

Advanced Data Analysis from an Elementary Point of View

Cosma Rohilla Shalizi

Spring 2013
Last L^AT_EX'd Wednesday 1st April, 2015

Contents

Introduction	16
To the Reader	16
Concepts You Should Know	17
I Regression and Its Generalizations	19
1 Regression Basics	20
1.1 Statistics, Data Analysis, Regression	20
1.2 Guessing the Value of a Random Variable	21
1.2.1 Estimating the Expected Value	22
1.3 The Regression Function	22
1.3.1 Some Disclaimers	23
1.4 Estimating the Regression Function	26
1.4.1 The Bias-Variance Tradeoff	26
1.4.2 The Bias-Variance Trade-Off in Action	28
1.4.3 Ordinary Least Squares Linear Regression as Smoothing	28
1.5 Linear Smoothers	33
1.5.1 k -Nearest-Neighbor Regression	33
1.5.2 Kernel Smoothers	35
1.6 Exercises	36
2 The Truth about Linear Regression	39
2.1 Optimal Linear Prediction: Multiple Variables	39
2.1.1 Collinearity	41
2.1.2 The Prediction and Its Error	41
2.1.3 Estimating the Optimal Linear Predictor	42
2.1.3.1 Unbiasedness and Variance of Ordinary Least Squares Estimates	43
2.2 Shifting Distributions, Omitted Variables, and Transformations	44
2.2.1 Changing Slopes	44
2.2.1.1 R^2 : Distraction or Nuisance?	44
2.2.2 Omitted Variables and Shifting Distributions	46
2.2.3 Errors in Variables	46

2.2.4	Transformation	50
2.3	Adding Probabilistic Assumptions	54
2.3.1	Examine the Residuals	55
2.3.2	On Significant Coefficients	55
2.4	Linear Regression Is Not the Philosopher’s Stone	57
2.5	Further Reading	58
2.6	Exercises	58
3	Model Evaluation	60
3.1	What Are Statistical Models For?	60
3.2	Errors, In and Out of Sample	61
3.3	Over-Fitting and Model Selection	65
3.4	Cross-Validation	70
3.4.1	Data-set Splitting	71
3.4.2	k -Fold Cross-Validation (CV)	73
3.4.3	Leave-one-out Cross-Validation	73
3.5	Warnings	76
3.5.1	Parameter Interpretation	76
3.6	Further Reading	78
3.7	Exercises	78
4	Smoothing in Regression	79
4.1	How Much Should We Smooth?	79
4.2	Adapting to Unknown Roughness	82
4.2.1	Bandwidth Selection by Cross-Validation	88
4.2.2	Convergence of Kernel Smoothing and Bandwidth Scaling	90
4.2.3	Summary on Kernel Smoothing in 1D	93
4.3	Kernel Regression with Multiple Inputs	93
4.4	Interpreting Smoothers: Plots	94
4.5	Average Predictive Comparisons	98
4.6	Computational Advice: <code>npreg</code>	101
4.7	Further Reading	107
4.8	Exercises	107
5	Simulation	108
5.1	What Do We Mean by “Simulation”?	108
5.2	How Do We Simulate Stochastic Models?	109
5.2.1	Chaining Together Random Variables	109
5.2.2	Random Variable Generation	109
5.2.2.1	Built-in Random Number Generators	109
5.2.2.2	Transformations	110
5.2.2.3	Quantile Method	110
5.2.3	Sampling	111
5.2.3.1	Sampling Rows from Data Frames	112
5.2.3.2	Multinomials and Multinoullis	112
5.2.3.3	Probabilities of Observation	112

