the physical sciences, and more or less goes back to Galileo and the beginning of the Scientific Revolution¹.

2. The younger strategy is to *randomize* over all the other variables but X . That is, to examine the contrast between $\Pr(Y|do(X = x))$ and $\Pr(Y|do(X = x'))$, we use an independent source of random noise to decide which experimental subjects will get $do(X = x)$ and which will get $do(X = x')$. It is easy to convince yourself that this makes $\Pr(Y|do(X = x))$ equal to $\Pr(Y|X = x)$. The great advantage of the randomization approach is that we can apply it even when we cannot actually control the other causally relevant variables, or even are unsure of what they are. Unsurprisingly, it has its origins in the biological sciences, especially agriculture. If we want to credit its invention to a single culture hero, it would not be too misleading² to attribute it to R. A. Fisher in the early 1900s.

Experimental evidence is compelling, but experiments are often slow, expensive, and difficult. Moreover, experimenting on people is hard, both because there are many experiments we *shouldn't* do, and because there are many experiments which would just be too hard to organize. We must therefore consider how to do causal inference from non-experimental, observational data.

22.2 Identification and Confounding

For today's purposes, the most important distinction between probabilistic and causal conditioning has to do with the **identification** (or **identifiability**), of the conditional distributions. An aspect of a statistical model is **identifiable** when it cannot be changed without there also being *some* change in the distribution of the observable variables. If we can alter part of a model with no observable consequences, that part of the model is **unidentifiable**³. Sometimes the lack of identification is trivial: in a two-component mixture model, we get the same observable distribution if we swap the labels of the two component distributions. The rotation problem for factor models is a less trivial identification problem⁴. If two variables are co-linear, then their coefficients in a linear regression are unidentifiable⁵. Note that identification is about the true distribution, not about what happens with finite data. A parameter might be identifiable, but we could have so little information about it in our data that

¹The anguished sound you hear as you read this is every historian of science wailing in protest as the oversimplification, but this will do as an origin myth for our purposes.

²See previous note.

³More formally, say that the model has two parameters, θ and ψ . The distinction between θ_1 and θ_2 is identifiable if, for all ψ_1, ψ_2 , the distribution over observables coming from (θ_1, ψ_1) is different from that coming from (θ_2, ψ_2) . If the right choice of ψ_1 and ψ_2 masks the distinction between θ_1 and θ_2 , then θ is unidentifiable.

⁴As this example suggests, what is identifiable depends on what is observed. If we could observe the factors directly, factor loadings would be identifiable.

⁵As that example suggests, whether one aspect of a model is identifiable or not can depend on other aspects of the model. If the co-linearity was broken, the two regression coefficients would become identifiable.

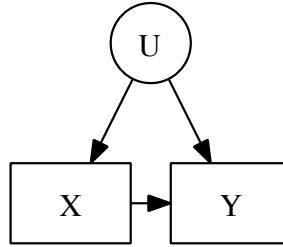


Figure 22.1: The distribution of Y given X , $\Pr(Y|X)$, **confounds** the actual causal effect of X on Y , $\Pr(Y|do(X=x))$, with the indirect dependence between X and Y created by their unobserved common cause U . (You may imagine that U is really more than one variable, with some internal sub-graph.)

our estimates are unusable, with immensely wide confidence intervals; that's unfortunate, but we just need more data. An unidentifiable parameter, however, cannot be estimated even with infinite data.⁶

When X and Y are both observable variables, $\Pr(Y|X=x)$ can't help being identifiable. (Changing this just *is* changing part of the distribution of observables.) Things are very different, however, for $\Pr(Y|do(X=x))$. In some models, it's entirely possible to change this drastically, and always have the same distribution of observables, by making compensating changes to other parts of the model. When this is the case, we simply cannot estimate causal effects from observational data. The basic problem is illustrated in Figure 22.1

In Figure 22.1, X is a parent of Y . But if we analyze the dependence of Y on X , say in the form of the conditional distribution $\Pr(Y|X=x)$, we see that there are two channels by which information flows from cause to effect. One is the direct, causal path, represented by $\Pr(Y|do(X=x))$. The other is the indirect path, where X gives information about its parent U , and U gives information about its child Y . If we just observe X and Y , we cannot separate the causal effect from the indirect inference. The causal effect is **confounded** with the indirect inference. More generally, the effect of X on Y is confounded whenever $\Pr(Y|do(X=x)) \neq \Pr(Y|X=x)$. If there is some way to write $\Pr(Y|do(X=x))$ in terms of distributions of observables, we say that the confounding can be removed by an **identification strategy**, which **de-confounds** the effect. If there is no way to de-confound, then this causal effect is unidentifiable.

The effect of X on Y in Figure 22.1 is unidentifiable. Even if we erased the arrow

⁶For more on identifiability, and what to do with unidentifiable problems, see the great book by Manski (2007).

from X to Y , we could get any joint distribution for X and Y we liked by picking $P(X|U)$, $P(Y|U)$ and $P(U)$ appropriately. So we cannot even, in this situation, use observations to tell whether X is actually a cause of Y . Notice, however, that even if U was observed, it would still not be the case that $\Pr(Y|X = x) = \Pr(Y|do(X = x))$. While the effect would be identifiable (via the back door criterion; see below), we would still need some sort of adjustment to recover it.

In the next section, we will look at such identification strategies and adjustments.

22.3 Identification Strategies

To recap, we want to calculate the causal effect of X on Y , $\Pr(Y|do(X = x))$, but we cannot do an experiment, and must rely on observations. In addition to X and Y , there will generally be some **covariates** Z which we know, and we'll assume we know the causal graph, which is a DAG. Is this enough to determine $\Pr(Y|do(X = x))$? That is, does the joint distribution **identify** the causal effect?

The answer is “yes” when the covariates Z contain all the other relevant variables⁷. The inferential problem is then no worse than any other statistical estimation problem. In fact, if we know the causal graph and get to observe all the variables, then we could (in principle) just use our favorite non-parametric conditional density estimate at each node in the graph, with its parent variables as the inputs and its own variable as the response. Multiplying conditional distributions together gives the whole distribution of the graph, and we can get any causal effects we want by surgery. Equivalently (Exercise 2), we have that

$$\Pr(Y|do(X = x)) = \sum_t \Pr(Y|X = x, \text{Pa}(X) = t) \Pr(\text{Pa}(X) = t) \quad (22.3)$$

where $\text{Pa}(X)$ is the complete set of parents of X .

If we're willing to assume more, we can get away with just using non-parametric regression or even just an additive model at each node. Assuming yet more, we could use parametric models at each node; the linear-Gaussian assumption is (alas) very popular.

If some variables are *not* observed, then the issue of which causal effects are observationally identifiable is considerably trickier. Apparently subtle changes in which variables are available to us and used can have profound consequences.

The basic principle underlying all considerations is that we would like to condition on adequate **control** variables, which will block paths linking X and Y *other than* those which would exist in the surgically-altered graph where all paths into X have been removed. If other unblocked paths exist, then there is some confounding of the causal effect of X on Y with their mutual dependence on other variables.

This is familiar to use from regression as the basic idea behind using additional variables in our regression, where the idea is that by introducing covariates, we “control for” other effects, until the regression coefficient for our favorite variable represents only its causal effect. Leaving aside the inadequacies of linear regression as such (Chapter 2), we need to be cautious here. Just conditioning on everything possible does *not* give us adequate control, or even necessarily bring us closer to it. As Figure 22.2 illustrates, and as the next homework will drive home, *adding* an ill-chosen covariate to a regression can create confounding.

⁷This condition is sometimes known as **causal sufficiency**. Strictly speaking, we do not have to suppose that *all* causes are included in the model and observable. What we have to assume is that all of the remaining causes have such an unsystematic relationship to the ones included in the DAG that they can be modeled as noise. (This does not mean that the noise is necessarily small.) In fact, what we really have to assume is that the relationships between the causes omitted from the DAG and those included is so intricate and convoluted that it might as well be noise, along the lines of algorithmic information theory (Li and Vitányi, 1997), whose key result might be summed up as “Any determinism distinguishable from randomness is insufficiently complex”. But here we verge on philosophy.

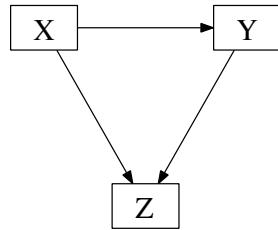


Figure 22.2: “Controlling for” additional variables can introduce bias into estimates of causal effects. Here the effect of X on Y is directly identifiable, $\Pr(Y|do(X=x)) = \Pr(Y|X=x)$. If we also condition on Z however, because it is a common effect of X and Y , we’d get $\Pr(Y|X=x, Z=z) \neq \Pr(Y|X=x)$. In fact, even if there were no arrow from X to Y , conditioning on Z would make Y depend on X .

There are three main ways we can find adequate controls, and so get both identifiability and appropriate adjustments:

1. We can condition on an intelligently-chosen set of covariates S , which block all the indirect paths from X to Y , but leave all the direct paths open. (That is, we can follow the regression strategy, but do it right.) To see whether a candidate set of controls S is adequate, we apply the **back-door criterion**.
2. We can find a set of variables M which **mediate** the causal influence of X on Y — all of the direct paths from X to Y pass through M . If we can identify the effect of M on Y , and of X on M , then we can combine these to get the effect of X on Y . (That is, we can just study the *mechanisms* by which X influences Y .) The test for whether we can do this combination is the **front-door criterion**.
3. We can find a variable I which affects X , and which *only* affects Y by influencing X . If we can identify the effect of I on Y , and of I on X , then we can, sometimes, “factor” them to get the effect of X on Y . (That is, I gives us variation in X which is independent of the common causes of X and Y .) I is then an **instrumental variable** for the effect of X on Y .

Let’s look at these three in turn.

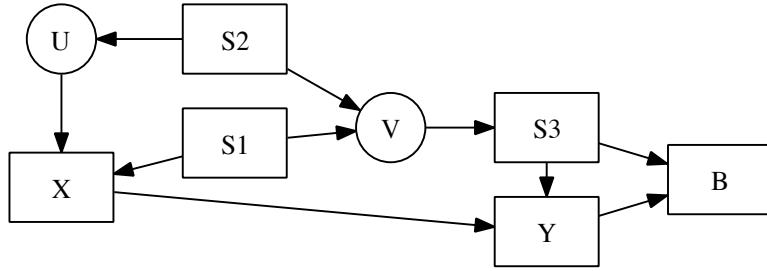


Figure 22.3: Illustration of the back-door criterion for identifying the causal effect of X on Y . Setting $S = \{S_1, S_2\}$ satisfies the criterion, but neither S_1 nor S_2 on their own would. Setting $S = \{S_3\}$, or $S = \{S_1, S_2, S_3\}$ also works. Adding B to any of the good sets makes them fail the criterion.

22.3.1 The Back-Door Criterion: Identification by Conditioning

When estimating the effect of X on Y , a **back-door path** is an undirected path between X and Y with an arrow *into* X . These are the paths which create confounding, by providing an indirect, non-causal channel along which information can flow. A set of conditioning variables or controls S satisfies the **back-door criterion** when (i) S blocks every back-door path between X and Y , and (ii) no node in S is a descendant of X . (Cf. Figure 22.3.) When S meets the back-door criterion,

$$\Pr(Y|do(X=x)) = \sum_s \Pr(Y|X=x, S=s) \Pr(S=s) \quad (22.4)$$

Notice that all the items on the right-hand side are observational conditional probabilities, not counterfactuals. Thus we have achieved identifiability, as well as having an adjustment strategy.

The motive for (i) is plain, but what about (ii)? We don't want to include descendants of X which are also ancestors of Y , because that blocks off some of the causal paths from X to Y , and we don't want to include descendants of X which are also descendants of Y , because they provide non-causal information about Y ⁸.

More formally, we can proceed as follows (Pearl, 2009b, §11.3.3). We know from Eq. 22.3 that

$$\Pr(Y|do(X=x)) = \sum_t \Pr(\text{Pa}(X)=t) \Pr(Y|X=x, \text{Pa}(X)=t) \quad (22.5)$$

⁸What about descendants of X which are neither ancestors nor descendants of Y ? Conditioning on them is either creates potential colliders, if they are also descended from ancestors of Y other than X , or needlessly complicates the adjustment in Eq. 22.4.

Now suppose we can always introduce another set of conditioned variables, if we sum out over them:

$$\Pr(Y|do(X=x)) = \sum_t \Pr(\text{Pa}(X)=t) \sum_s \Pr(Y, S=s | X=x, \text{Pa}(X)=t) \quad (22.6)$$

We can do this for *any* set of variables S , it's just probability. It's also just probability that

$$\begin{aligned} \Pr(Y, S | X=x, \text{Pa}(X)=t) &= \\ \Pr(Y | X=x, \text{Pa}(X)=t, S=s) \Pr(S=s | X=x, \text{Pa}(X)=t) \end{aligned} \quad (22.7)$$

so

$$\begin{aligned} \Pr(Y|do(X=x)) &= \\ \sum_t \Pr(\text{Pa}(X)=t) \sum_s \Pr(Y | X=x, \text{Pa}(X)=t, S=s) \Pr(S=s | X=x, \text{Pa}(X)=t) \end{aligned} \quad (22.8)$$

Now we invoke the fact that S satisfies the back-door criterion. Point (i) of the criterion, blocking back-door paths, implies that $Y \perp\!\!\!\perp \text{Pa}(X) | X, S$. Thus

$$\begin{aligned} \Pr(Y|do(X=x)) &= \\ \sum_t \Pr(\text{Pa}(X)=t) \sum_s \Pr(Y | X=x, S=s) \Pr(S=s | X=x, \text{Pa}(X)=t) \end{aligned} \quad (22.9)$$

Point (ii) of the criterion, not containing descendants of X , means (by the Markov property) that $X \perp\!\!\!\perp S | \text{Pa}(X)$. Therefore

$$\begin{aligned} \Pr(Y|do(X=x)) &= \\ \sum_t \Pr(\text{Pa}(X)=t) \sum_s \Pr(Y | X=x, S=s) \Pr(S=s | \text{Pa}(X)=t) \end{aligned} \quad (22.10)$$

Since $\sum_t \Pr(\text{Pa}(X)=t) \Pr(S=s | \text{Pa}(X)=t) = \Pr(S=s)$, we have, at last,

$$\Pr(Y|do(X=x)) = \sum_s \Pr(Y | X=x, S=s) \Pr(S=s) \quad (22.11)$$

as promised. \square

22.3.1.1 The Entner Rules

Using the back-door criterion requires us to know the causal graph. Recently, Entner *et al.* (2013) have given a simple set of rules which provide *sufficient* conditions for deciding that set of variables satisfy the back-door criterion, or that X actually has no effect on Y , which can be used without knowing the graph completely.

It makes no sense to control for anything which is a descendant of either Y or X ; that's either blocking a directed path or activating a collider. So let \mathcal{W} be the set of all observed variables which descend neither from X nor Y .

1. If there is a set of controls S such that $X \perp\!\!\!\perp Y|S$, then X has no causal effect on Y .

Reasoning: Y can't be a child of X if we can make them independent by conditioning on anything, and Y can't be a more remote descendant either, since S doesn't include any descendants of X . So in this situation all the paths linking X to Y must be back-door paths, and S , blocking them, shows there's no effect.

2. If there is a $W \in \mathcal{W}$ and a subset S of the \mathcal{W} , not including W , such that (i) $W \not\perp\!\!\!\perp Y|S$, but (ii) $W \perp\!\!\!\perp Y|S, X$, then X has an effect on Y , and S satisfies the back-door criterion for estimating the effect.

Reasoning: Point (i) shows that conditioning on S leaves open path from W to Y . By point (ii), these paths must all pass through X , since conditioning on X blocks them, hence X has an effect on Y . S must block all the back-door paths between X and Y , otherwise X would be a collider on paths between W and Y , so conditioning on X would activate those paths.

3. If there is a $W \in \mathcal{W}$ and a subset S of \mathcal{W} , excluding W , such that (i) $W \not\perp\!\!\!\perp X|S$ but (ii) $W \perp\!\!\!\perp Y|S$, then X has no effect on Y .

Reasoning: Point (i) shows that conditioning on S leaves open active paths from W to X . But by (ii), there cannot be any open paths from W to Y , so there cannot be any open paths from X to Y .

If none of these rules apply, whether X has an effect on Y , and if so what adequate controls are for finding it, will depend on the exact graph, and *cannot* be determined just from independence relations among the observables. (For proofs of everything, see the paper.)

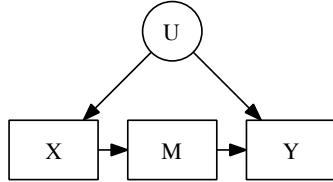


Figure 22.4: Illustration of the front-door criterion, after Pearl (2009b, Figure 3.5). X , Y and M are all observed, but U is an unobserved common cause of both X and Y . $X \leftarrow U \rightarrow Y$ is a back-door path confounding the effect of X on Y with their common cause. However, all of the effect of X on Y is mediated through X 's effect on M . M 's effect on Y is, in turn, confounded by the back-door path $M \leftarrow X \leftarrow U \rightarrow Y$, but X blocks this path. So we can use back-door adjustment to find $\Pr(Y|do(M = m))$, and directly find $\Pr(M|do(X = x)) = \Pr(M|X = x)$. Putting these together gives $\Pr(Y|do(X = x))$.

22.3.2 The Front-Door Criterion: Identification by Mechanisms

A set of variables M satisfies the **front-door criterion** when (i) M blocks all directed paths from X to Y , (ii) there are no unblocked back-door paths from X to M , and (iii) X blocks all back-door paths from M to Y . Then

$$\begin{aligned} \Pr(Y|do(X = x)) &= \\ &\sum_m \Pr(M = m|X = x) \sum_{x'} \Pr(Y|X = x', M = m) \Pr(X = x') \end{aligned} \tag{22.12}$$

A natural reaction to the front-door criterion is “Say what?”, but it becomes more comprehensible if we take it apart. Because, by clause (i), M blocks all *directed* paths from X to Y , any *causal* dependence of Y on X must be mediated by a dependence of Y on M :

$$\Pr(Y|do(X = x)) = \sum_m \Pr(Y|do(M = m)) \Pr(M = m|do(X = x)) \tag{22.13}$$

Clause (ii) says that we can get the effect of X on M directly,

$$\Pr(M = m|do(X = x)) = \Pr(M = m|X = x). \tag{22.14}$$

Clause (iii) say that X satisfies the back-door criterion for identifying the effect of M on Y , and the inner sum in Eq. 22.12 is just the back-door computation (Eq. 22.4) of $\Pr(Y|do(M = m))$. So really we *are* using the back door criterion, twice. (See Figure 22.4.)

For example, in the “does tooth-brushing prevent heart-disease?” example of §21.2.2, we have X = “frequency of tooth-brushing”, Y = “heart disease”, and we could take as the mediating M either “gum disease” or “inflammatory immune response”, according to Figure 21.2.

22.3.2.1 The Front-Door Criterion and Mechanistic Explanation

Morgan and Winship (2007, ch. 8) give a useful insight into the front-door criterion. Each directed path from X to Y is, or can be thought of as, a separate **mechanism** by which X influences Y . The requirement that all such paths be blocked by M , (i), is the requirement that the set of mechanisms included in M be “exhaustive”. The two back-door conditions, (ii) and (iii), require that the mechanisms be “isolated”, not interfered with by the rest of the data-generating process (at least once we condition on X). Once we identify an isolated and exhaustive set of mechanisms, we know all the ways in which X actually affects Y , and any indirect paths can be discounted, using the front-door adjustment 22.12.

One interesting possibility suggested by this is to elaborate mechanisms into sub-mechanisms, which could be used in some cases where the plain front-door criterion won’t apply⁹, such as Figure 22.5. Because U is a parent of M , we cannot use the front-door criterion to identify the effect of X on Y . (Clause (i) holds, but (ii) and (iii) both fail.) But we can use M_1 and the front-door criterion to find $\Pr(M|do(X=x))$, and we can use M_2 to find $\Pr(Y|do(M=m))$. Chaining those together, as in Eq. 22.13, would give $\Pr(Y|do(X=x))$. So even though the whole mechanism from X to Y is not isolated, we can still identify effects by breaking it into sub-mechanisms which *are* isolated. This suggests a natural point at which to stop refining our account of the mechanism into sub-sub-sub-mechanisms: when we can identify the causal effects we’re concerned with.

⁹The ideas in this paragraph come from conversation Prof. Winship, who I understand is currently preparing a paper on this.

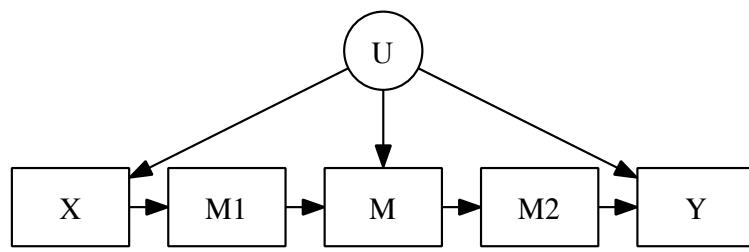


Figure 22.5: The path $X \rightarrow M \rightarrow Y$ contains all the mechanisms by which X influences Y , but is not isolated from the rest of the system ($U \rightarrow M$). The sub-mechanisms $X \rightarrow M_1 \rightarrow M$ and $M \rightarrow M_2 \rightarrow Y$ are isolated, and the original causal effect can be identified by composing them.

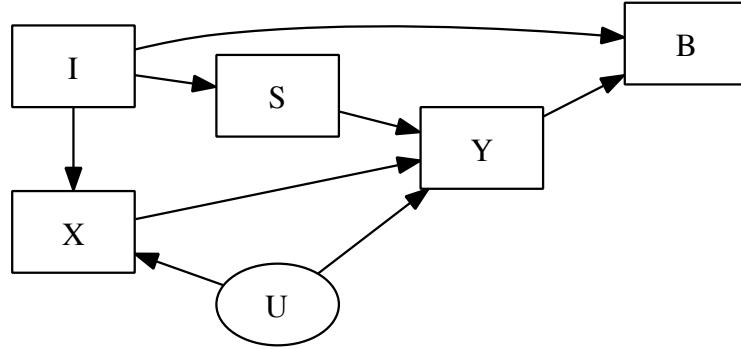


Figure 22.6: A valid instrumental variable, I , is related to the cause of interest, X , and influences Y only through its influence on X , at least once control variables block other paths. Here, to use I as an instrument, we *should* condition on S , but *should not* condition on B . (If we could condition on U , we would not need to use an instrument.)

22.3.3 Instrumental Variables

A variable I is an **instrument**¹⁰ for identifying the effect of X on Y when there is a set of controls S such that (i) $I \not\perp\!\!\!\perp X|S$, and (ii) every unblocked path from I to Y has an arrow pointing into to X . Another way to say (ii) is that $I \perp\!\!\!\perp Y|S, do(X)$. Colloquially, I influences Y , but only through first influencing X (at least once we control for S). (See Figure 22.6.)

How is this useful? By making back-door adjustments for S , we can identify $\Pr(Y|do(I = i))$ and $\Pr(X|do(I = i))$. Since all the causal influence of I on Y must be channeled through X (by point (ii)), we have

$$\Pr(Y|do(I = i)) = \sum_x \Pr(Y|do(X = x)) \Pr(X = x|do(I = i)) \quad (22.15)$$

as in Eq. 22.3. We can thus identify the causal effect of X on Y whenever Eq. 22.15 can be solved for $\Pr(Y|do(X = x))$ in terms of $\Pr(Y|do(I = i))$ and $\Pr(X|do(I = i))$. Figuring out when this is possible in general requires an excursion into the theory of

¹⁰The term “instrumental variables” comes from econometrics, where they were originally used, in the 1940s, to identify parameters in simultaneous equation models. (The metaphor was that I is a measuring instrument for the otherwise inaccessible parameters.) Definitions of instrumental variables are surprisingly murky and controversial outside of extremely simple linear systems; this one is taken from Galles and Pearl (1997), via Pearl (2009b, §7.4.5).

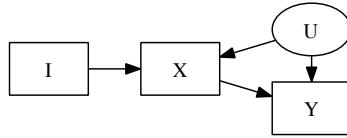


Figure 22.7: I acts as an instrument for estimating the effect of X on Y , despite the presence of the confounding, unobserved variable U .

integral equations¹¹, which is beyond the scope of this class; the upshot is that, in general, there are no solutions. However, in the special case where the relations between all variables are linear, we can do better.

Let's start with the most basic possible set-up for an instrumental variable, namely that in Figure 22.7, where we just have X , Y , the instrument I , and the unobserved confounders S . If everything is linear, identifying the causal effect of X on Y is equivalent to identifying the coefficient on the $X \rightarrow Y$ arrow. We can write

$$X = \alpha_0 + \alpha I + \delta U + \epsilon_X \quad (22.16)$$

and

$$Y = \beta_0 + \beta X + \gamma U + \epsilon_Y \quad (22.17)$$

where ϵ_X and ϵ_Y are mean-zero noise terms, independent of each other and of the other variables, and we can, without loss of generality, assume U has mean zero as well. We want to find β . Substituting,

$$Y = \beta_0 + \beta \alpha_0 + \beta \alpha I + (\beta \delta + \gamma)U + \beta \epsilon_X + \epsilon_Y \quad (22.18)$$

Since U , ϵ_X and ϵ_Y are all unobserved, we can re-write this as

$$Y = \gamma_0 + \beta \alpha I + \eta \quad (22.19)$$

where $\eta = (\beta \delta + \gamma)U + \beta \epsilon_X + \epsilon_Y$ has mean zero.

Now take the covariances:

$$\text{Cov}[I, X] = \alpha \text{Var}[I] + \text{Cov}[\epsilon_X, I] \quad (22.20)$$

$$\text{Cov}[I, Y] = \beta \alpha \text{Var}[I] + \text{Cov}[\eta, I] \quad (22.21)$$

$$\begin{aligned} &= \beta \alpha \text{Var}[I] + (\beta \delta + \gamma) \text{Cov}[U, I] \\ &\quad + \beta \text{Cov}[\epsilon_X, I] + \text{Cov}[\epsilon_Y, I] \end{aligned} \quad (22.22)$$

¹¹If X is continuous, then the analog of Eq. 22.15 is $\Pr(Y|do(I=i)) = \int p(Y|do(X=x))p(X=x|do(I=i))dx$, where the “integral operator” $\int \cdot p(X=x|do(I=i))dx$ is known, as is $\Pr(Y|do(I=i))$.

By condition (ii), however, we must have $\text{Cov}[U, I] = 0$, and of course $\text{Cov}[\epsilon_X, I] = \text{Cov}[\epsilon_Y, I] = 0$. Therefore $\text{Cov}[I, Y] = \beta\alpha\text{Var}[I]$. Solving,

$$\beta = \frac{\text{Cov}[I, Y]}{\text{Cov}[I, X]} \quad (22.23)$$

This can be estimated by substituting in the sample covariances, or any other consistent estimators of these two covariances.

On the other hand, the (true or population-level) coefficient for linearly regressing Y on X is

$$\frac{\text{Cov}[X, Y]}{\text{Var}[X]} = \frac{\beta\text{Var}[X] + \gamma\text{Cov}[U, X]}{\text{Var}[X]} \quad (22.24)$$

$$= \beta + \gamma \frac{\text{Cov}[U, X]}{\text{Var}[X]} \quad (22.25)$$

$$= \beta + \gamma \frac{\delta\text{Var}[U]}{\alpha^2\text{Var}[I] + \delta^2\text{Var}[U] + \text{Var}[\epsilon_X]} \quad (22.26)$$

That is, “OLS is biased for the causal effect when X is correlated with the noise”. In other words, simple regression is misleading in the presence of confounding¹².

The instrumental variable I provides a source of variation in X which is uncorrelated with the other common ancestors of X and Y . By seeing how both X and Y respond to these perturbations, and using the fact that I only influences Y through X , we can deduce something about how X influences Y , though linearity is very important to our ability to do so.

The simple line of reasoning above runs into trouble if we have multiple instruments, or need to include controls (as the definition of an instrument allows). In §23.2 we’ll look at the more complicated estimation methods which can handle this, again assuming linearity.

22.3.3.1 Some Invalid Instruments

Not everything which looks like an instrument actually works. If Y is indeed a descendant of I , but there is a line of descent that doesn’t go through X , then I is not a valid instrument for X (Figure 22.8). If there are unblocked back-door paths linking I and Y — if I and Y have common ancestors, for instance — then I is not a valid instrument (Figure 22.9).

Economists sometimes refer to both sets of problems with instruments as “violations of exclusion restrictions”. The second sort of problem, in particular, is a “failure of exogeneity”.

¹²But observe that if we want to make a linear prediction of Y and only have X available, i.e., to find the best r_1 in $E[Y|X = x] = r_0 + r_1x$, then Eq. 22.26 is *exactly* the coefficient we would want to use. OLS is doing its job.

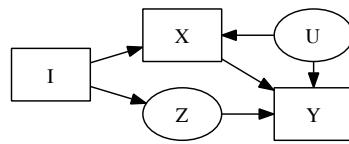


Figure 22.8: I is not a valid instrument for identifying the effect of X on Y , because I can influence Y through a path not going through X . If we could control for Z , however, I would become valid.

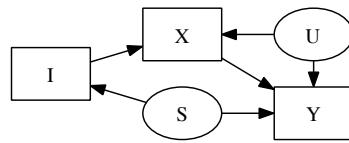


Figure 22.9: I is not a valid instrument for identifying the effect of X on Y , because there is an unblocked back-door path connecting I and Y . If we could control for S , however, I would become valid.

22.3.3.2 Critique of Instrumental Variables

By this point, you may well be thinking that instrumental variable estimation is very much like using the front-door criterion. There, the extra variable M came between X and Y ; here, X comes between I and Y . It is, perhaps, surprising (if not annoying) that using an instrument only lets us identify causal effects under extra assumptions, but that's life. Just as the front-door criterion relies on using our scientific knowledge, or rather theories, to find isolated and exhaustive mechanisms, finding valid instruments relies on theories about the world (or the part of it under study), and one would want to try to check those theories.

In fact, instrumental variable estimates of causal effects are often presented as more or less unquestionable, and free of theoretical assumptions; economists, and other social scientists influenced by them, are especially apt to do this. As the economist Daniel Davies puts it¹³, devotees of this approach

have a really bad habit of saying:
 "Whichever way you look at the numbers, X".
 when all they can really justify is:
 "Whichever way I look at the numbers, X".
 but in fact, I should have said that they could only really support:
 "Whichever way I look at **these** numbers, X".

(Emphasis in the original.) It will not surprise you to learn that I think this is very wrong.

I hope that, after four months of nonlinear models, if someone tries to sell you a linear regression, you should be very skeptical, but let's leave that to one side. (It's not *impossible* that everything really is linear.) The clue that instrumental variable estimation is a creature of theoretical assumptions is point (ii) in the definition of an instrument: $I \perp\!\!\!\perp Y | S, do(X)$. This says that if we eliminate all the arrows into X , the control variables S block all the other paths between I and Y . This is *exactly* as much an assertion about mechanisms as what we have to do with the front-door criterion. In fact it doesn't just say that every mechanism by which I influences Y is mediated by X , it also says that there are no common causes of I and Y (other than those blocked by S).

This assumption is most easily defended when I is genuinely random. For instance, if we do a randomized experiment, I might be a coin-toss which assigns each subject to be in either the treatment or control group, each with a different value of X . If "compliance" is not perfect (if some of those in the treatment group don't actually get the treatment, or some in the control group do), it is nonetheless plausible that the only route by which I influences the outcome is through X , so an instrumental variable regression is appropriate. (I here is sometimes called "intent to treat".)

Even here, we must be careful. If we are evaluating a new medicine, whether people *think* they are getting a medicine or not could change how they act, and medical outcomes. Knowing whether they were assigned to the treatment or the control

¹³In part four of his epic and insightful review of *Freakonomics*; see <http://d-squareddigest.blogspot.com/2007/09/freakiology-yes-folks-its-part-4-of.html>.

group would thus create another path from I to Y , not going through X . This is why randomized clinical trials are generally “double-blinded” (neither patients nor medical personnel know who is in the control group); but how effective the double-blinding is itself a theoretical assumption.

More generally, any argument that a candidate instrument is valid is really an argument that other channels of influence, apart from the favored one through X , can be ruled out. This generally cannot be done through analyzing the same variables used in the instrumental-variable estimation (see below), but involves some theory about the world, and rests on the strength of the evidence for that theory. As has been pointed out multiple times — for instance, by Rosenzweig and Wolpin (2000), and by Deaton (2010) — the theories needed to support instrumental variable estimates in particular concrete cases are often *not* very well-supported, and plausible rival theories can produce very different conclusions from the same data.

Many people have thought that one *can* test for the validity of an instrument, by looking at whether $I \perp\!\!\!\perp Y|X$ — the idea being that, if influence flows from I through X to Y , conditioning on X should block the channel. The problem is that, in the instrumental-variable set-up, X is a collider, so conditioning on X actually creates an indirect dependence *even if* I is valid. So $I \not\perp\!\!\!\perp Y|X$, whether or not the instrument is valid, and the test (even if performed perfectly with infinite data) tells us nothing¹⁴.

A final, more or less technical, issue with instrumental variable estimation is that many instruments are (even if valid) **weak** — they only have a little influence on X , and a small covariance with it. This means that the denominator in Eq. 22.23 is a number close to zero. Error in estimating the denominator, then, results in a much larger error in estimating the ratio. Weak instruments lead to noisy and imprecise estimates of causal effects. It is not hard to construct scenarios where, at reasonable sample sizes, one is actually better off using the biased OLS estimate than the unbiased but high-variance instrumental estimate.

¹⁴However, see Pearl (2009b, §8.4) for a different approach which can “screen out very bad would-be instruments”.

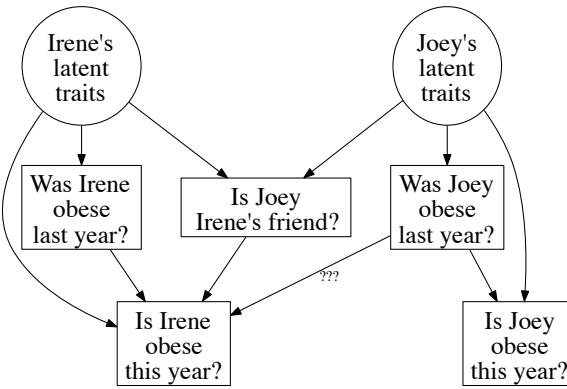


Figure 22.10: Social influence is confounded with selecting friends with similar traits, unobserved in the data.

22.3.4 Failures of Identification

The back-door and front-door criteria, and instrumental variables, are all *sufficient* for estimating causal effects from probabilistic distributions, but are not *necessary*. A necessary condition for *un*-identifiability is the presence of an unblockable back-door path from X to Y . However, this is not sufficient for lack of identification — we might, for instance, be able to use the front door criterion, as in Figure 22.4. There are necessary and sufficient conditions for the identifiability of causal effects in terms of the graph, and so for un-identifiability, but they are rather complex and I will not go over them (see Shpitser and Pearl (2008), and Pearl (2009b, §§3.4–3.5) for an overview).

As an example of the unidentifiable case, consider Figure 22.10. This DAG depicts the situation analyzed in Christakis and Fowler (2007), a famous paper claiming to show that obesity is contagious in social networks (or at least in the town in Massachusetts where the data was collected). At each observation, participants in the study get their weight taken, and so their obesity status is known over time. They also provide the name of a friend. This friend is often in the study. Christakis and Fowler were interested in the possibility that obesity is contagious, perhaps through some process of behavioral influence. If this is so, then Irene’s obesity status in year 2 should depend on Joey’s obesity status in year one, but *only* if Irene and Joey are friends — not if they are just random, unconnected people. It is indeed the case that if Joey becomes obese, this predicts a substantial increase in the odds of Joey’s friend Irene becoming obese, even controlling for Irene’s previous history of obesity¹⁵.

The difficulty arises from the latent variables for Irene and Joey (the round nodes

¹⁵The actual analysis was a bit more convoluted than that, but this is the general idea.

in Figure 22.10). These include all the traits of either person which (a) influence who they become friends with, and (b) influence whether or not they become obese. A very partial list of these would include: taste for recreational exercise, opportunity for recreational exercise, taste for alcohol, ability to consume alcohol, tastes in food, occupation and how physically demanding it is, ethnic background¹⁶, etc. Put simply, if Irene and Joey are friends because they spend two hours in the same bar every day drinking and eating chicken wings with ranch dressing, it's less surprising that both of them have an elevated chance of becoming obese, and likewise if they became friends because they both belong to the decathlete's club, they are both unusually unlikely to become obese. Irene's status is predictable from Joey's, then, not (or not just) because Joey influences Irene, but because seeing what kind of person Irene's friends are tells us about what kind of person Irene is. It is not too hard to convince oneself that there is just no way, in this DAG, to get at the causal effect of Joey's behavior on Irene's that isn't confounded with their latent traits (Shalizi and Thomas, 2011). To de-confound, we would need to actual measure those latent traits, which may not be impossible but is certainly was not done here¹⁷.

When identification is not possible — when we can't de-confound — it may still be possible to *bound* causal effects. That is, even if we can't say exactly that $\Pr(Y|do(X = x))$ must be, we can still say it has to fall within a certain (non-trivial!) range of possibilities. The development of bounds for non-identifiable quantities, what's sometimes called **partial identification**, is an active area of research, which I think is very likely to become more and more important in data analysis; the best introduction I know is Manski (2007).

¹⁶Friendships often run within ethnic communities. On the one hand, this means that friends tend to be more *genetically* similar than random members of the same town, so they will be usually apt to share genes which influence susceptibility to obesity (in that environment). On the other hand, ethnic communities transmit, non-genetically, traditions regarding food, alcohol, sports, exercise, etc., and (again non-genetically) influence employment and housing opportunities.

¹⁷Of course, the issue is not really about obesity. Studies of "viral marketing", and of social influence more broadly, all generically have the same problem. Predicting someone's behavior from that of their friend means conditioning on the existence of a social tie between them, but that social tie is a collider, and activating the collider creates confounding.

22.4 Summary

Of the four techniques I have introduced, instrumental variables are clever, but fragile and over-sold¹⁸. Experimentation is ideal, but often unavailable. The back-door and front-door criteria are, I think, the best observational approaches, when they can be made to work.

Often, nothing can be made to work. Many interesting causal effects are just not identifiable from observational data. More exactly, they only become identifiable under very strong modeling assumptions, typically ones which cannot be tested from the same data, and sometimes ones which cannot be tested by any sort of empirical data whatsoever. Sometimes, we have good reasons (from other parts of our scientific knowledge) to make such assumptions. Sometimes, we make such assumptions because we have a pressing need for *some* basis on which to act, and a wrong guess is better than nothing¹⁹. If you do make such assumptions, you need to make clear that you are doing so, and what they are; explain your reasons for making those assumptions, and not others²⁰; and indicate how different your conclusions could be if you made different assumptions.

22.4.1 Further Reading

My presentation of the three major criteria is heavily indebted to Morgan and Winship (2007), but I hope not a complete rip-off. Pearl (2009b) is also essential reading on this topic. Berk (2004) provides an excellent critique of naive (that is, overwhelmingly common) uses of regression for estimating causal effects.

Most econometrics texts devote considerable space to instrumental variables. Didelez *et al.* (2010) is a very good discussion of instrumental variable methods, with less-standard applications. There is some work on non-parametric versions of instrumental variables (e.g., Newey and Powell 2003), but the form of the models must be restricted or they are unidentifiable. On the limitations of instrumental variables, Rosenzweig and Wolpin (2000) and Deaton (2010) are particularly recommended; the latter reviews the issue in connection with important recent work in development economics and the alleviation of extreme poverty, an area where statistical estimates really do matter.

There is a large literature in the philosophy of science and in methodology on the notion of “mechanisms”. References I have found useful include, in general, Salmon (1984), and, specifically on social processes, Elster (1989), Hedström and Swedberg (1998) (especially Boudon 1998), Hedström (2005), Tilly (1984, 2008), and DeLanda (2006).

22.5 Exercises

To think through, not to hand in.

¹⁸I confess that I would probably not be so down on them if others did not push them up so excessively.

¹⁹As I once heard a distinguished public health expert put it, “This problem is too important to worry about getting it right.”

²⁰“My boss/textbook says so” and “so I can estimate β ” are not good reasons

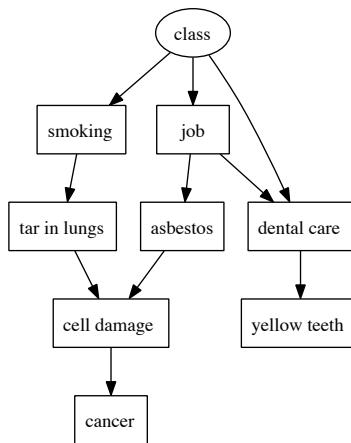


Figure 22.11: DAG for Exercise 3.

1. Draw a graphical model representing the situation where a causal variable X is set at random. Verify that $\Pr(Y|X = x)$ is then equal to $\Pr(Y|do(X = x))$. (*Hint:* Use the back door criterion.)
2. Prove Eq. 22.3 from the causal Markov property to the appropriate surgically-altered graph.
3. Refer to Figure 22.11. Can we use the front door criterion to estimate the effect of occupational prestige on cancer? If so, give a set S of variables that we would adjust for in the front-door method. Is there more than one such set? If so, can you find them all? Are there variables we could add to this set (or sets) which would violate the front-door criterion?
4. Read Salmon (1984). When does his “statistical relevance basis” provide enough information to identify causal effects?