5.3	Repeating Simulations	113
5.4	Why Simulate?	113
5.4.1	Understanding the Model; Monte Carlo	114
5.4.2	Checking the Model	114
5.4.2.1	“Exploratory” Analysis of Simulations	115
5.4.3	Sensitivity Analysis	117
5.5	Further Reading	119
5.6	Exercises	119
6	The Bootstrap	120
6.1	Stochastic Models, Uncertainty, Sampling Distributions	120
6.2	The Bootstrap Principle	122
6.2.1	Variances and Standard Errors	124
6.2.2	Bias Correction	125
6.2.3	Confidence Intervals	125
6.2.3.1	Other Bootstrap Confidence Intervals	126
6.2.4	Hypothesis Testing	128
6.2.4.1	Double bootstrap hypothesis testing	128
6.2.5	Parametric Bootstrapping Example: Pareto’s Law of Wealth Inequality	129
6.3	Resampling	132
6.3.1	Parametric vs. Nonparametric Bootstrapping	136
6.4	Bootstrapping Regression Models	136
6.4.1	Re-sampling Points: Parametric Example	136
6.4.2	Re-sampling Points: Non-parametric Example	139
6.4.3	Re-sampling Residuals: Example	142
6.5	Bootstrap with Dependent Data	144
6.6	Things Bootstrapping Does Poorly	144
6.7	Which Bootstrap When?	145
6.8	Further Reading	146
6.9	Exercises	146
7	Weighting and Variance	147
7.1	Weighted Least Squares	147
7.2	Heteroskedasticity	149
7.2.1	Weighted Least Squares as a Solution to Heteroskedasticity . .	151
7.2.2	Some Explanations for Weighted Least Squares	153
7.2.3	Finding the Variance and Weights	155
7.3	Conditional Variance Function Estimation	156
7.3.1	Iterative Refinement of Mean and Variance: An Example . . .	157
7.3.2	Real Data Example: Old Heteroskedastic	161
7.4	Re-sampling Residuals with Heteroskedasticity	161
7.5	Local Linear Regression	164
7.5.1	For and Against Locally Linear Regression	166
7.5.2	Lowess	167
7.6	Exercises	169

8 Splines	170
8.1 Smoothing by Penalizing Curve Flexibility	170
8.1.1 The Meaning of the Splines	171
8.2 Computational Example: Splines for Stock Returns	172
8.2.1 Confidence Bands for Splines	175
8.3 Basis Functions and Degrees of Freedom	178
8.3.1 Basis Functions	178
8.3.2 Degrees of Freedom	180
8.4 Splines in Multiple Dimensions	182
8.5 Smoothing Splines versus Kernel Regression	182
8.6 Some of the Math Behind Splines	182
8.7 Further Reading	184
8.8 Exercises	185
9 Additive Models	187
9.1 Additive Models	187
9.2 Partial Residuals and Back-fitting	188
9.2.1 Back-fitting for Linear Models	188
9.2.2 Backfitting Additive Models	190
9.3 The Curse of Dimensionality	190
9.4 Example: California House Prices Revisited	192
9.5 Closing Modeling Advice	206
9.6 Further Reading	206
9.7 Exercises	207
10 Testing Regression Specifications	208
10.1 Testing Functional Forms	208
10.1.1 Examples of Testing a Parametric Model	210
10.1.2 Remarks	215
10.2 Why Use Parametric Models At All?	220
10.3 Why We Sometimes Want Mis-Specified Parametric Models	220
10.4 Further Reading	224
10.5 Exercises	224
11 More about Hypothesis Testing	225
12 Logistic Regression	226
12.1 Modeling Conditional Probabilities	226
12.2 Logistic Regression	227
12.2.1 Likelihood Function for Logistic Regression	231
12.3 Numerical Optimization of the Likelihood	231
12.3.1 Iteratively Re-Weighted Least Squares	232
12.4 Generalized Linear and Additive Models	233
12.4.1 Generalized Additive Models	234
12.5 Model Checking	234
12.5.1 Residuals	234

12.5.2 Non-parametric Alternatives	235
12.5.3 Calibration	235
12.6 A Toy Example	236
12.7 Weather Forecasting in Snoqualmie Falls	242
12.8 Logistic Regression with More Than Two Classes	255
12.9 Exercises	255
13 GLMs and GAMs	256
13.1 Generalized Linear Models and Iterative Least Squares	256
13.1.1 GLMs in General	258
13.1.2 Examples of GLMs	258
13.1.2.1 Vanilla Linear Models	258
13.1.2.2 Binomial Regression	259
13.1.2.3 Poisson Regression	259
13.1.3 Uncertainty	260
13.1.4 Modeling Dispersion	260
13.1.5 Likelihood and Deviance	260
13.1.5.1 Maximum Likelihood and the Choice of Link Function	261
13.1.6 R: <code>glm</code>	262
13.2 Generalized Additive Models	262
13.3 Further Reading	262
13.4 Exercises	263
14 Trees	264
14.1 Prediction Trees	264
14.2 Regression Trees	267
14.2.1 Example: California Real Estate Again	267
14.2.2 Regression Tree Fitting	272
14.2.2.1 Cross-Validation and Pruning in R	275
14.2.2.3 Uncertainty in Regression Trees	276
14.3 Classification Trees	280
14.3.1 Measuring Information	281
14.3.2 Making Predictions	281
14.3.3 Measuring Error	282
14.3.3.1 Misclassification Rate	282
14.3.3.2 Average Loss	282
14.3.3.3 Likelihood and Cross-Entropy	283
14.3.3.4 Neyman-Pearson Approach	284
14.4 Further Reading	285
14.5 Exercises	286

II Multivariate Data, Distributions, and Latent Structure	287
15 Multivariate Distributions	288
15.1 Review of Definitions	288
15.2 Multivariate Gaussians	289
15.2.1 Linear Algebra and the Covariance Matrix	291
15.2.2 Conditional Distributions and Least Squares	292
15.2.3 Projections of Multivariate Gaussians	292
15.2.4 Computing with Multivariate Gaussians	292
15.3 Inference with Multivariate Distributions	293
15.3.1 Estimation	293
15.3.2 Model Comparison	294
15.3.3 Goodness-of-Fit	296
15.4 Exercises	297
16 Density Estimation	298
16.1 Histograms Revisited	298
16.2 “The Fundamental Theorem of Statistics”	299
16.3 Error for Density Estimates	300
16.3.1 Error Analysis for Histogram Density Estimates	301
16.4 Kernel Density Estimates	303
16.4.1 Analysis of Kernel Density Estimates	303
16.4.2 Joint Density Estimates	305
16.4.3 Categorical and Ordered Variables	306
16.4.4 Practicalities	307
16.4.5 Kernel Density Estimation in R: An Economic Example	307
16.5 Conditional Density Estimation	309
16.5.1 Practicalities and a Second Example	310
16.6 More on the Expected Log-Likelihood Ratio	313
16.7 Simulating from Density Estimates	314
16.7.1 Simulating from Kernel Density Estimates	314
16.7.1.1 Sampling from a Joint Density	315
16.7.1.2 Sampling from a Conditional Density	315
16.7.2 Drawing from Histogram Estimates	316
16.7.3 Examples of Simulating from Kernel Density Estimates	316
16.8 Exercises	321
17 Relative Distributions and Smooth Tests	323
17.1 Smooth Tests of Goodness of Fit	323
17.1.1 From Continuous CDFs to Uniform Distributions	323
17.1.2 Testing Uniformity	324
17.1.3 Neyman’s Smooth Test	324
17.1.3.1 Choice of Function Basis	326
17.1.3.2 Choice of Number of Basis Functions	327
17.1.3.3 Application: Combining p -Values	327
17.1.3.4 Density Estimation by Series Expansion	330