Chapter 23

Estimating Causal Effects from Observations

Chapter 22 gave us ways of identifying causal effects, that is, of knowing when quantities like $\Pr(Y = y|do(X = x))$ are functions of the distribution of observable variables. Once we know that something is identifiable, the next question is how we can actually estimate it from data.

23.1 Estimators in the Back- and Front- Door Criteria

The back-door and front-door criteria for identification not only show us when causal effects are identifiable, they actually give us formulas for representing the causal effects in terms of ordinary conditional probabilities. When S satisfies the back-door criterion, for instance,

$$\Pr(Y = y|do(X = x)) = \sum_s \Pr(S = s) \Pr(Y = y|X = x, S = s) \quad (23.1)$$

Everything on the right-hand side refers to the distribution of observables, following the usual DAG without any surgery.

This is *very handy*, because we have spent the whole first part of the class learning different ways of estimating distributions like $\Pr(S = s)$ and $\Pr(Y = y|X = x, S = s)$. We can do fully non-parametric density estimation (Chapter 15), we can use parametric density models, we can model $Y|X, S = f(X, S) + \epsilon_Y$ and use regression, etc. If $\hat{\Pr}(Y = y|X = x, S = s)$ is a consistent estimator of $\Pr(Y = y|X = x, S = s)$, and $\hat{\Pr}(S = s)$ is a consistent estimator of $\Pr(S = s)$, then

$$\sum_s \hat{\Pr}(S = s) \hat{\Pr}(Y = y|X = x, S = s) \quad (23.2)$$

will be a consistent estimator of $\Pr(Y|do(X = x))$.

In principle, I could end this section right here, but there are some special cases and tricks which are worth knowing about. For simplicity, I will in this section only work with the back-door criterion, since estimating with the front-door criterion amounts to doing two rounds of back-door adjustment.

23.1.1 Estimating Average Causal Effects

Because $\Pr(Y|do(X = x))$ is a probability distribution, one can ask about $E[Y|do(X = x)]$, when it makes sense for Y to have an expectation value; it's just

$$E[Y|do(X = x)] = \sum_y y \Pr(Y = y|do(X = x)) \quad (23.3)$$

as you'd hope. This is the **average effect**, or sometimes just **the effect** of $do(X = x)$. While it is certainly not *always* the case that it summarizes all there is to know about the effect of X on Y , it is often useful.

If we identify the effect of X on Y through the back-door criterion, with control variables S , then some algebra shows

$$E[Y|do(X = x)] = \sum_y y \Pr(Y = y|do(X = x)) \quad (23.4)$$

$$= \sum_y y \sum_s \Pr(Y = y|X = x, S = s) \Pr(S = s) \quad (23.5)$$

$$= \sum_s \Pr(S = s) \sum_y y \Pr(Y = y|X = x, S = s) \quad (23.6)$$

$$= \sum_s \Pr(S = s) E[Y|X = x, S = s] \quad (23.7)$$

The inner conditional expectation is just the regression function, for when we try to make a point-prediction of Y from X and S , so now all of the regression methods from Part I come into play. We would, however, still need to know the distribution $\Pr(S)$, so as to average appropriately. Let's turn to this.

23.1.2 Avoiding Estimating Marginal Distributions

We'll continue to focus on estimating the causal effect of X on Y using the back-door criterion, i.e., assuming we've found a set of control variables S such that

$$\Pr(Y = y|do(X = x)) = \sum_s \Pr(Y = y|X = x, S = s) \Pr(S = s) \quad (23.8)$$

S will generally contain multiple variables, so we are committed to estimating two potentially quite high-dimensional distributions, $\Pr(S)$ and $\Pr(Y|X, S)$. Even assuming that we knew all the distributions, just enumerating possible values s and summing over them would be computationally demanding. (Similarly, if S is continuous, we would need to do a high-dimensional integral.) Can we reduce these burdens?

One useful short-cut is to use the law of large numbers, rather than exhaustively enumerating all possible values of s . Notice that the left-hand side fixes y and x ,

so $\Pr(Y = y|X = x, S = s)$ is just some function of s . If we have an IID sample of realizations of S , say s_1, s_2, \dots, s_n , then the law of large numbers says that, for all well-behaved function f ,

$$\frac{1}{n} \sum_{i=1}^n f(s_i) \rightarrow \sum_s f(s) \Pr(S = s) \quad (23.9)$$

Therefore, with a large sample,

$$\Pr(Y = y|do(X = x)) \approx \frac{1}{n} \sum_{i=1}^n \Pr(Y = y|X = x, S = s_i) \quad (23.10)$$

and this will still be (approximately) true when we use a consistent estimate of the conditional probability, rather than its true value.

The same reasoning applies for estimating $E[Y|do(X = x)]$. Moreover, we can use the same reasoning to avoid explicitly summing over all possible s if we *do* have $\Pr(S)$, by simulating from it¹. Even if our sample (or simulation) is not completely IID, but is statistically stationary, in the sense we will cover in Chapter 25 (strictly speaking: “ergodic”), then we can still use this trick.

None of this gets us away from having to estimate $\Pr(Y|X, S)$, which is still going to be a high-dimensional object, if S has many variables.

23.1.3 Propensity Scores

The problems of having to estimate high-dimensional conditional distributions and of averaging over large sets of control values are both reduced if the set of control variables has in fact only a few dimensions. If we have two sets of control variables, S and R , both of which satisfy the back-door criterion for identifying $\Pr(Y|do(X = x))$, all else being equal we should R if it contains fewer variables than S ²

An important special instance of this is when we can set $R = f(S)$, for some function f , and have

$$X \perp\!\!\!\perp S|R \quad (23.11)$$

In the jargon, R is a **sufficient statistic**³ for predicting X from S . To see why this matters, suppose now that we try to identify $\Pr(Y = y|do(X = x))$ from a back-door

¹This is a “Monte Carlo” approximation to the full expectation value.

²Other things which might not be equal: the completeness of data on R and S ; parametric assumptions might be more plausible for the variables in S , giving a better rate of convergence; we might be more confident that S really does satisfy the back-door criterion.

³This is not the same sense of the word “sufficient” as in “causal sufficiency”.

adjustment for R alone, not for S . We have⁴

$$\sum_r \Pr(Y = y | X = x, R = r) \Pr(R = r) \quad (23.12)$$

$$= \sum_{r,s} \Pr(Y = y, S = s | X = x, R = r) \Pr(R = r) \quad (23.13)$$

$$= \sum_{r,s} \Pr(Y = y | X = x, R = r, S = s) \Pr(S = s | X = x, R = r) \Pr(R = r) \quad (23.14)$$

$$= \sum_{r,s} \Pr(Y = y | X = x, S = s) \Pr(S = s | X = x, R = r) \Pr(R = r) \quad (23.15)$$

$$= \sum_s \Pr(Y = y | X = x, S = s) \sum_r \Pr(S = s, R = r) \quad (23.16)$$

$$= \sum_s \Pr(Y = y | X = x, S = s) \Pr(S = s) \quad (23.17)$$

$$= \Pr(Y = y | do(X = x)) \quad (23.18)$$

That is to say, if S satisfies the back-door criterion, then so does R . Since R is a function of S , both the computational and the statistical problems which come from using R are no worse than those of using S , and possibly much better, if R has much lower dimension.

It may seem far-fetched that such a summary score should exist, but really all that's required is that some combinations of the variables in S carry the same information. Consider for instance, the set-up where

$$X \leftarrow \sum_{j=1}^p V_j + \epsilon_X \quad (23.19)$$

$$Y \leftarrow f(X, V_1, V_2, \dots, V_p) + \epsilon_Y \quad (23.20)$$

To identify the effect of X on Y , we need to block the back-door paths between them. Each one of the V_j provides such a back-door path, so we nee to condition on *all* of them. However, if $R = \sum_{j=1}^p V_j$, then $X \perp\!\!\!\perp \{V_1, V_2, \dots, V_p\} | R$, so we could reduce a p -dimensional set of control variables to a one-dimensional set.

Often, as here, finding summary scores will depend on the functional form, and so not be available in the general, non-parametric case. There is, however, an important special case where, if we can use the back-door criterion at all, we can use a one-dimensional summary.

This is the case where X is binary. If we set $f(S) = \Pr(X = 1 | S = s)$, and then take this as our summary R , it is not hard to convince oneself that $X \perp\!\!\!\perp S | R$. This $f(S)$ is called the **propensity score**. It is remarkable, and remarkably convenient, that an arbitrarily large set of control variables S , perhaps with very complicated

⁴Going from Eq. 23.13 to Eq. 23.14 uses the fact that $R = f(S)$, so conditioning on both R and S is the same as just conditioning on S . Going from Eq. 23.14 uses the fact that $S \perp\!\!\!\perp X | R$.

relationships with X and Y , can always be boiled down to a single number between 0 and 1, but there it is.

That said, except in very special circumstances, there is no analytical formula for $f(S)$. This means that it must be modeled and estimated. The most common model seems to be logistic regression, but so far as I can see this is just because many people know no other way to model a binary variable. Since accurate propensity scores are needed to make the method work, it would seem to be worthwhile to model R very carefully, and to consider GAM or fully non-parametric estimates.

23.1.4 Matching and Propensity Scores

Suppose that our causal variable of interest X is binary, or (almost equivalent) that we are only interested in comparing the effect of two levels, $do(X = 1)$ and $do(X = 0)$. Let's call these the "treatment" and "control" groups for definiteness, though nothing really hinges on one of them being in any sense a normal or default value (as "control" suggests) — for instance, we might want to know not just whether men get paid more than women, but whether they are paid more *because* of their sex⁵. In situations like this, we are often not so interested in the full distributions $\Pr(Y|do(X = 1))$ and $\Pr(Y|do(X = 0))$, but just in the expectations, $E[Y|do(X = 1)]$ and $E[Y|do(X = 0)]$. In fact, we are often interested just in the *difference* between these expectations, $E[Y|do(X = 1)] - E[Y|do(X = 0)]$.

Suppose we are the happy possessors of a set of control variables S which satisfy the back-door criterion. How might we use them to estimate this average causal effect?

$$E[Y|do(X = 1)] - E[Y|do(X = 0)] \quad (23.21)$$

$$\begin{aligned} &= \sum_s \Pr(S = s) E[Y|X = 1, S = s] - \sum_s \Pr(S = s) E[Y|X = 0, S = s] \\ &= \sum_s \Pr(S = s) (E[Y|X = 1, S = s] - E[Y|X = 0, S = s]) \end{aligned} \quad (23.22)$$

Clearly, we need to estimate $E[Y|X = 1, S = s] - E[Y|X = 0, S = s]$. The simplest way to do this would be to find all the individuals in the sample with $S = s$, and

⁵The example is both imperfect and controversial. It is imperfect because biological sex (never mind cultural gender) is not *quite* binary, even in mammals, but it's close enough for a good approximation. It is controversial because many statisticians insist that there is no sense in talking about causal effects unless there is some actual manipulation or intervention one could do to change X for an actually-existing "unit" — see, for instance, Holland (1986), which seems to be the source of the slogan "No causation without manipulation". I will just note that (i) this is the kind of metaphysical argument which statisticians usually avoid (if we can't talk about sex or race as causes, because changing those makes the subject a "different person", how about native language? the shape of the nose? hair color? whether they go to college?); (ii) genetic variables are highly manipulable with modern experimental techniques, though we don't use them on people; (iii) real scientists routinely talk about causal effects with no feasible manipulation (e.g., "continental drift causes earthquakes"), or even imaginable manipulation (e.g., "the solar system formed because of gravitational attraction"); and, finally (iv) many of the statisticians who make such pronouncements work or have worked for the Educational Testing Service, and so have a vested interest in being able to testify in court that, strictly speaking, sex and race cannot have any *causal* role in the score anyone gets on the SAT. Whether their views are the cause or the effect of their employment, I am not able to say. (Points (i)–(iii) follow Glymour (1986), but he was too polite to mention (iv).)

compare the mean Y for those who are treated ($X = 1$) to the mean Y for those who are untreated ($X = 0$). This is a sort of a paired comparison, which is called “matching”, because members of the treatment group are being compared to with members of the control group with matching values of the covariates S .

If the number of covariates in S is large, the curse of dimensionality settles upon us. Many values of S will have few or no individuals at all, let alone a large number in both the treatment and the control groups. Even if the real difference $E[Y|X = 1, S = s] - E[Y|X = 0, S = s]$ is small, with only a few individuals in either sub-group we could easily get a large difference in sample means. And of course with continuous covariates in S , each individual will generally have no exact matches at all.

The very clever idea of Rosenbaum and Rubin (1983) is to solve this by matching not on S , but on the propensity score defined in the last section. We have seen already that when X is binary, adjusting for the propensity score is just as good as adjusting for the full set of covariates S . It is easy to double-check (Exercise 1) that

$$\begin{aligned} & \sum_s \Pr(S = s)(E[Y|X = 1, S = s] - E[Y|X = 0, S = s]) \\ &= \sum_r \Pr(R = r)(E[Y|X = 1, R = r] - E[Y|X = 0, R = r]) \end{aligned} \quad (23.23)$$

when $R = \Pr(X = 1|S = s)$, so we lose no essential information by matching on the propensity score R rather than on the covariates S . Intuitively, we now compare each treated individual with one who was just as likely to have received the treatment, but, by chance, did not. On average, the differences between such matched individuals have to be due to the treatment.

What have we gained by doing this? Since R is always a one-dimensional variable, no matter how big S is, it is going to be *much* easier to find matches on R than on S — the curse of dimensionality has been broken⁶. This is a *tremendous* advantage, which makes matching actually feasible.

It is important to be clear, however, that the gain here is in computational tractability and statistical efficiency, not in fundamental identification. With $R = \Pr(X = 1|S = s)$, it will always be true that $X \perp\!\!\!\perp S|R$, whether or not the back-door criterion is satisfied. If the criterion is satisfied, in principle there is nothing stopping us from using matching on S to estimate the effect, except our own impatience. If the criterion is not satisfied, having a compact one-dimensional summary of the wrong set of control variables is just going to let us get the wrong answer faster.

Some confusion seems to have arisen on this point, because, conditional on the propensity score, the treated group and the control group have the same distribution of covariates. (Again, recall that $X \perp\!\!\!\perp S|R$.) Since treatment and control groups have the same distribution of covariates in a randomized experiment, some people have concluded that propensity score matching is just as good as randomization⁷. That this is emphatically *not* the case.

⁶If no exact match is available, we might match to within some distance, or do some sort of kernel-weighted matching. (It's not a good idea to use these directly on S , because they become very inefficient in high dimensions.) See, e.g., Stuart (2010) for details.

⁷These people do not include Rubin and Rosenbaum, but it is easy to see how their readers could come away with this impression. See Pearl (2009b, §11.3.5), and especially Pearl (2009a).

The propensity score matching method has become incredibly popular since Rosenbaum and Rubin (1983), and there are a huge number of implementations of various versions of it. The `MatchIt` package in R is one of the most common, but see Stuart (2010) for a fairly recent listing of relevant software in R and other languages.

23.2 Instrumental-Variables Estimates

§22.3.3 introduced the idea of using instrumental variables to identify causal effects. Roughly speaking, I is an instrument for identifying the effect of X on Y when I is a cause of X , but the only way I is associated with Y is through directed paths which go through X . To the extent that variation in I predicts variation in X and Y , this can only be because X has a causal influence on Y . More precisely, given some controls S , I is a valid instrument when $I \not\perp\!\!\!\perp X|S$, and every path from I to Y left open by S has an arrow into X .

In the simplest case, of Figure 22.7, we saw that when everything is linear, we can find the causal coefficient of Y on X as

$$\beta = \frac{\text{Cov}[I, Y]}{\text{Cov}[I, X]} \quad (23.24)$$

A one-unit change in I causes (on average) an α -unit change in X , and an $\alpha\beta$ -unit change in Y , so β is, as it were, the gearing ratio or leverage of the mechanism connecting I to Y .

Estimating β by plugging in the sample values of the covariances into Eq. 23.24 is called the **Wald estimator** of β . In more complex situations, we might have multiple instruments, and be interested in the causal effects of multiple variables, and we might have to control for some covariates to block undesired paths and get valid instruments. In such situations, the Wald estimator breaks down.

There is however a more general procedure which still works, provided the linearity assumption holds. This is called **two-stage regression**, or **two-stage least squares** (2SLS).

1. Regress X on I and S . Call the fitted values \hat{x} .
2. Regress Y on \hat{x} and S , but *not* on I . The coefficient of Y on \hat{x} is a consistent estimate of β .

The logic is very much as in the Wald estimator: conditional on S , variations in I are independent of the rest of the system. The only way they can affect Y is through their effect on X . In the first stage, then, we see how much changes in the instruments affect X . In the second stage, we see how much these I -caused changes in X change Y ; and this gives us what we want.

To actually prove that this works, we would need to go through some heroic linear algebra to show that the population version of the two-stage estimator is actually equal to β , and then a straight-forward argument that plugging in the appropriate sample covariance matrices is consistent. The details can be found in any econometrics textbook, so I'll skip them. (But see Exercise 3.)

As mentioned in §23.2, there are circumstances where it is possible to use instrumental variables in nonlinear and even nonparametric models. The technique becomes far more complicated, however, because finding $\Pr(Y = y|do(X = x))$ requires solving Eq. 22.15,

$$\Pr(Y|do(I = i)) = \sum_x \Pr(Y|do(X = x))\Pr(X = x|do(I = i))$$

and likewise finding $E[Y|do(X = x)]$ means solving

$$E[Y|do(I = i)] = \sum_x E[Y|do(X = x)]\Pr(X = x|do(I = i)) \quad (23.25)$$

When, as is generally the case, x is continuous, we have rather an integral equation,

$$E[Y|do(I = i)] = \int E[Y|do(X = x)] p(x|do(I = i)) dx \quad (23.26)$$

Solving such integral equations is not (in general) impossible, but it is hard, and the techniques needed are much more complicated than even two-stage least squares. I will not go over them here, but see Li and Racine (2007, chs. 16–17).

23.3 Uncertainty and Inference

The point of the identification strategies from Chapter 22 is to reduce the problem of causal inference to that of ordinary statistical inference. Having done so, we can assess our uncertainty about any of our estimates of causal effects the same way we would assess any other statistical inference. If we want confidence intervals or standard errors for $E[Y|do(X = 1)] - E[Y|do(X = 0)]$, for instance, we can treat our estimate of this like any other point estimate, and proceed accordingly. In particular, we can use the bootstrap (Chapter 6), if analytical formulas are not available or unappealing.

The one wrinkle to the use of analytical formulas comes from two-stage least-squares. Taking standard errors, confidence intervals, etc., for β from the usual formulas for the second regression neglects the fact that this estimate of β comes from regressing Y on \hat{x} , which is itself an estimate and so uncertain.

23.4 Recommendations

Instrumental variables are a very clever idea, but they need to be treated with caution. They only work if the instruments are valid, and that validity is rests just as much on assumptions about the underlying DAG as any of the other identification strategies. The crucial point, after all, is that the instrument is an indirect cause of Y , but *only* through X , with no other (unblocked) paths connecting I to Y . This can only too easily fail, if some indirect path has been neglected.

Matching, especially propensity score matching, is just as ingenious, and just as much at the mercy of the correctness of the DAG. Whether we match directly on

covariates, or indirectly through the propensity score, what matters is whether the covariates really block off the back-door pathways between X and Y . If they do, well and good. If they do not, then ingenuity is not going to help you.

There is a curious divide, among practitioners, between those who lean mostly on instrumental variables, and those who lean mostly on matching. The former tend to suspect that (in our terms) the covariates used in matching are not enough to block all the back-door paths⁸, and to think that the business is more or less over once an exogenous variable has been found. The matchers, for their part, think the instrumentalists are too quick to discount the possibility that their instruments are connected to Y through unmeasured pathways⁹, but that if you match on enough variables, you've got to block the back-door paths. (They don't often worry that they might be conditioning on colliders in doing so.) As is often the case in the sciences, there is much truth to each faction's criticism of the other side. You are now in a position to think more clearly, and act more intelligently, in these matters than many practitioners.

Throughout these chapters, we have been assuming that we know the correct DAG. Without such assumptions, or ones equivalent to them, none of these ideas can be used. In the next chapter, then, we will look at how to actually begin *discovering* causal structure from data.

23.5 Further Reading

The material in §23.1 is largely “folklore”, though see Morgan and Winship (2007).

Rubin (2006) collects Rubin's major papers on matching, including propensity score matching. Rubin and Waterman (2006) is an extremely clear easy-to-follow introduction to propensity score matching as a method of causal inference.

⁸As an example for their side, Arceneaux *et al.* (2010) applied matching methods to an actual experiment, where the real causal relations could be worked out straightforwardly for comparison. Well-conducted propensity-score “matching suggests that [a] pre-election phone call that encouraged people to wear their seat belts also generated huge increases in voter turnout”. Their paper provides a convincing explanation of where this illusory effect comes from, i.e., of what the unblocked back-door path is, which I will not spoil for you.

⁹For instance, a recent and widely-promoted preprint by three economists argued that watching television caused autism in children. (I leave tracking down the paper as an exercise for the reader.) The economists used the variation in how much it rains across different locations in California, Oregon and Washington as an instrument to predict average TV-watching (X) and its affects on the prevalence of autism (Y). It is certainly plausible that kids watch more TV when it rains, and that neither TV-watching nor autism causes rain. But this leaves open the question of whether rain and the prevalence of autism might not have some common cause, and for the West Coast in particular it is easy to find one. It is well-established that the risk of autism is higher among children of older parents, and that more-educated people tend to have children later in life. All three states have, of course, a striking contrast between large, rainy cities full of educated people (San Francisco, Portland, Seattle), and very dry, very rural locations on the other side of the mountains. Thus there is a (potential) uncontrolled common cause of rain and autism, namely geographic location, and the situation is as in Figure 22.9. — For a rather more convincing effort to apply ideas about causal inference to understanding the changing prevalence of autism, see Liu *et al.* (2010).

23.6 Exercises

1. Prove Eq. 23.23.
2. Suppose that X has three levels, say 0, 1, 2. Let R be the vector $(\Pr(X = 0|S = s), \Pr(X = 1|S = s))$. Prove that $X \perp\!\!\!\perp S|R$. (This is how to generalize propensity scores to non-binary X .)
3. For the situation in Figure 22.7, prove that the two-stage least-squares estimate of β is the same as the Wald estimate.

Chapter 24

Discovering Causal Structure from Observations

The last few chapters have, hopefully, convinced you that when you want to do causal inference, knowing the causal graph is very helpful. We have looked at how it would let us calculate the effects of actual or hypothetical manipulations of the variables in the system. Furthermore, knowing the graph tells us about what causal effects we can and cannot identify, and estimate, from observational data. But everything has posited that we know the graph somehow. This chapter finally deals with where the graph comes from.

There are fundamentally three ways to get the DAG:

- Prior knowledge
- Guessing-and-testing
- Discovery algorithms

There is only a little to say about the first, because, while it's important, it's not very statistical. As functioning adult human beings, you have a lot of everyday causal knowledge, which does not disappear the moment you start doing data analysis. Moreover, you are the inheritor of a vast scientific tradition which has, through patient observation and toilsome experiments, acquired even more causal knowledge. You can and should use this. Someone's sex or race or caste might be causes of the job they get or their pay, but not the other way around. Running an electric current through a wire produces heat at a rate proportional to the square of the current. Malaria is due to a parasite transmitted by mosquitoes, and spraying mosquitoes with insecticides makes the survivors more resistant to those chemicals. All of these sorts of ideas can be expressed graphically, or at least as constraints on graphs.

We can, and should, also use graphs to represent scientific ideas which are not as secure as Ohm's law or the epidemiology of malaria. The ideas people work with in areas like psychology or economics, are really quite tentative, but they are ideas

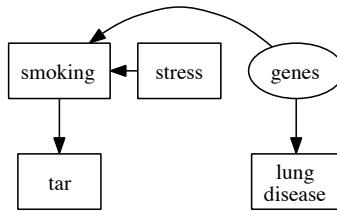


Figure 24.1: A hypothetical causal model in which smoking is associated with lung disease, but does not cause it. Rather, both smoking and lung disease are caused by common genetic variants. (This idea was due to R. A. Fisher.) Smoking is also caused, in this model, by stress.

about the causal structure of parts of the world, and so graphical models are implicit in them.

All of which said, even if we think we know very well what's going on, we will often still want to check it, and that brings us the guess-and-test route.

24.1 Testing DAGs

A graphical causal model makes two kinds of qualitative claims. One is about direct causation. If the model says X is a parent of Y , then it says that changing X will change the (distribution of) Y . If we experiment on X (alone), moving it back and forth, and yet Y is unaltered, we know the model is wrong and can throw it out.

The other kind of claim a DAG model makes is about probabilistic conditional independence. If S d-separates X from Y , then $X \perp\!\!\!\perp Y | S$. If we observed X , Y and S , and see that $X \not\perp\!\!\!\perp Y | S$, then we know the model is wrong and can throw it out. (More: we know that there is a path linking X and Y which isn't blocked by S .) Thus in the model of Figure 24.1, $\text{lungdisease} \perp\!\!\!\perp \text{tar} | \text{smoking}$. If lung disease and tar turn out to be dependent when conditioning on smoking, the model must be wrong.

This then is the basis for the guess-and-test approach to getting the DAG:

- Start with an initial guess about the DAG.
- Deduce conditional independence relations from d-separation.
- Test these, and reject the DAG if variables which ought to be conditionally independent turn out to be dependent.

This is a distillation of primary-school scientific method: formulate a hypotheses (the DAG), work out what the hypothesis implies, test those predictions, reject hypotheses which make wrong predictions.

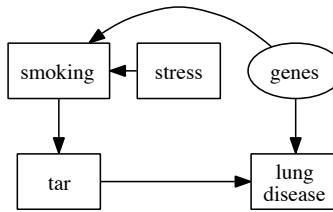


Figure 24.2: As in Figure 24.1, but now tar in the lungs does cause lung disease.

It may happen that there are only a few competing, scientifically-plausible models, and so only a few, competing DAGs. Then it is usually a good idea to focus on checking predictions which *differ* between them. So in both Figure 24.1 and in Figure 24.2, $\text{stress} \perp\!\!\!\perp \text{tar} | \text{smoking}$. Checking that independence thus does nothing to help us distinguish between the two graphs. In particular, confirming that stress and tar are independent given smoking really doesn't give us evidence *for* the model from Figure 24.1, since it equally follows from the other model. If we want such evidence, we have to look for something they *disagree* about.

In any case, testing a DAG means testing conditional independence, so let's turn to that next.

24.2 Testing Conditional Independence

Recall from §21.4 that conditional independence is equivalent to zero conditional information: $X \perp\!\!\!\perp Y | Z$ if and only if $I[X; Y | Z] = 0$. In principle, this solves the problem. In practice, estimating mutual information is non-trivial, and in particular the sample mutual information often has a very complicated distribution. You *could* always bootstrap it, but often something more tractable is desirable. Completely general conditional independence testing is actually an active area of research. Some of this work is still quite mathematical (Sriperumbudur *et al.*, 2010), but it has already led to practical tests (Székely and Rizzo, 2009; Gretton *et al.*, 2012; Zhang *et al.*, 2011) and no doubt more are coming soon.

If all the variables are discrete, one just has a big contingency table problem, and could use a G^2 or χ^2 test. If everything is linear and multivariate Gaussian, $X \perp\!\!\!\perp Y | Z$ is equivalent to zero partial correlation¹. Nonlinearly, if $X \perp\!\!\!\perp Y | Z$, then $E[Y | Z] = E[Y | X, Z]$, so if smoothing Y on X and Z leads to different predictions than just smoothing on Z , conditional independence fails. To reverse this, and go from $E[Y | Z] = E[Y | X, Z]$ to $X \perp\!\!\!\perp Y | Z$, requires the extra assumption that Y doesn't

¹Recall that the partial correlation between X and Y given Z is the correlation between X and Y , after linearly regressing each of them on Z separately. That is, it is the correlation of their residuals.

depend on X through its variance or any other moment. (This is weaker than the linear-and-Gaussian assumption, of course.)

The conditional independence relation $X \perp\!\!\!\perp Y|Z$ is fully equivalent to $\Pr(Y|X, Z) = \Pr(Y|Z)$. We could check this using non-parametric density estimation, though we would have to bootstrap the distribution of the test statistic. A more automatic, if slightly less rigorous, procedure comes from the idea mentioned in Chapter 15: If X is in fact useless for predicting Y given Z , then an adaptive bandwidth selection procedure (like cross-validation) should realize that giving any finite bandwidth to X just leads to over-fitting. The bandwidth given to X should tend to the maximum allowed, smoothing X away altogether. This argument can be made more formal, and made into the basis of a test (Hall *et al.*, 2004; Li and Racine, 2007).

24.3 Faithfulness and Equivalence

In graphical models, d-separation implies conditional independence: if S blocks all paths from U to V , then $U \perp\!\!\!\perp V|S$. To reverse this, and conclude that if $U \perp\!\!\!\perp V|S$ then S must d-separate U and V , we need an additional assumption, already referred to in §21.2, called **faithfulness**. More exactly, if the distribution is faithful to the graph, then if S does not d-separate U from V , $U \not\perp\!\!\!\perp V|S$. The combination of faithfulness and the Markov property means that $U \perp\!\!\!\perp V|S$ if and only if S d-separates U and V .

This seems extremely promising. We can test whether $U \perp\!\!\!\perp V|S$ for any sets of variables we like. We could in particular test whether each pair of variables is independent, given all sorts of conditioning variable sets S . If we assume faithfulness, when we find that $X \perp\!\!\!\perp Y|S$, we know that S blocks all paths linking X and Y , so we learn something about the graph. If $X \not\perp\!\!\!\perp Y|S$ for all S , we would seem to have little choice but to conclude that X and Y are directly connected. Might it not be possible to reconstruct or discover the right DAG from knowing all the conditional independence and dependence relations?

This is on the right track, but too hasty. Start with just two variables:

$$X \rightarrow Y \Rightarrow X \not\perp\!\!\!\perp Y \quad (24.1)$$

$$X \leftarrow Y \Rightarrow X \not\perp\!\!\!\perp Y \quad (24.2)$$

With only two variables, there is only one independence (or dependence) relation to worry about, and it's the same no matter which way the arrow points.

Similarly, consider these arrangements of three variables:

$$X \rightarrow Y \rightarrow Z \quad (24.3)$$

$$X \leftarrow Y \leftarrow Z \quad (24.4)$$

$$X \leftarrow Y \rightarrow Z \quad (24.5)$$

$$X \rightarrow Y \leftarrow Z \quad (24.6)$$

The first two are chains, the third is a fork, the last is a collider. It is not hard to check (Exercise 1) that the first three DAGs all imply exactly the same set of conditional

independence relations, which are different from those implied by the fourth².

These examples illustrate a general problem. There may be multiple graphs which imply the same independence relations, even when we assume faithfulness. When this happens, the exact same distribution of observables can factor according to, and be faithful to, all of those graphs. The graphs are thus said to be **equivalent**, or **Markov equivalent**. Observational alone cannot distinguish between equivalent DAGs. Experiment can, of course — changing Y alters both X and Z in a fork, but not a chain — which shows that there really is a difference between the DAGs, just not one *observational* data can track.

24.3.1 Partial Identification of Effects

Chapters 22–23 considered the identification and estimation of causal effects under the assumption that there was a single known graph. If there are multiple equivalent DAGs, then, as mentioned above, no amount of purely observational data can select a single graph. Background knowledge lets us rule out some equivalent DAGs³, but it may not narrow the set of possibilities to a single graph. How then are we to actually do our causal estimation?

We *could* just pick one of the equivalent graphs, and do all of our calculations as though it were the only possible graph. This is often what people seem to do. The kindest thing one can say about it is that it shows confidence; phrases like “lying by omission” also come to mind.

A more principled alternative is to admit that the uncertainty about the DAG means that causal effects are only *partially* identified. Simply put, one does the estimation in each of the equivalent graphs, and reports the range of results⁴. If each estimate is consistent, then this gives a consistent estimate of the range of possible effects. Because the effects are not fully identified, this range will not narrow to a single point, even in the limit of infinite data, but admitting this, rather than claiming a non-existent precision, is simple scientific honesty.

24.4 Causal Discovery with Known Variables

Section 24.1 talks about how we can test a DAG, once we have it. This lets us eliminate some DAGs, but still leaves mysterious where they come from in the first place. While in principle there is nothing wrong which deriving your DAG from a vision of serpents biting each others’ tails, so long as you test it, it would be nice to have a systematic way of finding good models. This is the problem of model discovery, and especially of causal discovery.

²In all of the first three, $X \not\perp\!\!\!\perp Z$ but $X \perp\!\!\!\perp Z|Y$, while in the collider, $X \perp\!\!\!\perp Z$ but $X \not\perp\!\!\!\perp Z|Y$. Remarkably enough, the work which introduced the notion of forks and colliders, Reichenbach (1956), missed this — he thought that $X \perp\!\!\!\perp Z|Y$ in a collider as well as a fork. Arguably, this one mistake delayed the development of causal inference by thirty years or more.

³If we know that X , Y and Z have to be in either a chain or a fork, with Y in the middle, and we know that X comes before Y in time, then we can rule out the fork and the chain $X \leftarrow Y \rightarrow Z$.

⁴Sometimes the different graphs will give the same estimates of certain effects. For example, the chain $X \rightarrow Y \rightarrow Z$ and the fork $X \leftarrow Y \rightarrow Z$ will agree on the effect of Y on Z .

Causal discovery is silly with just one variable, and too hard for us with just two.⁵

With three or more variables, we have however a very basic principle. If there is no edge between X and Y , in either direction, then X is neither Y 's parent nor its child. But any variable is independent of its non-descendants given its parents. Thus, for some set⁶ of variables S , $X \perp\!\!\!\perp Y|S$ (Exercise 2). If we assume faithfulness, then the converse holds: if $X \perp\!\!\!\perp Y|S$, then there cannot be an edge between X and Y . Thus, there is no edge between X and Y if and only if we can make X and Y independent by conditioning on some S . Said another way, there is an edge between X and Y if and only if we cannot make the dependence between them go away, no matter what we condition on⁷.

So let's start with three variables, X , Y and Z . By testing for independence and conditional independence, we could learn that there had to be edges between X and Y and Y and Z , but not between X and Z . But conditional independence is a symmetric relationship, so how could we **orient** those edges, give them direction? Well, to rehearse a point from the last section, there are only four possible directed graphs corresponding to that undirected graph:

- $X \rightarrow Y \rightarrow Z$ (a chain);
- $X \leftarrow Y \leftarrow Z$ (the other chain);
- $X \leftarrow Y \rightarrow Z$ (a fork on Y);
- $X \rightarrow Y \leftarrow Z$ (a collision at Y)

With the fork or either chain, we have $X \perp\!\!\!\perp Z|Y$. On the other hand, with the collider we have $X \not\perp\!\!\!\perp Z|Y$. Thus $X \not\perp\!\!\!\perp Z|Y$ if and only if there is a collision at Y . By testing for *this* conditional dependence, we can either definitely orient the edges, or rule out an orientation. If $X - Y - Z$ is just a subgraph of a larger graph, we can still identify it as a collider if $X \not\perp\!\!\!\perp Z|\{Y, S\}$ for *all* collections of nodes S (not including X and Z themselves, of course).

With more nodes and edges, we can **induce** more orientations of edges by consistency with orientations we get by identifying colliders. For example, suppose we know that X, Y, Z is either a chain or a fork on Y . If we learn that $X \rightarrow Y$, then the triple *cannot* be a fork, and must be the chain $X \rightarrow Y \rightarrow Z$. So orienting the $X - Y$ edge induces an orientation of the $Y - Z$ edge. We can also sometimes orient edges through background knowledge; for instance we might know that Y comes later in time than X , so if there is an edge between them it *cannot* run from Y to X .⁸ We can

⁵But see Janzing (2007); Hoyer *et al.* (2009) for some ideas on how you could do it if you're willing to make some extra assumptions. The basic idea of these papers is that the distribution of effects given causes should be simpler, in some sense, than the distribution of causes given effects.

⁶Possibly empty: conditioning on the empty set of variables is the same as not conditioning at all.

⁷"No causation without association", as it were.

⁸Some have argued, or at least entertained the idea, that the logic here is backwards: rather than order in time constraining causal relations, causal order *defines* time order. (Versions of this idea are discussed by, inter alia, Russell (1927); Wiener (1961); Reichenbach (1956); Pearl (2009b); Janzing (2007) makes a related suggestion). Arguably then using order in time to orient edges in a causal graph begs the question, or commits the fallacy of *petitio principii*. But of course every syllogism does, so this isn't a distinctively *statistical* issue. (Take the classic: "All men are mortal; Socrates is a man; therefore Socrates is mortal.")

eliminate other edges based on similar sorts of background knowledge: men tend to be heavier than women, but changing weight does not change sex, so there can't be an edge (or even a directed path!) from weight to sex, though there could be one the other way around.

To sum up, we can rule out an edge between X and Y whenever we can make them independent by conditioning on other variables; and when we have an $X - Y - Z$ pattern, we can identify colliders by testing whether X and Z are dependent given Y . Having oriented the arrows going into colliders, we induce more orientations of other edges.

Putting these three things — edge elimination by testing, collider finding, and inducing orientations — gives the most basic causal discovery procedure, the SGS (Spirtes-Glymour-Scheines) algorithm (Spirtes *et al.*, 2001, §5.4.1, p. 82). This assumes:

1. The data-generating distribution has the causal Markov property on a graph G .
2. The data-generating distribution is faithful to G .
3. Every member of the population has the same distribution.
4. All relevant variables are in G .
5. There is only *one* graph G to which the distribution is faithful.

Abstractly, the algorithm works as follows:

- Start with a complete undirected graph on all p variables, with edges between all nodes.
- For each pair of variables X and Y , and each set of other variables S , see if $X \perp\!\!\!\perp Y | S$; if so, remove the edge between X and Y .
- Find colliders by checking for conditional dependence; orient the edges of colliders.
- Try to orient undirected edges by consistency with already-oriented edges; do this recursively until no more edges can be oriented.

Pseudo-code is in Appendix G.

Call the result of the SGS algorithm \widehat{G} . If all of the assumptions above hold, and the algorithm is correct in its guesses about when variables are conditionally independent, then $\widehat{G} = G$. In practice, of course, conditional independence guesses are really statistical tests based on finite data, so we should write the output as \widehat{G}_n , to indicate that it is based on only n samples. If the conditional independence test is consistent, then

$$\lim_{n \rightarrow \infty} \Pr(\widehat{G}_n \neq G) = 0 \quad (24.7)$$

How can we know that *all* men are mortal until we know about the mortality of this particular man, Socrates? Isn't this just like asserting that tomatoes and peppers must be poisonous, because they belong to the nightshade family of plants, all of which are poisonous?) While these philosophical issues are genuinely fascinating, this footnote has gone on long enough, and it is time to return to the main text.

In other words, the SGS algorithm converges in probability on the correct causal structure; it is consistent for all graphs G . Of course, at finite n , the probability of error — of having the wrong structure — is (generally!) not zero, but this just means that, like any statistical procedure, we cannot be absolutely certain that it's not making a mistake.

One consequence of the independence tests making errors on finite data can be that we fail to orient some edges — perhaps we missed some colliders. These unoriented edges in \hat{G}_n can be thought of as something like a confidence region — they have *some* orientation, but multiple orientations are all compatible with the data.⁹ As more and more edges get oriented, the confidence region shrinks.

If the fifth assumption above fails to hold, then there are multiple graphs G to which the distribution is faithful. This is just a more complicated version of the difficulty of distinguishing between the graphs $X \rightarrow Y$ and $X \leftarrow Y$. All the graphs in the equivalence class may have some arrows in common; in that case the SGS algorithm will identify those arrows. If some edges differ in orientation across the equivalence class, SGS will not orient them, even in the limit. In terms of the previous paragraph, the confidence region never shrinks to a single point, just because the data doesn't provide the information needed to do this. The graph is only partially identified.

If there *are* unmeasured relevant variables, we can get not just unoriented edges, but actually arrows pointing in both directions. This is an excellent sign that some basic assumption is being violated.

24.4.1 The PC Algorithm

The SGS algorithm is statistically consistent, but very computationally inefficient; the number of tests it does grows exponentially in the number of variables p . This is the worst-case complexity for *any* consistent causal-discovery procedure, but this algorithm just proceeds immediately to the worst case, not taking advantage of any possible short-cuts.

Since it's enough to find *one* S making X and Y independent to remove their edge, one obvious short-cut is to do the tests in some order, and skip unnecessary tests. On the principle of doing the easy work first, the revised edge-removal step would look something like this:

- For each X and Y , see if $X \perp\!\!\!\perp Y$; if so, remove their edge.
- For each X and Y which are still connected, and each third variable Z , see if $X \perp\!\!\!\perp Y|Z$; if so, remove the edge between X and Y .
- For each X and Y which are still connected, and each third and fourth variables Z_1 and Z_2 , see if $X \perp\!\!\!\perp Y|Z_1, Z_2$; if so, remove their edge.
- ...