17.1.4	Smooth Tests of Non-Uniform Parametric Families	330
17.1.4.1	Estimated Parameters	332
17.1.5	Implementation in R	332
17.1.5.1	Some Examples	333
17.1.6	Conditional Distributions and Calibration	336
17.2	Relative Distributions	337
17.2.1	Estimating the Relative Distribution	339
17.2.2	R Implementation and Examples	339
17.2.2.1	Example: Conservative versus Liberal Brains	339
17.2.2.2	Example: Economic Growth Rates	343
17.2.3	Adjusting for Covariates	344
17.2.3.1	Example: Adjusting Growth Rates	346
17.3	Further Reading	349
17.4	Exercises	349
18	Principal Components Analysis	350
18.1	Mathematics of Principal Components	350
18.1.1	Minimizing Projection Residuals	351
18.1.2	Maximizing Variance	353
18.1.3	More Geometry; Back to the Residuals	354
18.1.3.1	Scree Plots	355
18.1.4	Statistical Inference, or Not	356
18.2	Example 1: Cars	356
18.3	Example 2: The United States <i>circa</i> 1977	361
18.4	Latent Semantic Analysis	364
18.4.1	Principal Components of the New York <i>Times</i>	365
18.5	PCA for Visualization	367
18.6	PCA Cautions	370
18.7	Further Reading	370
18.8	Exercises	373
19	Factor Analysis	375
19.1	From PCA to Factor Analysis	375
19.1.1	Preserving correlations	377
19.2	The Graphical Model	377
19.2.1	Observables Are Correlated Through the Factors	379
19.2.2	Geometry: Approximation by Linear Subspaces	380
19.3	Roots of Factor Analysis in Causal Discovery	382
19.4	Estimation	383
19.4.1	Degrees of Freedom	384
19.4.2	A Clue from Spearman's One-Factor Model	385
19.4.3	Estimating Factor Loadings and Specific Variances	386
19.5	Maximum Likelihood Estimation	387
19.5.1	Alternative Approaches	388
19.5.2	Estimating Factor Scores	388
19.6	The Rotation Problem	388

19.7	Factor Analysis as a Predictive Model	389
19.7.1	How Many Factors?	390
19.7.1.1	R^2 and Goodness of Fit	391
19.8	Factor Models versus PCA Once More	392
19.9	Examples in R	393
19.9.1	Example 1: Back to the US <i>circa</i> 1977	393
19.9.2	Example 2: Stocks	397
19.10	Reification, and Alternatives to Factor Models	397
19.10.1	The Rotation Problem Again	397
19.10.2	Factors or Mixtures?	398
19.10.3	The Thomson Sampling Model	399
19.11	Further Reading	404
20	Nonlinear Dimensionality Reduction	406
20.1	Why We Need Nonlinear Dimensionality Reduction	406
20.2	Local Linearity and Manifolds	410
20.3	Locally Linear Embedding (LLE)	412
20.3.1	Finding Neighborhoods	413
20.3.2	Finding Weights	414
20.3.2.1	$k > p$	415
20.3.3	Finding Coordinates	415
20.4	More Fun with Eigenvalues and Eigenvectors	417
20.4.1	Finding the Weights	417
20.4.1.1	$k > p$	418
20.4.2	Finding the Coordinates	419
20.5	Calculation	420
20.5.1	Finding the Nearest Neighbors	420
20.5.2	Calculating the Weights	423
20.5.3	Calculating the Coordinates	427
20.6	Diffusion Maps	429
20.6.1	Diffusion-Map Coordinates	433
20.6.1.1	Fun with Transition Matrices	434
20.6.1.2	Multiple Scales	437
20.6.1.3	Choosing q	438
20.6.2	What to Do with the Diffusion Map Once You Have It	438
20.6.2.1	Spectral Clustering	438
20.6.2.2	Asymmetry	439
20.6.3	The Kernel Trick	440
20.7	Exercises	441
21	Mixture Models	442
21.1	Two Routes to Mixture Models	442
21.1.1	From Factor Analysis to Mixture Models	442
21.1.2	From Kernel Density Estimates to Mixture Models	442
21.1.3	Mixture Models	443
21.1.4	Geometry	444