⁹I say “multiple orientations” rather than “all orientations”, because picking a direction for one edge might induce an orientation for others.

- For each X and Y which are still connected, see if $X \perp\!\!\!\perp Y | S$ all the $p - 2$ other variables; if so, remove their edge.

If all the tests are done correctly, this will give the same result as the SGS procedure (Exercise 3). And if some of the tests give erroneous results, conditioning on a small number of variables will tend to be more reliable than conditioning on more (why?).

We can be even more efficient, however. If $X \perp\!\!\!\perp Y | S$ for any S at all, then $X \perp\!\!\!\perp Y | S'$, where all the variables in S' are adjacent to X or Y (or both) (Exercise 4). To see the sense of this, suppose that there is a single long directed path running from X to Y . If we condition on any of the variables along the chain, we make X and Y independent, but we could always move the point where we block the chain to be either right next to X or right next to Y . So when we are trying to remove edges and make X and Y independent, we only need to condition on variables which are still connected to X and Y , not ones in totally different parts of the graph.

This then gives us the PC¹⁰ algorithm (Spirtes *et al.* 2001, §5.4.2, pp. 84–88; see also Appendix G). It works exactly like the SGS algorithm, except for the edge-removal step, where it tries to condition on as few variables as possible (as above), and only conditions on adjacent variables. The PC algorithm has the same assumptions as the SGS algorithm, and the same consistency properties, but generally runs much faster, and does many fewer statistical tests. It should be the default algorithm for attempting causal discovery.

24.4.2 Causal Discovery with Hidden Variables

Suppose that the set of variables we measure is *not* causally sufficient. Could we at least discover this? Could we possibly get hold of *some* of the causal relationships? Algorithms which can do this exist (e.g., the CI and FCI algorithms of Spirtes *et al.* (2001, ch. 6)), but they require considerably more graph-fu. (The RFCI algorithm (Colombo *et al.*, 2012) is a modern, fast successor to FCI.) The results of these algorithms can succeed in removing *some* edges between observable variables, and definitely orienting some of the remaining edges. If there are actually no latent common causes, they end up acting like the SGS or PC algorithms.

24.4.2.0.1 Partial identification of effects When all relevant variables are observed, all effects are identified within one graph; partial identification happens because multiple graphs are equivalent. When some variables are not observed, we may have to use the identification strategies to get at the same effect. In fact, the same effect may be identified in one graph and not identified in another, equivalent graph. This is, again, unfortunate, but when it happens it needs to be admitted.

24.4.3 On Conditional Independence Tests

The abstract algorithms for causal discovery assume the existence of consistent tests for conditional independence. The implementations known to me mostly assume either that variables are discrete (so that one can basically use the χ^2 test), or that

¹⁰Peter-Clark

they are continuous, Gaussian, and linearly related (so that one can test for vanishing partial correlations), though the `pca` package does allow users to provide their own conditional independence tests as arguments. It bears emphasizing that these restrictions are *not* essential. As soon as you have a consistent independence test, you are, in principle, in business. In particular, consistent *non-parametric* tests of conditional independence would work perfectly well. An interesting example of this is the paper by Chu and Glymour (2008), on finding causal models for the time series, assuming additive but non-linear models.

24.5 Software and Examples

The PC and FCI algorithms are implemented in the stand-alone Java program Tetrad (<http://www.phil.cmu.edu/projects/tetrad/>). They are also implemented in the `pca` package on CRAN (Kalisch *et al.*, 2010, 2012). This package also includes functions for calculating the effects of interventions from fitted graphs, assuming linear models. The documentation for the functions is somewhat confusing; rather see Kalisch *et al.* (2012) for a tutorial introduction.

It's worth going through how `pca` works¹¹. The code is designed to take advantage of the modularity and abstraction of the PC algorithm itself; it separates actually finding the graph completely from performing the conditional independence test, which is rather a function the user supplies. (Some common ones are built in.) For reasons of computational efficiency, in turn, the conditional independence tests are set up so that the user can just supply a set of sufficient statistics, rather than the raw data.

Let's walk through an example¹², using the `mathmarks` data set you saw in the second exam. There we had grades ("marks") from 88 students in five mathematical subjects, algebra, analysis, mechanics, statistics and vectors. All five variables are positively correlated with each other.

```
library(pca)
library(SMPRACTICALS)
data(mathmarks)
suffStat <- list(C=cor(mathmarks), n=nrow(mathmarks))
pc.fit <- pc(suffStat, indepTest=gaussCITest, p=ncol(mathmarks), alpha=0.005)
```

This uses a Gaussian (-and-linear) test for conditional independence, `gaussCITest`, which is built into the `pca` package. Basically, it tests whether $X \perp\!\!\!\perp Y | Z$ by testing whether the partial correlation of X and Y given Z is close to zero. These partial correlations can all be calculated from the correlation matrix, so the line before creates the sufficient statistics needed by `gaussCITest` — the matrix of correlations and

¹¹A word about installing the package: you'll need the package `Rgraphviz` for drawing graphs, which is hosted not on CRAN (like `pca`) but on BioConductor. Try installing it, and its dependencies, before installing `pca`. See <http://www.bioconductor.org/packages/2.10/bioc/readmes/Rgraphviz/README> for help on installing `Rgraphviz`.

¹²After Spirtes *et al.* (2001, §6.12, pp. 152–154).

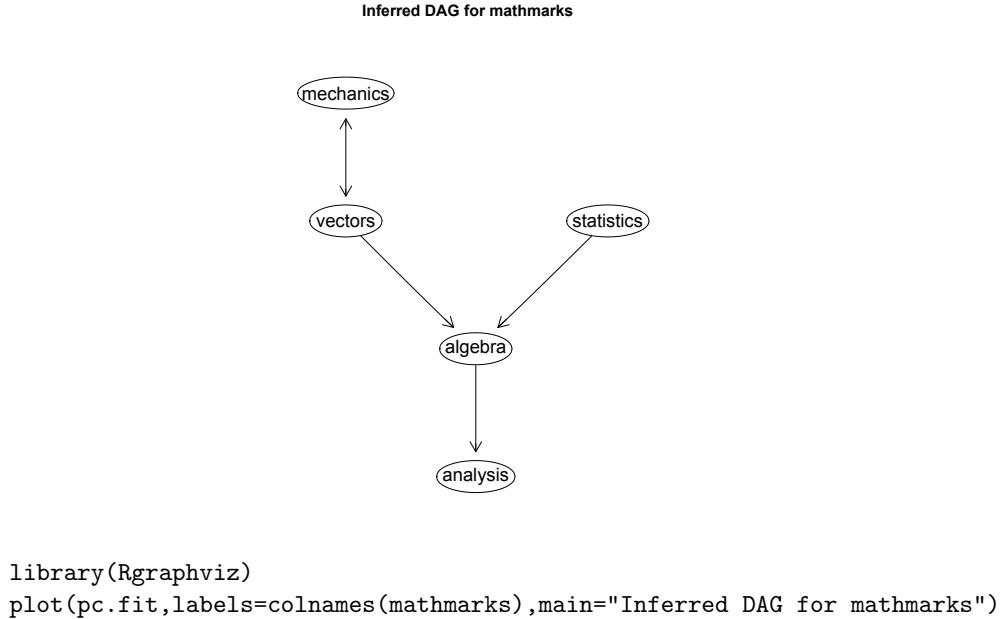


Figure 24.3: DAG inferred by the PC algorithm from the `mathmarks` data. Two-headed arrows, like undirected edges, indicate that the algorithm was unable to orient the edge. (It is obscure why `pcalg` sometimes gives an edge it cannot orient no heads and sometimes two.)

the number of data points. We also have to tell `pc` how many variables there are, and what significance level to use in the test (here, 0.5%).

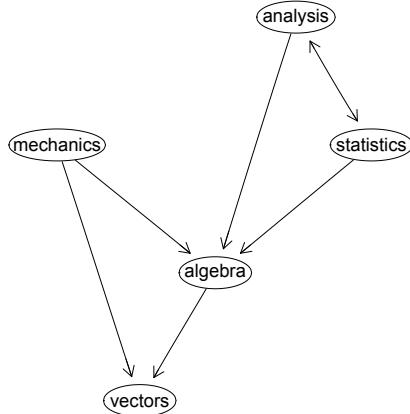
Before going on, I encourage you to run `pc` as above, but with `verbose=TRUE`, and to study the output.

Figure 24.3 shows what it looks like. If we take it seriously, it says that grades in analysis are driven by grades in algebra, while algebra in turn is driven by statistics and vectors. While one could make up stories for why this would be so (perhaps something about the curriculum?), it seems safer to regard this as a warning against *blindly* trusting any algorithm — a key assumption of the PC algorithm, after all, is that there are no unmeasured but causally-relevant variables, and it is easy to believe these are violated. For instance, while *knowledge* of different mathematical fields may be causally linked (it would indeed be hard to learn much mechanics without knowing about vectors), test scores are only imperfect measurements of knowledge.

The size of the test may seem low, but remember we are doing a lot of tests:

```

> summary(pc.fit)
Object of class 'pcAlgo', from Call:
skeleton(suffStat = suffStat, indepTest = indepTest, p = p, alpha = alpha,
verbose = verbose, fixedGaps = fixedGaps, fixedEdges = fixedEdges,
  
```



```
plot(pc(suffStat, indepTest=gaussCItest, p=ncol(mathmarks), alpha=0.05),
     labels=colnames(mathmarks), main="")
```

Figure 24.4: Inferred DAG when the size of the test is 0.05.

```

NAdelete = NAdelete, m.max = m.max)

Nmb. edgetests during skeleton estimation:
=====
Max. order of algorithm: 3
Number of edgetests from m = 0 up to m = 3 :  20 31 4 0

Graphical properties of skeleton:
=====
Max. number of neighbours: 2 at node(s) 2
Avg. number of neighbours: 1
  
```

This tells us that it considered going up to conditioning on three variables (the maximum possible, since there are only five variables), that it did twenty tests of unconditional independence, 31 tests where it conditioned on one variable, four tests where it conditioned on two, and none where it conditioned on three. This 55 tests in all, so a simple Bonferroni correction suggests the over-all size is $55 \times 0.005 = 0.275$. This is probably pessimistic (the Bonferroni correction typically is). Setting $\alpha = 0.05$ gives a somewhat different graph (Figure 24.4).

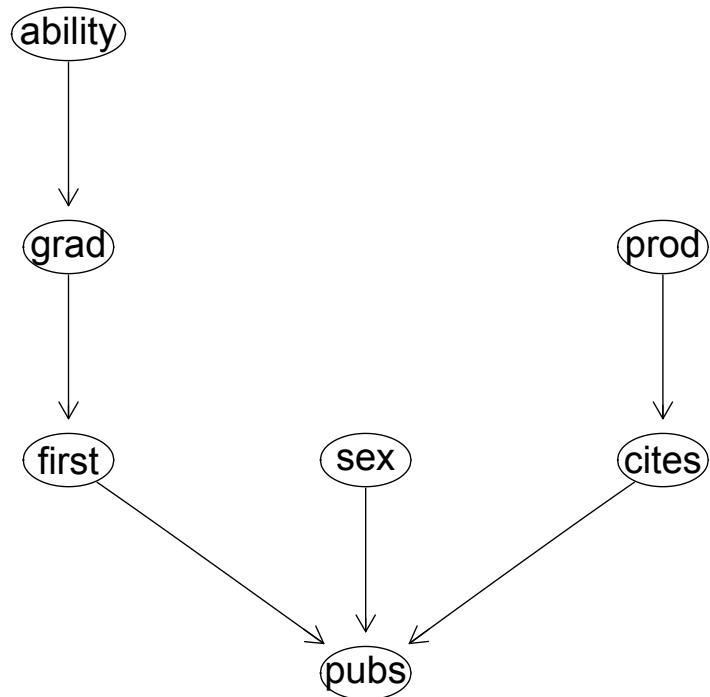
For a second example¹³, let's use some data on academic productivity among psy-

¹³Following Spirtes *et al.* (2001, §5.8.1, pp. 98–102).

chologists. The two variables of ultimate interest were the publication (`pubs`) and citation (`cites`) rates, with possible measured causes including ability (basically, standardized test scores), graduate program quality `grad` (basically, the program's national rank), the quality of the psychologist's first job, `first`, a measure of productivity `prod`, and sex. There were 162 subjects, and while the actual data isn't reported, the correlation matrix is.

```
> rm
      ability grad prod first   sex cites pubs
ability     1.00 0.62 0.25  0.16 -0.10  0.29 0.18
grad        0.62 1.00 0.09  0.28  0.00  0.25 0.15
prod        0.25 0.09 1.00  0.07  0.03  0.34 0.19
first       0.16 0.28 0.07  1.00  0.10  0.37 0.41
sex         -0.10 0.00 0.03  0.10  1.00  0.13 0.43
cites       0.29 0.25 0.34  0.37  0.13  1.00 0.55
pubs        0.18 0.15 0.19  0.41  0.43  0.55 1.00
```

The model found by `pcaalg` is fairly reasonable (Figure 24.5). Of course, the linear-and-Gaussian assumption has no particular support here, and there is at least one variable for which it must be wrong (`which?`), but unfortunately with just the correlation matrix we cannot go further.



```
plot(pc(list(C=rm,n=162),indepTest=gaussCItest,p=7,alpha=0.01),
     labels=colnames(rm),main="")
```

Figure 24.5: Causes of academic success among psychologists. The arrow from citations to publications is a bit odd, but not impossible — people who get cited more might get more opportunities to do research and so to publish.

24.6 Limitations on Consistency of Causal Discovery

There are some important limitations to causal discovery algorithms (Spirtes *et al.*, 2001, §12.4). They are *universally* consistent: for all causal graphs G ,¹⁴

$$\lim_{n \rightarrow \infty} \Pr(\widehat{G}_n \neq G) = 0 \quad (24.8)$$

The probability of getting the graph wrong can be made arbitrarily small by using enough data. However, this says nothing about *how much* data we need to achieve a given level of confidence, i.e., the *rate* of convergence. *Uniform* consistency would mean that we could put a bound on the probability of error as a function of n which did not depend on the true graph G . Robins *et al.* (2003) proved that *no* uniformly-consistent causal discovery algorithm can exist. The issue, basically, is that the Adversary could make the convergence in Eq. 24.8 arbitrarily slow by selecting a distribution which, while faithful to G , came *very close* to being unfaithful, making some of the dependencies implied by the graph arbitrarily small. For any given dependence strength, there's some amount of data which will let us recognize it with high confidence, but the Adversary can make the required data size as large as he likes by weakening the dependence, without ever setting it to zero.¹⁵

The upshot is that so *uniform, universal* consistency is out of the question; we can be *universally* consistent, but without a uniform rate of convergence; or we can converge *uniformly*, but only on some less-than-universal class of distributions. These might be ones where all the dependencies which do exist are not too weak (and so not too hard to learn reliably from data), or the number of true edges is not too large (so that if we haven't seen edges yet they probably don't exist; Janzing and Herrmann, 2003; Kalisch and Bühlmann, 2007).

It's worth emphasizing that the Robins *et al.* (2003) no-uniform-consistency result applies to *any* method of discovering causal structure from data. Invoking human judgment, Bayesian priors over causal structures, etc., etc., won't get you out of it.

¹⁴If the true distribution is faithful to multiple graphs, then we should read G as their equivalence class, which has some undirected edges.

¹⁵Roughly speaking, if X and Y are dependent given Z , the probability of missing this conditional dependence with a sample of size n should go to zero like $O(2^{-nI[X;Y|Z]})$, I being mutual information. To make this probability equal to, say, α we thus need $n = O(-\log \alpha/I)$ samples. The Adversary can thus make n extremely large by making I very small, yet positive.

24.7 Further Reading

The best single reference on causal discovery algorithms remains Spirtes *et al.* (2001). A lot of work has been done in recent years by the group centered around ETH-Zürich, beginning with Kalisch and Bühlmann (2007), connecting this to modern statistical concerns about sparse effects and high-dimensional data.

As already mentioned, the best reference on partial identification is Manski (2007). Partial identification of causal effects due to multiple equivalent DAGs is considered in Maathuis *et al.* (2009), along with efficient algorithms for linear systems, which are applied in Maathuis *et al.* (2010), and implemented in the `pcalg` package as `ida()`.

Discovery is possible for directed cyclic graphs, though since it's harder to understand what such models mean, it less well-developed. Important papers on this topic include Richardson (1996) and Lacerda *et al.* (2008).

24.8 Exercises

To think through, not to hand in.

1. Prove that, assuming faithfulness, a three-variable chain and a three-variable fork imply exactly the same set of dependence and independence relations, but that these are different from those implied by a three-variable collider. Are any implications common to chains, forks, and colliders? Could colliders be distinguished from chains and forks without assuming faithfulness?
2. Prove that if X and Y are not parent and child, then either $X \perp\!\!\!\perp Y$, or there exists a set of variables S such that $X \perp\!\!\!\perp Y|S$. *Hint:* start with the Markov property, that any X is independent of all its non-descendants given its parents, and consider separately the cases where Y a descendant of X and those where it is not.
3. Prove that the graph produced by the edge-removal step of the PC algorithm is exactly the same as the graph produced by the edge-removal step of the SGS algorithm. *Hint:* SGS removes the edge between X and Y when $X \perp\!\!\!\perp Y|S$ for even one set S .
4. Prove that if $X \perp\!\!\!\perp Y|S$ for some set of variables S , then $X \perp\!\!\!\perp Y|S'$, where every variable in S' is a neighbor of X or Y .
5. When, exactly, does $E[Y|X, Z] = E[Y|Z]$ imply $Y \perp\!\!\!\perp X|Z$?
6. Would the SGS algorithm work on a non-causal, merely-probabilistic DAG? If so, in what sense is it a *causal* discovery algorithm? If not, why not?
7. Describe how to use bandwidth selection as a conditional independence test.
8. Read Kalisch *et al.* (2012) and write a conditional independence test function based on bandwidth selection. Check that your test function gives the right size when run on test cases where you know the variables are conditionally independent. Check that your test function works with `pcalg::pc`.

14:41 Thursday 25th April, 2013

Part IV

Dependent Data

Chapter 25

Time Series

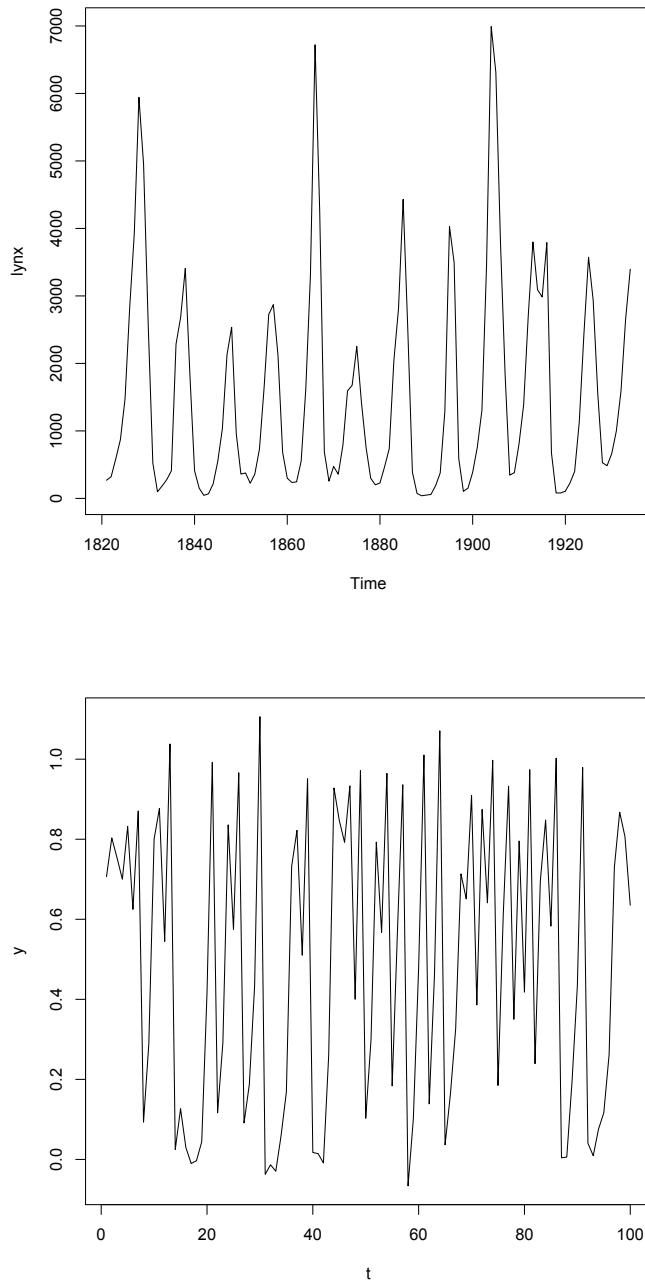
So far, we have assumed that all data points are pretty much independent of each other. In the chapters on regression, we assumed that each Y_i was independent of every other, given its X_i , and we often assumed that the X_i were themselves independent. In the chapters on multivariate distributions and even on causal inference, we allowed for arbitrarily complicated dependence between the *variables*, but each *data-point* was assumed to be generated independently. We will now relax this assumption, and see what sense we can make of dependent data.

25.1 Time Series, What They Are

The simplest form of dependent data are time series, which are just what they sound like: a series of values recorded over time. The most common version of this, in statistical applications, is to have measurements of a variable or variables X at equally-spaced time-points starting from t , written say $X_t, X_{t+h}, X_{t+2h}, \dots$, or $X(t), X(t+h), X(t+2h), \dots$. Here h , the amount of time between observations, is called the “sampling interval”, and $1/h$ is the “sampling frequency” or “sampling rate”.

Figure 25.1 shows two fairly typical time series. One of them is actual data (the number of lynxes trapped each year in a particular region of Canada); the other is the output of a purely artificial model. (Without the labels, it might not be obvious which one was which.) The basic idea of all of time series analysis is one which we’re already familiar with from the rest of statistics: we regard the actual time series we see as one realization of some underlying, partially-random (“stochastic”) process, which generated the data. We use the data to make guesses (“inferences”) about the process, and want to make *reliable* guesses while being clear about the uncertainty involved. The complication is that each observation is dependent on all the other observations; in fact it’s usually this dependence that we want to draw inferences about.

25.1.0.0.2 Other kinds of time series One sometimes encounters irregularly-sampled time series, $X(t_1), X(t_2), \dots$, where $t_i - t_{i-1} \neq t_{i+1} - t_i$. This is mostly an annoyance, unless the observation times are somehow dependent on the values.



```
data(lynx); plot(lynx)
```

Figure 25.1: Above: annual number of trapped lynxes in the Mackenzie River region of Canada. Below: a toy dynamical model. (See code online for the toy.)
14:41 Thursday, 23rd April, 2013

Continuously-observed processes are rarer — especially now that digital sampling has replaced analog measurement in so many applications. (It is more common to model the process as evolving continuously in time, but observe it at discrete times.) We skip both of these in the interest of space.

Regular, irregular or continuous time series all record the same variable at every moment of time. An alternative is to just record the sequence of times at which some event happened; this is called a “point process”. More refined data might record the time of each event and its type — a “marked point process”. Point processes include very important kinds of data (e.g., earthquakes), but they need special techniques, and we’ll skip them.

25.1.0.0.3 Notation For a regularly-sampled time series, it’s convenient not to have to keep writing the actual time, but just the position in the series, as X_1, X_2, \dots , or $X(1), X(2), \dots$. This leads to a useful short-hand, that $X_i^j = (X_i, X_{i+1}, \dots, X_{j-1}, X_j)$, a whole block of time; some people write $X_{i:j}$ with the same meaning.

25.2 Stationarity

In our old IID world, the distribution of each observation is the same as the distribution of every other data point. It would be nice to have something like this for time series. The property is called **stationarity**, which doesn’t mean that the time series never changes, but that its *distribution* doesn’t.

More precisely, a time series is **strictly stationary** or **strongly stationary** when X_1^k and X_t^{t+k-1} have the same distribution, for all k and t — the distribution of blocks of length k is **time-invariant**. Again, this doesn’t mean that every block of length k has the same value, just that it has the same distribution of values.

If there is strong or strict stationarity, there should be **weak** or **loose** (or **wide-sense**) stationarity, and there is. All it requires is that $E[X_1] = E[X_t]$, and that $\text{Cov}[X_1, X_k] = \text{Cov}[X_t, X_{t+k-1}]$. (Notice that it’s not dealing with whole blocks of time any more, just single time-points.) Clearly (exercise!), strong stationarity implies weak stationarity, but not, in general, the other way around, hence the names. It may not surprise you to learn that strong and weak stationarity coincide when X_t is a Gaussian process, but not, in general, otherwise.

You should convince yourself that an IID sequence is strongly stationary.

25.2.1 Autocorrelation

Time series are **serially dependent**: X_t is in general dependent on all earlier values in time, and on all later ones. Typically, however, there is **decay of dependence** (sometimes called **decay of correlations**): X_t and X_{t+h} become more and more nearly independent as $h \rightarrow \infty$. The oldest way of measuring this is the **autocovariance**,

$$\gamma(h) = \text{Cov}[X_t, X_{t+h}] \quad (25.1)$$

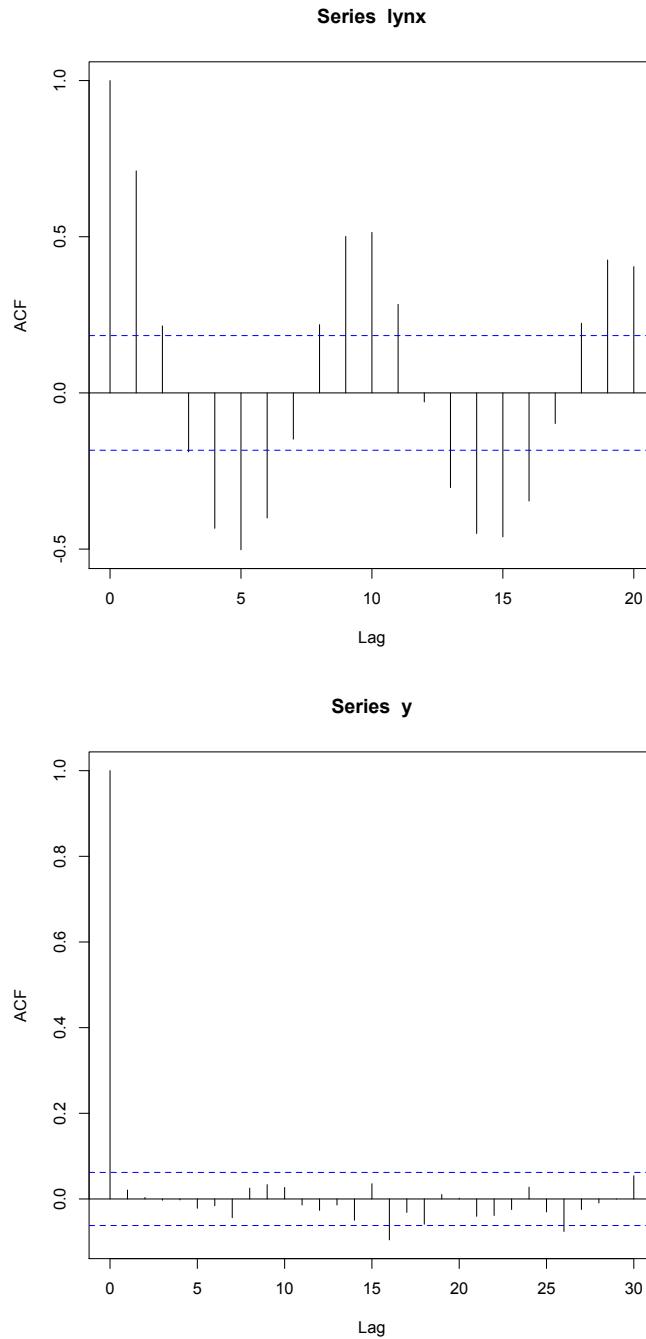
which is well-defined just when the process is weakly stationary. We could equally well use the **autocorrelation**,

$$\rho(b) = \frac{\text{Cov}[X_t, X_{t+b}]}{\text{Var}[X_t]} = \frac{\gamma(b)}{\gamma(0)} \quad (25.2)$$

again using stationarity to simplify the denominator.

As I said, for most time series $\gamma(b) \rightarrow 0$ as b grows. Of course, $\gamma(b)$ could be exactly zero while X_t and X_{t+b} are strongly dependent. Figure 25.2 shows the auto-correlation functions (ACFs) of the lynx data and the simulation model; the correlation for the latter is basically never distinguishable from zero, which doesn't accord at all with the visual impression of the series. Indeed, we can confirm that *something* is going on the series by the simple device of plotting X_{t+1} against X_t (Figure 25.3). More general measures of dependence would include looking at the Spearman rank-correlation of X_t and X_{t+b} , or quantities like mutual information.

Autocorrelation is important for four reasons, however. First, because it is the oldest measure of serial dependence, it has a “large installed base”: everybody knows about it, they use it to communicate, and they’ll ask you about it. Second, in the rather special case of Gaussian processes, it really does tell us everything we need to know. Third, in the somewhat less special case of linear prediction, it tells us everything we need to know. Fourth and finally, it plays an important role in a crucial theoretical result, which we’ll go over next.



```
acf(lynx); acf(y)
```

Figure 25.2: Autocorrelation functions of the lynx data (above) and the simulation (below). The `acf` function plots the autocorrelation function as an automatic side-effect; it actually returns the actual value of the autocorrelations, which you can capture. The 95% confidence interval around zero is computed under Gaussian assumptions which shouldn't be taken too seriously, unless the sample size is quite large, but are useful as guides to the eye.

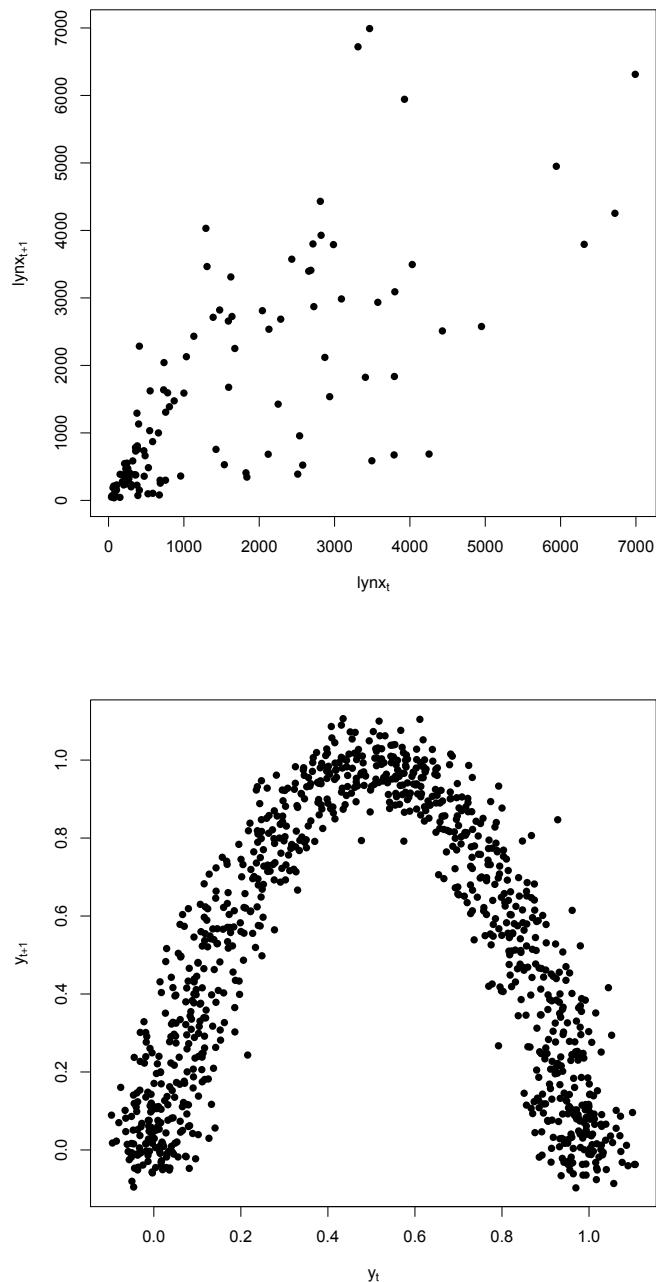


Figure 25.3: Plots of X_{t+1} versus X_t , for the lynx (above) and the simulation (below). (See code online.) Note that even though the correlation between successive iterates is next to zero for the simulation, there is clearly a lot of dependence.

14:41 Thursday 25th April, 2013

25.2.2 The Ergodic Theorem

With IID data, the ultimate basis of all our statistical inference is the law of large numbers, which told us that

$$\frac{1}{n} \sum_{i=1}^n X_i \rightarrow E[X_1] \quad (25.3)$$

For complicated historical reasons, the corresponding result for time series is called the **ergodic theorem**¹. The most general and powerful versions of it are quite formidable, and have very subtle proofs, but there is a simple version which gives the flavor of them all, and is often useful enough.

25.2.2.1 The World's Simplest Ergodic Theorem

Suppose X_t is weakly stationary, and that

$$\sum_{b=0}^{\infty} |\gamma(b)| = \gamma(0)\tau < \infty \quad (25.4)$$

(Remember that $\gamma(0) = \text{Var}[X_t]$.) The quantity τ is called the **correlation time**, or **integrated autocorrelation time**.

Now consider the average of the first n observations,

$$\bar{X}_n = \frac{1}{n} \sum_{t=1}^n X_t \quad (25.5)$$

This **time average** is a random variable. Its expectation value is

$$E[\bar{X}_n] = \frac{1}{n} \sum_{t=1}^n E[X_t] = E[X_1] \quad (25.6)$$

¹In the late 1800s, the physicist Ludwig Boltzmann needed a word to express the idea that if you took an isolated system at constant energy and let it run, any one trajectory, continued long enough, would be representative of the system as a whole. Being a highly-educated nineteenth century German-speaker, Boltzmann knew far too much ancient Greek, so he called this the “ergodic property”, from *ergon* “energy, work” and *hodos* “way, path”. The name stuck.

because the mean is stationary. What about its variance?

$$\text{Var}[\bar{X}_n] = \text{Var}\left[\frac{1}{n} \sum_{t=1}^n X_t\right] \quad (25.7)$$

$$= \frac{1}{n^2} \left[\sum_{t=1}^n \text{Var}[X_t] + 2 \sum_{t=1}^n \sum_{s=t+1}^n \text{Cov}[X_t, X_s] \right] \quad (25.8)$$

$$= \frac{1}{n^2} \left[n \text{Var}[X_1] + 2 \sum_{t=1}^n \sum_{s=t+1}^n \gamma(s-t) \right] \quad (25.9)$$

$$\leq \frac{1}{n^2} \left[n \gamma(0) + 2 \sum_{t=1}^n \sum_{s=t+1}^n |\gamma(s-t)| \right] \quad (25.10)$$

$$\leq \frac{1}{n^2} \left[n \gamma(0) + 2 \sum_{t=1}^n \sum_{b=1}^n |\gamma(b)| \right] \quad (25.11)$$

$$\leq \frac{1}{n^2} \left[n \gamma(0) + 2 \sum_{t=1}^n \sum_{b=1}^{\infty} |\gamma(b)| \right] \quad (25.12)$$

$$= \frac{n \gamma(0)(1+2\tau)}{n^2} \quad (25.13)$$

$$= \frac{\gamma(0)(1+2\tau)}{n} \quad (25.14)$$

Eq. 25.9 uses stationarity again, and then Eq. 25.13 uses the assumption that the correlation time τ is finite.

Since $E[\bar{X}_n] = E[X_1]$, and $\text{Var}[\bar{X}_n] \rightarrow 0$, we have that $\bar{X}_n \rightarrow E[X_1]$, exactly as in the IID case. ("Time averages converge on expected values.") In fact, we can say a bit more. Remember Chebyshev's inequality: for any random variable Z ,

$$\Pr(|Z - E[Z]| > \epsilon) \leq \frac{\text{Var}[Z]}{\epsilon^2} \quad (25.15)$$

so

$$\Pr(|\bar{X}_n - E[X_1]| > \epsilon) \leq \frac{\gamma(0)(1+2\tau)}{n \epsilon^2} \quad (25.16)$$

which goes to zero as n grows for any given ϵ .

You may wonder whether the condition that $\sum_{b=0}^{\infty} |\gamma(b)| < \infty$ is as weak as possible. It turns out that it can in fact be weakened to just $\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{b=0}^n \gamma(b) = 0$, as indeed the proof above might suggest.

25.2.2.2 Rate of Convergence

If the X_t were all IID, or even just uncorrelated, we would have $\text{Var}[\bar{X}_n] = \gamma(0)/n$ exactly. Our bound on the variance is larger by a factor of $(1+2\tau)$, which reflects the influence of the correlations. Said another way, we can more or less pretend

that instead of having n correlated data points, we have $n/(1+2\tau)$ independent data points, that $n/(1+2\tau)$ is our **effective sample size**²

Generally speaking, dependence between observations reduces the effective sample size, and the stronger the dependence, the greater the reduction. (For an extreme, consider the situation where X_1 is randomly drawn, but thereafter $X_{t+1} = X_t$.) In more complicated situations, finding the effective sample size is itself a tricky undertaking, but it's often got this general flavor.

25.2.2.3 Why Ergodicity Matters

The ergodic theorem is important, because it tells us that a single long time series becomes representative of the whole data-generating process, just the same way that a large IID sample becomes representative of the whole population or distribution. We can therefore actually learn about the process from empirical data.

Strictly speaking, we have established that time-averages converge on expectations only for X_t itself, not even for $f(X_t)$ where the function f is non-linear. It might be that $f(X_t)$ doesn't have a finite correlation time even though X_t does, or indeed vice versa. This is annoying; we don't want to have to go through the analysis of the last section for every different function we might want to calculate.

When people say that the whole process is **ergodic**, they roughly speaking mean that

$$\frac{1}{n} \sum_{t=1}^n f(X_t^{t+k-1}) \rightarrow \mathbf{E}[f(X_1^k)] \quad (25.17)$$

for any reasonable function f . This is (again very roughly) equivalent to

$$\frac{1}{n} \sum_{t=1}^n \Pr(X_1^k \in A, X_t^{t+k-1} \in B) \rightarrow \Pr(X_1^k \in A) \Pr(X_1^l \in B) \quad (25.18)$$

which is a kind of asymptotic independence-on-average³

If our data source is ergodic, then what Eq. 25.17 tells us is that sample averages of any reasonable function are representative of expectation values, which is what we need to be in business statistically. This in turn is basically implied by stationarity.⁴ What does this let us do?

²Some people like to define the correlation time as, in this notation, $1+2\tau$ for just this reason.

³It's worth sketching a less rough statement. Instead of working with X_t , work with the whole future trajectory $Y_t = (X_t, X_{t+1}, X_{t+2}, \dots)$. Now the dynamics, the rule which moves us into the future, can be summed up in very simple, and deterministic, operation T : $Y_{t+1} = TY_t = (X_{t+1}, X_{t+2}, X_{t+3}, \dots)$. A set of trajectories is **invariant** if it is left unchanged by T : for every $y \in A$, there is another y' in A where $Ty' = y$. A process is **ergodic** if every invariant set either has probability 0 or probability 1. What this means is that (almost) all trajectories generated by an ergodic process belong to a single invariant set, and they all wander from every part of that set to every other part — they are **metrically transitive**. (Because: no smaller set with any probability is invariant.) Metric transitivity, in turn, is equivalent, assuming stationarity, to $n^{-1} \sum_{t=0}^{n-1} \Pr(Y \in A, T^t Y \in B) \rightarrow \Pr(Y \in A) \Pr(Y \in B)$. From metric transitivity follows Birkhoff's “individual” ergodic theorem, that $n^{-1} \sum_{t=0}^{n-1} f(T^t Y) \rightarrow \mathbf{E}[f(Y)]$, with probability 1. Since a function of the trajectory can be a function of a block of length k , we get Eq. 25.17.

⁴Again, a sketch of a less rough statement. Use Y again for whole trajectories. Every stationary distribution for Y can be written as a mixture of stationary and ergodic distributions, rather as we wrote complicated distributions as mixtures of simple Gaussians in Chapter 19. (This is called the “ergodic

25.3 Markov Models

For this section, we'll assume that X_t comes from a stationary, ergodic time series. So for any reasonable function f , the time-average of $f(X_t)$ converges on $E[f(X_1)]$. Among the “reasonable” functions are the indicators, so

$$\frac{1}{n} \sum_{t=1}^n \mathbf{1}_A(X_t) \rightarrow \Pr(X_1 \in A) \quad (25.19)$$

Since this also applies to functions of blocks,

$$\frac{1}{n} \sum_{t=1}^n \mathbf{1}_{A,B}(X_t, X_{t+1}) \rightarrow \Pr(X_1 \in A, X_2 \in B) \quad (25.20)$$

and so on. If we can learn joint and marginal probabilities, and we remember how to divide, then we can learn conditional probabilities.

It turns out that pretty much any density estimation method which works for IID data will also work for getting the marginal and conditional distributions of time series (though, again, the effective sample size depends on how quickly dependence decays). So if we want to know $p(x_t)$, or $p(x_{t+1}|x_t)$, we can estimate it just as we learned how to do in Chapter 15.

Now, the conditional pdf $p(x_{t+1}|x_t)$ always exists, and we can always estimate it. But why stop just one step back into the past? Why not look at $p(x_{t+1}|x_t, x_{t-1})$, or for that matter $p(x_{t+1}|x_{t-999}^t)$? There are three reasons, in decreasing order of pragmatism.

- Estimating $p(x_{t+1}|x_{t-999}^t)$ means estimating a thousand-and-one-dimensional distribution. The curse of dimensionality will crush us.
- Because of the decay of dependence, there shouldn't be much difference, much of the time, between $p(x_{t+1}|x_{t-999}^t)$ and $p(x_{t+1}|x_{t-998}^t)$, etc. Even if we could go very far back into the past, it shouldn't, usually, change our predictions very much.
- Sometimes, a finite, short block of the past completely screens off the remote past.

You will remember the Markov property from your previous probability classes:

$$X_{t+1} \perp\!\!\!\perp X_1^{t-1} | X_t \quad (25.21)$$

(decomposition” of the process.) We can think of this as first picking an ergodic process according to some fixed distribution, and then generating Y from that process. Time averages computed along any one trajectory thus converge according to Eq. 25.17. If we have only a single trajectory, it looks just like a stationary and ergodic process. If we have multiple trajectories from the same source, each one may be converging to a different ergodic component. It is common, and only rarely a problem, to assume that the data source is not only stationary but also ergodic.

When the Markov property holds, there is simply no point in looking at $p(x_{t+1}|x_t, x_{t-1})$, because it's got to be just the same as $p(x_{t+1}|x_t)$. If the process isn't a simple Markov chain but has a higher-order Markov property,

$$X_{t+1} \perp\!\!\!\perp X_1^{t-k} | X_{t-k+1}^t \quad (25.22)$$

then we never have to condition on more than the last k steps to learn all that there is to know. The Markov property means that the current state screens off the future from the past.

It is *always* an option to model X_t as a Markov process, or a higher-order Markov process. If it isn't exactly Markov, if there's really some dependence between the past and the future even given the current state, then we're introducing some bias, but it can be small, and dominated by the reduced variance of not having to worry about higher-order dependencies.

25.3.1 Meaning of the Markov Property

The Markov property is a weakening of both being strictly IID and being strictly deterministic.

That being Markov is weaker than being IID is obvious: an IID sequence satisfies the Markov property, because everything is independent of everything else no matter what we condition on.

In a deterministic dynamical system, on the other hand, we have $X_{t+1} = g(X_t)$ for some fixed function g . Iterating this equation, the current state X_t fixes the whole future trajectory X_{t+1}, X_{t+2}, \dots . In a Markov chain, we weaken this to $X_{t+1} = g(X_t, U_t)$, where the U_t are IID noise variables (which we can take to be uniform for simplicity). The current state of a Markov chain doesn't fix the exact future trajectory, but it does fix the *distribution* over trajectories.

The real meaning of the Markov property, then, is about information flow: the current state is the only channel through which the past can affect the future.

t	x				
		lag0	lag1	lag2	lag3
1821	269				
1822	321	871	585	321	269
1823	585				
1824	871	1475	871	585	321
1825	1475	\Rightarrow	2821	1475	871
1826	2821	3928	2821	1475	871
1827	3928	5943	3928	2821	1475
1828	5943		4950	5943	3928
1829	4950			2821	
	...				

Figure 25.4: Turning a time series (here, the beginning of lynx) into a regression-suitable matrix.

```
design.matrix.from.ts <- function(ts,order) {
  n <- length(ts)
  x <- ts[(order+1):n]
  for (lag in 1:order) {
    x <- cbind(x,ts[(order+1-lag):(n-lag)])
  }
  colnames(x) <- c("lag0",paste("lag",1:order,sep=""))
  return(as.data.frame(x))
}
```

Code Example 32: Example code for turning a time series into a design matrix, suitable for regression.

25.4 Autoregressive Models

Instead of trying to estimate the whole conditional distribution of X_t , we can just look at its conditional expectation. This is a regression problem, but since we are regressing X_t on earlier values of the series, it's called an **autoregression**:

$$\mathbb{E} \left[X_t | X_{t-p}^{t-1} = x_1^p \right] = r(x_1^p) \quad (25.23)$$

If we think the process is Markov of order p , then of course there is no point in conditioning on more than p steps of the past when doing an autoregression. But even if we don't think the process is Markov, the same reasons which inclined us towards Markov approximations also make limited-order autoregressions attractive.

Since this is a regression problem, we can employ all the tools we know for regression analysis: linear models, kernel regression, spline smoothing, additive models, etc., mixtures of regressions, etc. Since we are regressing X_t on earlier values from the same series, it is useful to have tools for turning a time series into a regression-style design matrix (as in Figure 25.4); see Code Example 32.

Suppose $p = 1$. Then we essentially want to draw regression curves through plots like those in Figure 25.3. Figure 25.5 shows an example for the artificial series.

25.4.1 Autoregressions with Covariates

Nothing keeps us from adding a variable other than the past of X_t to the regression:

$$\mathbb{E}[X_{t+1}|X_{t-k+1}^t, Z] \quad (25.24)$$

or even another time series:

$$\mathbb{E}[X_{t+1}|X_{t-k+1}^t, Z_{t-l+1}^t] \quad (25.25)$$

These are perfectly well-defined conditional expectations, and quite estimable in principle. Of course, adding more variables to a regression means having to estimate more, so again the curse of dimensionality comes up, but our methods are very much the same as in the basic regression analyses.

25.4.2 Additive Autoregressions

As before, if we want some of the flexibility of non-parametric smoothing, without the curse of dimensionality, we can try to approximate the conditional expectation as an additive function:

$$\mathbb{E}[X_t|X_{t-p}^{t-1}] \approx \alpha_0 + \sum_{j=1}^p g_j(X_{t-j}) \quad (25.26)$$

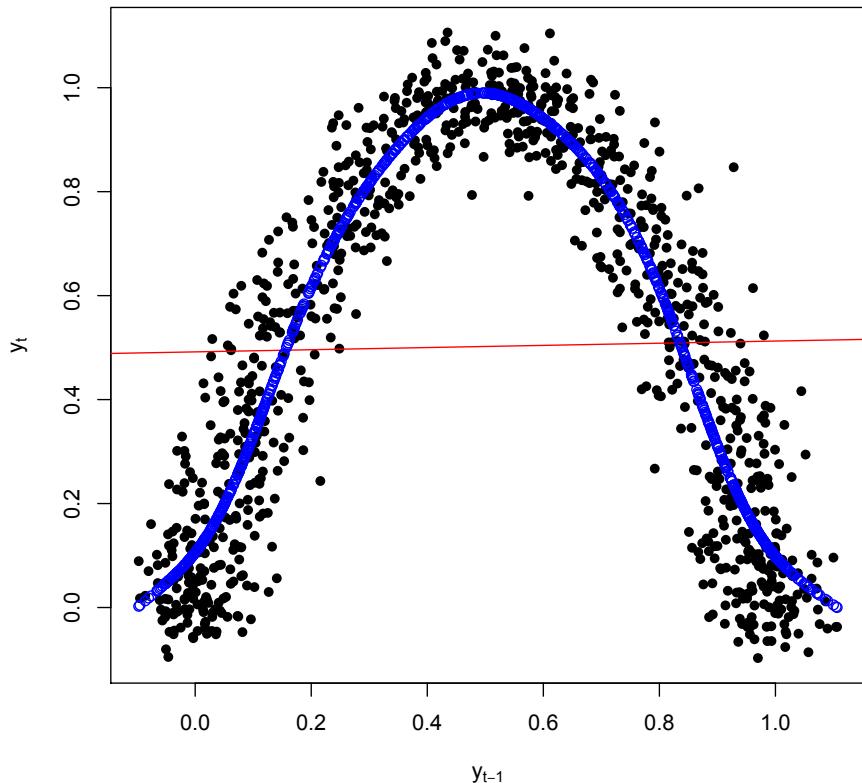
25.4.2.0.1 Example: The lynx Let's try fitting an additive model for the lynx. Code Example 33 shows some code for doing this. (Most of the work is re-shaping the time series into a data frame, and then automatically generating the right formula for `gam`.) Let's try out $p = 2$.

```
lynx.aar2 <- aar(lynx, 2)
```

This inherits everything we can do with a GAM, so we can do things like plot the partial response functions (Figure 25.6), plot the fitted values against the actual (Figure 25.7), etc. To get a sense of how well it can actually extrapolate, Figure 25.8 re-fits the model to just the first 80 data points, and then predicts the remaining 34.

25.4.3 Linear Autoregression

When people talk about autoregressive models, they usually (alas) just mean linear autoregressions. There is almost never any justification in scientific theory for this preference, but we can always ask for the best linear approximation to the true autoregression, if only because it's fast to compute and fast to converge.

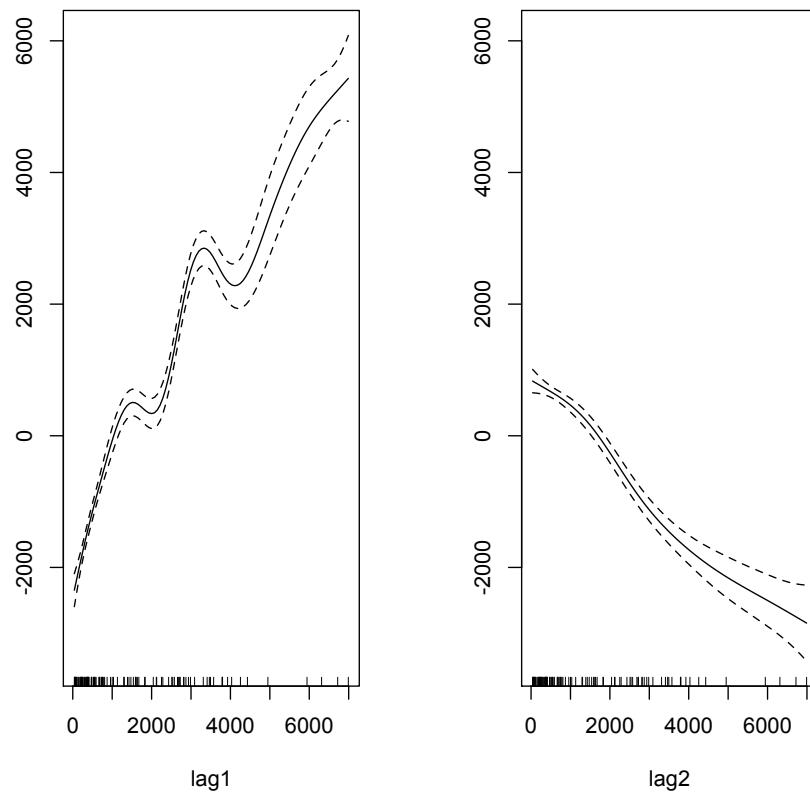


```

plot(lag0 ~ lag1,data=design.matrix.from.ts(y,1),xlab=expression(y[t-1]),
      ylab=expression(y[t]),pch=16)
abline(lm(lag0~lag1,data=design.matrix.from.ts(y,1)),col="red")
yaar1 <- aar(y,order=1)
points(y[-length(y)],fitted(yaar1),col="blue")

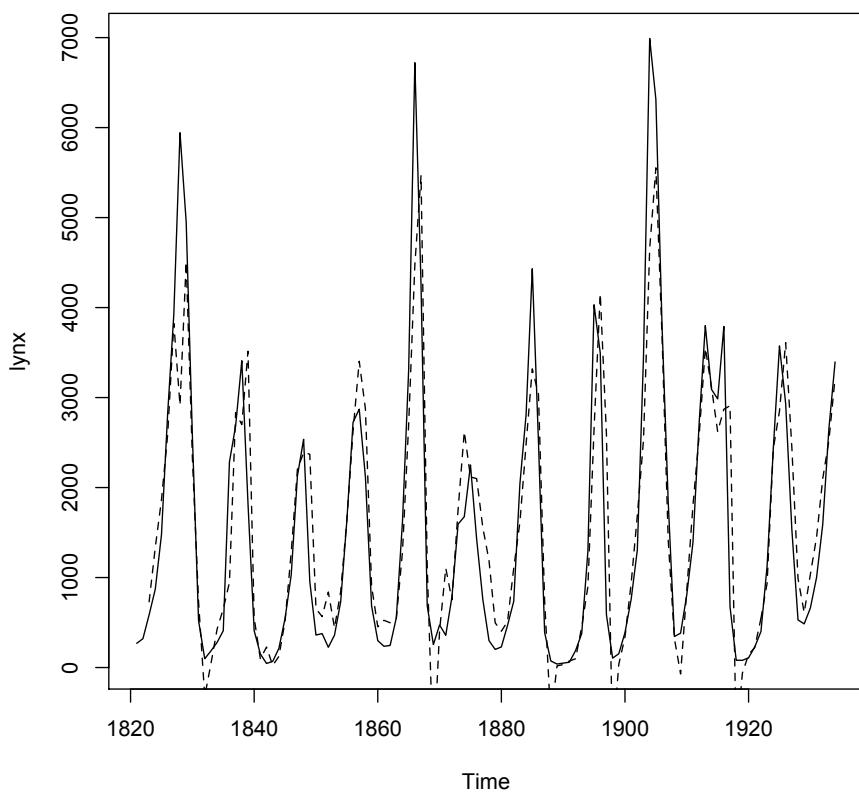
```

Figure 25.5: Plotting successive values of the artificial time series against each other, along with the linear regression, and a spline curve (see below for the `aar` function, which fits additive autoregressive models; with `order=1`, it just fits a spline).



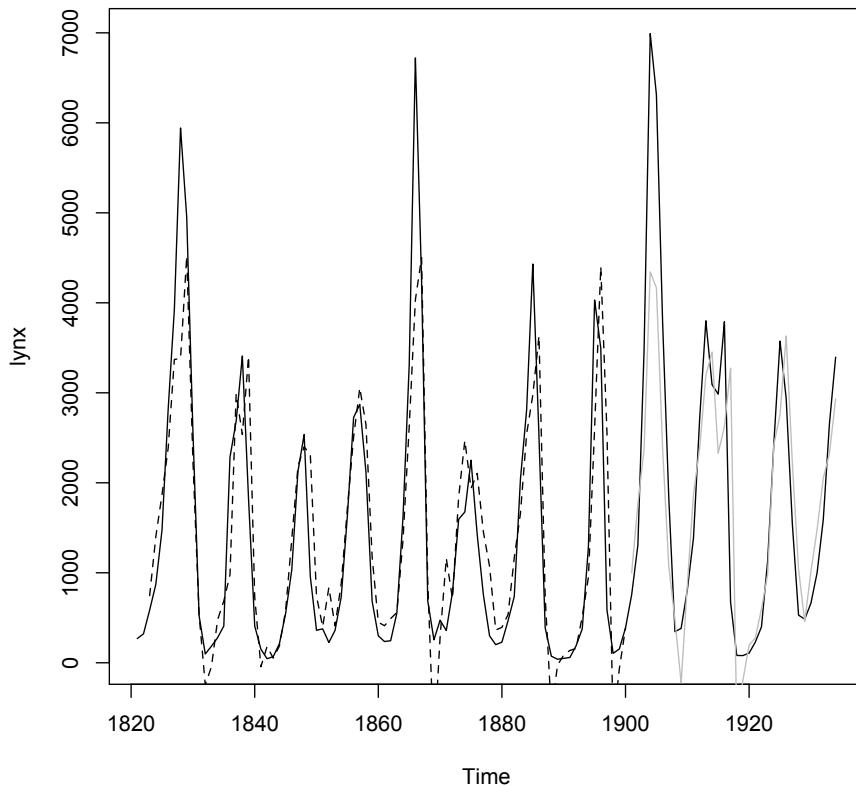
```
plot(lynx.aar2,pages=1)
```

Figure 25.6: Partial response functions for the second-order additive autoregression model of the lynx. Notice that a high count last year predicts a higher count this year, but a high count *two* years ago predicts a lower count this year. This is the sort of alternation which will tend to drive oscillations.



```
plot(lynx)
lines(1823:1934,fitted(lynx.aar2),lty="dashed")
```

Figure 25.7: Actual time series (solid line) and predicted values (dashed) for the second-order additive autoregression model of the lynx. The match is quite good, but of course every one of these points was used to learn the model, so it's not quite as impressive as all that. (Also, the occasional prediction of a negative number of lynxes is less than ideal.)



```

lynx.aar2b <- aar(lynx[1:80],2)
out.of.sample <- design.matrix.from.ts(lynx[-(1:78)],2)
lynx.preds <- predict(lynx.aar2b,newdata=out.of.sample)
plot(lynx)
lines(1823:1900,fitted(lynx.aar2b),lty="dashed")
lines(1901:1934,lynx.preds,col="grey")

```

Figure 25.8: Out-of-sample forecasting. The same model specification as before is estimated on the first 80 years of the lynx data, then used to predict the remaining 34 years. Solid black line, data; dashed line, the in-sample prediction on the training data; grey lines, predictions on the testing data. The RMS errors are 723 lynxes/year in-sample, 922 lynxes/year out-of-sample.

```

aar <- function(ts,order) {
  stopifnot(require(mgcv))
  fit <- gam(as.formula(auto.formula(order)),
             data=design.matrix.from.ts(ts,order))
  return(fit)
}

auto.formula <- function(order) {
  inputs <- paste("s(lag",1:order,")",sep="",collapse="+")
  form <- paste("lag0 ~ ",inputs)
  return(form)
}

```

Code Example 33: Fitting an additive autoregression of arbitrary order to a time series. See online for comments.

The analysis we did in Chapter 2 of how to find the optimal linear predictor carries over with no change whatsoever. If we want to predict X_t as a linear combination of the last k observations, $X_{t-1}, X_{t-2}, \dots, X_{t-p}$, then the ideal coefficients β are

$$\beta = \left(\text{Var} \left[X_{t-p}^{t-1} \right] \right)^{-1} \text{Cov} \left[X_{t-p}^{t-1}, X_t \right] \quad (25.27)$$

where $\text{Var} \left[X_{t-p}^t \right]$ is the variance-covariance matrix of $(X_{t-1}, \dots, X_{t-p})$ and similarly $\text{Cov} \left[X_{t-p}^{t-1}, X_t \right]$ is a vector of covariances. Assume stationarity, $\text{Var} \left[X_t \right]$ is constant in t , and so the common factor of the over-all variance goes away, and β could be written entirely in terms of the correlation function ρ . Stationarity also lets us estimate these covariances, by taking time-averages.

A huge amount of effort is given over to using linear AR models, which in my opinion is out of all proportion to their utility — but very reflective of what was computationally feasible up to about 1980. My experience is that results like Figure 25.9 is pretty typical.

25.4.3.1 “Unit Roots” and Stationary Solutions

Suppose we really believed a first-order linear autoregression,

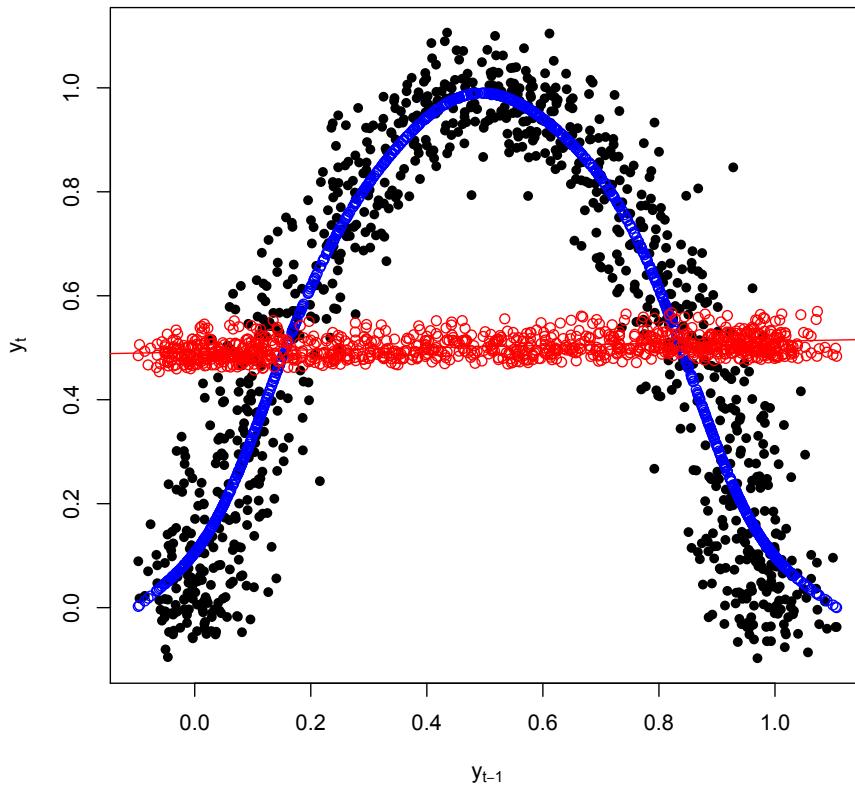
$$X_{t+1} = \alpha + \beta X_t + \epsilon_t \quad (25.28)$$

with ϵ_t some IID noise sequence. Let’s suppose that the mean is zero for simplicity, so $\alpha = 0$. Then

$$X_{t+2} = \beta^2 X_t + \beta \epsilon_t + \epsilon_{t+1} \quad (25.29)$$

$$X_{t+3} = \beta^3 X_t + \beta^2 \epsilon_t + \beta \epsilon_{t+1} + \epsilon_{t+2}, \quad (25.30)$$

etc. If this is going to be stationary, it’d better be the case that what happened at time t doesn’t go on to dominate what happens at all later times, but clearly that



```
library(tseries)
yar8 <- arma(y,order=c(8,0))
points(y[-length(y)],fitted(yar8)[-1],col="red")
```

Figure 25.9: Adding the predictions of an eighth-order linear AR model (red dots) to Figure 25.5. We will see the `arma` function in more detail next time; for now, it's enough to know that when the second component of its `order` argument is 0, it estimates and fits a linear AR model.

will happen if $|\beta| > 1$, whereas if $|\beta| < 1$, eventually all memory of X_t (and ϵ_t) fades away. The linear AR(1) model in fact can only produce stationary distributions when $|\beta| < 1$.

For higher-order linear AR models, with parameters $\beta_1, \beta_2, \dots, \beta_p$, the corresponding condition is that all the roots of the polynomial

$$\sum_{j=1}^p \beta_j z^j - 1 \quad (25.31)$$

must be outside the unit circle. When this fails, when there is a “unit root”, the linear AR model cannot generate a stationary process.

There is a fairly elaborate machinery for testing for unit roots, which is sometimes also used to test for non-stationarity. It is not clear how much this really matters. A non-stationary but truly linear AR model can certainly be estimated⁵; a linear AR model can be non-stationary even if it has no unit roots⁶; and if the linear model is just an approximation to a non-linear one, the unit-root criterion doesn’t apply to the true model anyway.

25.4.4 Conditional Variance

Having estimated the conditional expectation, we can estimate the conditional variance $\text{Var}[X_t | X_{t-p}^{t-1}]$ just as we estimated other conditional variances, in Chapter 7.

25.4.4.0.1 Example: lynx The lynx series seems ripe for fitting conditional variance functions — presumably when there are a few thousand lynxes, the noise is going to be larger than when there are only a few hundred.

```
sq.res <- residuals(lynx.aar2)^2
lynx.condvar1 <- gam(sq.res ~ s(lynx[-(1:2)]))
lynx.condvar2 <- gam(sq.res ~ s(lag1)+s(lag2),
  data=design.matrix.from.ts(lynx,2))
```

I have fit two different models for the conditional variance here, just because. Figure 25.10 shows the data, and the predictions of the second-order additive AR model, but with just the standard deviation bands corresponding to the first of these two models; you can try making the analogous plot for `lynx.condvar2`.

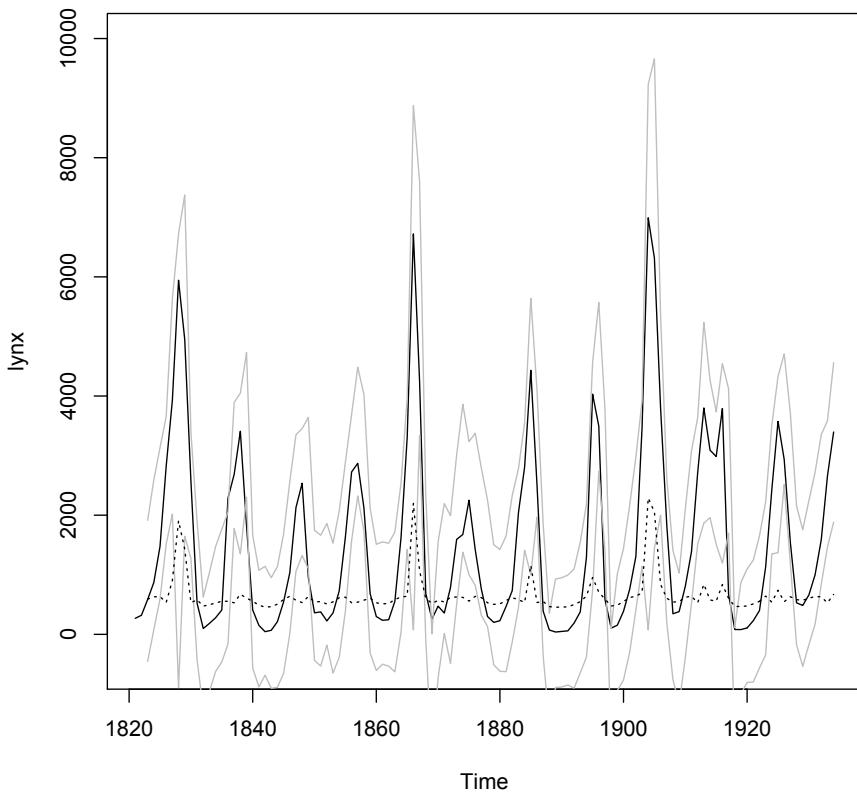
25.4.5 Regression with Correlated Noise; Generalized Least Squares

Suppose we have an old-fashioned regression problem

$$Y_t = r(X_t) + \epsilon_t \quad (25.32)$$

⁵Because the correlation structure stays the same, even as the means and variances can change. Consider $X_t = X_{t-1} + \epsilon_t$, with ϵ_t IID.

⁶Start it with X_1 very far from the expected value.



```

plot(lynx,ylim=c(-500,10000))
sd1 <- sqrt(fitted(lynx.condvar1))
lines(1823:1934,fitted(lynx.aar2)+2*sd1,col="grey")
lines(1823:1934,fitted(lynx.aar2)-2*sd1,col="grey")
lines(1823:1934, sd1,lty="dotted")

```

Figure 25.10: The lynx data (black line), together with the predictions of the additive autoregression ± 2 conditional standard deviations. The dotted line shows how the conditional standard deviation changes over time; notice how it ticks upwards around the big spikes in population.

only now the noise terms ϵ_t are themselves a dependent time series. Ignoring this dependence, and trying to estimate m by minimizing the mean squared error, is very much like ignoring heteroskedasticity. (In fact, heteroskedastic ϵ_t are a special case.) What we saw in Chapter 7 is that ignoring heteroskedasticity doesn't lead to bias, but it does mess up our understanding of the uncertainty of our estimates, and is generally inefficient. The solution was to weight observations, with weights inversely proportional to the variance of the noise.

With correlated noise, we do something very similar. Suppose we knew the covariance function $\gamma(b)$ of the noise. From this, we could construct the variance-covariance matrix Γ of the ϵ_t (since $\Gamma_{ij} = \gamma(i - j)$, of course).

We can use this as follows. Say that our guess about the regression function is m . Stacking y_1, y_2, \dots, y_n into a matrix y as usual in regression, and likewise creating $\mathbf{m}(x)$, the Gauss-Markov theorem (Appendix D) tells us that the most efficient estimate is the solution to the **generalized least squares** problem,

$$\hat{m}_{GLS} = \underset{m}{\operatorname{argmin}} \frac{1}{n} (y - \mathbf{m}(x))^T \Gamma^{-1} (y - \mathbf{m}(x)) \quad (25.33)$$

as opposed to just minimizing the mean-squared error,

$$\hat{m}_{OLS} = \underset{m}{\operatorname{argmin}} \frac{1}{n} (y - \mathbf{m}(x))^T (y - \mathbf{m}(x)) \quad (25.34)$$

Multiplying by the inverse of Γ appropriately discounts for observations which are very noisy, *and* discounts for correlations between observations introduced by the noise.⁷

This raises the question of how to get $\gamma(b)$ in the first place. If we knew the true regression function r , we could use the covariance of $Y_t - r(X_t)$ across different t . Since we don't know r , but have only an estimate \hat{m} , we can try alternating between using a guess at γ to estimate \hat{m} , and using \hat{m} to improve our guess at γ . We used this sort of iterative approximation for weighted least squares, and it can work here, too.

⁷If you want to use a linear model for m , this can be carried through to an explicit modification of the usual ordinary-least-squares estimate — Exercise 1.

25.5 Bootstrapping Time Series

The big picture of bootstrapping doesn't change: simulate a distribution which is close to the true one, repeat our estimate (or test or whatever) on the simulation, and then look at the distribution of this statistic over many simulations. The catch is that the surrogate data from the simulation has to have the same sort of dependence as the original time series. This means that simple resampling is just wrong (unless the data are independent), and our simulations will have to be more complicated.

25.5.1 Parametric or Model-Based Bootstrap

Conceptually, the simplest situation is when we fit a full, generative model — something which we could step through to generate a new time series. If we are confident in the model specification, then we can bootstrap by, in fact, simulating from the fitted model. This is the parametric bootstrap we saw in Chapter 6.

25.5.2 Block Bootstraps

Simple resampling won't work, because it destroys the dependence between successive values in the time series. There is, however, a clever trick which does work, and is almost as simple. Take the full time series x_1^n and divide it up into overlapping blocks of length k , so $x_1^k, x_2^{k+1}, \dots, x_{n-k+1}^n$. Now draw $m = n/k$ of these *blocks* with replacement⁸, and set them down in order. Call the new time series \tilde{x}_1^n .

Within each block, we have preserved *all* of the dependence between observations. It's true that successive observations are now completely independent, which generally wasn't true of the original data, so we're introducing some inaccuracy, but we're certainly coming closer than just resampling individual observations (which would be $k = 1$). Moreover, we can make this inaccuracy smaller and smaller by letting k grow as n grows. One can show⁹ that the optimal $k = O(n^{1/3})$; this gives a growing number ($O(n^{2/3})$) of increasingly long blocks, capturing more and more of the dependence. (We will consider how exactly to pick k in the next chapter.)

The block bootstrap scheme is extremely clever, and has led to a great many variants. Three in particular are worth mentioning.

1. In the **circular block bootstrap** (or **circular bootstrap**), we "wrap the time series around a circle", so that it goes $x_1, x_2, \dots, x_{n_1}, x_n, x_1, x_2, \dots$. We then sample the n blocks of length k this gives us, rather than the merely $n - k$ blocks of the simple block bootstrap. This makes better use of the information we have about dependence on distances $< k$.
2. In the **block-of-blocks bootstrap**, we first divide the series into blocks of length k_2 , and then subdivide each of those into sub-blocks of length $k_1 < k_2$. To generate a new series, we sample blocks with replacement, and then sample

⁸If n/k isn't a whole number, round.

⁹I.e., I will not show.

t	x	t	\tilde{x}
1821	269	lag2	
1822	321	321	585
1823	585	585	871
1824	871	871	1475
1825	1475	1475	2821
1826	2821	2821	3928
1827	3928	3928	5943
1828	5943		
		\Rightarrow	
lag2			
269			
\Rightarrow			
871			
585			
			\Rightarrow
1821			1821
1822			321
1823			585
1824			871
1825			1475
1826			2821
1827			585
1828			871

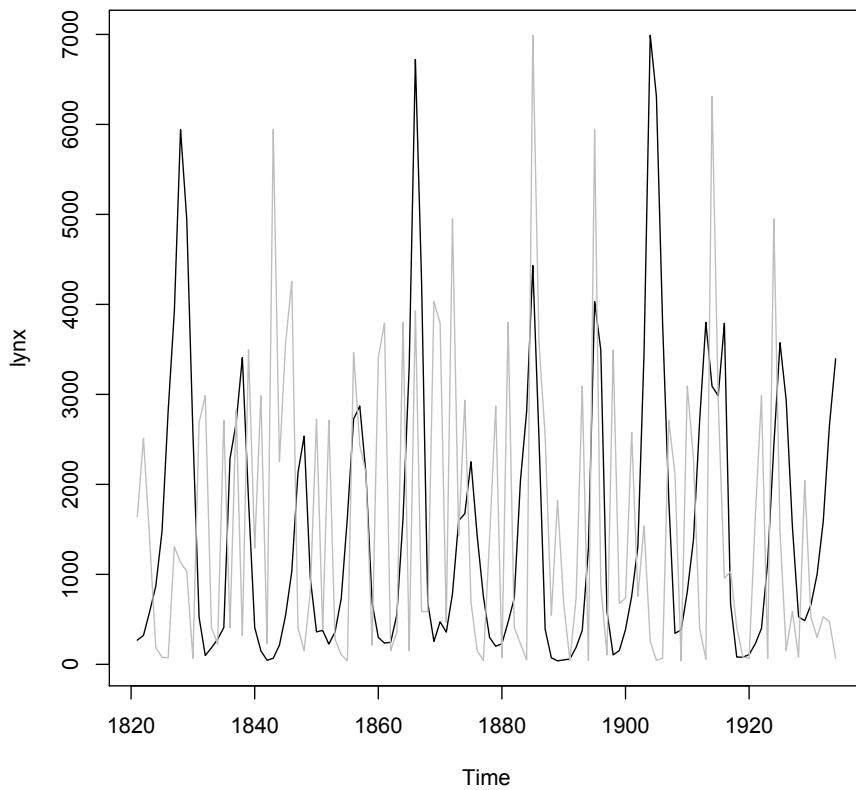
Figure 25.11: Scheme for block bootstrapping: turn the time series (here, the first eight years of lynx) into blocks of consecutive values; randomly resample enough of these blocks to get a series as long as the original; then string the blocks together in order. See `rblockboot` online for code.

sub-blocks within each block with replacement. This gives a somewhat better idea of longer-range dependence, though we have to pick two block-lengths.

3. In the **stationary bootstrap**, the length of each block is random, chosen from a geometric distribution of mean k . Once we have chosen a sequence of block lengths, we sample the appropriate blocks with replacement. The advantage of this is that the ordinary block bootstrap doesn't quite give us a stationary time series. (The distribution of X_{k-1}^k is not the same as the distribution of X_k^{k+1} .) Averaging over the random choices of block lengths, the stationary bootstrap does. It tends to be slightly slower to converge than the block or circular bootstrap, but there are some applications where the surrogate data really needs to be strictly stationary.

25.5.3 Sieve Bootstrap

A compromise between model-based and non-parametric bootstraps is to use a **sieve bootstrap**. This also simulates from models, but we don't really believe in them; rather, we just want them to be reasonable easy to fit and simulate, yet flexible enough that they can capture a wide range of processes if we just give them enough capacity.



```
plot(lynx)
lines(1821:1934, rblockboot(lynx,4), col="grey")
```

Figure 25.12: The lynx time series, and one run of resampling it with a block bootstrap, block length = 4. (See online for the code to `rblockboot`.)

We then (slowly) let them get more complicated as we get more data¹⁰. One popular choice is to use linear AR(p) models, and let p grow with n — but there is nothing special about linear AR models, other than that they are very easy to fit and simulate from. Additive autoregressive models, for instance, would often work at least as well.

25.6 Trends and De-Trending

The sad fact is that a lot of important real time series are not even approximately stationary. For instance, Figure 25.13 shows US national income per person (adjusted for inflation) over the period from 1952 (when the data begins) until now. It is *possible* that this is sample from a stationary process. But in that case, the correlation time is evidently much longer than 50 years, on the order of centuries, and so the theoretical stationarity is irrelevant for anyone but a very ambitious quantitative historian.

More sensibly, we should try to treat data like this as a non-stationary time series. The conventional approach is to try to separate time series like this into a persistent **trend**, and stationary **fluctuations** (or **deviations**) around the trend,

$$\begin{aligned} Y_t &= X_t + Z_t \\ \text{series} &= \text{fluctuations} + \text{trend} \end{aligned} \tag{25.35}$$

Since we could add or subtract a constant to each X_t without changing whether they are stationary, we'll stipulate that $E[X_t] = 0$, so $E[Y_t] = E[Z_t]$. (In other situations, the decomposition might be multiplicative instead of additive, etc.) How might we find such a decomposition?

If we have multiple independent realizations $Y_{i,t}$ of the same process, say m of them, and they all have the same trend Z_t , then we can try to find the common trend by averaging the time series:

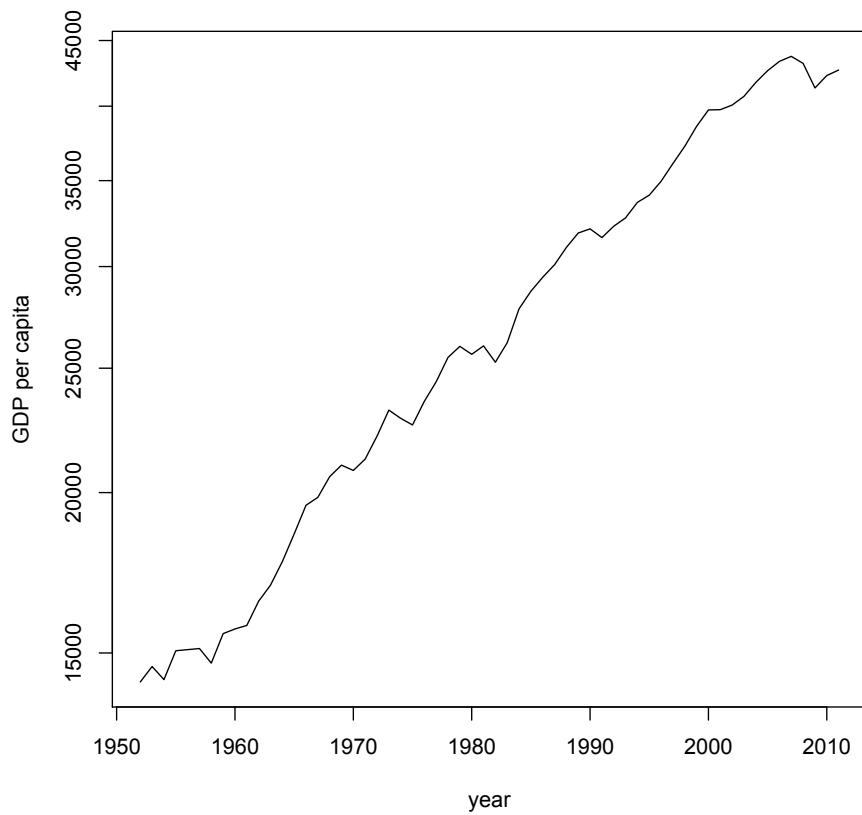
$$Z_t = E[Y_{\cdot,t}] \approx \sum_{i=1}^m Y_{i,t} \tag{25.36}$$

Multiple time series with the same trend do exist, especially in the experimental sciences. $Y_{i,t}$ might be the measurement of some chemical in a reactor at time t in the i^{th} repetition of the experiment, and then it would make sense to average the $Y_{i,t}$ to get the common Z_t trend, the average trajectory of the chemical concentration. One can tell similar stories about experiments in biology or even psychology, though those are complicated by the tendency of animals to get tired and to learn¹¹.

For better or for worse, however, we have only one realization of the post-WWII US economy, so we can't average multiple runs of the experiment together. If we have a theoretical model of the trend, we can try to fit that model. For instance,

¹⁰This is where the metaphor of the “sieve” comes in: the idea is that the mesh of the sieve gets finer and finer, catching more and more subtle features of the data.

¹¹Even if we do have multiple independent experimental runs, it is very important to get them aligned in time, so that $Y_{i,t}$ and $Y_{j,t}$ refer to the *same* point in time relative to the start of the experiment; otherwise, averaging them is just mush. It can also be important to ensure that the initial state, before the experiment, is the same for every run.



```
gdppc <- read.csv("gdp-pc.csv")
gdppc$y <- gdppc$y*1e6
plot(gdppc,log="y",type="l",ylab="GDP per capita")
```

Figure 25.13: US GDP per capita, constant dollars (consumer price index deflator). Note logarithmic scale of vertical axis. (The values were initially recorded in the file in millions of dollars per person per year, hence the correction.)

some (simple) models of economic growth predict that series like the one in Figure 25.13 should, on average, grow at a steady exponential rate¹². We could then estimate Z_t by fitting a model to Y_t of the form $\beta_0 e^{\beta t}$, or even by doing a linear regression of $\log Y_t$ on t . The fluctuations X_t are then taken to be the residuals of this model.

If we only have one time series (no replicates), and we don't have a good theory which tells us what the trend should be, we fall back on curve fitting. In other words, we regress Y_t on t , call the fitted values Z_t , and call the residuals X_t . This is frankly rests more on hope than on theorems. The hope is that the characteristic time-scale for the fluctuations X_t (say, their correlation time τ) is short compared to the characteristic time-scale for the trend Z_t ¹³. Then if we average Y_t over a band-width which is large compared to τ , but small compared to the scale of Z_t , we should get something which is mostly Z_t — there won't be too much bias from averaging, and the fluctuations should mostly cancel out.