CONTENTS	10
----------	----

21.1.5 Identifiability	444
21.1.6 Probabilistic Clustering	445
21.1.7 Simulation	446
21.2 Estimating Parametric Mixture Models	446
21.2.1 More about the EM Algorithm	448
21.2.2 Further Reading on and Applications of EM	451
21.2.3 Topic Models and Probabilistic LSA	451
21.3 Non-parametric Mixture Modeling	451
21.4 Worked Computating Example	452
21.4.1 Mixture Models in R	452
21.4.2 Fitting a Mixture of Gaussians to Real Data	452
21.4.3 Calibration-checking for the Mixture	456
21.4.4 Selecting the Number of Components by Cross-Validation	458
21.4.5 Interpreting the Mixture Components, or Not	463
21.4.6 Hypothesis Testing for Mixture-Model Selection	468
21.5 Exercises	471
22 Graphical Models	472
22.1 Conditional Independence and Factor Models	472
22.2 Directed Acyclic Graph (DAG) Models	473
22.2.1 Conditional Independence and the Markov Property	474
22.3 Examples of DAG Models and Their Uses	475
22.3.1 Missing Variables	478
22.4 Non-DAG Graphical Models	479
22.4.1 Undirected Graphs	479
22.4.2 Directed but Cyclic Graphs	481
22.5 Further Reading	481
III Causal Inference	484
23 Graphical Causal Models	485
23.1 Causation and Counterfactuals	485
23.2 Causal Graphical Models	486
23.2.1 Calculating the “effects of causes”	487
23.2.2 Back to Teeth	488
23.3 Conditional Independence and <i>d</i> -Separation	491
23.3.1 D-Separation Illustrated	492
23.3.2 Linear Graphical Models and Path Coefficients	495
23.3.3 Positive and Negative Associations	496
23.4 Independence and Information	497
23.5 Further Reading	498
23.6 Exercises	499

24 Identifying Causal Effects	500
24.1 Causal Effects, Interventions and Experiments	500
24.1.1 The Special Role of Experiment	501
24.2 Identification and Confounding	502
24.3 Identification Strategies	505
24.3.1 The Back-Door Criterion: Identification by Conditioning	507
24.3.1.1 The Entner Rules	508
24.3.2 The Front-Door Criterion: Identification by Mechanisms	510
24.3.2.1 The Front-Door Criterion and Mechanistic Explanation	511
24.3.3 Instrumental Variables	512
24.3.3.1 Some Invalid Instruments	514
24.3.3.2 Critique of Instrumental Variables	514
24.3.4 Failures of Identification	518
24.4 Summary	520
24.4.1 Further Reading	520
24.5 Exercises	521
25 Estimating Causal Effects	522
25.1 Estimators in the Back- and Front- Door Criteria	522
25.1.1 Estimating Average Causal Effects	523
25.1.2 Avoiding Estimating Marginal Distributions	523
25.1.3 Propensity Scores	524
25.1.4 Matching and Propensity Scores	526
25.2 Instrumental-Variables Estimates	528
25.3 Uncertainty and Inference	529
25.4 Recommendations	529
25.5 Further Reading	530
25.6 Exercises	531
26 Discovering Causal Structure	532
26.1 Testing DAGs	533
26.2 Testing Conditional Independence	534
26.3 Faithfulness and Equivalence	535
26.3.1 Partial Identification of Effects	536
26.4 Causal Discovery with Known Variables	536
26.4.1 The PC Algorithm	539
26.4.2 Causal Discovery with Hidden Variables	540
26.4.3 On Conditional Independence Tests	540
26.5 Software and Examples	541
26.6 Limitations on Consistency of Causal Discovery	546
26.7 Further Reading	547
26.8 Exercises	547
26.9 Pseudo-code for the SGS and PC Algorithms	548
26.9.1 The SGS Algorithm	548
26.9.2 The PC Algorithm	549