Once we have the fluctuations, and are reasonably satisfied that they're stationary, we can model them like any other stationary time series. Of course, to actually make predictions, we need to extrapolate the trend, which is a harder business.

25.6.1 Forecasting Trends

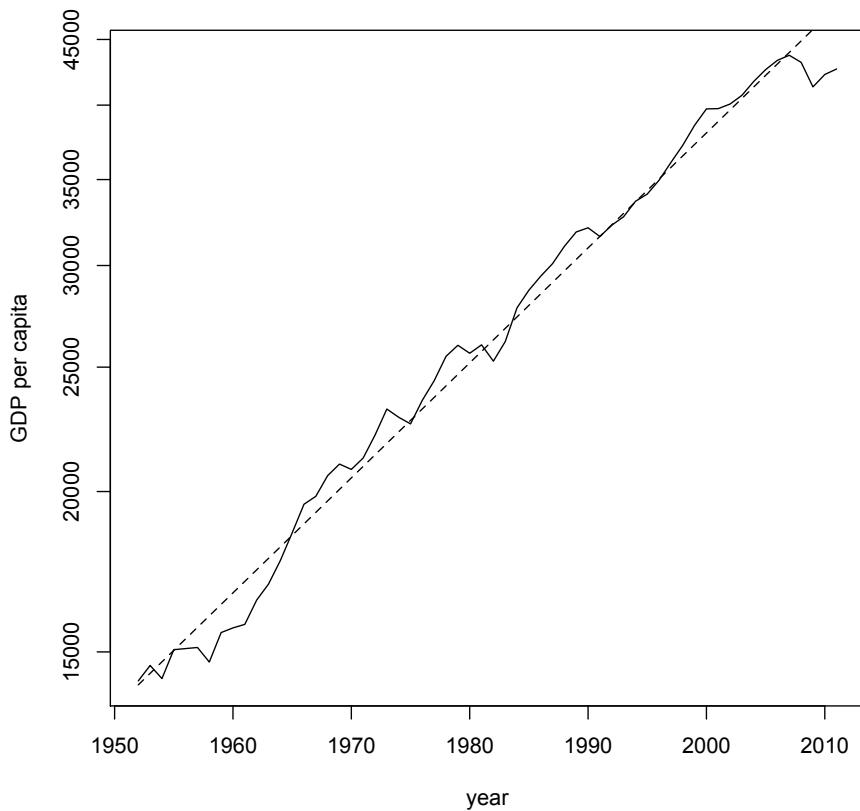
The problem with making predictions when there is a substantial trend is that it is usually hard to know how to continue or extrapolate the trend beyond the last data point. If we are in the situation where we have multiple runs of the same process, we can at least extrapolate up to the limits of the different runs. If we have an actual model which tells us that the trend should follow a certain functional form, and we've estimated that model, we can use it to extrapolate. But if we have found the trend purely through curve-fitting, we have a problem.

Suppose that we've found the trend by spline smoothing, as in Figure 25.16. The fitted spline model will cheerfully make predictions for the what the trend of GDP per capita will be in, say, 2252, far outside the data. This will be a simple linear extrapolation, because splines are always linear outside the data range (Chapter 8, p. 178). This is just because of the way splines are set up, not because linear extrapolation is such a good idea. Had we used kernel regression, or any of many other ways of fitting the curve, we'd get different extrapolations. People in 2252 could look back and see whether the spline had fit well, or some other curve would have done better. (But why would they want to?) Right now, if *all* we have is curve-fitting, we are in a dubious position even as regards 2013, never mind 2252¹⁴

¹²This is not *quite* what is claimed by Solow (1970), which you should read anyway if this kind of question is at all interesting to you.

¹³I am being deliberately vague about what “the characteristic time scale of Z_t ” means. Intuitively, it's the amount of time required for Z_t to change substantially. You might think of it as something like $n^{-1} \sum_{t=1}^{n-1} 1/|Z_{t+1} - Z_t|$, if you promise not to treat that too seriously. Trying to get an exact statement of what's involved in identifying trends requires being very precise, and getting into topics at the intersection of statistics and functional analysis which are beyond the scope of this class.

¹⁴Yet again, we hit a basic philosophical obstacle, which is the problem of induction. We have so far evaded it, by assuming that we're dealing with IID or a stationary probability distribution; these assumptions let us *deductively* extrapolate from past data to future observations, with more or less confidence. (For more on this line of thought, see Hacking (2001); Spanos (2011); Gelman and Shalizi (2013).) If we



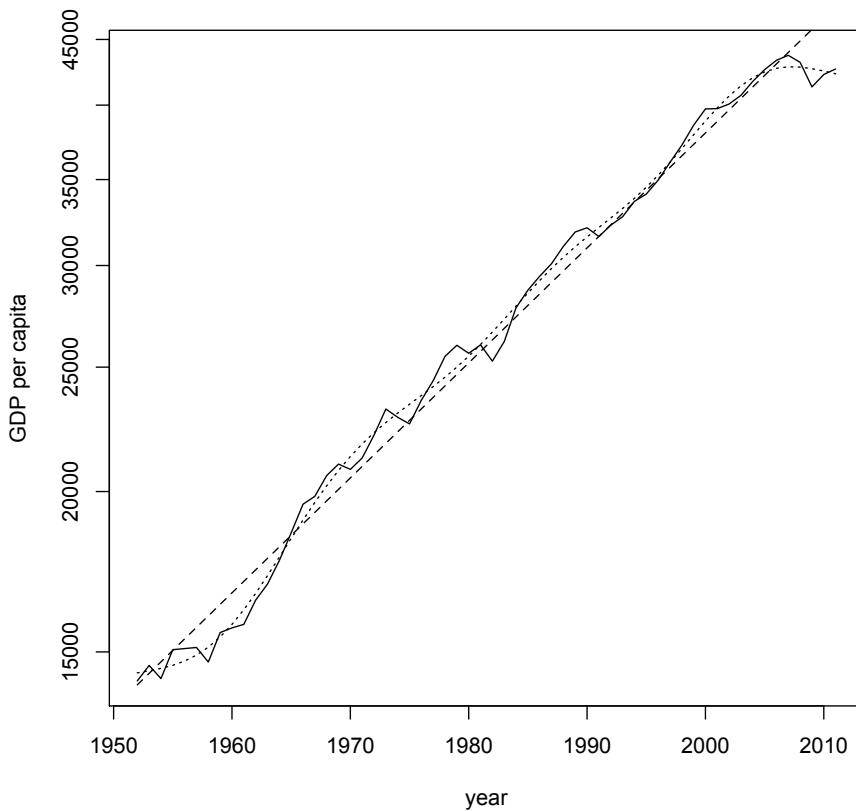
```
gdppc.exp <- lm(log(y) ~ year,data=gdppc)
beta0 <- exp(coefficients(gdppc.exp)[1])
beta <- coefficients(gdppc.exp)[2]
curve(beta0*exp(beta*x),lty="dashed",add=TRUE)
```

Figure 25.14: As in Figure 25.13, but with an exponential trend fitted.



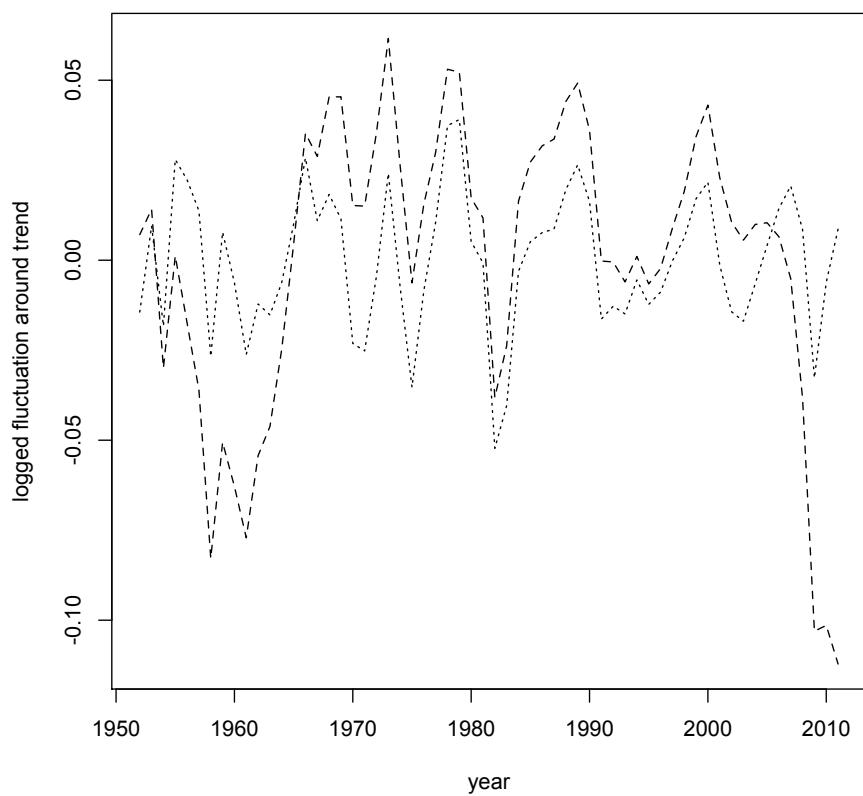
```
plot(gdppc$year,residuals(gdppc.exp),xlab="year",
     ylab="logged fluctuation around trend",type="l",lty="dashed")
```

Figure 25.15: The hopefully-stationary fluctuations around the exponential growth trend in Figure 25.14. Note that these are $\log \frac{Y_t}{\hat{\beta}_0 e^{\hat{\beta} t}}$, and so unitless.



```
gdp.spline <- fitted(gam(y~s(year), data=gdppc))
lines(gdppc$year, gdp.spline, lty="dotted")
```

Figure 25.16: Figure 25.14, but with the addition of a spline curve for the time trend (dotted line). This is, perhaps unsurprisingly, not all *that* different from the simple exponential-growth trend.



```
lines(gdppc$year,log(gdppc$y/gdp.spline),xlab="year",
      ylab="logged fluctuations around trend",lty="dotted")
```

Figure 25.17: Adding the logged deviations from the spline trend (dotted) to Figure 25.15.

25.6.2 Seasonal Components

Sometimes we know that time series contain components which repeat, pretty exactly, over regular periods. These are called **seasonal** components, after the obvious example of trends which cycle each year with the season. But they could cycle over months, weeks, days, etc.

The decomposition of the process is thus

$$Y_t = X_t + Z_t + S_t \quad (25.37)$$

where X_t handles the stationary fluctuations, Z_t the long-term trends, and S_t the repeating seasonal component.

If $Z_t = 0$, or equivalently if we have a good estimate of it and can subtract it out, we can find S_t by averaging over multiple cycles of the seasonal trend. Suppose that we know the period of the cycle is T , and we can observe $m = n/T$ full cycles. Then

$$S_t \approx \frac{1}{m} \sum_{j=0}^{m-1} Y_{t+jT} \quad (25.38)$$

This works because, with Z_t out of the picture, $Y_t = X_t + S_t$, and S_t is periodic, $S_t = S_{t+T}$. Averaging over multiple cycles, the stationary fluctuations tend to cancel out (by the ergodic theorem), but the seasonal component does not.

For this trick to work, we need to know the period. If the true $T = 355$, but we use $T = 365$ without thinking¹⁵, we can get mush.

We also need to know the over-all trend. Of course, if there are seasonal components, we really ought to subtract them out before trying to find Z_t . So we have yet another vicious cycle, or, more optimistically, another case for iterative approximation.

25.6.3 Detrending by Differencing

Suppose that Y_t has a linear time trend:

$$Y_t = \beta_0 + \beta t + X_t \quad (25.39)$$

with X_t stationary. Then if we take the difference between successive values of Y_t , the trend goes away:

$$Y_t - Y_{t-1} = \beta + X_t - X_{t-1} \quad (25.40)$$

Since X_t is stationary, $\beta + X_t - X_{t-1}$ is also stationary. Taking differences has removed the trend.

Differencing will not only get rid of linear time trends. Suppose that

$$Z_t = Z_{t-1} + \epsilon_t \quad (25.41)$$

assume a certain form or model for the trend, then again we can deduce future behavior on that basis. But if we have neither probabilistic nor mechanistic assumptions, we are, to use a technical term, stuck with induction. Whether there is some principle which might help — perhaps a form of Occam's Razor (Kelly, 2007)? — is a nice question.

¹⁵Exercise: come up with an example of a time series where the periodicity *should* be 355 days.

where the “innovations” or “shocks” ϵ_t are IID, and that

$$Y_t = Z_t + X_t \quad (25.42)$$

with X_t stationary, and independent of the ϵ_t . It is easy to check that (i) Z_t is not stationary (Exercise 2), but that (ii) the first difference

$$Y_t - Y_{t-1} = \epsilon_t + X_t - X_{t-1} \quad (25.43)$$

is stationary. So differencing can get rid of trends which are built out of the summation of *persistent* random shocks.

This gives us another way of making a time series stationary: instead of trying to model the time trend, take the difference between successive values, and see if that is stationary. (The `diff()` function in R does this; see Figure 25.18.) If such “first differences” don’t look stationary, take differences among differences, third differences, etc., until you have something satisfying.

Notice that now we can continue to the trend: once we predict $Y_{t+1} - Y_t$, we add it on to Y_t (which we observed) to get Y_{t+1} .

Differencing is like taking the discrete version of a derivative. It will eventually get rid of trends if they correspond to curves (e.g., polynomials) with only finitely many non-zero derivatives. It fails for trends which aren’t like that, like exponentials or sinusoids, though you can hope that eventually the higher differences are small enough that they don’t matter much.

25.7 Further Reading

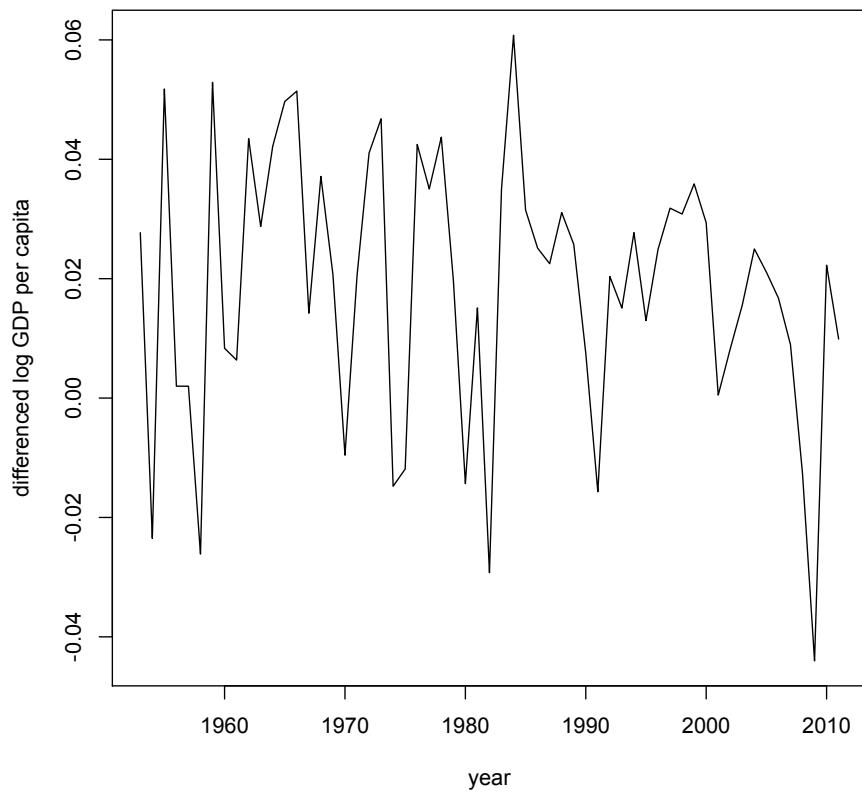
Shumway and Stoffer (2000) is a good introduction to conventional time series analysis, covering R practicalities. Lindsey (2004) surveys a broader range of situations in less depth; it is readable, but opinionated, and I don’t always agree with the opinions. Fan and Yao (2003) is a deservedly-standard reference on nonparametric time series models. The theoretical portions would be challenging for most readers of this book, but the methodology isn’t, and it devotes about the right amount of space (no more than a quarter of the book) to the usual linear-model theory.

The block bootstrap was introduced by Künsch (1989). Davison and Hinkley (1997, §8.2) has a characteristically-clear treatment of the main flavors of bootstrap for time series; Lahiri (2003) is a thorough but theoretical. Bühlmann (2002) is also useful.

The best introduction to stochastic processes I know of, by a very wide margin, is Grimmett and Stirzaker (1992). However, like most textbooks on stochastic processes, it says next to nothing about how to use them as models of data. Two exceptions I can recommend are the old but insightful Bartlett (1955), and the excellent Guttorp (1995).

The basic ergodic theorem in §25.2.2.1 follows a continuous-time argument in Frisch (1995). My general treatment of ergodicity is heavily shaped by Gray (1988) and Shields (1996).

In parallel to the treatment of time series by statisticians, physicists and mathematicians developed their own tradition of time-series analysis (Packard *et al.*, 1980),



```
plot(gdppc$year[-1],diff(log(gdppc$y)),type="l",xlab="year",
     ylab="differenced log GDP per capita")
```

Figure 25.18: First differences of log GDP per capita, i.e., the year-to-year growth rate of GDP per capita.

where the basic models are not stochastic processes but deterministic, yet unstable, dynamical systems. Perhaps the best treatment of this are Abarbanel (1996); Kantz and Schreiber (2004). There are in fact very deep connections between this approach and the question of why probability theory works in the first place (Ruelle, 1991), but that's not a subject for *data analysis*.

25.8 Exercises

1. In Eq. 25.33, assume that $m(x)$ has to be a linear function, $m(x) = \beta \cdot x$. Solve for the optimal β in terms of \mathbf{y} , \mathbf{x} , and Γ . This “generalized least squares” (GLS) solution should reduce to ordinary least squares when $\Gamma = \sigma^2 \mathbf{I}$.
2. If $Z_t = Z_{t-1} + \epsilon_t$, with ϵ_t IID, prove that Z_t is not stationary. *Hint:* consider $\text{Var}[Z_t]$.

14:41 Thursday 25th April, 2013

Chapter 26

Time Series with Latent Variables

[[To come]]

Chapter 27

Longitudinal, Spatial and Network Data

[[To come]]

[[Data arranged in space: spatial autocorrelation, spatial smoothing ("kriging"; also Kafadar 1996). Data arranged on a network; network smoothing. Multiple time series: the distinction of "Granger causality" and the possibility of real causality.]]

14:41 Thursday 25th April, 2013

Appendices

[[Appendix to come: helpful facts from linear algebra about special matrices, eigen-decompositions, and orthonormal bases]]

[[Appendix to come: optimization by Newton's method (from logistic regression chapter)]]

[[Appendix to come: asymptotics of estimation by optimization, especially of maximum likelihood]]

Appendix A

Writing R Functions

The ability to read, understand, modify and write simple pieces of code is an essential skill for modern data analysis. Lots of high-quality software already exists for specific purposes, which you can and should use, but statisticians need to grasp how such software works, tweak it to suit their needs, recombine existing pieces of code, and when needed create their own tools. Someone who just knows how to run canned routines is not a data analyst but a technician who tends a machine they do not understand.

Fortunately, writing code is not actually very hard, especially not in R. All it demands is the discipline to think logically, and the patience to practice. This chapter tries to illustrate what's involved, starting from the very beginning. It is redundant for many students, but included through popular demand.

A.1 Functions

Programming in R is organized around **functions**. You all know what a mathematical function is, like $\log x$ or $\phi(z)$ or $\sin \theta$: it is a rule which takes some **inputs** and delivers a definite **output**. A function in R, like a mathematical function, takes zero or more inputs, also called **arguments**, and **returns** an output. The output is arrived at by going through a series of calculations, based on the input, which we specify in the body of the function. As the computer follows our instructions, it may do other things to the system; these are called **side-effects**. (The most common sort of side-effect, in R, is probably making or updating a plot on the screen.) The basic **declaration** or **definition** of a function looks like so:

```
my.function <- function(argument.1, argument.2, ...) {
  # clever manipulations of arguments
  return(the.return.value)
}
```

Strictly speaking, we often don't need the `return()` command; without it, the function will return the last thing it evaluated. But it's usually clearer, and never hurts, to be explicit.

We write functions because we often find ourselves going through the same sequence of steps at the command line, perhaps with small variations. It saves mental effort on our part to take that sequence and bind it together into an integrated procedure, the function, so that then we can think about the function as a whole, rather than the individual steps. It also reduces error, because, by invoking the same function every time, we don't have to worry about missing a step, or wondering whether we forgot to change the third step to be consistent with the second, and so on.

A.2 First Example: Pareto Quantiles

Let me give a really concrete example. In Chapter 6, I mentioned the **Pareto distribution**, which has the probability density function

$$f(x; \alpha, x_0) = \begin{cases} \frac{\alpha-1}{x_0} \left(\frac{x}{x_0}\right)^{-\alpha} & x \geq x_0 \\ 0 & x < x_0 \end{cases} \quad (\text{A.1})$$

Consequently, the CDF is

$$F(x; \alpha, x_0) = 1 - \left(\frac{x}{x_0}\right)^{-\alpha+1} \quad (\text{A.2})$$

and the quantile function is

$$Q(p; \alpha, x_0) = x_0(1-p)^{-\frac{1}{\alpha-1}} \quad (\text{A.3})$$

Say I want to find the median of a Pareto distribution with $\alpha = 2.34$ and $x_0 = 6 \times 10^8$. I can do that:

```
> 6e8 * (1-0.5)^(-1/(2.33-1))
[1] 1010391288
```

If I decide I want the 40th percentile of the same distribution, I can do that:

```
> 6e8 * (1-0.4)^(-1/(2.33-1))
[1] 880957225
```

If I decide to raise the exponent to 2.5, lower the threshold to 1×10^6 , and ask about the 92nd percentile, I can do that, too:

```
> 1e6 * (1-0.92)^(-1/(2.5-1))
[1] 5386087
```

But doing this all by hand gets quite tiresome, and at some point I'm going to mess up and write when I meant \wedge . I'll write a function to do this for me, and so that there is only *one* place for me to make a mistake:

```
qpareto.1 <- function(p, exponent, threshold) {
  q <- threshold*((1-p)^(-1/(exponent-1)))
  return(q)
}
```

The name of the function is what goes on the left of the assignment `<-`, with the declaration (beginning `function`) on the right. (I called this `qpareto.1` to distinguish it from later modifications.) The three terms in the parenthesis after `function` are the arguments to `qpareto` — the inputs it has to work with. The body of the function is just like some R code we would type into the command line, after assigning values to the arguments. The very last line tells the function, explicitly, what its output or return value should be. Here, of course, the body of the function calculates the p th quantile of the Pareto distribution with the exponent and threshold we ask for.

When I enter the code above, defining `qpareto.1`, into the command line, R just accepts it without outputting anything. It thinks of this as assigning certain value to the name `qpareto.1`, and it doesn't produce outputs for assignments when they succeed, just as if I'd said `alpha <- 2.5`.

All that successfully creating a function means, however, is that we didn't make a huge error in the syntax. We should still check that it works, by invoking the function with values of the arguments where we know, by other means, what the output should be. I just calculated three quantiles of Pareto distributions above, so let's see if we can reproduce them.

```
> qpareto.1(p=0.5,exponent=2.33,threshold=6e8)
[1] 1010391288
> qpareto.1(p=0.4,exponent=2.33,threshold=6e8)
[1] 880957225
> qpareto.1(p=0.92,exponent=2.5,threshold=1e6)
[1] 5386087
```

So, our first function seems to work successfully.

A.3 Functions Which Call Functions

If we examine other quantile functions (e.g., `qnorm`), we see that most of them take an argument called `lower.tail`, which controls whether p is a probability from the lower tail or the upper tail. `qpareto.1` implicitly assumes that it's the lower tail, but let's add the ability to change this.

```
qpareto.2 <- function(p, exponent, threshold, lower.tail=TRUE) {
  if(lower.tail==FALSE) {
    p <- 1-p
  }
  q <- threshold*((1-p)^(-1/(exponent-1)))
  return(q)
}
```

When, in a function declaration, an argument is followed by `=` and an expression, the expression sets the **default value** of the argument, the one which will be used unless explicitly over-ridden. The default value of `lower.tail` is `TRUE`, so, unless it is explicitly set to false, we will assume p is a probability counted from $-\infty$ on up.

The `if` command is a **control structure** — if the condition in parenthesis is true, then the commands in the following braces will be executed; if not, not. Since lower tail probabilities plus upper tail probabilities must add to one, if we are given an upper tail probability, we just find the lower tail probability and proceed as before.

Let's try it:

```
> qpareto.2(p=0.5,exponent=2.33,threshold=6e8,lower.tail=TRUE)
[1] 1010391288
> qpareto.2(p=0.5,exponent=2.33,threshold=6e8)
[1] 1010391288
> qpareto.2(p=0.92,exponent=2.5,threshold=1e6)
[1] 5386087
> qpareto.2(p=0.5,exponent=2.33,threshold=6e8,lower.tail=FALSE)
[1] 1010391288
> qpareto.2(p=0.92,exponent=2.5,threshold=1e6,lower.tail=FALSE)
[1] 1057162
```

First: the answer `qpareto.2` gives with `lower.tail` explicitly set to true matches what we already got from `qpareto.1`. Second and third: the default value for `lower.tail` works, and it works for two different values of the other arguments. Fourth and fifth: setting `lower.tail` to FALSE works properly (since the 50th percentile is the same from above or from below, but the 92nd percentile is different, and smaller from above than from below).

The function `qpareto.2` is equivalent to this:

```
qpareto.3 <- function(p, exponent, threshold, lower.tail=TRUE) {
  if(lower.tail==FALSE) {
    p <- 1-p
  }
  q <- qpareto.1(p, exponent, threshold)
  return(q)
}
```

When R tries to execute this, it will look for a function named `qpareto.1` in the workspace. If we have already defined such a function, then R will execute it, with the arguments we have provided, and `q` will become whatever is returned by `qpareto.1`. When we give R the above function definition for `qpareto.3`, it does not check whether `qpareto.1` exists — it only has to be there at run time. If `qpareto.1` changes, then the behavior of `qpareto.3` will change with it, *without our having to redefine qpareto.3*.

This is *extremely useful*. It means that we can take our programming problem and sub-divide it into smaller tasks efficiently. If I made a mistake in writing `qpareto.1`, when I fix it, `qpareto.3` automatically gets fixed as well — along with any other function which calls `qpareto.1`, or `qpareto.3` for that matter. If I discover a more efficient way to calculate the quantiles and modify `qpareto.1`, the improvements are likewise passed along to everything else. But when I *write* `qpareto.3`, I don't have to worry about how `qpareto.1` works, I can just assume it does what I need somehow.

A.3.1 Sanity-Checking Arguments

It is good practice, though not *strictly* necessary, to write functions which check that their arguments make sense before going through possibly long and complicated calculations. For the Pareto quantile function, for instance, p must be in $[0, 1]$, the exponent α must be at least 1, and the threshold x_0 must be positive, or else the mathematical function just doesn't make sense.

Here is how to check all these requirements:

```
qpareto.4 <- function(p, exponent, threshold, lower.tail=TRUE) {
  stopifnot(p >= 0, p <= 1, exponent > 1, threshold > 0)
  q <- qpareto.3(p,exponent,threshold,lower.tail)
  return(q)
}
```

The function `stopifnot` halts the execution of the function, with an error message, if all of its arguments do not evaluate to TRUE. If all those conditions are met, however, R just goes on to the next command, which here happens to be running `qpareto.3`. Of course, I could have written the checks on the arguments directly into the latter.

Let's see this in action:

```
> qpareto.4(p=0.5,exponent=2.33,threshold=6e8,lower.tail=TRUE)
[1] 1010391288
> qpareto.4(p=0.92,exponent=2.5,threshold=1e6,lower.tail=FALSE)
[1] 1057162
> qpareto.4(p=1.92,exponent=2.5,threshold=1e6,lower.tail=FALSE)
Error: p <= 1 is not TRUE
> qpareto.4(p=-0.02,exponent=2.5,threshold=1e6,lower.tail=FALSE)
Error: p >= 0 is not TRUE
> qpareto.4(p=0.92,exponent=0.5,threshold=1e6,lower.tail=FALSE)
Error: exponent > 1 is not TRUE
> qpareto.4(p=0.92,exponent=2.5,threshold=-1,lower.tail=FALSE)
Error: threshold > 0 is not TRUE
> qpareto.4(p=-0.92,exponent=2.5,threshold=-1,lower.tail=FALSE)
Error: p >= 0 is not TRUE
```

The first two lines give the same results as our earlier functions — as they should, because all the arguments are in the valid range. The third, fourth, fifth and sixth lines all show that `qpareto.4` stops with an error message when one of the conditions in the `stopifnot` is violated. Notice that the error message says *which* condition was violated. The seventh line shows one limitation of this: the arguments violate *two* conditions, but `stopifnot`'s error message will only mention the *first* one. (What is the other violation?)

A.4 Layering Functions and Debugging

Functions can call functions which call functions, and so on indefinitely. To illustrate, I'll write a function which generates Pareto-distributed random numbers, using

the “quantile transform” method from Lecture 7. This, remember, is to generate a uniform random number U on $[0, 1]$, and produce $Q(U)$, with Q being the quantile function of the desired distribution.

The first version contains a deliberate bug, which I will show how to track down and fix.

```
rpareto <- function(n, exponent, threshold) {
  x <- vector(length=n)
  for (i in 1:n) {
    x[i] <- qpareto.4(p=rnorm(1), exponent=exponent, threshold=threshold)
  }
  return(x)
}
```

Notice that this calls `qpareto.4`, which calls `qpareto.3`, which calls `qpareto.1`.

Let's this out:

```
> rpareto(10)
Error in exponent > 1 : 'exponent' is missing
```

This is a puzzling error message — the expression `exponent > 1` never appears in `rpareto!` The error is coming from further down the chain of execution. We can see where it happens by using the `traceback()` function, which gives the chain of function calls leading to the latest error:

```
> rpareto(10)
Error in exponent > 1 : 'exponent' is missing
> traceback()
3: stopifnot(p >= 0, p <= 1, exponent > 1, threshold > 0)
2: qpareto.4(p = rnorm(1), exponent = exponent, threshold = threshold)
1: rpareto(10)
```

`traceback()` outputs the sequence of function calls leading up to the error in reverse order, so that the last line, numbered 1, is what we actually entered on the command line. This tells us that the error is happening when `qpareto.4` tries to check the arguments to the quantile function. And the reason it is happening is that we are not providing `qpareto.4` with any value of `exponent`. And the reason *that* is happening is that we didn't give `rpareto` any value of `exponent` as an explicit argument when we called it, and our definition didn't set a default.

Let's try this again.

```
> rpareto(n=10, exponent=2.5, threshold=1)
Error: p <= 1 is not TRUE
> traceback()
4: stop(paste(ch, " is not ", if (length(r) > 1L) "all ", "TRUE",
           sep = ""), call. = FALSE)
3: stopifnot(p >= 0, p <= 1, exponent > 1, threshold > 0)
2: qpareto.4(p = rnorm(1), exponent = exponent, threshold = threshold)
1: rpareto(n = 10, exponent = 2.5, threshold = 1)
```

This is progress! The `stopifnot` in `qpareto.4` is at least able to evaluate all the conditions — it just happens that one of them is false. (The line 4 here comes from the internal workings of `stopifnot`.) The problem, then, is that `qpareto.4` is being passed a negative value of `p`. This tells us that the problem is coming from the part of `rpareto.1` which sets `p`. Looking at that,

```
p = rnorm(1)
```

the culprit is obvious: I stupidly wrote `rnorm`, which generates a *Gaussian* random number, when I meant to write `runif`, which generates a *uniform* random number.¹

The obvious fix is just to replace `rnorm` with `runif`

```
rpareto <- function(n,exponent,threshold) {
  x <- vector(length=n)
  for (i in 1:n) {
    x[i] <- qpareto.4(p=runif(1),exponent=exponent,threshold=threshold)
  }
  return(x)
}
```

Let's see if this is enough to fix things, or if I have any other errors:

```
> rpareto(n=10,exponent=2.5,threshold=1)
[1] 1.000736 2.764087 2.775880 1.058910 1.061712 2.142950 4.220731
[8] 1.496793 3.004766 1.194545
```

This function at least produces numerical return values rather than errors! Are they the right values?

We can't expect a random number generator to always give the same results, so I can't cross-check this function against direct calculation, the way I could check `qpareto.1`. (Actually, one way to check a random number generator is to make sure it *doesn't* give identical results when run twice!) It's at least encouraging that all the numbers are above `threshold`, but that's not much of a test. However, since this *is* a random number generator, if I use it to produce a lot of random numbers, the quantiles of the output should be close to the theoretical quantiles, which I *do* know how to calculate.

```
> r <- rpareto(n=1e4,exponent=2.5,threshold=1)
> qpareto.4(p=0.5,exponent=2.5,threshold=1)
[1] 1.587401
> quantile(r,0.5)
  50%
1.598253
> qpareto.4(p=0.1,exponent=2.5,threshold=1)
[1] 1.072766
> quantile(r,0.1)
  10%
```

¹I actually made this exact mistake the first time I wrote the function, back in 2004.

```

1.072972
> qpareto.4(p=0.9,exponent=2.5,threshold=1)
[1] 4.641589
> quantile(r,0.9)
 90%
4.526464

```

This looks pretty good. Figure A.1 shows a plot comparing all the theoretical percentiles to the simulated ones, confirming that we didn't just get lucky with choosing particular percentiles above.

A.4.1 More on Debugging

Everyone who writes their own code spends a lot of time debugging². There are some guidelines for making it easier and less painful.

Characterize the Bug We've got a bug when the code we've written won't do what we want. To fix this, it helps a lot to know exactly what error we're seeing. The first step to this is to make the error reproducible. Can we always get the error when re-running the same code and values? If we start the same code in a clean copy of R, does the same thing happen? Once we can reproduce the error, we map its boundaries. How much can we change the inputs and get the same error? A different error? For what inputs (if any) does the bug go away? How big is the error?

Localize the Bug The problem *may* be a diffuse all-pervading wrongness, but often it's a lot more localized, to a few lines or even just one line of code; it helps to know where! We have seen some tools for localizing the bug above: `traceback()` and `stopifnot()`. Another very helpful one is to add `print` statements, so that our function gives us messages about the progress of its calculations, selected variables, etc., as it goes; the `warning` command can be used to much the same effect³.

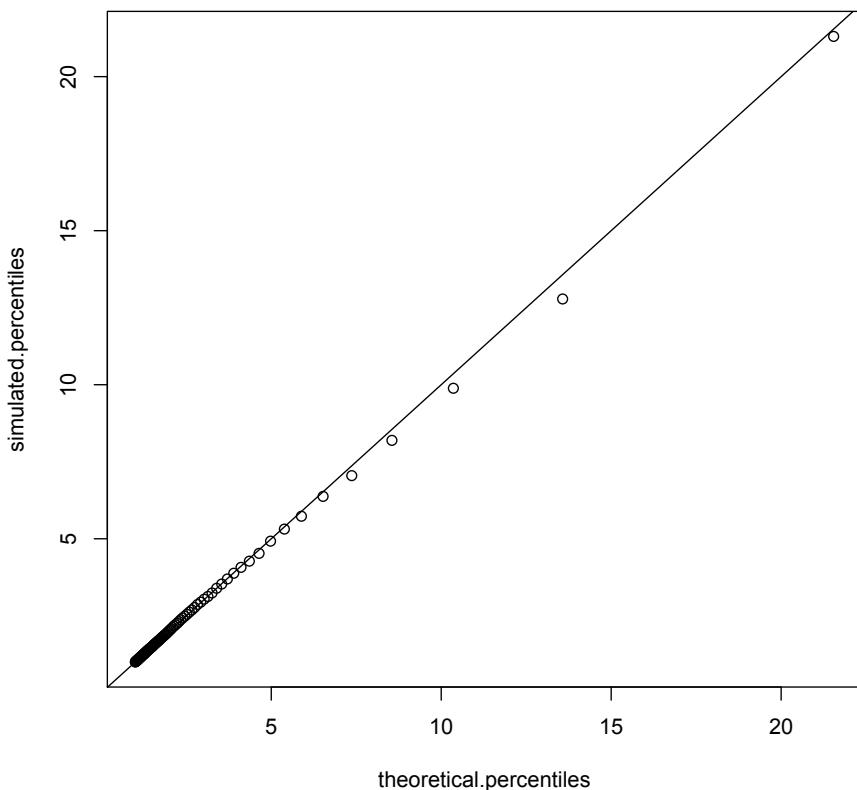
Fix the Bug Once you know what's going wrong and where it's going wrong, it's often not too hard to spot the error, either one of syntax (say `=` vs. `==`) or logic. Try a fix and see if it makes it better. Do the inputs which gave you the bugs before now work properly? Are you getting different errors?

A.5 Automating Repetition and Passing Arguments

The match between the theoretical quantiles and the simulated ones in Figure A.1 is close, but it's not perfect. On the one hand, this might indicate some subtle mistake. On the other hand, it might just be random sampling noise — `rpareto` is supposed

²Those who don't write their own code but use computers anyway spend a lot of time putting up with other people's bugs.

³Real software engineers look down on this, in favor of more sophisticated tools, like interactive debuggers. They have something of a point, but that's usually over-kill for the purposes of this class.



```

simulated.percentiles <- quantile(r,(0:99)/100)
theoretical.percentiles <- qpareto.4((0:99)/100,exponent=2.5,threshold=1)
plot(theoretical.percentiles,simulated.percentiles)
abline(0,1)

```

Figure A.1: Theoretical percentiles of the Pareto distribution with $\alpha = 2.5$, $x_0 = 1$, and empirical percentiles from a sample of 10^4 values simulated from it with the `rpareto` function. (The solid line is the $x = y$ diagonal, for visual reference.)

to be a random number generator, after all. We could check this by seeing whether we get *different* deviations around the line with different runs of `rpareto`, or if on the contrary they all pull in the same direction. We could just make many plots by hand, the way we made that plot by hand, but since we're doing almost exactly the same thing many times, let's write a function.

```
pareto.sim.vs.theory <- function() {
  r <- rpareto(n=1e4, exponent=2.5, threshold=1)
  simulated.percentiles <- quantile(r, (0:99)/100)
  points(theoretical.percentiles, simulated.percentiles)
}
```

This doesn't return anything. All it does is draw a new sample from the same Pareto distribution as before, re-calculate the simulated percentiles, and add them to an existing plot — this is an example of a side-effect. Notice also that the function presumes that `theoretical.percentiles` already exists. (The theoretical percentiles won't need to change from one simulation draw to the next, so it makes sense to only calculate them once.)

Figure A.2 shows how we can use it to produce multiple simulation runs. We can see that, looking over many simulation runs, the quantiles seem to be too large about as often, and as much, as they are too low, which is reassuring.

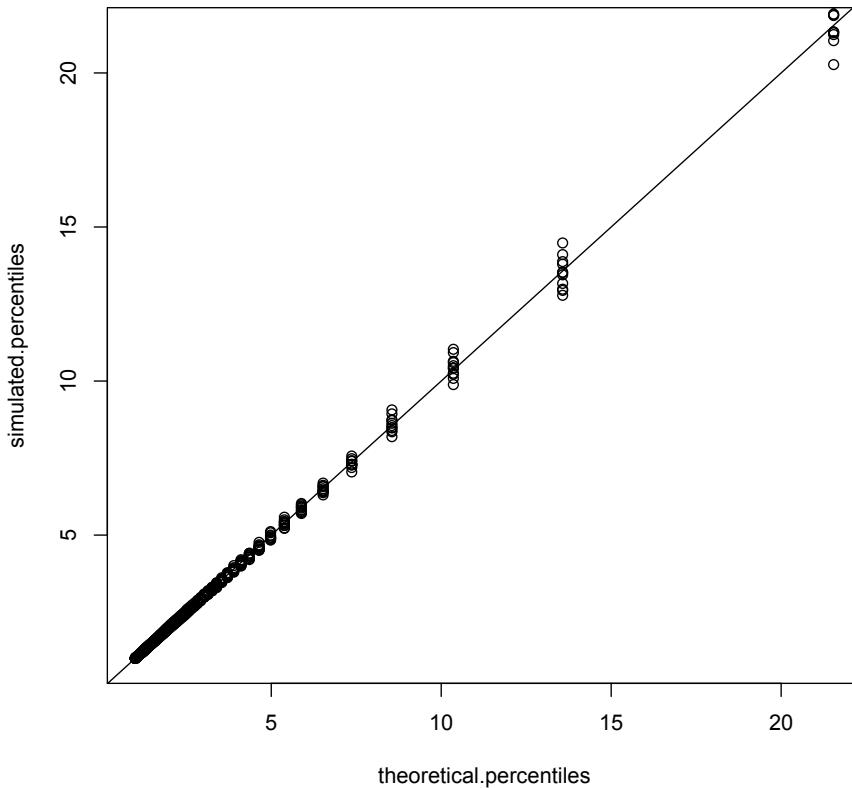
One thing which that figure doesn't do is let us trace the connections between points from the same simulation. More generally, we can't modify the plotting properties, which is kind of annoying. This is easily fixed modifying the function to **pass along arguments**:

```
pareto.sim.vs.theory <- function(...) {
  r <- rpareto(n=1e4, exponent=2.5, threshold=1)
  simulated.percentiles <- quantile(r, (0:99)/100)
  points(theoretical.percentiles, simulated.percentiles, ...)
}
```

Putting the ellipses (...) in the argument list means that we can give `pareto.sim.vs.theory`.*2* an arbitrary collection of arguments, but with the expectation that it will pass them along unchanged to some other function that it will call with ... — here, that's the `points` function. Figure A.3 shows how we can use this, by passing along graphical arguments to `points` — in particular, telling it to connect the points by lines (`type="b"`), varying the shape of the points (`pch=i`) and the line style (`lty=i`).

These figures are reasonably convincing that nothing is going seriously wrong with the simulation for *these* parameter values. To check other parameter settings, again, I could repeat all these steps by hand, or I could write another function:

```
check.rpareto <- function(n=1e4, exponent=2.5, threshold=1, B=10) {
  # One set of percentiles for everything
  theoretical.percentiles <- qpareto.4((0:99)/100, exponent=exponent,
                                         threshold=threshold)
  # Set up plotting window, but don't put anything in it:
```

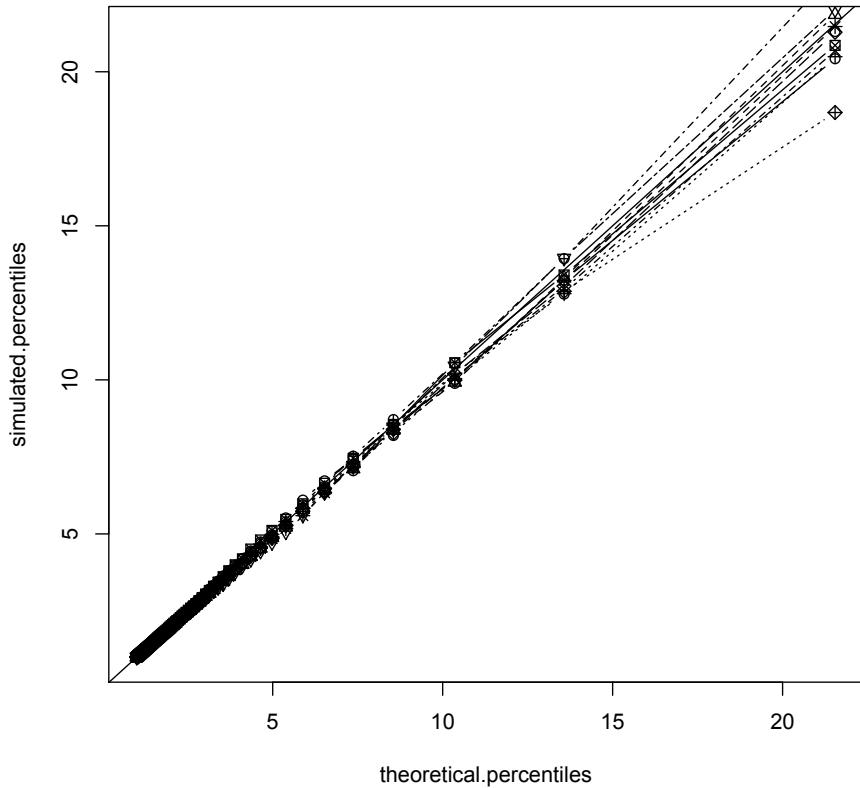


```

simulated.percentiles <- quantile(r,(0:99)/100)
theoretical.percentiles <- qpareto.4((0:99)/100,exponent=2.5,threshold=1)
plot(theoretical.percentiles,simulated.percentiles)
abline(0,1)
for (i in 1:10) {
  pareto.sim.vs.theory()
}

```

Figure A.2: Comparing multiple simulated quantile values to the theoretical quantiles.



```

simulated.percentiles <- quantile(r,(0:99)/100)
theoretical.percentiles <- qpareto.4((0:99)/100,exponent=2.5,threshold=1)
plot(theoretical.percentiles,simulated.percentiles)
abline(0,1)
for (i in 1:10) {
  pareto.sim.vs.theory(pch=i,type="b",lty=i)
}

```

Figure A.3: As Figure A.2, but using the ability to pass along arguments to a subsidiary function to distinguish separate simulation runs.

```

plot(0,type="n", xlim=c(0,max(theoretical.percentiles)),
  # No more horizontal room than we need
  ylim=c(0,1.1*max(theoretical.percentiles)),
  # Allow some extra vertical room for noise
  xlab="theoretical percentiles", ylab="simulated percentiles",
  main = paste("exponent = ", exponent, ", threshold = ", threshold))
# Diagonal, for visual reference
abline(0,1)
for (i in 1:B) {
  pareto.sim.vs.theory(n=n,exponent=exponent,threshold=threshold,
    pch=i,type="b",lty=i)
}
}

```

Defining this will work just fine, but it won't work properly until we re-defined `pareto.sim.vs.theory` to take the arguments `n`, `exponent` and `threshold`.⁴

It seems like a simple modification of the old definition should do the trick:

```

pareto.sim.vs.theory <- function(n,exponent,threshold,...) {
  r <- rpareto(n=n,exponent=exponent,threshold=threshold)
  simulated.percentiles <- quantile(r,(0:99)/100)
  points(theoretical.percentiles,simulated.percentiles,...)
}

```

After defining this, the checker function seems to work fine. The following commands produce the plot in Figure A.4, which looks very like the manually-created one. (Random noise means it won't be exactly the same.) Putting in the default arguments explicitly gives the same results (not shown).

```

> check.rpareto()
> check.rpareto(n=1e4,exponent=2.5,threshold=1)

```

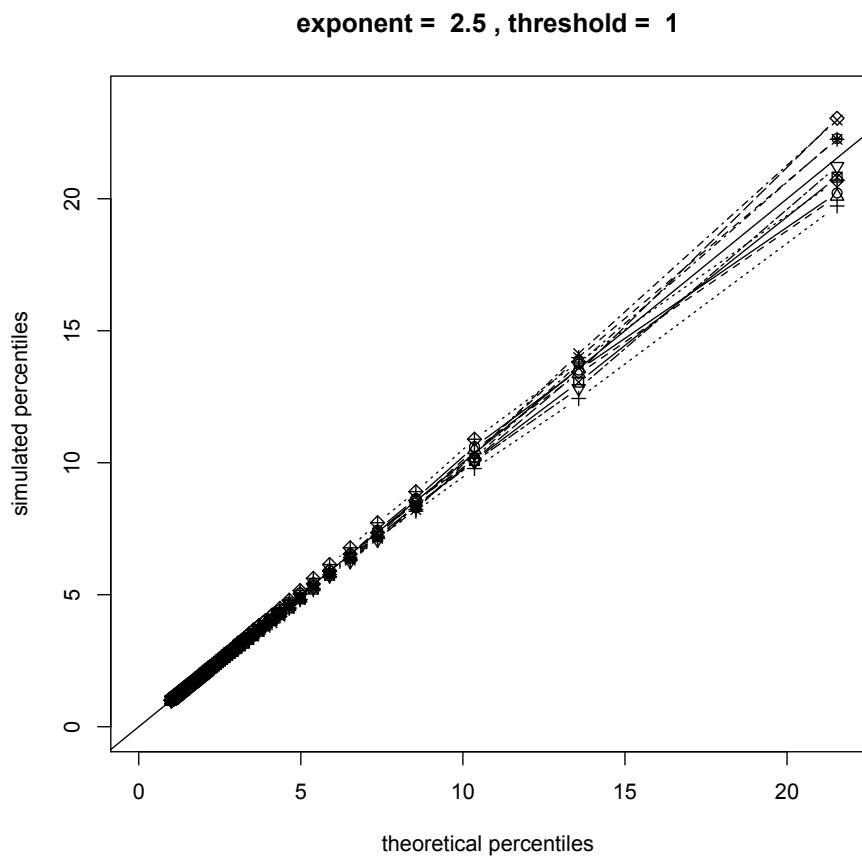
Unfortunately, changing the arguments reveals a bug (Figure A.5). Notice that the vertical coordinates of the points, coming from the simulation, look like they have about the same range as the theoretical quantiles, used to lay out the plotting window. But the horizontal coordinates are all pretty much the same (on a scale of tens of billions, anyway). What's going on?

The horizontal coordinates for the points being plotted are set in `pareto.sim.vs.theory`.³:

```
points(theoretical.percentiles,simulated.percentiles,...)
```

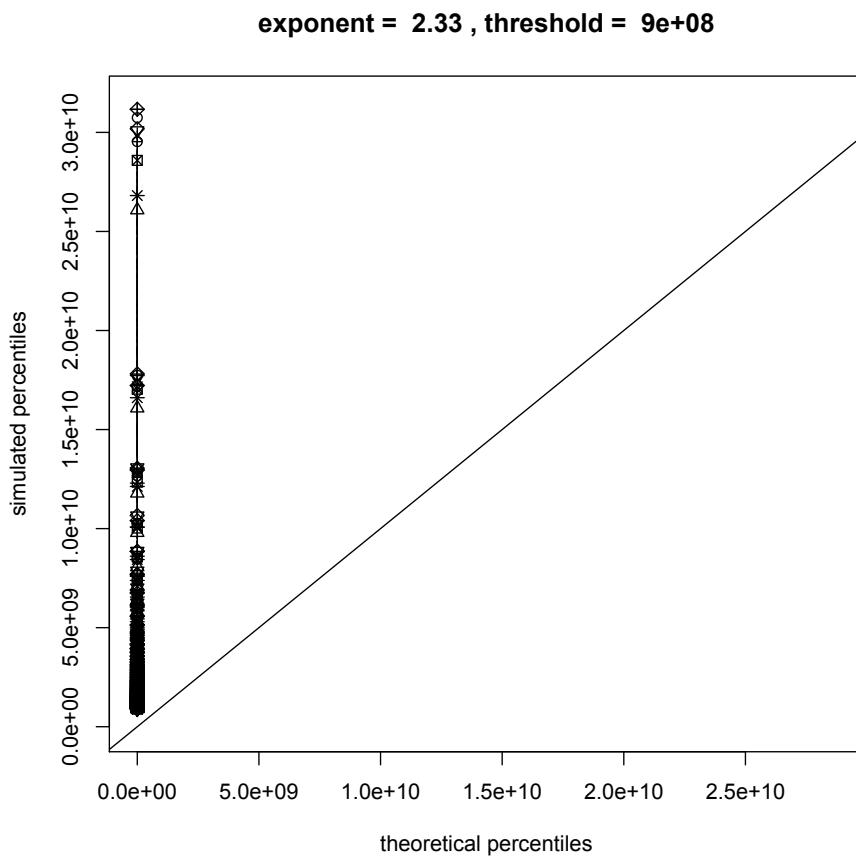
Where does this function get `theoretical.percentiles` from? Since the variable isn't assigned inside the function, R tries to figure it out from context. Since `pareto.sim.vs.theory` was defined on the command line, the context R uses to interpret it is the global workspace — where there is, in fact, a variable called `theoretical.percentiles`, which I set by hand for the previous plots. So the *plotted* theoretical quantiles are all

⁴Try running `check.rpareto()`, followed by `warnings()`.



```
check.rpareto()
```

Figure A.4: Automating the checking of `rpareto`.



```
check.rpareto(n=1e4,exponent=2.33,threshold=9e8)
```

Figure A.5: A bug in `check.rpareto`.

too small in Figure A.5, because they're for a distribution with a much lower threshold.

Didn't `check.rpareto` assign its own value to `theoretical.percentiles`, which it used to set the plot boundaries? Yes, but that assignment only applied *in the context of the function*. Assignments inside a function have limited `scope`, they leave values in the broader context alone. Try this:

```
> x <- 7
> x
[1] 7
> square <- function(y) { x <- y^2; return(x) }
> square(7)
[1] 49
> x
[1] 7
```

The function `square` assigns `x` to be the square of its argument. This assignment holds within the scope of the function, as we can see from the fact that the returned value is always the square of the argument, and not what we assigned `x` to be in the global, command-line context. However, this does not over-write that global value, as the last line shows.⁵

There are two ways to fix this problem. One is to re-define `pareto.sim.vs.theory` to calculate the theoretical quantiles:

```
pareto.sim.vs.theory <- function(n,exponent,threshold,...) {
  r <- rpareto(n=n,exponent=exponent,threshold=threshold)
  theoretical.percentiles <- qpareto.4((0:99)/100,exponent=exponent,
                                         threshold=threshold)
  simulated.percentiles <- quantile(r,(0:99)/100)
  points(theoretical.percentiles,simulated.percentiles,...)
}
```

This will work (try running `check.rpareto(1e4,2.33,9e8)` now), but it's very redundant — every time we call this, we're recalculating the same percentiles, which we already calculated in `check.rpareto`. A cleaner solution is to make the vector of theoretical percentiles an argument to `pareto.sim.vs.theory`, and change `check.rpareto` to provide it.

```
check.rpareto <- function(n=1e4,exponent=2.5,threshold=1,B=10) {
  # One set of percentiles for everything
  theoretical.percentiles <- qpareto.4((0:99)/100,exponent=exponent,
                                         threshold=threshold)
  # Set up plotting window, but don't put anything in it:
  plot(0,type="n", xlim=c(0,max(theoretical.percentiles)),
```

⁵There are techniques by which functions can change assignments outside of their scope. They are tricky, rare, and best avoided except by those who really know what they are doing. (If you think you do, you are probably wrong.)

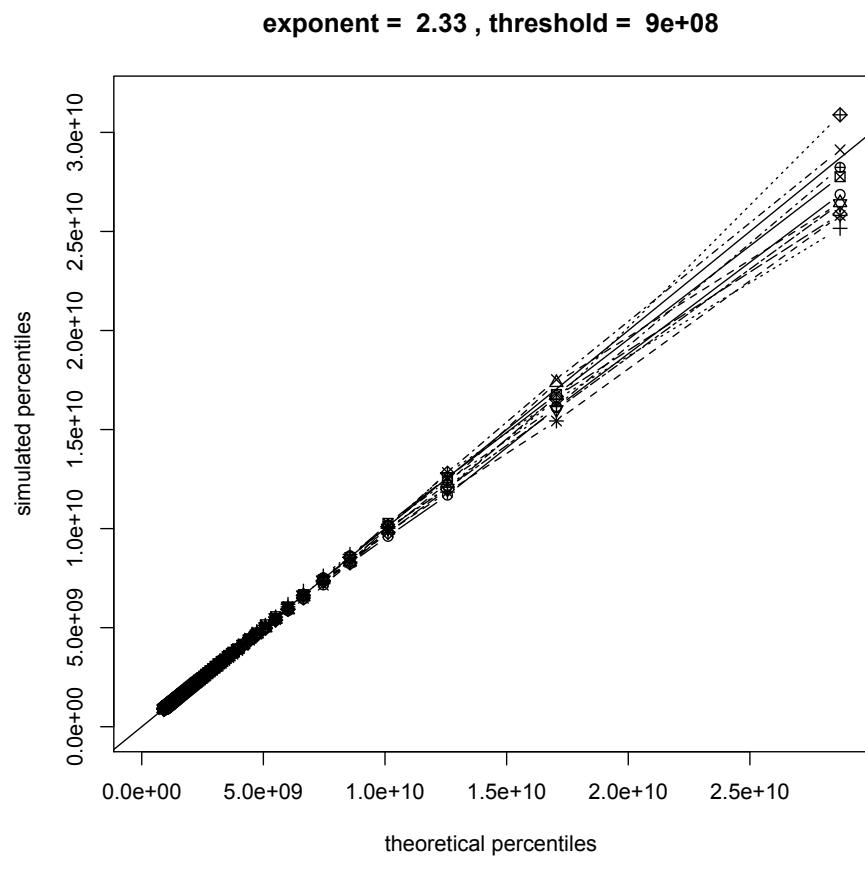
```

# No more horizontal room than we need
ylim=c(0,1.1*max(theoretical.percentiles)),
# Allow some extra vertical room for noise
xlab="theoretical percentiles", ylab="simulated percentiles",
main = paste("exponent = ", exponent, ", threshold = ", threshold))
# Diagonal, for visual reference
abline(0,1)
for (i in 1:B) {
  pareto.sim.vs.theory.4(n=n,exponent=exponent,threshold=threshold,
    theoretical.percentiles=theoretical.percentiles,
    pch=i,type="b",lty=i)
}
}

pareto.sim.vs.theory <- function(n,exponent,threshold,
  theoretical.percentiles,...) {
  r <- rpareto(n=n,exponent=exponent,threshold=threshold)
  simulated.percentiles <- quantile(r,(0:99)/100)
  points(theoretical.percentiles,simulated.percentiles,...)
}

```

Figure A.6 shows that this succeeds.



```
check.rpareto(1e4,2.33,9e8)
```

Figure A.6: Using the corrected simulation checker.

A.6 Avoiding Iteration: Manipulating Objects

Let's go back to the declaration of `rpareto`, which I repeat here, unchanged, for convenience:

```
rpareto <- function(n,exponent,threshold) {
  x <- vector(length=n)
  for (i in 1:n) {
    x[i] <- qpareto.4(p=runif(1),exponent=exponent,threshold=threshold)
  }
  return(x)
}
```

We've confirmed that this works, but it involves explicit iteration in the form of the `for` loop. Because of the way R carries out iteration⁶, it is slow, and better avoided when possible. Many of the utility functions in R, like `replicate`, are designed to avoid explicit iteration. We could re-write `rpareto` using `replicate`, for example:

```
rpareto <- function(n,exponent,threshold) {
  x <- replicate(n,qpareto.4(p=runif(1),exponent=exponent,threshold=threshold))
  return(x)
}
```

(The outstanding use of `replicate` is when we want to repeat the same random experiment many times — there are examples in the notes for Chapters 6.)

An even clearer alternative makes use of the way R automatically **vectorizes** arithmetic:

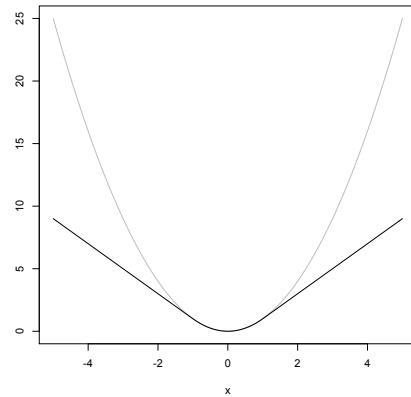
```
rpareto <- function(n,exponent,threshold) {
  x <- qpareto.4(p=runif(n),exponent=exponent,threshold=threshold)
  return(x)
}
```

This feeds `qpareto.4` a *vector* of quantiles `p`, of length `n`, which in turn gets passed along to `qpareto.1`, which finally tries to evaluate

```
threshold*((1-p)^(-1/(exponent-1)))
```

With `p` being a vector, R hopes that `threshold` and `exponent` are also vectors, and of the same length, so that it evaluates this arithmetic expression component-wise. If `exponent` and `threshold` are shorter, it will “recycle” their values, in order, until it has vectors equal in length to `p`. In particular, if `exponent` and `threshold` have length 1, it will repeat both of them `length(p)` times, and then evaluate everything component by component. (See the “Introduction to R” manual for more on this “recycling rule”.) The quantile functions we have defined inherit this ability to recycle, without any special work on our part. The final version of `rpareto` we have written is not only faster, it is clearer and easier to read. It focuses our attention on what is being done, and not on the mechanics of doing it.

⁶Roughly speaking, it ends up having to create and destroy a whole copy of everything which gets changed in the course of one pass around the iteration loop, which can involve lots of memory and time.



```
curve(x^2,col="grey",from=-5,to=5,ylab="")
curve(huber,add=TRUE)
```

Figure A.7: The Huber loss function ψ (black) versus the squared error loss (grey).

A.6.1 ifelse and which

Sometimes we want to do different things to different parts of a vector (or larger structure) depending on its values. For instance, in robust regression one often replaces the squared error loss with what's called the Huber loss⁷,

$$\psi(x) = \begin{cases} x^2 & \text{if } |x| \leq 1 \\ 2|x| - 1 & \text{if } |x| > 1 \end{cases} \quad (\text{A.4})$$

which isn't so vulnerable to outliers, as in Figure A.7.

We might code this up like so:

```
huber <- function(x) {
  n <- length(x)
  y <- vector(n)
  for (i in 1:n) {
    if (abs(x) <= 1) {
      y[i] <- x[i]^2
    } else {
      y[i] <- 2*abs(x[i])-1
    }
  }
  return(y)
}
```

⁷One applies this not to the residuals directly, but to residuals divided by some robust measure of dispersion.

This is not very easy follow. R provides a very useful function, `ifelse`, which lets us apply a binary test, and then draw from either of two calculations. Using it, we re-write `huber` like so:

```
huber <- function(x) {
  return(ifelse(abs(x) <= 1, x^2, 2*abs(x)-1))
}
```

The first argument needs to produce a vector of TRUE/FALSE values; the second argument provides the outputs for the TRUE positions, the third the outputs for the FALSE positions. Here all three are expressions involving the same variable, but that's not essential.

Another useful device is the `which` function, whose argument is a vector of TRUE/FALSE values, returning a vector of the indices where the argument is TRUE, e.g.,

```
incomplete.cases <- which(is.na(cholesterol))
```

would give us the positions at which the vector `cholesterol` had NA values. This is equivalent to

```
incomplete.cases <- c()
for (i in 1:length(cholesterol)) {
  if (is.na(cholesterol[i])) {
    incomplete.cases <- c(incomplete.cases,i)
  }
}
```

A.6.2 `apply` and Its Variants

Particularly useful ways of avoiding iteration come from the function `apply`, and the closely related `sapply` and `lapply` functions. We saw `apply` in Chapter 6:

```
x <- replicate(10,rpareto(100,2.5,1))
apply(x,2,quantile,probs=0.9)
```

Each call to `rpareto` inside the `replicate` creates a vector of length 100. Replicate then stacks these, as columns, into an array. The `apply` function applies the same function to each row or column of the array, depending on whether its second argument is 1 (rows) or 2 (columns). So this will find the 90th percentile of each of the 10 random-number draws, and give that back to us as a vector.

`array` only works for arrays, matrices and data frames (and works on them by treating them as arrays). If we want to apply the same function to every element of a vector or list, we use `lapply`. This gives us back a list, which can be inconvenient:

```
> y <- c(0.9,0.99,0.999,0.99999)
> lapply(y,qpareto.4,exponent=2.5,threshold=1)
[[1]]
[1] 4.641589
```

```

[[2]]
[1] 21.54435

[[3]]
[1] 100

[[4]]
[1] 2154.435

```

The function `sapply` works like `lapply`, but tries to simplify its output down to a vector or array:

```
> sapply(y,qpareto.4,exponent=2.5,threshold=1)
[1] 4.641589 21.544347 100.000000 2154.434690
```

With this function, this is equivalent to `qpareto.4(y,exponent=2.5,threshold=1)`, but `sapply` can take considerably more complicated functions:

```
# Suppose we have models lm.1 and lm.2 hanging around
some.models <- list(model.1=lm.1, model.2=lm.2)
# Extract all the coefficients from all the models
sapply(some.models,coefficients)
```

`sapply` has a `simplify` argument, which defaults to TRUE; setting it to FALSE turns off the simplification. `replicate` actually has the same argument. Usually, simplifying the output of `replicate` is a good thing, but it can weirdness when what's being replicated is a complicated value itself.

For instance, here's a little bit of bootstrapping regression models, using the fossil-animal data set from homework 3.

```
resample <- function(x) { sample(x,size=length(x),replace=TRUE) }
nampd.lm.subset <- function(s) {
  lm(delta_ln_mass ~ ln_old_mass,data=nampd,subset=s)
}
boot.models.1 <- replicate(10,nampd.lm.subset(resample(1:nrow(nampd))))
```

Working with `boot.models.1` is going to be very hard, because it wants to be an array, but isn't quite, and is generally very confused. (Try it!) Instead do it this way:

```
boot.models.2 <- replicate(10,nampd.lm.subset(resample(1:nrow(nampd))),simplify=FALSE)
```

`boot.models.2` is simply a list with 10 elements, each one of which is an `lm`-style model. Now it's easy extract information about any particular one, or use `sapply`:

```
> sapply(boot.models.2,coefficients)
      [,1]      [,2]      [,3]      [,4]
(Intercept) 0.21613522 0.092359537 0.184610989 0.15530334
ln_old_mass -0.01379554 -0.002729451 -0.007396701 -0.01078759
```

	[,5]	[,6]	[,7]	[,8]
(Intercept)	0.124932040	0.115330144	0.192097575	0.0880172496
ln_old_mass	-0.003754933	-0.007362125	-0.008486858	0.0008434435
	[,9]	[,10]		
(Intercept)	0.17065043	0.207331222		
ln_old_mass	-0.01430204	-0.009881709		

A.7 More Complicated Return Values

So far, all the functions we have written have returned either a single value, or a simple vector, or nothing at all. The built-in functions return much more complicated things, like matrices, data frames, or lists, and we can too.

To illustrate, let's switch gears away from the Pareto distribution, and think about the Gaussian for a change. As you know, if we have data x_1, x_2, \dots, x_n and we want to fit a Gaussian distribution to them by maximizing the likelihood, the best-fitting Gaussian has mean

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i \quad (\text{A.5})$$

which is just the sample mean, and variance

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2 \quad (\text{A.6})$$

which differs from the usual way of defining the sample variance by having a factor of n in the denominator, instead of $n - 1$. Let's write a function which takes in a vector of data points and returns the maximum-likelihood parameter estimates for a Gaussian.

```
gaussian.mle <- function(x) {
  n <- length(x)
  mean.est <- mean(x)
  var.est <- var(x)*(n-1)/n
  est <- list(mean=mean.est, sd=sqrt(var.est))
  return(est)
}
```

There is one argument, which is the vector of data. To be cautious, I should probably check that it *is* a vector of numbers, but skip that to be clear here. The first line figures out how many data points we have. The second takes the mean. The third finds the estimated variance — the definition of the built-in `var` function uses $n - 1$ in its denominator, so I scale it down by the appropriate factor⁸. The fourth line creates a list, called `est`, with two components, named `mean` and `sd`, since those are the names R likes to use for the parameters of Gaussians. The first component is

⁸Clearly, if n is large, $\frac{n-1}{n} = 1 - 1/n$ will be very close to one, but why not be precise?

our estimated mean, and the second is the standard deviation corresponding to our estimated variance⁹. Finally, the function returns the list.

As always, it's a good idea to check the function on a case where we know the answer.

```
> x <- 1:10
> mean(x)
[1] 5.5
> var(x) * (9/10)
[1] 8.25
> sqrt(var(x) * (9/10))
[1] 2.872281
> gaussian.mle(x)
$mean
[1] 5.5

$sd
[1] 2.872281
```

A.8 Re-Writing Your Code: An Extended Example

Suppose we want to find a standard error for the median of a Gaussian distribution. We know, somehow, that the mean of the Gaussian is 3, the standard deviation is 2, and the sample size is one hundred. If we do

```
x <- rnorm(n=100,mean=3,sd=2)
```

we'll get a draw from that distribution in `x`. If we do

```
x <- rnorm(n=100,mean=3,sd=2)
median(x)
```

we'll calculate the median on one random draw. Following the general idea of bootstrapping we can approximate the standard error of the median by repeating this many times and taking the standard deviation. We'll do this by explicitly iterating, so we need to set up a vector to store our intermediate results first.

```
medians <- vector(length=100)
for (i in 1:100) {
  x <- rnorm(n=100,mean=3,sd=2)
  medians[i] <- median(x)
}
se.in.median <- sd(medians)
```

⁹If n is large, $\sqrt{\frac{n-1}{n}} = \sqrt{1 - \frac{1}{n}} \approx 1 - \frac{1}{2n}$ (using the binomial theorem in the last step). For reasonable data sets, the error of just using `sd(x)` would have been small — but why have it at all?

Well, how do we know that 100 replicates is enough to get a good approximation? We'd need to run this a couple of times, typing it in or at least pasting it in many times. Instead, we can write a function which just gives everything we've done a single name. (I'll add comments as I go on.)

```
# Inputs: None; everything is hard-coded
# Output: the standard error in the median
find.se.in.median <- function() {
  # Set up a vector to store the simulated medians
  medians <- vector(length=100)
  # Do the simulation 100 times
  for (i in 1:100) {
    x <- rnorm(n=100,mean=3,sd=2) # Simulate
    medians[i] <- median(x) # Calculate the median of the simulation
  }
  se.in.median <- sd(medians) # Take standard deviation
  return(se.in.median)
}
```

If we decide that 100 replicates isn't enough and we want 1000, we need to change this function. We could just change the first two appearances of "100" to "1000", but we have to catch all of them; we have to remember that the 100 in `rnorm` is there for a different reason and leave it alone; and if we later decide that actually 500 replicates would be enough, we have to do everything all over again.

It is easier, safer, clearer and more flexible to abstract a little and add an argument to the function, which is the number of replicates. I'll add comments as I go.

```
# Inputs: Number of bootstrap replicates B
# Output: the standard error in the median
find.se.in.median <- function(B) {
  # Set up a vector to store the simulated medians
  medians <- vector(length=B)
  # Do the simulation B times
  for (i in 1:B) {
    x <- rnorm(n=100,mean=3,sd=2) # Simulate
    medians[i] <- median(x) # Calculate median of the simulation
  }
  se.in.median <- sd(medians) # Take standard deviation
  return(se.in.median)
}
```

Now suppose we want to find the standard error of the median for an exponential distribution with rate 2 and sample size 37. We could write another function,

```
find.se.in.median.exp <- function(B) {
  # Set up a vector to store the simulated medians
  medians <- vector(length=B)
  # Do the simulation B times
```

```

for (i in 1:B) {
  x <- rexp(n=37,rate=2) # Simulate
  medians[i] <- median(x) # Calculate median of the simulation
}
se.in.median <- sd(medians) # Take standard deviation
return(se.in.median)
}

```

but it is wasteful to define two functions which do almost the same job. It's not just inelegant; it invites mistakes, it's harder to read (imagine coming back to this in two weeks — was there a big reason why we had two separate functions here?), and it's harder to improve. We need to abstract a bit more.

We *could* put in some kind of switch which would simulate from either of these two distributions, maybe like this:

```

# Inputs: number of replicates (B)
# flag for whether to use a normal or an exponential (use.norm)
# Output: The standard error in the median
find.se.in.median <- function(B,use.norm=TRUE) {
  medians <- vector(length=B)
  for (i in 1:B) {
    if (use.norm) {
      x <- rnorm(100,3,2)
    } else {
      x <- rexp(37,2)
    }
    medians[i] <- median(x)
  }
  se.in.median <- sd(medians)
  return(se.in.median)
}

```

but why just these two? If we wanted any other distribution whatsoever, plainly all we'd have to do is change how *x* is simulated. So we really want to be able to *give* a simulator to the function as an argument.

Fortunately, in R you can give one function as an argument to another, so we'd do something like this.

```

# Inputs: Number of replicates (B)
# Simulator function (simulator)
# Presumes: simulator is a no-argument function which produce a vector of
# numbers
# Output: The standard error in the media
find.se.in.median <- function(B,simulator) {
  median <- vector(length=B)
  for (i in 1:B) {
    x <- simulator()
  }
  se.in.median <- sd(median)
  return(se.in.median)
}

```

```

    medians[i[ <- median(x)
  }
se.in.median <- sd(medians)
return(se.in.medians)
}

```

Now to repeat our original calculations, we define a simulator function:

```

# Inputs: None
# Output: ten draws from the mean 3, s.d. 2 Gaussian
simulator.1 <- function() {
  return(rnorm(10,3,2))
}

```

If we now call

```
find.se.in.median(B=100,simulator=simulator.1)
```

then every time `find.se.in.median` goes through the `for` loop, it will call `simulator.1`, which in turn will produce the right random numbers. If we also define

```

# Inputs: None
# Output: 37 draws from the rate 2 exponential
simulator.2 <- function() {
  return(rexp(37,2))
}

```

then to find the standard error in the median of *this*, we just call

```
find.se.in.median(B=100,simulator=simulator.2)
```

This same approach works if we want to sample from a much more complicated distribution. If we fit a locally-linear kernel regression to the Old Faithful data, and want a standard error in the median of the predicted waiting times, with noise coming from resampling cases, we would do something like this for the simulator

```

# Inputs: None
# Output: The fitted waiting times of a bootstrapped kernel smooth from the
# geyser data
simulator.3 <- function() {
  if (!exists("geyser")) {
    require(MASS)
    data(geyser)
  }
  n <- nrow(geyser)
  resampled.rows <- sample(1:n,size=n,replace=TRUE)
  geyser.r <- geyser[resampled.rows,]
  fit <- npreg(waiting~duration,data=geyser.r,regtype="ll")
  waiting.times <- npreg$mean
  return(waiting.times)
}

```

and then this for to find the standard error in the median:

```
find.se.in.median(B=100,simulator=simulator.3)
```

By breaking up the task this way, if we encounter errors or just general trouble when we run that last command, it is easier to localize the problem. We can check whether `find.se.in.median` seems to work properly with other simulator functions. (For instance, we might write a “simulator” that either does `rep(10, 1)` or `rep(10, -1)` with equal probability, since then we can work out what the standard error of the median ought to be.) We can also check whether `simulator.3` is working properly, and finally whether there is some issue with putting them together, say that the output from the simulator is not quite in a format that `find.se.in.median` can handle. If we just have one big ball of code, it is much harder to read, to understand, to debug, and to improve.

To turn to that last point, one of the things R does poorly is explicit iteration with `for` loops. As mentioned above, it’s generally better to replace such loops with “vectorized” functions, which do the iteration using fast code outside of R. One of these, especially for this situation, is the function `replicate`. We can re-write `find.se.in.median` using it:

```
# Inputs: number of replicates (B)
# Simulator function (simulator)
# Presumes: simulator is a no-argument function which produces a vector of
# numbers
# Outputs: Standard error in the median of the output of simulator
find.se.in.median <- function(B,simulator) {
  medians <- replicate(B,median(simulator()))
  se.in.median <- sd(medians)
  return(se.in.median)
}
```

Again: shorter, faster, and easier to understand (if you know what `replicate` does). Also, because we are telling this what simulation function to use, and writing those functions separately, we do not have to change any of our simulators. They don’t care how `find.se.in.median` works. In fact, they don’t care that there is any such function — they could be used as components in many other functions which can also process their outputs. So long as these *interfaces* are maintained, the inner workings of the functions are irrelevant to each other.

Suppose for instance that we want not the standard error of the median, but the interquartile range of the median — the median is after all a “robust”, outlier-resistant measure of the central tendency, and the IQR is likewise a robust measure of dispersion. This is now easy:

```
# Inputs: number of replicates (B)
# Simulator function (simulator)
# Presumes: simulator is a no-argument function which produces a vector of
# numbers
# Outputs: Interquartile range of the median of the output of simulator
```

```
find.iqr.of.median <- function(B,simulator) {
  medians <- replicate(B,median(simulator()))
  iqr.of.median <- IQR(medians)
  return(iqr.of.median)
}
```

Or for that matter the good old standard error of the mean:

```
# Inputs: number of replicates (B)
# Simulator function (simulator)
# Presumes: simulator is a no-argument function which produces a vector of
# numbers
# Outputs: Standard error of the mean of the output of simulator
find.se.of.mean <- function(B,simulator) {
  means <- replicate(B,mean(simulator()))
  se.of.mean <- sd(means)
  return(se.of.mean)
}
```

These last few examples suggest that we could abstract even further, by swapping in and out different estimators (like `median` and `mean`) and different summarizing functions (like `se` or `IQR`).

```
# Inputs: number of replicates (B)
# Simulator function (simulator)
# Estimator function (estimator)
# Sample summarizer function (summarizer)
# Presumes: simulator is a no-argument function which produces a vector of
# numbers
# estimator is a function that takes a vector of numbers and produces one
# output
# summarizer takes a vector of outputs from estimator
# Outputs: Summary of the simulated distribution of estimates
summarize.sampling.dist.of.estimates <- function(B,simulator,estimator,
                                                 summarizer) {
  estimates <- replicate(B,estimator(simulator()))
  return(summarizer(estimates))
}
```

The name is too long, of course, so we should replace it with something catchier:

```
bootstrap <- function(B,simulator,estimator,summarizer) {
  estimates <- replicate(B,estimator(simulator()))
  return(summarizer(estimates))
}
```

Our very first example is equivalent to

```
bootstrap(B=100,simulator=simulator.1,estimator=median,summarizer=sd)
```

`bootstrap` is just two lines: one simulates and re-estimates, the other summarizes the re-estimates. This is the essence of what we are trying to do, and is logically distinct from the details of particular simulators, estimators and summaries.

We started with a particular special case and generalized it. The alternative route is to start with a very general framework — here, writing `bootstrap` — and then figure out what lower-level functions we would need to make it work in a the case at hand, writing them if necessary. (We need to write a simulator, but someone's already written `median` for us.) Getting the first stage right involves a certain amount of reflection on how to solve the problem — it's rather like the strategy of doing a “show that” math problem by starting from the desired conclusion and working backwards.

It is still somewhat clunky to have to write a new function every time we want to change the settings in the simulation, but this has gone on long enough.

A.9 General Advice on Programming

Programming is an act of communication: with the computer, of course, but also with your co-workers, and with yourself in the future¹⁰. Clear and effective communication is a valuable skill in itself; it also tends to make it easier to do the job, and to make debugging easier.

A.9.1 Comment your code

Comments lengthen your file, but they make it immensely easier for other people to understand. (“Other people” includes your future self; there are few experiences more frustrating than coming back to a program after a break only to wonder what you were thinking.) Comments should say what each part of the code does, and how it does it. The “what” is more important; you can change the “how” more often and more easily.

Every function (or subroutine, etc.) should have comments at the beginning saying:

- what it does;
- what all its inputs are (in order);
- what it requires of the inputs and the state of the system (“presumes”);
- what side-effects it may have (e.g., “plots histogram of residuals”);
- what all its outputs are (in order)

Listing what other functions or routines the function calls (“dependencies”) is optional; this can be useful, but it’s easy to let it get out of date.

You should treat “Thou shalt comment thy code” as a commandment which Moses brought down from Mt. Sinai, written on stone by a fiery Hand.

¹⁰And, in this class, with your graders.

A.9.2 Use meaningful names

Unlike some older languages, R lets you give variables and functions names of essentially arbitrary length and form. So give them meaningful names. Writing `loglikelihood`, or even `loglike`, instead of `L` makes your code a little longer, but generally a lot clearer, and it runs just the same.

This rule is lower down in the list because there are exceptions and qualifications. If your code is tightly associated to a mathematical paper, or to a field where certain symbols are conventionally bound to certain variables, you may as well use those names (e.g., call the probability of success in a binomial `p`). You should, however, explain what those symbols are in your comments. In fact, since what you regard as a meaningful name may be obscure to others (e.g., those grading your work), you should use comments to explain variables in any case. Finally, it's OK to use single-letter variable names for counters in loops (but see the advice on iteration below).

A.9.3 Check whether your program works

It's not enough — in fact it's very little — to have a program which runs and gives you some output. It needs to be the right output. You should therefore construct tests, which are things that the correct program should be able to do, but an incorrect program should not. This means that:

- you need to be able to check whether the output is right;
- your tests should be reasonably severe, so that it's hard for an incorrect program to pass them;
- your tests should help you figure out what isn't working;
- you should think hard about programming the test, so it checks whether the output is right, and you can easily repeat the test as many times as you need.

Try to write tests for the component functions, as well as the program as a whole. That way you can see where failures are. Also, it's easier to figure out what the right answers should be for small parts of the problem than the whole.

Try to write tests as very small function which call the component you're testing with controlled input values. For instance, we tested `qpareto` by looking at what it returned for selected arguments with manually carrying out the computation. With statistical procedures, tests can look at average or distributional results — we saw an example of this with checking `rpareto`.

Of course, unless you are very clever, or the problem is very simple, a program could pass all your tests and still be wrong, but a program which fails your tests is definitely not right.

(Some people would actually advise writing your tests before writing any actual functions. They have their reasons but I think that's overkill for this class.)

A.9.4 Avoid writing the same thing twice

Many data-analysis tasks involve doing the same thing multiple times, either as iteration, or to slightly different pieces of data, or with some parameters adjusted, etc. Try to avoid writing two pieces of code to do the same job. If you find yourself copying the same piece of code into two places in your program, look into writing *one* function, and *calling* it twice.

Doing this means that there is only one place to make a mistake, rather than many. It also means that when you fix your mistake, you only have one piece of code to correct, rather than many. (Even if you don't make a mistake, you can always make improvements, and then there's only one piece of code you have to work on.) It also leads to shorter, more comprehensible and more adaptable code.

A.9.5 Start from the beginning and break it down

When you have a big problem, start by thinking about what you want your program to do. Then figure out a set of slightly smaller steps which, put together, would accomplish that. Then take each of those steps and break them down into yet smaller ones. Keep going until the pieces you're left with are so small that you can see how to do each of them with only a few lines of code. Then write the code for the smallest bits, check it, once it works write the code for the next larger bits, and so on.

In slogan form:

- Think before you write.
- What first, then how.
- Design from the top down, code from the bottom up.