CONTENTS	12
27 Experimental Design	550
IV Dependent Data	551
28 Time Series	552
28.1 Time Series, What They Are	552
28.2 Stationarity	553
28.2.1 Autocorrelation	554
28.2.2 The Ergodic Theorem	556
28.2.2.1 The World's Simplest Ergodic Theorem	556
28.2.2.2 Rate of Convergence	557
28.2.2.3 Why Ergodicity Matters	558
28.3 Markov Models	559
28.3.1 Meaning of the Markov Property	560
28.4 Autoregressive Models	561
28.4.1 Autoregressions with Covariates	563
28.4.2 Additive Autoregressions	563
28.4.3 Linear Autoregression	563
28.4.3.1 "Unit Roots" and Stationary Solutions	567
28.4.4 Conditional Variance	569
28.4.5 Regression with Correlated Noise; Generalized Least Squares .	569
28.5 Bootstrapping Time Series	572
28.5.1 Parametric or Model-Based Bootstrap	572
28.5.2 Block Bootstraps	572
28.5.3 Sieve Bootstrap	573
28.6 Trends and De-Trending	575
28.6.1 Forecasting Trends	577
28.6.2 Seasonal Components	582
28.6.3 Detrending by Differencing	582
28.6.4 Bootstrapping with Trends	583
28.7 Further Reading	583
28.8 Exercises	585
29 Time Series with Latent Variables	586
30 Simulation-Based Inference	587
30.1 The Method of Simulated Moments	587
30.1.1 The Method of Moments	587
30.1.2 Adding in the Simulation	588
30.1.3 An Example: Moving Average Models and the Stock Market .	588
30.2 Exercises	593
30.3 Appendix: Some Design Notes on the Method of Moments Code .	596
31 Longitudinal, Spatial and Network Data	598

V Data-Analysis Problem Sets	599
32 What's That Got to Do with the Price of Condos in California?	601
33 The Advantages of Backwardness	605
34 The Size of a Cat's Heart	609
35 It's Not the Heat that Gets You	612
36 Nice Demo City, but Will It Scale?	616
36.1 Version 1	616
36.1.1 Background	616
36.1.2 Data	617
36.1.3 Tasks and Questions	618
36.2 Version 2	618
36.2.1	618
36.2.2	620
36.2.2.1 Data	620
36.2.3 Problems	621
37 Diabetes	623
38 Fair's Affairs	626
39 How the North American Paleofauna Got a Crook in Its Regression Line	628
40 How the <i>Hyracotherium</i> Got Its Mass	632
41 How the Recent Mammals Got Their Size Distribution	635
42 Red Brain, Blue Brain	638
43 Patterns of Exchange	641
44 Is This Assignment Really Necessary?	644
45 Mystery Multivariate Analysis	646
46 Separated at Birth	649
47 Brought to You by the Letters D, A and G	653
48 Estimating with DAGs	659
49 Use and Abuse of Conditioning	663
50 What Makes the Union Strong?	665

CONTENTS	14
51 An Insufficiently Random Walk Down Wall Street	669
52 Macroeconomic Forecasting	673
53 Debt Needs Time for What It Kills to Grow In	676
 Appendices	 679
A Linear Algebra	679
A.1 Eigenvalues and Eigenvectors of Matrices	679
A.1.1 Singular Value Decomposition	679
A.1.2 Square Root of a Matrix	680
A.2 Special Kinds of Matrix	680
A.3 Orthonormal Bases	680
A.4 Orthogonal Projections	680
A.5 Function Spaces	681
A.5.1 Bases	681
A.5.2 Eigenvalues and Eigenfunctions of Operators	681
A.6 Further Reading	682
B Big O and Little o Notation	683
C Taylor Expansions	685
D Propagation of Error	687
E Optimization Theory	689
E.1 Basic Concepts of Optimization	689
E.2 Small-Noise Asymptotics for Optimization	691
E.2.1 Application to Maximum Likelihood	696
E.2.2 Aside: The Akaike Information Criterion	696
E.3 Constrained and Penalized Optimization	697
E.3.1 Constrained Optimization	697
E.3.2 Lagrange Multipliers	697
E.3.3 Penalized Optimization	699
E.3.4 Constrained Linear Regression	699
E.3.5 Statistical Remark: “Ridge Regression” and “The Lasso”	703
F Optimization Methods	705
F.1 Optimization with First- and Second-Derivatives	705
F.1.1 Gradient Descent	705
F.1.2 Newton’s Method	706
F.1.2.1 Newton’s Method in More than One Dimension	709
F.1.3 Stochastic Approximation	711
F.1.3.1 Stochastic Newton’s Method	712
F.1.4 Pros and Cons of Stochastic Gradient Methods	713