(Not everyone likes to design code this way, and it's not in the written-in-stone-atop-Sinai category, but there are many much worse ways to start.)

A.9.6 Break your code into many short, meaningful functions

Since you have broken your programming problem into many small pieces, try to make each piece a short function. (In other languages you might make them subroutines or methods, but in R they should be functions.)

Each function should achieve a single coherent task — its function, if you will. The division of code into functions should respect this division of the problem into sub-problems. More exactly, the way you break your code into functions is how you have divided your problem.

Each function should be short, generally less than a page of print-out. The function should do one single meaningful thing. (Do not just break the calculation into arbitrary thirty-line chunks and call each one a function.) These functions should generally be separate, not nested one inside the other.

Using functions has many advantages:

- you can re-use the same code many times, either at different places in this program or in other programs

- the rest of your code only has to care about the inputs and outputs to the function (its interfaces), not about the internal machinery that turns inputs into outputs. This makes it easier to design the rest of the program, and it means you can change that machinery without having to re-design the rest of the program.
- it makes your code easier to test (see below), to debug, and to understand.

Of course, every function should be commented, as described above.

A.10 Further Reading

Matloff (2011) is a good introduction to programming for total novices using R. Braun and Murdoch (2008) has more on statistical calculations and related topics, but can also work as an introduction for absolute beginners. Adler (2009) is an introduction to R for those with some prior knowledge of other programming languages. Chambers (2008) is excellent for anyone who wants to be serious about programming in R.

Appendix B

Big O and Little o Notation

It is often useful to talk about the *rate* at which some function changes as its argument grows (or shrinks), without worrying too much about the detailed form. This is what the $O(\cdot)$ and $o(\cdot)$ notation lets us do.

A function $f(n)$ is “of constant order”, or “of order 1” when there exists some non-zero constant c such that

$$\frac{f(n)}{c} \rightarrow 1 \quad (\text{B.1})$$

as $n \rightarrow \infty$; equivalently, since c is a constant, $f(n) \rightarrow c$ as $n \rightarrow \infty$. It doesn’t matter how big or how small c is, just so long as there is some such constant. We then write

$$f(n) = O(1) \quad (\text{B.2})$$

and say that “the proportionality constant c gets absorbed into the big O ”. For example, if $f(n) = 37$, then $f(n) = O(1)$. But if $g(n) = 37(1 - \frac{2}{n})$, then $g(n) = O(1)$ also.

The other orders are defined recursively. Saying

$$g(n) = O(f(n)) \quad (\text{B.3})$$

means

$$\frac{g(n)}{f(n)} = O(1) \quad (\text{B.4})$$

or

$$\frac{g(n)}{f(n)} \rightarrow c \quad (\text{B.5})$$

as $n \rightarrow \infty$ — that is to say, $g(n)$ is “of the same order” as $f(n)$, and they “grow at the same rate”, or “shrink at the same rate”. For example, a quadratic function $a_1 n^2 + a_2 n + a_3 = O(n^2)$, no matter what the coefficients are. On the other hand, $b_1 n^{-2} + b_2 n^{-1}$ is $O(n^{-1})$.

Big- O means “is of the same order as”. The corresponding little- o means “is ultimately smaller than”: $f(n) = o(1)$ means that $f(n)/c \rightarrow 0$ for any constant c . Recursively, $g(n) = o(f(n))$ means $g(n)/f(n) = o(1)$, or $g(n)/f(n) \rightarrow 0$. We also read $g(n) = o(f(n))$ as “ $g(n)$ is ultimately negligible compared to $f(n)$ ”.

There are some rules for arithmetic with big- O symbols:

- If $g(n) = O(f(n))$, then $cg(n) = O(f(n))$ for any constant c .
- If $g_1(n)$ and $g_2(n)$ are both $O(f(n))$, then so is $g_1(n) + g_2(n)$.
- If $g_1(n) = O(f(n))$ but $g_2(n) = o(f(n))$, then $g_1(n) + g_2(n) = O(f(n))$.
- If $g(n) = O(f(n))$, and $f(n) = o(h(n))$, then $g(n) = o(h(n))$.

These are not *all* of the rules, but they’re enough for most purposes.

Appendix C

Where the χ^2 Null Distribution for the Log Likelihood-Ratio Test Comes From

Here is a very hand-wavy explanation for Eq. 2.34. We're assuming that the true parameter value, call it θ , lies in the restricted class of models ω . So there are q components to θ which matter, and the other $p - q$ are fixed by the constraints defining ω . To simplify the book-keeping, let's say those constraints are all that the extra parameters are zero, so $\theta = (\theta_1, \theta_2, \dots, \theta_q, 0, \dots, 0)$, with $p - q$ zeroes at the end.

The restricted MLE $\hat{\theta}$ obeys the constraints, so

$$\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_q, 0, \dots, 0) \quad (\text{C.1})$$

The unrestricted MLE does not have to obey the constraints, so it's

$$\hat{\Theta} = (\hat{\Theta}_1, \hat{\Theta}_2, \dots, \hat{\Theta}_q, \hat{\Theta}_{q+1}, \dots, \hat{\Theta}_p) \quad (\text{C.2})$$

Because both MLEs are consistent, we know that $\hat{\theta}_i \rightarrow \theta_i$, $\hat{\Theta}_i \rightarrow \theta_i$ if $1 \leq i \leq q$, and that $\hat{\Theta}_i \rightarrow 0$ if $q + 1 \leq i \leq p$.

Very roughly speaking, it's the last extra terms which end up making $L(\hat{\Theta})$ larger than $L(\hat{\theta})$. Each of them tends towards a mean-zero Gaussian whose variance is $O(1/n)$, but their impact on the log-likelihood depends on the square of their sizes, and the square of a mean-zero Gaussian has a χ^2 distribution with one degree of freedom. A whole bunch of factors cancel out, leaving us with a sum of $p - q$ independent χ^2_1 variables, which has a χ^2_{p-q} distribution.

In slightly more detail, we know that $L(\hat{\Theta}) \geq L(\hat{\theta})$, because the former is a maximum over a larger space than the latter. Let's try to see how big the difference is by

doing a Taylor expansion around $\hat{\Theta}$, which we'll take out to second order.

$$\begin{aligned} L(\hat{\theta}) &\approx L(\hat{\Theta}) + \sum_{i=1}^p (\hat{\Theta}_i - \hat{\theta}_i) \left(\frac{\partial L}{\partial \theta_i} \Big|_{\hat{\Theta}} \right) + \frac{1}{2} \sum_{i=1}^p \sum_{j=1}^p (\hat{\Theta}_i - \hat{\theta}_i) \left(\frac{\partial^2 L}{\partial \theta_i \partial \theta_j} \Big|_{\hat{\Theta}} \right) (\hat{\Theta}_j - \hat{\theta}_j) \\ &= L(\hat{\Theta}) + \frac{1}{2} \sum_{i=1}^p \sum_{j=1}^p (\hat{\Theta}_i - \hat{\theta}_i) \left(\frac{\partial^2 L}{\partial \theta_i \partial \theta_j} \Big|_{\hat{\Theta}} \right) (\hat{\Theta}_j - \hat{\theta}_j) \end{aligned} \quad (\text{C.3})$$

All the first-order terms go away, because $\hat{\Theta}$ is a maximum of the likelihood, and so the first derivatives are all zero there. Now we're left with the second-order terms. Writing all the partials out repeatedly gets tiresome, so abbreviate $\partial^2 L / \partial \theta_i \partial \theta_j$ as $L_{,ij}$.

To simplify the book-keeping, suppose that the second-derivative matrix, or **Hessian**, is diagonal. (This should seem like a swindle, but we get the same conclusion without this supposition, only we need to use a lot more algebra — we diagonalize the Hessian by an orthogonal transformation.) That is, suppose $L_{,ij} = 0$ unless $i = j$. Now we can write

$$L(\hat{\Theta}) - L(\hat{\theta}) \approx -\frac{1}{2} \sum_{i=1}^p (\hat{\Theta}_i - \hat{\theta}_i)^2 L_{,ii} \quad (\text{C.4})$$

$$2 \left[L(\hat{\Theta}) - L(\hat{\theta}) \right] \approx -\sum_{i=1}^q (\hat{\Theta}_i - \hat{\theta}_i)^2 L_{,ii} - \sum_{i=q+1}^p (\hat{\Theta}_i)^2 L_{,ii} \quad (\text{C.5})$$

At this point, we need a fact about the asymptotic distribution of maximum likelihood estimates: they're generally Gaussian, centered around the true value, and with a shrinking variance that depends on the Hessian evaluated at the true parameter value; this is called the **Fisher information**, F or I . (Call it F .) If the Hessian is diagonal, then we can say that

$$\hat{\Theta}_i \rightsquigarrow \mathcal{N}(\theta_i, -1/nF_{ii}) \quad (\text{C.6})$$

$$\hat{\theta}_i \rightsquigarrow \mathcal{N}(\theta_1, -1/nF_{ii}) \quad 1 \leq i \leq q \quad (\text{C.7})$$

$$\hat{\theta}_i = 0 \quad q+1 \leq i \leq p \quad (\text{C.8})$$

Also, $(1/n)L_{,ii} \rightarrow -F_{ii}$.

Putting all this together, we see that each term in the second summation in Eq. C.5 is (to abuse notation a little)

$$\frac{-1}{nF_{ii}} (\mathcal{N}(0, 1))^2 L_{,ii} \rightarrow \chi_1^2 \quad (\text{C.9})$$

so the whole second summation has a χ_{p-q}^2 distribution¹. The first summation, meanwhile, goes to zero because $\hat{\Theta}_i$ and $\hat{\theta}_i$ are actually strongly correlated, so their

¹Thanks to Xiaoran Yan for catching a typo in a previous version here.

difference is $O(1/n)$, and their difference squared is $O(1/n^2)$. Since $L_{,ii}$ is only $O(n)$, that summation drops out.

A somewhat less hand-wavy version of the argument uses the fact that the MLE is really a vector, with a multivariate normal distribution which depends on the inverse of the Fisher information matrix:

$$\widehat{\Theta} \rightsquigarrow \mathcal{N}(\theta, (1/n)F^{-1}) \quad (\text{C.10})$$

Then, at the cost of more linear algebra, we don't have to assume that the Hessian is diagonal.

Appendix D

Proof of the Gauss-Markov Theorem

We want to prove that, when we are doing weighted least squares for linear regression, the best choice of weights $w_i = 1/\sigma_{x_i}^2$. We have already established that WLS is unbiased (Eq. 7.8), so “best” here means minimizing the variance. We have also already established that

$$\hat{\beta}_{WLS} = \mathbf{h}(w)\mathbf{y} \quad (\text{D.1})$$

where the matrix $\mathbf{h}(w)$ is

$$\mathbf{h}(w) = (\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w} \quad (\text{D.2})$$

It would be natural to try to write out the variance as a function of the weights w , set the derivative equal to zero, and solve. This is tricky, partly because we need to make sure that all the weights are positive and add up to one, but mostly because of the matrix inversion in the definition of \mathbf{h} . A slightly indirect approach is actually much easier.

Write \mathbf{w}_0 for the inverse-variance weight matrix, and \mathbf{h}_0 for the hat matrix we get with those weights. Then for any other choice of weights, we have $\mathbf{h}(w) = \mathbf{h}_0 + \mathbf{c}$. Since we know WLS estimates are all unbiased, we must have

$$(\mathbf{h}_0 + \mathbf{c})\mathbf{x}\beta = \beta \quad (\text{D.3})$$

but using the inverse-variance weights is a particular WLS estimate so

$$\mathbf{h}_0 \mathbf{x} \beta = \beta \quad (\text{D.4})$$

and so we can deduce that

$$\mathbf{c}\mathbf{x} = 0 \quad (\text{D.5})$$

from unbiasedness.

Now consider the covariance matrix of the estimates, $\text{Var}[\tilde{\beta}]$. This will be $\text{Var}[(\mathbf{h}_0 + \mathbf{c})\mathbf{Y}]$, which we can expand:

$$\text{Var}[\tilde{\beta}] = \text{Var}[(\mathbf{h}_0 + \mathbf{c})\mathbf{Y}] \quad (\text{D.6})$$

$$= (\mathbf{h}_0 + \mathbf{c})\text{Var}[Y](\mathbf{h}_0 + \mathbf{c})^T \quad (\text{D.7})$$

$$= (\mathbf{h}_0 + \mathbf{c})\mathbf{w}_0^{-1}(\mathbf{h}_0 + \mathbf{c})^T \quad (\text{D.8})$$

$$= \mathbf{h}_0\mathbf{w}_0^{-1}\mathbf{h}_0^T + \mathbf{c}\mathbf{w}_0^{-1}\mathbf{h}_0^T + \mathbf{h}_0\mathbf{w}_0^{-1}\mathbf{c}^T + \mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T \quad (\text{D.9})$$

$$= (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}_0\mathbf{w}_0^{-1}\mathbf{w}_0\mathbf{x}(\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1} \quad (\text{D.10})$$

$$+ \mathbf{c}\mathbf{w}_0^{-1}\mathbf{w}_0\mathbf{x}(\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}$$

$$+ (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}_0\mathbf{w}_0^{-1}\mathbf{c}^T$$

$$+ \mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T$$

$$= (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}_0\mathbf{x}(\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1} \quad (\text{D.11})$$

$$+ \mathbf{c}\mathbf{x}(\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1} + (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}\mathbf{x}^T\mathbf{c}^T$$

$$+ \mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T$$

$$= (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1} + \mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T \quad (\text{D.12})$$

where in the last step we use the fact that $\mathbf{c}\mathbf{x} = 0$ (and so $\mathbf{x}^T\mathbf{c}^T = 0^T = 0$). Since $\mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T \geq 0$, we see that the variance is minimized by setting $\mathbf{c} = 0$, and using the inverse variance weights.

Notes:

1. The proof actually works when comparing the inverse-variance weights to any other linear, unbiased estimator; WLS with different weights is just a special case.
2. If all the variances are equal, then we've proved the optimality of OLS.
3. We can write the WLS problem as that of minimizing $(\mathbf{y} - \mathbf{x}\beta)^T\mathbf{w}(\mathbf{y} - \mathbf{x}\beta)$. If we allow \mathbf{w} to be a non-diagonal, but still positive-definite, matrix, then we have the **generalized least squares** problem. This is appropriate when there are correlations between the noise terms at different observations, i.e., when $\text{Cov}[\epsilon_i, \epsilon_j] \neq 0$ even though $i \neq j$. In this case, the proof is easily adapted to show that the optimal weight matrix \mathbf{w} is the inverse of the noise covariance matrix.

Appendix E

Constrained Optimization, Lagrange Multipliers, and Penalized Optimization

E.1 Constrained Optimization

Suppose we want to minimize¹ a function $L(u, v)$ of two variables u and v . (It could be more, but this will illustrate the pattern.) Ordinarily, we know exactly what to do: we take the derivatives of L with respect to u and to v , and solve for the u^*, v^* which makes the derivatives equal to zero, i.e., solve the system of equations

$$\frac{\partial L}{\partial u} = 0 \quad (\text{E.1})$$

$$\frac{\partial L}{\partial v} = 0 \quad (\text{E.2})$$

If necessary, we take the second derivative matrix of L and check that it is positive.

Suppose however that we want to impose a constraint on u and v , to demand that they satisfy some condition which we can express as an equation, $g(u, v) = c$. The old, unconstrained minimum u^*, v^* generally will not satisfy the constraint, so there will be a different, constrained minimum, say \hat{u}, \hat{v} . How do we find it?

We could attempt to use the constraint to eliminate either u or v — take the equation $g(u, v) = c$ and solve for u as a function of v , say $u = h(v, c)$. Then $L(u, v) = L(h(v, c), v)$, and we can minimize this over v , using the chain rule:

$$\frac{dL}{dv} = \frac{\partial L}{\partial v} + \frac{\partial L}{\partial u} \frac{\partial h}{\partial v} \quad (\text{E.3})$$

which we then set to zero and solve for v . Except in quite rare cases, this is messy.

¹Maximizing L is of course just minimizing $-L$.

E.2 Lagrange Multipliers

A superior alternative is the method of **Lagrange multipliers**. We introduce a new variable λ , the Lagrange multiplier, and a new objective function, the **Lagrangian**,

$$\mathcal{L}(u, v, \lambda) = L(u, v) + \lambda(g(u, v) - c) \quad (\text{E.4})$$

which we minimize with respect to λ , u and v and λ . That is, we solve

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \quad (\text{E.5})$$

$$\frac{\partial \mathcal{L}}{\partial u} = 0 \quad (\text{E.6})$$

$$\frac{\partial \mathcal{L}}{\partial v} = 0 \quad (\text{E.7})$$

Notice that minimize \mathcal{L} with respect to λ always gives us back the constraint equation, because $\frac{\partial \mathcal{L}}{\partial \lambda} = g(u, v) - c$. Moreover, when the constraint is satisfied, $\mathcal{L}(u, v, \lambda) = L(u, v)$. Taken together, these facts mean that the \hat{u}, \hat{v} we get from the *unconstrained* minimization of \mathcal{L} is equal to what we would find from the *constrained* minimization of L . We have encoded the constraint into the Lagrangian.

Practically, the value of this is that we know how to solve unconstrained optimization problems. The derivative with respect to λ yields, as I said, the constraint equation. The other derivatives are however yields

$$\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial L}{\partial u} + \lambda \frac{\partial g}{\partial u} \quad (\text{E.8})$$

$$\frac{\partial \mathcal{L}}{\partial v} = \frac{\partial L}{\partial v} + \lambda \frac{\partial g}{\partial v} \quad (\text{E.9})$$

Together with the constraint, this gives us as many equations as unknowns, so a solution exists.

If $\lambda = 0$, then the constraint doesn't matter — we could just as well have ignored it. When $\lambda \neq 0$, the size (and sign) of the constraint tells us about how it affects the value of the objective function at the minimum. The value of the objective function L at the constrained minimum is bigger than at the unconstrained minimum, $L(\hat{u}, \hat{v}) > L(u^*, v^*)$. Changing the level of constraint c changes how big this gap is. As we saw, $\mathcal{L}(\hat{u}, \hat{v}) = L(\hat{u}, \hat{v})$, so we can see how much influence the level of the constraint on the value of the objective function by taking the derivative of \mathcal{L} with respect to c ,

$$\frac{\partial L}{\partial c} = -\lambda \quad (\text{E.10})$$

That is, at the constrained minimum, increasing the constraint level from c to $c + \delta$, with δ very small, would change the value of the objective function by $-\lambda\delta$. (Note that λ might be negative.) This makes λ the “price”, in units of L , which we would be

willing to pay for a marginal increase in c — what economists would call the **shadow price**².

If there is more than one constraint equation, then we just introduce more multipliers, and more terms, into the Lagrangian. Each multiplier corresponds to a different constraint. The size of each multiplier indicates how much lower the objective function L could be if we relaxed that constraint — the set of shadow prices.

What about inequality constraints, $g(u, v) \leq c$? Well, either the unconstrained minimum exists in that set, in which case we don't need to worry about it, or it does not, in which case the constraint is “binding”, and we can treat this as an equality constraint³.

E.3 Penalized Optimization

So much for constrained optimization; how does this relate to penalties? Well, once we fix λ , the (u, v) which minimizes the full Lagrangian

$$L(u, v) + \lambda g(u, v) + \lambda c \quad (\text{E.11})$$

has to be the same as the one which minimizes

$$L(u, v) + \lambda g(u, v) \quad (\text{E.12})$$

This is a *penalized* optimization problem. Changing the magnitude of the penalty λ corresponds to changing the level c of the constraint. Conversely, if we start with a penalized problem, it implicitly corresponds to a constraint on the value of the penalty function $g(u, v)$. So, generally speaking, constrained optimization corresponds to penalized optimization, and vice versa.

E.4 Mini-Example: Constrained Linear Regression

To make this more concrete, let's tackle a simple one-variable statistical optimization problem, namely univariate regression through the origin, with a constraint on the slope. That is, we have the statistical model

$$Y = \beta X + \epsilon \quad (\text{E.13})$$

where ϵ is noise, and X and Y are both scalars. We want to estimate the optimal value of the slope β , but subject to the constraint that it not be too large, say $\beta^2 < c$. The unconstrained optimization problem is just least squares, i.e.,

$$L(\beta) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta x_i)^2 \quad (\text{E.14})$$

²In economics, shadow prices are internal to each decision maker, and depend on their values and resources; they are distinct from market prices, which depend on exchange and are common to all decision makers.

³A full and precise statement of this idea is the Karush-Kuhn-Tucker theorem of optimization, which you can look up.

Call the unconstrained optimum $\hat{\beta}$:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} L(\beta) \quad (\text{E.15})$$

As was said above in §E.3, there are really only two cases. Either the unconstrained optimum is inside the constraint set, i.e., $\hat{\beta}^2 < c$, or it isn't, in which case we can treat the inequality constraint like an equality. So we write out the Lagrangian

$$\mathcal{L}(\beta, \lambda) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta x_i)^2 + \lambda(\beta^2 - c) \quad (\text{E.16})$$

and we optimize:

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \quad (\text{E.17})$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = 0 \quad (\text{E.18})$$

$$(\text{E.19})$$

The first of these just gives us the constraint back again,

$$\tilde{\beta}^2 = c \quad (\text{E.20})$$

writing $\tilde{\beta}$ for the constrained optimum. The second equation is

$$\frac{1}{n} \sum_{i=1}^n 2(y_i - \tilde{\beta} x_i)(-x_i) + 2\lambda \tilde{\beta} = 0 \quad (\text{E.21})$$

(If it weren't for the λ term, we'd just solve for the slope and get, as usual, $\hat{\beta} = \frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2}$.) Now we have two unknowns, $\tilde{\beta}$ and λ , and two equations. Let's solve for λ . The equation $\tilde{\beta}^2 = c$ can also be written $\tilde{\beta} = \sqrt{c} \operatorname{sgn} \tilde{\beta}$, so, plugging in to Eq. E.21,

$$0 = \frac{2}{n} \sum_{i=1}^n x_i y_i - \sqrt{c} \operatorname{sgn} \tilde{\beta} \frac{2}{n} \sum_{i=1}^n x_i^2 + \lambda \sqrt{c} \operatorname{sgn} \tilde{\beta} \quad (\text{E.22})$$

$$\lambda = \frac{\frac{2}{n} \sum_{i=1}^n x_i y_i - c \operatorname{sgn} \tilde{\beta} \frac{2}{n} \sum_{i=1}^n x_i^2}{\sqrt{c} \operatorname{sgn} \tilde{\beta}} \quad (\text{E.23})$$

The only thing left to figure out then is $\operatorname{sgn} \tilde{\beta}$, but this just has to be the same as $\operatorname{sgn} \hat{\beta}$. (Why?)

To illustrate, I generate 100 observations from the model in Eq. E.13, with the true $\beta = 4$, X uniformly distributed on $[-1, 1]$, and ϵ having a t distribution with

2 degrees of freedom (Figure E.1). Figure E.2 shows the MSE as a function of β , i.e., the $L(\beta)$ of Eq. E.14. If \sqrt{c} is smaller than $\hat{\beta} \approx 3.95$, then the constraint is active and λ is non-zero. Figure E.3 plots λ against c from Eq. E.23. Notice how, as the constraint comes closer and closer to including the unconstrained optimum, the Lagrange multiplier λ becomes closer and closer to 0, finally crossing when $c = \hat{\beta}^2 \approx 15.6$.

Turned around, we could fix λ and try to solve the penalized optimization problem

$$\tilde{\beta} = \underset{\beta}{\operatorname{argmin}} \mathcal{L}(\beta, \lambda) \quad (\text{E.24})$$

$$= \underset{\beta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (y_i - \beta x_i)^2 + \lambda \beta^2 \quad (\text{E.25})$$

Taking the derivative with respect to β ,

$$0 = \frac{\partial \mathcal{L}}{\partial \beta} \quad (\text{E.26})$$

$$0 = \frac{1}{n} \sum_{i=1}^n 2(y_i - \tilde{\beta} x_i)(-x_i) + 2\lambda \tilde{\beta} \quad (\text{E.27})$$

$$\tilde{\beta} = \frac{\frac{1}{n} \sum_{i=1}^n x_i y_i}{\lambda + \frac{1}{n} \sum_{i=1}^n x_i^2} \quad (\text{E.28})$$

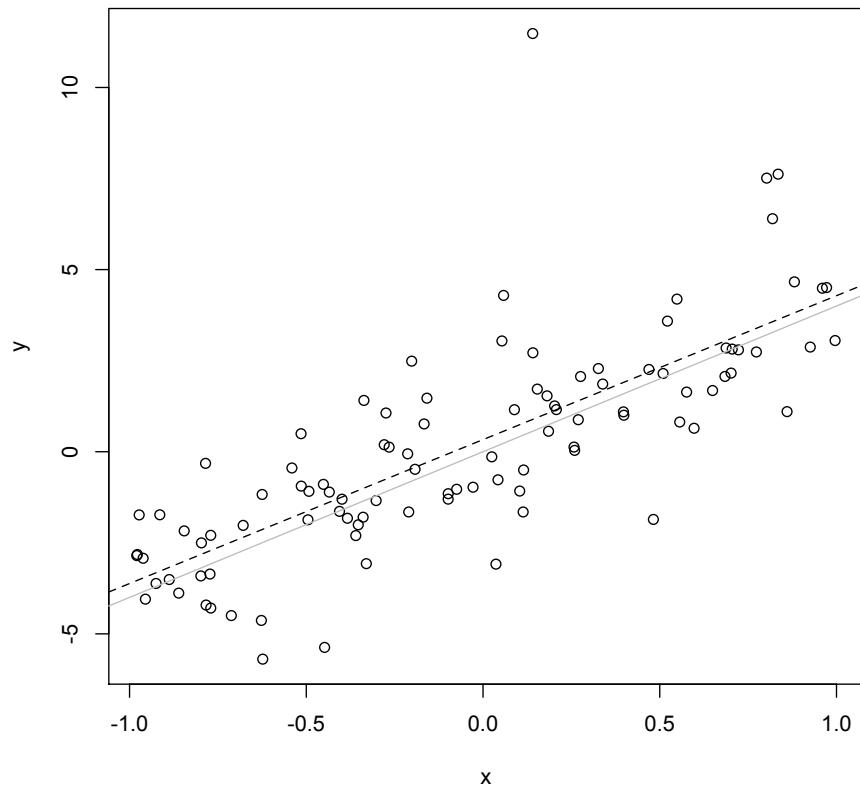
which is of course just Eq. E.21 again. Figure E.4 shows how $\tilde{\beta}$ changes with λ , while Figure E.4.1 shows how $\tilde{\beta}^2$ depends on λ . The fact that Figure E.4.1 shows the same curve as Figure E.3 only turned on its side reflects the general correspondence between penalized and constrained optimization.

E.4.1 Statistical Remark: “Ridge Regression” and “The Lasso”

The idea of penalizing or constraining the coefficients of a linear regression model can be extended to having more than one coefficient. The general case, with p covariates, is that one penalizes the sum of the squared coefficients, $\beta_1^2 + \dots + \beta_p^2$, which of course is just the squared length of the coefficient vector, $\|\beta\|^2$. This is called **ridge regression** (Hoerl and Kennard, 1970), and yields the estimates

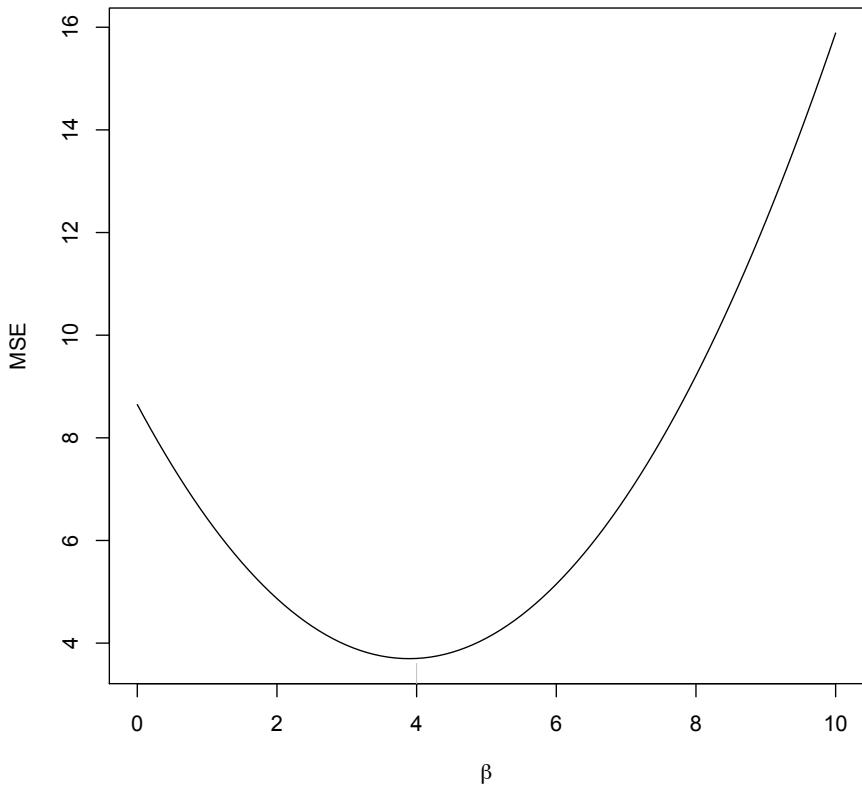
$$\tilde{\beta} = (\mathbf{x}^T \mathbf{x} + \lambda \mathbf{I})^{-1} \mathbf{x}^T \mathbf{y} \quad (\text{E.29})$$

where \mathbf{I} is the $p \times p$ identity matrix. Instead of penalizing or constraining the sum of squared coefficients, we could penalize or constrain the sum of the *absolute values* of the coefficients, $|\beta_1| + |\beta_2| + \dots + |\beta_p|$, abbreviated $\|\beta\|_1$. This is called the **lasso** (Tibshirani, 1996). It doesn’t have a nice formula like Eq. E.29, but it can be computed efficiently.



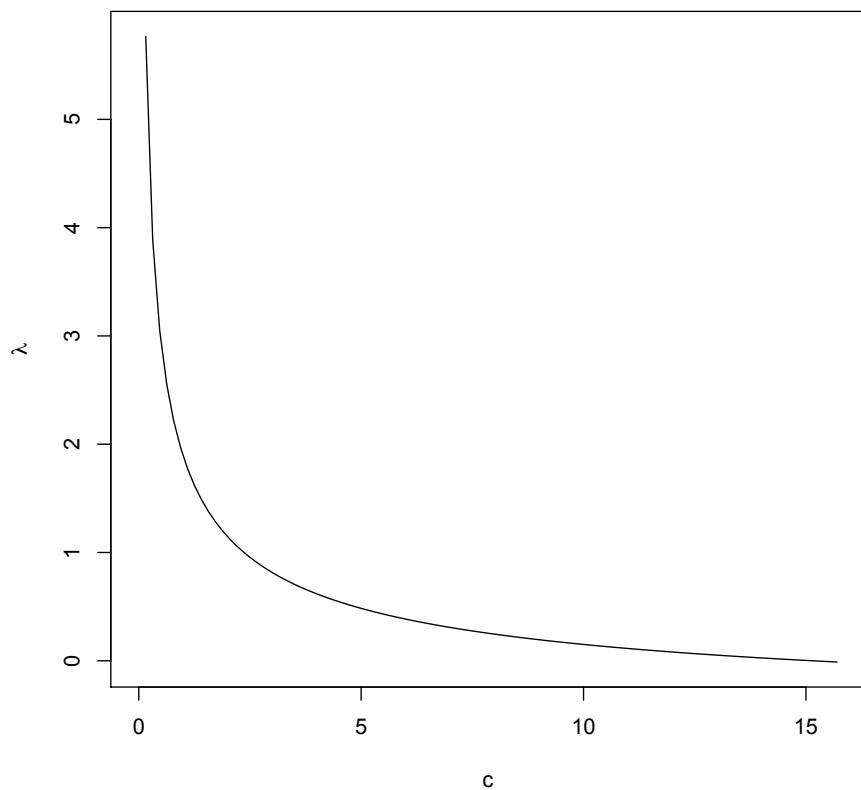
```
x <- runif(n=100,min=-1,max=1)
beta.true <- 4
y <- beta.true*x + rt(n=100,df=2)
plot(y~x)
abline(0,beta.true,col="grey")
abline(lm(y~x),lty=2)
```

Figure E.1: Example for constrained regression. Dots are data points, the grey line is the true regression line, and the dashed line is the ordinary least squares fit through the origin, without a constraint on the slope.



```
demo.mse <- function(b) { return(mean((y-b*x)^2)) }
curve(Vectorize(demo.mse)(x),from=0,to=10,xlab=expression(beta),ylab="MSE")
rug(x=beta.true,side=1,col="grey")
```

Figure E.2: Mean squared error as a function of β . The grey tick marks the true $\beta = 4$; the minimum of the curve is at $\hat{\beta} = 3.95$.

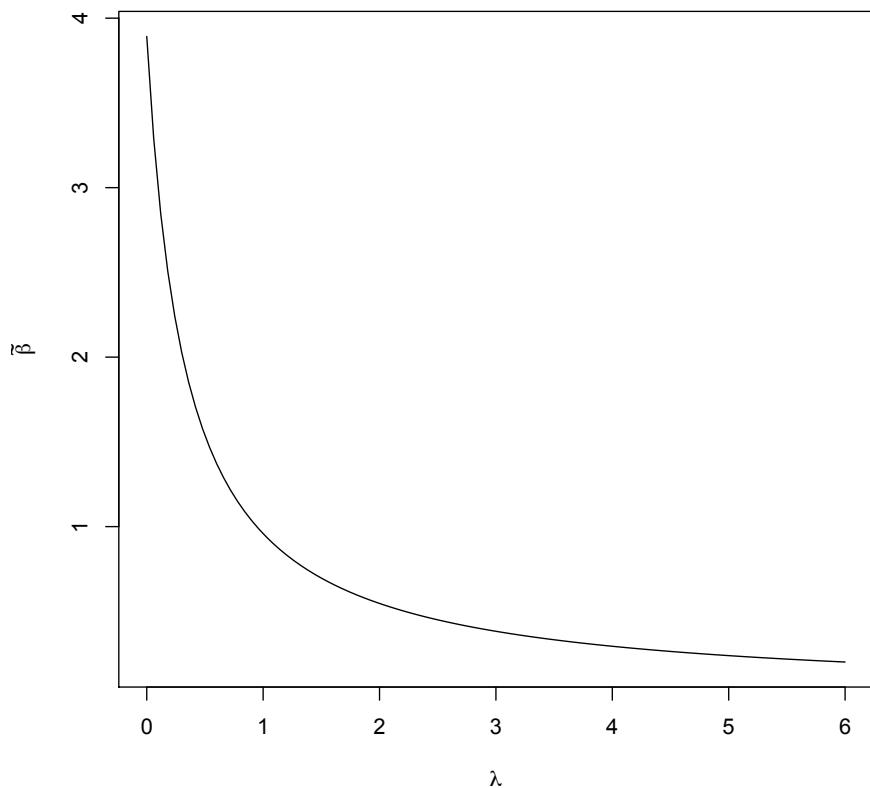


```
lambda.from.c <- function(c) {  
  return((2*mean(x*y) - sqrt(c)*2*mean(x^2))/sqrt(c))  
}  
curve(lambda.from.c(x),from=0,to=15.7,xlab="c",  
      ylab=expression(lambda))
```

Figure E.3: Calculation of λ as a function of the constraint level c , according to Eq. E.23 and the data in Figure E.1.

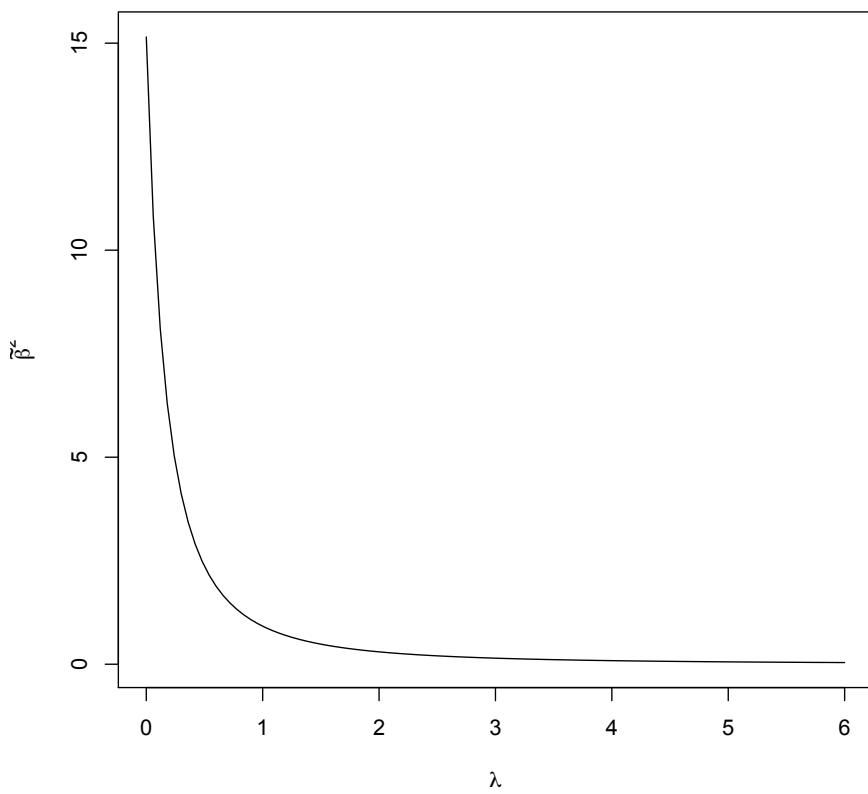
Examining Eq. E.29 should convince you that $\tilde{\beta}$ is generally smaller than the unpenalized estimate $\hat{\beta}$. (This may be easier to see from Eq. E.28.) The same is true for the lasso penalty. Both are examples of **shrinkage** estimators, called that because the usual estimate is “shrunk” towards the null model of an all-0 parameter vector. This introduces a bias, but it also reduces the variance. Shrinkage estimators are rarely very helpful in situations like the simulation example above, where the number of observations n (here = 100) is large compared to the number of parameters to estimate p (here = 1), but they can be very handy when n is close to p , and $p > n$, ordinary least squares is useless, but shrinkage estimators can still work. (Ridge regression in particular can be handy in the face of collinearity, even when $p \ll n$.) While the lasso is a bit harder to deal with mathematically and computationally than is ridge regression, it has the nice property of shrinking small coefficients to zero exactly, so that they drop out of the problem; this is especially helpful when there are really only a few predictor variables that matter, but you don’t know which.

For much more on the lasso, ridge regression, shrinkage, etc., see Hastie *et al.* (2009).



```
beta.from.lambda <- function(l) {
  return(mean(x*y)/(1+mean(x^2)))
}
curve(beta.from.lambda(x),from=0,to=6,
      xlab=expression(lambda),ylab=expression(tilde(beta)))
```

Figure E.4: The penalized estimation of the regression slope, as a function of the strength of the penalty λ .



```
curve(beta.from.lambda(x)^2,from=0,to=6,
      xlab=expression(lambda),ylab=expression(tilde(beta)^2))
```

Appendix F

Rudimentary Graph Theory

A **graph** G is built out of a set of **nodes** or **vertices**, and **edges** or **links** connecting them. The edges can either be directed or undirected. A graph with undirected edges, or an undirected graph, represents a symmetric binary relation among the nodes. For instance, in a social network, the nodes might be people, and the relationship might be “spends time with”. A graph with directed edges, or **arrows**, is called a directed graph or **digraph**¹, and represents an asymmetric relation among the nodes. To continue the social example, the arrows might mean “admires”, pointing from the admirer to the object of admiration. If the relationship is reciprocal, that is indicated by drawing a *pair* of arrows between the nodes, one in each direction (as between A and B in Figure F.1).

A **directed path** from node V_1 to node V_2 is a sequence of edges, beginning at V_1 and ending at V_2 , which is connected and which follows the orientation of the edges at each step. An **undirected path** is a sequence of connected edges ignoring orientation. (Every path in an undirected graph is undirected.) If there is a directed path from V_1 to V_2 and from V_2 to V_1 , then those two nodes are **strongly connected**. (In Figure F.1, A and C are strongly connected, but A and D are not.) If there are undirected paths in both directions, they are **weakly connected**. (A and D are weakly connected.) Strong connection implies weak connection. (EXERCISE: Prove this.) We also stipulate that every node is strongly connected to itself.

Strong connection is an equivalence relation, i.e., it is reflective, symmetric and transitive. (EXERCISE: Prove this.) Weak connection is also an equivalence relation. (EXERCISE: Prove this.) Therefore, a graph can be divided into non-overlapping **strongly connected components**, consisting of maximal sets of nodes which are all strongly connected to each other. (In Figure F.1, A , B and C form one strongly connected component, and D and E form components with just one node.) It can also be divided into **weakly connected components**, maximal sets of nodes which are all weakly connected to each other. (There is only one weakly connected component in the graph. If either of the edges into D were removed, there would be two weakly connected components.)

¹Or, more rarely, a **Guthrie diagram**.

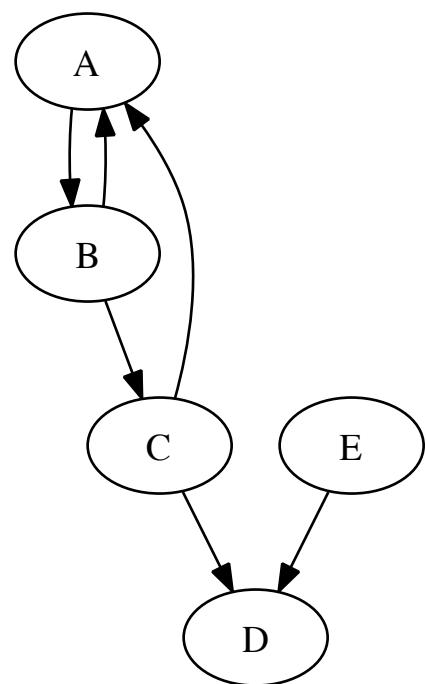


Figure F.1: Example for illustrating the concepts of graph theory.

A **cycle** is a directed path from a node to itself. The existence of two distinct nodes which are strongly connected to each other implies the existence of a cycle, and vice versa. A directed graph without cycles is called **acyclic**. Said another way, an acyclic graph is one where all the strongly connected components consist of individual nodes. The weakly connected components can however contain an unlimited number of nodes.

In a directed acyclic graph, or DAG, it is common to refer to the nodes connected by an edge as “parent” and “child” (so that the arrow runs from the parent to the child). If there is a directed path from V_1 to V_2 , then V_1 is the **ancestor** of V_2 , which is the **descendant** of V_1 . In the jargon, the ancestor/descendant relation is the **transitive closure** of the parent/child relation.

Appendix G

Pseudo-code for the SGS Algorithm

G.1 Pseudo-code for the SGS Algorithm

When you see a loop, assume that it gets entered at least once. “Replace” in the sub-functions always refers to the input graph.

```

SGS = function(set of variables V) {
     $\hat{G}$  = colliders(prune( complete undirected graph on V))
    until ( $\hat{G} == G'$ ) {
         $\hat{G} = G'$ 
         $G' = \text{orient}(\hat{G})$ 
    }
    return( $\hat{G}$ )
}

prune = function(G) {
    for each  $A, B \in V$  {
        for each  $S \subseteq V \setminus \{A, B\}$  {
            if  $A \perp\!\!\!\perp B | S$  {  $G = G \setminus (A - B)$  }
        }
    }
    return(G)
}

colliders = function(G) {
    for each  $(A - B) \in G$  {
        for each  $(B - C) \in G$  {
            if  $(A - C) \notin G$  {

```

```

collision = TRUE
for each  $S \subset B \cap V \setminus \{A, C\}$  {
    if  $A \perp\!\!\!\perp C | S$  { collision = FALSE }
}
if (collision) { replace  $(A - B)$  with  $(A \rightarrow B)$ ,  $(B - C)$  with  $(B \leftarrow C)$  }
}
}
return( $G$ )
}

orient = function( $G$ ) {
    if  $((A \rightarrow B) \in G \& (B - C) \in G \& (A - C) \notin G)$  { replace  $(B - C)$  with  $(B \rightarrow C)$  }
    if  $((\text{directed path from } A \text{ to } B) \in G \& (A - B) \in G)$  { replace  $(A - B)$  with  $(A \rightarrow B)$  }
    return( $G$ )
}

```

G.2 Pseudo-code for the PC Algorithm

[[To come]]

14:41 Thursday 25th April, 2013

Acknowledgments

Thanks to Martin Gould and especially Danny Yee for their detailed comments on earlier versions.

Thanks for specific comments and corrections to Bob Carpenter, Kathy Chen, Brad DeLong, Beatriz Estefania Etchegaray, Njál Foldnes, Rafael Izbicki, Kent Johnson, Kidong “Justin” Kim, Terra Mack, Brendan O’Connor, David Pugh, Janet E. Rosenbaum, Donald Schoolmaster, Jr., and Howard Seltman.

Bibliography

- Abarbanel, Henry D. I. (1996). *Analysis of Observed Chaotic Data*. Berlin: Springer-Verlag.
- Adler, Joseph (2009). *R in a Nutshell*. Sebastopol, California: O'Reilly.
- al Ghazali, Abu Hamid Muhammad ibn Muhammad at-Tusi (1100/1997). *The Incoherence of the Philosophers = Tahafut al-Falasifah: A Parallel English-Arabic Text*. Provo, Utah: Brigham Young University Press. Translated by Michael E. Marmura.
- Alford, J. R., C. L. Funk and J. R. Hibbing (2005). "Are Political Orientations Genetically Transmitted?" *American Political Science Review*, **99**: 153–167.
- Arceneaux, Kevin, Alan S. Gerber and Donald P. Green (2010). "A Cautionary Note on the Use of Matching to Estimate Causal Effects: An Empirical Example Comparing Matching Estimates to an Experimental Benchmark." *Sociological Methods and Research*, **39**: 256–282. doi:10.1177/0049124110378098.
- Arnold, Barry C. (1983). *Pareto Distributions*. Fairland, Maryland: International Cooperative Publishing House.
- Bai, Jushan (2003). "Testing Parametric Conditional Distributions of Dynamic Models." *The Review of Economics and Statistics*, **85**: 531–549. doi:10.1162/003465303322369704.
- Bartholomew, David J. (1987). *Latent Variable Models and Factor Analysis*. New York: Oxford University Press.
- Bartholomew, David J., Ian J. Deary and Martin Lawn (2009). "A New Lease on Life for Thomson's Bonds Model of Intelligence." *Psychological Review*, **116**: 567–579. doi:10.1037/a0016262.
- Bartlett, M. S. (1955). *An Introduction to Stochastic Processes, with Special Reference to Methods and Applications*. Cambridge, England: Cambridge University Press.
- Basharin, Gely P., Amy N. Langville and Valeriy A. Naumov (2004). "The Life and Work of A. A. Markov." *Linear Algebra and its Applications*, **386**: 3–26. URL <http://decision.cs1.uiuc.edu/~meyn/pages/Markov-Work-and-life.pdf>.

- Benaglia, Tatiana, Didier Chauveau, David R. Hunter and Derek S. Young (2009). “mixtools: An R Package for Analyzing Mixture Models.” *Journal of Statistical Software*, 32. URL <http://www.jstatsoft.org/v32/i06>.
- Bera, Anil K. and Aurobindo Ghosh (2002). “Neyman’s Smooth Test and Its Applications in Econometrics.” In *Handbook of Applied Econometrics and Statistical Inference* (Aman Ullah and Alan T. K. Wan and Anoop Chaturvedi, eds.), pp. 177–230. New York: Marcel Dekker. URL <http://ssrn.com/abstract=272888>.
- Berk, Richard A. (2004). *Regression Analysis: A Constructive Critique*. Thousand Oaks, California: Sage.
- Biecek, Przemyslaw and Teresa Ledwina (2010). *ddst: Data driven smooth test*. URL <http://CRAN.R-project.org/package=ddst>. R package, version 1.02.
- Blei, David M. and John D. Lafferty (2009). “Topic Models.” In *Text Mining: Theory and Applications* (A. Srivastava and M. Sahami, eds.). London: Taylor and Francis. URL <http://www.cs.princeton.edu/~blei/papers/BleiLafferty2009.pdf>.
- Blei, David M., Andrew Y. Ng and Michael I. Jordan (2003). “Latent Dirichlet Allocation.” *Journal of Machine Learning Research*, 3: 993–1022. URL <http://jmlr.csail.mit.edu/papers/v3/blei03a.html>.
- Boudon, Raymond (1998). “Social Mechanisms without Black Boxes.” In Hedström and Swedberg (1998), pp. 172–203.
- Bousquet, Olivier, Stéphane Boucheron and Gábor Lugosi (2004). “Introduction to Statistical Learning Theory.” In *Advanced Lectures in Machine Learning* (Olivier Bousquet and Ulrike von Luxburg and Gunnar Rätsch, eds.), pp. 169–207. Berlin: Springer-Verlag. URL http://www.econ.upf.edu/~lugosi/mlss_slt.pdf.
- Braun, W. John and Duncan J. Murdoch (2008). *A First Course in Statistical Programming with R*. Cambridge University Press.
- Bühlmann, Peter (2002). “Bootstraps for Time Series.” *Statistical Science*, 17: 52–72. URL <http://projecteuclid.org/euclid.ss/1023798998>. doi:10.1214/ss/1023798998.
- Buja, Andreas, Trevor Hastie and Robert Tibshirani (1989). “Linear Smoothers and Additive Models.” *Annals of Statistics*, 17: 453–555. URL <http://projecteuclid.org/euclid.aos/1176347115>.
- Canty, Angelo J., Anthony C. Davison, David V. Hinkley and Valérie Ventura (2006). “Bootstrap Diagnostics and Remedies.” *The Canadian Journal of Statistics*, 34: 5–27. URL <http://www.stat.cmu.edu/tr/tr726/tr726.html>.
- Carroll, Raymond J., Aurore Delaigle and Peter Hall (2009). “Nonparametric Prediction in Measurement Error Models.” *Journal of the American Statistical Association*, 104: 993–1003. doi:10.1198/jasa.2009.tm07543.

- Cavalli-Sforza, L. L., P. Menozzi and A. Piazza (1994). *The History and Geography of Human Genes*. Princeton: Princeton University Press.
- Chambers, John M. (2008). *Software for Data Analysis: Programming with R*. New York: Springer.
- Christakis, Nicholas A. and James H. Fowler (2007). “The Spread of Obesity in a Large Social Network over 32 Years.” *The New England Journal of Medicine*, 357: 370–379. URL <http://content.nejm.org/cgi/content/abstract/357/4/370>.
- Chu, Tianjiao and Clark Glymour (2008). “Search for Additive Nonlinear Time Series Causal Models.” *Journal of Machine Learning Research*, 9: 967–991. URL <http://jmlr.csail.mit.edu/papers/v9/chu08a.html>.
- Claeskens, Gerda and Nils Lid Hjort (2008). *Model Selection and Model Averaging*. Cambridge, England: Cambridge University Press.
- Clauset, Aaron, Cosma Rohilla Shalizi and M. E. J. Newman (2009). “Power-law Distributions in Empirical Data.” *SIAM Review*, 51: 661–703. URL <http://arxiv.org/abs/0706.1062>.
- Colombo, Diego, Marloes H. Maathuis, Markus Kalisch and Thomas S. Richardson (2012). “Learning High-dimensional Directed Acyclic Graphs with Latent And Selection Variables.” *Annals of Statistics*, 40: 249–321. URL <http://arxiv.org/abs/1104.5617>. doi:10.1214/11-AOS940.
- Cover, Thomas M. and Joy A. Thomas (2006). *Elements of Information Theory*. New York: John Wiley, 2nd edn.
- Cristianini, Nello and John Shawe-Taylor (2000). *An Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods*. Cambridge, England: Cambridge University Press.
- Davison, A. C. and D. V. Hinkley (1997). *Bootstrap Methods and their Applications*. Cambridge, England: Cambridge University Press.
- de Oliveira, Cesar, Richard Watt and Mark Hamer (2010). “Toothbrushing, inflammation, and risk of cardiovascular disease: results from Scottish Health Survey.” *British Medical Journal*, 340: c2451. doi:10.1136/bmj.c2451.
- Deaton, Angus (2010). “Instruments, Randomization, and Learning about Development.” *Journal of Economic Literature*, 48: 424–455. URL <http://www.princeton.edu/~deaton/downloads/deaton%20instruments%20randomization%20learning%20about%20development%20jel%202010.pdf>. doi:10.1257/jel.48.2.424.
- Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer and Richard Harshman (1990). “Indexing by Latent Semantic Analysis.” *Journal of the American Society for Information Science*, 41: 391–407. URL <http://>

- //lsa.colorado.edu/papers/JASIS.lsi.90.pdf. doi:10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.
- DeLand, Manuel (2006). *A New Philosophy of Society: Assemblage Theory and Social Complexity*. London: Continuum.
- Devroye, Luc and Gábor Lugosi (2001). *Combinatorial Methods in Density Estimation*. Berlin: Springer-Verlag.
- Didelez, Vanessa, Sha Meng and Nuala A. Sheehan (2010). “Assumptions of IV Methods for Observational Epidemiology.” *Statistical Science*, **25**: 22–40. URL <http://arxiv.org/abs/1011.0595>.
- Dinno, Alexis (2009). *LoopAnalyst: A collection of tools to conduct Levins' Loop Analysis*. URL <http://CRAN.R-project.org/package=LoopAnalyst>. R package version 1.2-2.
- Efron, Bradley (1979). “Bootstrap Methods: Another Look at the Jackknife.” *Annals of Statistics*, **7**: 1–26. URL <http://projecteuclid.org/euclid-aos/1176344552>.
- (1982). *The Jackknife, the Bootstrap, and Other Resampling Plans*. Philadelphia: SIAM Press.
- Efron, Bradley and Robert J. Tibshirani (1993). *An Introduction to the Bootstrap*. New York: Chapman and Hall.
- Eliade, Mircea (1971). *The Forge and the Crucible: The Origin and Structure of Alchemy*. New York: Harper and Row.
- Elster, Jon (1989). *Nuts and Bolts for the Social Sciences*. Cambridge, England: Cambridge University Press.
- Entner, Doris, Patrik O. Hoyer and Peter Spirtes (2013). “Data-driven Covariate Selection for Nonparametric Estimation of Causal Effects.” In *Sixteenth International Conference on Artificial Intelligence and Statistics [AIStats 2013]* (Carlos M. Carvalho and Pradeep Ravikumar, eds.). URL http://www.cs.helsinki.fi/u/entner/CovariateSelection/AISTATS2013_110.pdf.
- Ezekiel, Mordecai (1924). “A Method of Handling Curvilinear Correlation for Any Number of Variables.” *Journal of the American Statistical Association*, **19**: 431–453. URL <http://www.jstor.org/stable/2281561>.
- Fan, Jianqing and Qiwei Yao (2003). *Nonlinear Time Series: Nonparametric and Parametric Methods*. Berlin: Springer-Verlag.
- Fraser, Andrew M. (2008). *Hidden Markov Models and Dynamical Systems*. Philadelphia: SIAM Press. URL <http://www.siam.org/books/ot107/>.
- Frisch, Uriel (1995). *Turbulence: The Legacy of A. N. Kolmogorov*. Cambridge, England: Cambridge University Press.

- Galles, David and Judea Pearl (1997). “Axioms of Causal Relevance.” *Artificial Intelligence*, **97**: 9–43. URL <http://nexus.cs.usfca.edu/~galles/research/relaxiom.ps>.
- Gelman, Andrew and Iain Pardoe (2007). “Average predictive comparisons for models with nonlinearity, interactions, and variance components.” *Sociological Methodology*, **37**: 23–51. URL <http://www.stat.columbia.edu/~gelman/research/published/ape17.pdf>.
- Gelman, Andrew and Cosma Rohilla Shalizi (2013). “Philosophy and the Practice of Bayesian Statistics.” *British Journal of Mathematical and Statistical Psychology*, **66**: 8–38. URL <http://arxiv.org/abs/1006.3868>. doi:10.1111/j.2044-8317.2011.02037.x.
- Glymour, Clark (1986). “Statistics and Metaphysics.” *Journal of the American Statistical Association*, **81**: 964–966. URL <http://www.hss.cmu.edu/philosophy/glymour/glymour1986.pdf>.
- (2001). *The Mind’s Arrows: Bayes Nets and Graphical Causal Models in Psychology*. Cambridge, Massachusetts: MIT Press.
- Gray, Robert M. (1988). *Probability, Random Processes, and Ergodic Properties*. New York: Springer-Verlag. URL <http://ee.stanford.edu/~gray/arp.html>.
- Gretton, Arthur, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf and Alexander Smola (2012). “A Kernel Two-Sample Test.” *Journal of Machine Learning Research*, **13**: 723–773. URL <http://jmlr.csail.mit.edu/papers/v13/gretton12a.html>.
- Griffeath, David (1976). “Introduction to Markov Random Fields.” In *Denumerable Markov Chains* (John G. Kemeny and J. Laurie Snell and Anthony W. Knapp, eds.), pp. 425–457. Berlin: Springer-Verlag, 2nd edn.
- Grimmett, G. R. and D. R. Stirzaker (1992). *Probability and Random Processes*. Oxford: Oxford University Press, 2nd edn.
- Guttorp, Peter (1995). *Stochastic Modeling of Scientific Data*. London: Chapman and Hall.
- Guyon, Xavier (1995). *Random Fields on a Network: Modeling, Statistics, and Applications*. Berlin: Springer-Verlag.
- Hacking, Ian (1990). *The Taming of Chance*, vol. 17 of *Ideas in Context*. Cambridge, England: Cambridge University Press.
- (2001). *An Introduction to Probability and Inductive Logic*. Cambridge, England: Cambridge University Press.

- Hall, Peter, Jeff Racine and Qi Li (2004). "Cross-Validation and the Estimation of Conditional Probability Densities." *Journal of the American Statistical Association*, **99**: 1015–1026. URL <http://www.ssc.wisc.edu/~bhansen/workshop/QiLi.pdf>.
- Handcock, Mark S. and Martina Morris (1998). "Relative Distribution Methods." *Sociological Methodology*, **28**: 53–97. URL <http://www.jstor.org/pss/270964>.
- (1999). *Relative Distribution Methods in the Social Sciences*. Berlin: Springer-Verlag.
- Hart, Jeffrey D. (1997). *Nonparametric Smoothing and Lack-of-Fit Tests*. Berlin: Springer-Verlag.
- Hastie, Trevor, Robert Tibshirani and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Berlin: Springer, 2nd edn. URL <http://www-stat.stanford.edu/~tibs/ElemStatLearn/>.
- Hayfield, Tristen and Jeffrey S. Racine (2008). "Nonparametric Econometrics: The np Package." *Journal of Statistical Software*, **27**(5): 1–32. URL <http://www.jstatsoft.org/v27/i05>.
- Hedström, Peter (2005). *Dissecting the Social: On the Principles of Analytical Sociology*. Cambridge, England: Cambridge University Press.
- Hedström, Peter and Richard Swedberg (eds.) (1998). *Social Mechanisms: An Analytical Approach to Social Theory*, Studies in Rationality and Social Change, Cambridge, England. Cambridge University Press.
- Hoerl, Arthur E. and Robert W. Kennard (1970). "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics*, **12**. URL <http://www.jstor.org/pss/1267351>.
- Hofmann, Thomas (1999). "Probabilistic Latent Semantic Analysis." In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference [UAI 1999]* (Kathryn Laskey and Henri Prade, eds.), pp. 289–296. San Francisco: Morgan Kaufmann. URL http://uai.sis.pitt.edu/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=179&proceeding_id=15.
- Holland, Paul W. (1986). "Statistics and Causal Inference." *Journal of the American Statistical Association*, **81**: 945–970.
- Honerkamp, Josef (2002). *Statistical Physics: An Advanced Approach with Applications*. Berlin: Springer-Verlag, 2nd edn. Translated by Thomas Filk.
- Hoyer, Patrik O., Dominik Janzing, Joris Mooij, Jonas Peters and Bernhard Schölkopf (2009). "Nonlinear causal discovery with additive noise models." In *Advances in Neural Information Processing Systems 21 [NIPS 2008]* (Daphne Koller and D. Schuurmans and Y. Bengio and L. Bottou, eds.), pp. 689–696. Cambridge, Massachusetts: MIT Press. URL http://books.nips.cc/papers/files/nips21/NIPS2008_0266.pdf.

- Hume, David (1739). *A Treatise of Human Nature: Being an Attempt to Introduce the Experimental Method of Reasoning into Moral Subjects*. London: John Noon. Reprint (Oxford: Clarendon Press, 1951) of original edition, with notes and analytical index.
- Iyigun, Murat (2008). "Luther and Suleyman." *Quarterly Journal of Economics*, **123**: 1465–1494. URL <http://www.colorado.edu/Economics/courses/iyigun/ottoman081506.pdf>. doi:10.1162/qjec.2008.123.4.1465.
- Jacobs, Robert A. (1997). "Bias/Variance Analyses of Mixtures-of-Experts Architectures." *Neural Computation*, **9**: 369–383.
- Janzing, Dominik (2007). "On causally asymmetric versions of Occam's Razor and their relation to thermodynamics." E-print, arxiv.org. URL <http://arxiv.org/abs/0708.3411>.
- Janzing, Dominik and Daniel Herrmann (2003). "Reliable and Efficient Inference of Bayesian Networks from Sparse Data by Statistical Learning Theory." Electronic preprint. URL <http://arxiv.org/abs/cs.LG/0309015>.
- Jordan, Michael I. (ed.) (1998). *Learning in Graphical Models*, Dordrecht. Kluwer Academic.
- Jordan, Michael I. and Robert A. Jacobs (1994). "Hierarchical Mixtures of Experts and the EM Algorithm." *Neural Computation*, **6**: 181–214.
- Jordan, Michael I. and Terrence J. Sejnowski (eds.) (2001). *Graphical Models: Foundations of Neural Computation*, Computational Neuroscience, Cambridge, Massachusetts. MIT Press.
- Kalisch, Markus and Peter Bühlmann (2007). "Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm." *Journal of Machine Learning Research*, **8**: 616–636. URL <http://jmlr.csail.mit.edu/papers/v8/kalisch07a.html>.
- Kalisch, Markus, Martin Mächler and Diego Colombo (2010). *pcalg: Estimation of CPDAG/PAG and causal inference using the IDA algorithm*. URL <http://CRAN.R-project.org/package=pcalg>. R package version 1.1-2.
- Kalisch, Markus, Martin Mächler, Diego Colombo, Marloes H. Maathuis and Peter Bühlmann (2012). "Causal Inference Using Graphical Models with the R Package pcalg." *Journal of Statistical Software*, **47**(11): 1–26. URL <http://www.jstatsoft.org/v47/i11>.
- Kallenberg, Wilbert C. M. and Teresa Ledwina (1997). "Data-Driven Smooth Tests When the Hypothesis Is Composite." *Journal of the American Statistical Association*, **92**: 1094–1104. URL <http://doc.utwente.nl/62408/>.
- Kanai, Ryota, Tom Feilden, Colin Firth and Geraint Rees (2011). "Political Orientations Are Correlated with Brain Structure in Young Adults." *Current Biology*, **21**: 677–680. doi:10.1016/j.cub.2011.03.017.

- Kantz, Holger and Thomas Schreiber (2004). *Nonlinear Time Series Analysis*. Cambridge, England: Cambridge University Press, 2nd edn.
- Kao, Yi-hao and Benjamin Van Roy (2011). “Learning a Factor Model via Regularized PCA.” *Journal of Machine Learning Research*, submitted. URL <http://arxiv.org/abs/1111.6201>.
- Kearns, Michael J. and Umesh V. Vazirani (1994). *An Introduction to Computational Learning Theory*. Cambridge, Massachusetts: MIT Press.
- Kelly, Kevin T. (2007). “Ockham’s razor, empirical complexity, and truth-finding efficiency.” *Theoretical Computer Science*, 383: 270–289. doi:10.1016/j.tcs.2007.04.009.
- Kindermann, Ross and J. Laurie Snell (1980). *Markov Random Fields and their Applications*. Providence, Rhode Island: American Mathematical Society. URL http://www.ams.org/online_bks/conm1/.
- Kogan, Barry S. (1985). *Averroes and the Metaphysics of Causation*. Albany, New York: State University of New York Press.
- Kullback, Solomon (1968). *Information Theory and Statistics*. New York: Dover Books, 2nd edn.
- Künsch, Hans R. (1989). “The Jackknife and the Bootstrap for General Stationary Observations.” *Annals of Statistics*, 17: 1217–1241. URL <http://projecteuclid.org/euclid.aos/1176347265>.
- Lacerda, Gustavo, Peter Spirtes, Joseph Ramsey and Patrik Hoyer (2008). “Discovering Cyclic Causal Models by Independent Components Analysis.” In *Proceedings of the Twenty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-08)*, pp. 366–374. Corvallis, Oregon: AUAI Press. URL <http://uai.sis.pitt.edu/papers/08/p366-lacerda.pdf>.
- Lahiri, S. N. (2003). *Resampling Methods for Dependent Data*. New York: Springer-Verlag.
- Landauer, Thomas K. and Susan T. Dumais (1997). “A Solution to Plato’s Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge.” *Psychological Review*, 104: 211–240. URL <http://lsa.colorado.edu/papers/plato.annotate.html>.
- Lauritzen, Steffen L. (1984). “Extreme Point Models in Statistics.” *Scandinavian Journal of Statistics*, 11: 65–91. URL <http://www.jstor.org/pss/4615945>. With discussion and response.
- (1996). *Graphical Models*. New York: Oxford University Press.
- Leisch, Friedrich (2004). “FlexMix: A General Framework for Finite Mixture Models and Latent Class Regression in R.” *Journal of Statistical Software*, 11. URL <http://www.jstatsoft.org/v11/i08>.

- Li, Ching Chun (1975). *Path Analysis: A Primer*. Pacific Grove, California: The Boxwood Press.
- Li, Ching Chun, Sati Mazumdar and B. Raja Rao (1975). "Partial Correlation in Terms of Path Coefficients." *The American Statistician*, 29: 89–90. URL <http://www.jstor.org/stable/2683271>.
- Li, Ming and Paul M. B. Vitányi (1997). *An Introduction to Kolmogorov Complexity and Its Applications*. New York: Springer-Verlag, 2nd edn.
- Li, Qi and Jeffrey Scott Racine (2007). *Nonparametric Econometrics: Theory and Practice*. Princeton, New Jersey: Princeton University Press.
- Lindsey, J. K. (2004). *Statistical Analysis of Stochastic Processes in Time*. Cambridge, England: Cambridge University Press.
- Liu, Han, John Lafferty and Larry Wasserman (2009). "The Nonparanormal: Semiparametric Estimation of High Dimensional Undirected Graphs." *Journal of Machine Learning Research*, 10: 2295–2328. URL <http://jmlr.csail.mit.edu/papers/v10/liu09a.html>.
- Liu, Ka-Yuet, Marissa King and Peter S. Bearman (2010). "Social Influence and the Autism Epidemic." *American Journal of Sociology*, 115: 1387–1434. URL [http://www.understandingautism.columbia.edu/papers/social-influence-and-the-autism-epidemic-\(2010\).pdf](http://www.understandingautism.columbia.edu/papers/social-influence-and-the-autism-epidemic-(2010).pdf). doi:10.1086/651448.
- Loehlin, John C. (1992). *Latent Variable Models: An Introduction to Factor, Path, and Structural Analysis*. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 2nd edn.
- Maathuis, Marloes H., Diego Colombo, Markus Kalisch and Peter Bühlmann (2010). "Predicting Causal Effects in Large-scale Systems from Observational Data." *Nature Methods*, 7: 247–248. URL <http://stat.ethz.ch/Manuscripts/buhlmanna/maathuisetal2010.pdf>. doi:10.1038/nmeth0410-247. See also <http://stat.ethz.ch/Manuscripts/buhlmanna/maathuisetal2010SI.pdf>.
- Maathuis, Marloes H., Markus Kalisch and Peter Bühlmann (2009). "Estimating High-Dimensional Intervention Effects from Observational Data." *Annals of Statistics*, 37: 3133–3164. URL <http://arxiv.org/abs/0810.4214>. doi:10.1214/09-AOS685.
- MacCulloch, Diarmaid (2004). *The Reformation: A History*. New York: Penguin.
- Maguire, B. A., E. S. Pearson and A. H. A. Wynn (1952). "The Time Intervals between Industrial Accidents." *Biometrika*, 39: 168–180. URL <http://www.jstor.org/pss/2332475>.
- Mandelbrot, Benoit (1962). "The Role of Sufficiency and of Estimation in Thermodynamics." *Annals of Mathematical Statistics*, 33: 1021–1038. URL <http://projecteuclid.org/euclid.aoms/1177704470>.

- Manski, Charles F. (2007). *Identification for Prediction and Decision*. Cambridge, Massachusetts: Harvard University Press.
- Matloff, Norman (2011). *The Art of R Programming: A Tour of Statistical Software Design*. San Francisco: No Starch Press.
- McGee, Leonard A. and Stanley F. Schmidt (1985). *Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry*. Tech. Rep. 86847, NASA Technical Memorandum. URL http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19860003843_1986003843.pdf.
- Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller (1953). “Equations of State Calculations by Fast Computing Machines.” *Journal of Chemical Physics*, **21**: 1087–1092. doi:10.1063/1.1699114.
- Mohri, Mehryar, Afshin Rostamizadeh and Ameet Talwalkar (2012). *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: MIT Press.
- Morgan, Stephen L. and Christopher Winship (2007). *Counterfactuals and Causal Inference: Methods and Principles for Social Research*. Cambridge, England: Cambridge University Press.
- Neal, Radford M. and Geoffrey E. Hinton (1998). “A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants.” In Jordan (1998), pp. 355–368. URL <http://www.cs.toronto.edu/~radford/em.abstract.html>.
- Newey, Whitney K. and James L. Powell (2003). “Instrumental Variable Estimation of Nonparametric Models.” *Econometrica*, **71**: 1565–1578. doi:10.1111/1468-0262.00459.
- Novembre, John and Matthew Stephens (2008). “Interpreting principal component analyses of spatial population genetic variation.” *Nature Genetics*, **40**: 646–649. doi:10.1038/ng.139.
- Packard, Norman H., James P. Crutchfield, J. Doyne Farmer and Robert S. Shaw (1980). “Geometry from a Time Series.” *Physical Review Letters*, **45**: 712–716.
- Paige, Robert L. and A. Alexandre Trindade (2010). “The Hodrick-Prescott Filter: A special case of penalized spline smoothing.” *Electronic Journal of Statistics*, **4**: 856–874. URL <http://projecteuclid.org/euclid.ejs/1284557751>.
- Pearl, Judea (1988). *Probabilistic Reasoning in Intelligent Systems*. New York: Morgan Kaufmann.
- (2000). *Causality: Models, Reasoning, and Inference*. Cambridge, England: Cambridge University Press.
- (2009a). “Causal inference in statistics: An overview.” *Statistics Surveys*, **3**: 96–146. URL <http://projecteuclid.org/euclid.ssu/1255440554>.

- (2009b). *Causality: Models, Reasoning, and Inference*. Cambridge, England: Cambridge University Press, 2nd edn.
- Peterson, Robert A. (2000). “A Meta-Analysis of Variance Accounted for and Factor Loadings in Exploratory Factor Analysis.” *Marketing Letters*, 11: 261–275.
- Pitman, E. J. G. (1979). *Some Basic Theory for Statistical Inference*. London: Chapman and Hall.
- Pollard, David (1989). “Asymptotics via Empirical Processes.” *Statistical Science*, 4: 341–354. URL <http://projecteuclid.org/euclid.ss/1177012394>. doi:10.1214/ss/1177012394.
- Porter, Theodore M. (1986). *The Rise of Statistical Thinking, 1820–1900*. Princeton, New Jersey: Princeton University Press.
- Puccia, Charles J. and Richard Levins (1985). *Qualitative Modeling of Complex Systems: An Introduction to Loop Analysis and Time Averaging*. Cambridge, Massachusetts: Harvard University Press.
- Quiñonero-Candela, Joaquin, Masashi Sugiyama, Anton Schwaighofer and Neil D. Lawrence (eds.) (2009). *Dataset Shift in Machine Learning*. Cambridge, Massachusetts: MIT Press.
- Raginsky, Maxim (2011). “Directed Information and Pearl’s Causal Calculus.” In *Proceedings of the 49th Annual Allerton Conference on Communication, Control and Computing*, p. forthcoming. URL <http://arxiv.org/abs/1110.0718>.
- Rayner, J. C. W. and D. J. Best (1989). *Smooth Tests of Goodness of Fit*. Oxford: Oxford University Press.
- Reichenbach, Hans (1956). *The Direction of Time*. Berkeley: University of California Press. Edited by Maria Reichenbach.
- Richardson, Thomas (1996). “A Discovery Algorithm for Directed Cyclic Graphs.” In *Proceedings of the Proceedings of the Twelfth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pp. 454–446. San Francisco, CA: Morgan Kaufmann. URL <ftp://ftp.andrew.cmu.edu/pub/phil/thomas/TR68.ps>. URL is for expanded version.
- Robins, James M., Richard Scheines, Peter Spirtes and Larry Wasserman (2003). “Uniform Consistency in Causal Inference.” *Biometrika*, 90: 491–515. URL <http://www.stat.cmu.edu/tr/tr725/tr725.html>.
- Rosenbaum, Paul and Donald Rubin (1983). “The Central Role of the Propensity Score in Observational Studies for Causal Effects.” *Biometrika*, 70: 41–55. URL <http://www.jstor.org/stable/2335942>.
- Rosenzweig, Mark R. and Kenneth I. Wolpin (2000). “Natural ‘Natural Experiments’ in Economics.” *Journal of Economic Literature*, 38: 827–874. doi:10.1257/jel.38.4.827.

- Rubin, Donald B. (2006). *Matched Sampling for Causal Effects*. Cambridge, England: Cambridge University Press.
- Rubin, Donald B. and Richard P. Waterman (2006). “Estimating the Causal Effects of Marketing Interventions Using Propensity Score Methodology.” *Statistical Science*, 21: 206–222. URL <http://arxiv.org/abs/math.ST/0609201>.
- Ruelle, David (1991). *Chance and Chaos*. Princeton, New Jersey: Princeton University Press.
- Russell, Bertrand (1927). *The Analysis of Matter*. International Library of Philosophy, Psychology and Scientific Method. London: K. Paul Trench, Trubner and Co. Reprinted New York: Dover Books, 1954.
- Salmon, Wesley C. (1984). *Scientific Explanation and the Causal Structure of the World*. Princeton: Princeton University Press.
- Sandhaus, Evan (2008). “The New York Times Annotated Corpus.” Electronic database. URL <http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2008T19>.
- Schwarz, Gideon (1978). “Estimating the Dimension of a Model.” *Annals of Statistics*, 6: 461–464. URL <http://projecteuclid.org/euclid-aos/1176344136>.
- Sethna, James P. (2006). *Statistical Mechanics: Entropy, Order Parameters, and Complexity*. Oxford: Oxford University Press. URL <http://pages.physics.cornell.edu/sethna/StatMech/>.
- Shalizi, Cosma Rohilla (2007). “Maximum Likelihood Estimation and Model Testing for q -Exponential Distributions.” *Physical Review E*, submitted. URL <http://arxiv.org/abs/math.ST/0701854>.
- Shalizi, Cosma Rohilla and Andrew C. Thomas (2011). “Homophily and Contagion Are Generically Confounded in Observational Social Network Studies.” *Sociological Methods and Research*, 40: 211–239. URL <http://arxiv.org/abs/1004.4704>. doi:10.1177/0049124111404820.
- Shannon, Claude E. (1948). “A Mathematical Theory of Communication.” *Bell System Technical Journal*, 27: 379–423. URL <http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html>. Reprinted in Shannon and Weaver (1963).
- Shannon, Claude E. and Warren Weaver (1963). *The Mathematical Theory of Communication*. Urbana, Illinois: University of Illinois Press.
- Shields, Paul C. (1996). *The Ergodic Theory of Discrete Sample Paths*. Providence, Rhode Island: American Mathematical Society.
- Shpitser, Ilya and Judea Pearl (2008). “Complete Identification Methods for the Causal Hierarchy.” *Journal of Machine Learning Research*, 9: 1941–1979. URL <http://jmlr.csail.mit.edu/papers/v9/shpitser08a.html>.

- Shumway, Robert H. and David S. Stoffer (2000). *Time Series Analysis and Its Applications*. New York: Springer-Verlag.
- Simonoff, Jeffrey S. (1996). *Smoothing Methods in Statistics*. Berlin: Springer-Verlag.
- Solow, Robert M. (1970). *Growth Theory: An Exposition*. Radcliffe Lectures, University of Warwick, 1969. Oxford: Oxford University Press. New edition with the 1987 Nobel lecture.
- Spanos, Aris (2011). “A Frequentist Interpretation of Probability for Model-based Inductive Inference.” *Synthese*, forthcoming. URL http://www.econ.vt.edu/faculty/2008vitas_research/Spanos/1Spanos-2011-Synthese.pdf. doi:10.1007/s11229-011-9892-x.
- Spearman, Charles (1904). ““General Intelligence,” Objectively Determined and Measured.” *American Journal of Psychology*, 15: 201–293. URL <http://psychclassics.yorku.ca/Spearman/>.
- Spirites, Peter, Clark Glymour and Richard Scheines (1993). *Causation, Prediction, and Search*. Berlin: Springer-Verlag, 1st edn.
- (2001). *Causation, Prediction, and Search*. Cambridge, Massachusetts: MIT Press, 2nd edn.
- Sriperumbudur, Bharath K., Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf and Gert R.G. Lanckriet (2010). “Hilbert Space Embeddings and Metrics on Probability Measures.” *Journal of Machine Learning Research*, 11: 1517–1561. URL <http://jmlr.csail.mit.edu/papers/v11/sriperumbudur10a.html>.
- Stuart, Elizabeth A. (2010). “Matching Methods for Causal Inference: A Review and a Look Forward.” *Statistical Science*, 25: 1–21. URL <http://arxiv.org/abs/1010.5586>. doi:10.1214/09-STS313.
- Székely, Gábor J. and Maria L. Rizzo (2009). “Brownian Distance Covariance.” *Annals of Applied Statistics*, 3: 1236–1265. URL <http://arxiv.org/abs/1010.0297>. doi:10.1214/09-AOAS312. With discussion and reply.
- Thomson, Godfrey H. (1916). “A Hierarchy without a General Factor” *British Journal of Psychology*, 8: 271–281.
- (1939). *The Factorial Analysis of Human Ability*. Boston: Houghton Mifflin Company. URL <http://www.archive.org/details/factorialanal032965mbp>.
- Thurstone, L. L. (1934). “The Vectors of Mind.” *Psychological Review*, 41: 1–32. URL <http://psychclassics.yorku.ca/Thurstone/>.
- Tibshirani, Robert (1996). “Regression Shrinkage and Selection via the Lasso.” *Journal of the Royal Statistical Society B*, 58: 267–288. URL <http://www-stat.stanford.edu/~tibs/lasso/lasso.pdf>.

- Tibshirani, Ryan J. and Robert Tibshirani (2009). “A Bias Correction for the Minimum Error Rate in Cross-Validation.” *Annals of Applied Statistics*, 3: 822–829. URL <http://arxiv.org/abs/0908.2904>.
- Tilly, Charles (1984). *Big Structures, Large Processes, Huge Comparisons*. New York: Russell Sage Foundation.
- (2008). *Explaining Social Processes*. Boulder, Colorado: Paradigm Publishers.
- Tukey, John W. (1954). “Unsolved Problems of Experimental Statistics.” *Journal of the American Statistical Association*, 49: 706–731. URL <http://www.jstor.org/pss/2281535>.
- Vapnik, Vladimir N. (2000). *The Nature of Statistical Learning Theory*. Berlin: Springer-Verlag, 2nd edn.
- Vidyasagar, M. (2003). *Learning and Generalization: With Applications to Neural Networks*. Berlin: Springer-Verlag, 2nd edn.
- von Luxburg, Ulrike and Bernhard Schölkopf (2008). “Statistical Learning Theory: Models, Concepts, and Results.” E-print, arxiv.org. URL <http://arxiv.org/abs/0810.4752>.
- von Plato, Jan (1994). *Creating Modern Probability: Its Mathematics, Physics and Philosophy in Historical Perspective*. Cambridge, England: Cambridge University Press.
- Vuong, Quang H. (1989). “Likelihood Ratio Tests for Model Selection and Non-Nested Hypotheses.” *Econometrica*, 57: 307–333. URL <http://www.jstor.org/pss/1912557>.
- Wahba, Grace (1990). *Spline Models for Observational Data*. Philadelphia: Society for Industrial and Applied Mathematics.
- Wasserman, Larry (2003). *All of Statistics: A Concise Course in Statistical Inference*. Berlin: Springer-Verlag.
- (2006). *All of Nonparametric Statistics*. Berlin: Springer-Verlag.
- Whittaker, E. T. (1922). “On a New Method of Graduation.” *Proceedings of the Edinburgh Mathematical Society*, 41: 63–75. doi:10.1017/S001309150000359X.
- Wiener, Norbert (1961). *Cybernetics: Or, Control and Communication in the Animal and the Machine*. Cambridge, Massachusetts: MIT Press, 2nd edn. First edition New York: Wiley, 1948.
- Winkler, Gerhard (1995). *Image Analysis, Random Fields and Dynamic Monte Carlo Methods: A Mathematical Introduction*. Berlin: Springer-Verlag.
- Wood, Simon N. (2006). *Generalized Additive Models: An Introduction with R*. Boca Raton, Florida: Chapman and Hall/CRC.

Wright, Sewall (1934). “The Method of Path Coefficients.” *Annals of Mathematical Statistics*, 5: 161–215. URL <http://projecteuclid.org/euclid.aoms/1177732676>.

Zhang, Kun, Jonas Peters, Dominik Janzing and Bernhard Schölkopf (2011). “Kernel-based Conditional Independence Test and Application in Causal Discovery.” In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)* (Fabio Gagliardi Cozman and Avi Pfeffer, eds.), pp. 804–813. Corvallis, Oregon: AUAI Press. URL <http://arxiv.org/abs/1202.3775>.