F.2	Derivative-Free Optimization Techniques	713
F.2.1	Nelder-Mead, a.k.a. the Simplex Method	713
F.2.2	Simulated Annealing	714
F.3	Methods for Constraints	714
F.4	R Notes	714
G	χ^2 and Likelihood Ratios	715
H	Proof of the Gauss-Markov Theorem	718
I	Rudimentary Graph Theory	720
J	Uncorrelated versus Independent	723
K	Information Theory	726
L	Programming	727
L.1	Functions	727
L.2	First Example: Pareto Quantiles	728
L.3	Functions Which Call Functions	729
L.3.1	Sanity-Checking Arguments	731
L.4	Layering Functions and Debugging	732
L.4.1	More on Debugging	734
L.5	Automating Repetition and Passing Arguments	736
L.6	Avoiding Iteration: Manipulating Objects	745
L.6.1	<code>ifelse</code> and <code>which</code>	746
L.6.2	<code>apply</code> and Its Variants	747
L.7	More Complicated Return Values	749
L.8	Re-Writing Your Code: An Extended Example	750
L.9	General Advice on Programming	756
L.9.1	Comment your code	756
L.9.2	Use meaningful names	757
L.9.3	Check whether your program works	757
L.9.4	Avoid writing the same thing twice	758
L.9.5	Start from the beginning and break it down	758
L.9.6	Break your code into many short, meaningful functions	758
L.10	Further Reading	759
M	Generating Random Variables	760
M.1	Rejection Method	760
M.2	The Metropolis Algorithm and Markov Chain Monte Carlo	763
M.3	Generating Uniform Random Numbers	764
M.4	Further Reading	765
M.5	Exercises	768

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 once learned but have 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.

Exercises and Problem Sets

There are two kinds of assignments included here. Mathematical and computational exercises go at the end of chapters, since they are mostly connected to those pieces of content. (Many of them are complements to, or filling in details of, material in the chapters.) There are also data-centric problem sets, in Part V; most of these draw on material from multiple chapters, and also many of them are based on specific papers. If you are teaching a course based on the book, feel free to write for solutions; unfortunately, providing solutions to those using the book for self-study is not feasible.

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.

Drafts and Updates The page for this book is <http://www.stat.cmu.edu/~cshalizi/ADAFaEPoV/>. The latest version will live there. The book 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. While the book is still in a draft, the PDF contains a lot of notes to myself for revisions, [[like so]]; you can ignore them.

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 — we also want to answer “what if?” questions — 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**:

$$\text{MSE}(\alpha) \equiv E[(Y - \alpha)^2] \tag{1.1}$$

²The origin of the name is instructive (Stigler, 1986). 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 .

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

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

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

[[TODO: Replace this with real data?]]

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

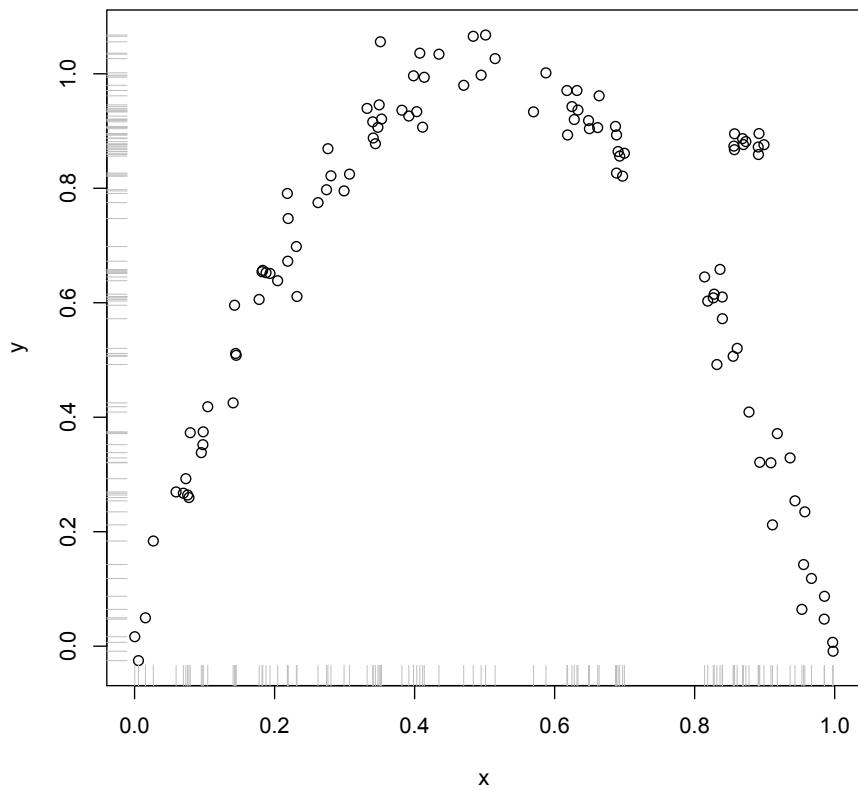
$$Y \leftarrow r(X) + \epsilon \quad (1.15)$$

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

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

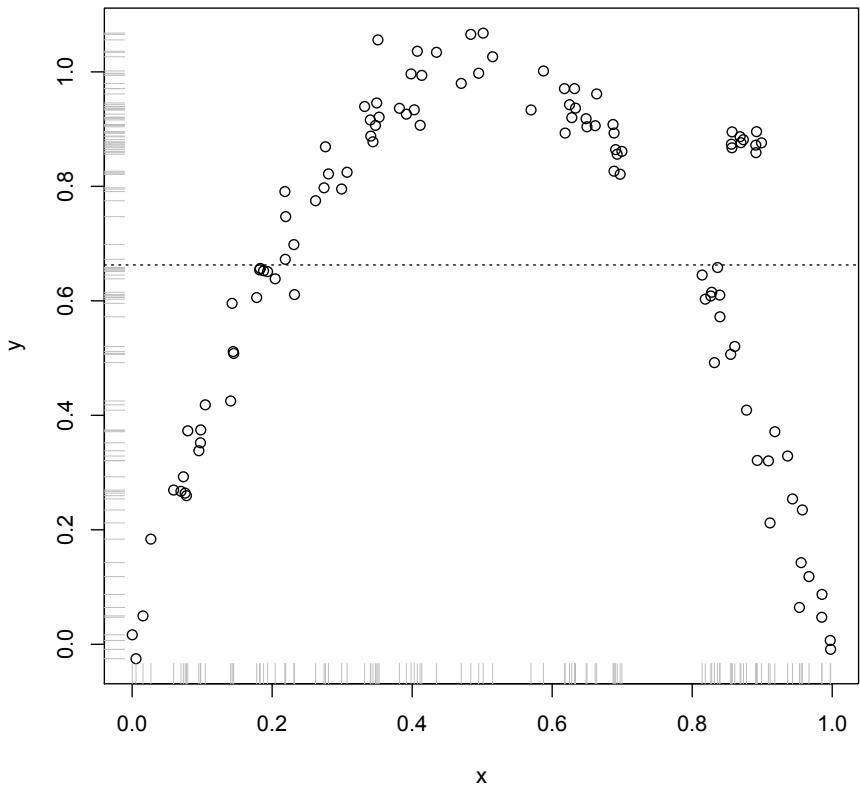
where $\eta(X)$ is a noise variable with mean zero, but as the notation indicates the distribution of the noise generally depends on X .

³We will cover causal inference in 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 (made up) running example data. `rug()` 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 $r(x) = E[Y|X = x]$, 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 $\hat{r}(x) \rightarrow E[Y|X = x]$.

Unfortunately, this *only* works when X takes values in a finite set. If X is continuous, then in general the probability of our getting a sample at any *particular* value is zero, as 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 generally not equal to $E[Y|X = x_i]$. 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.

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

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

Eq. 1.16 tells us 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 not just on the method, but also on how well the method matches the data-generating process, 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

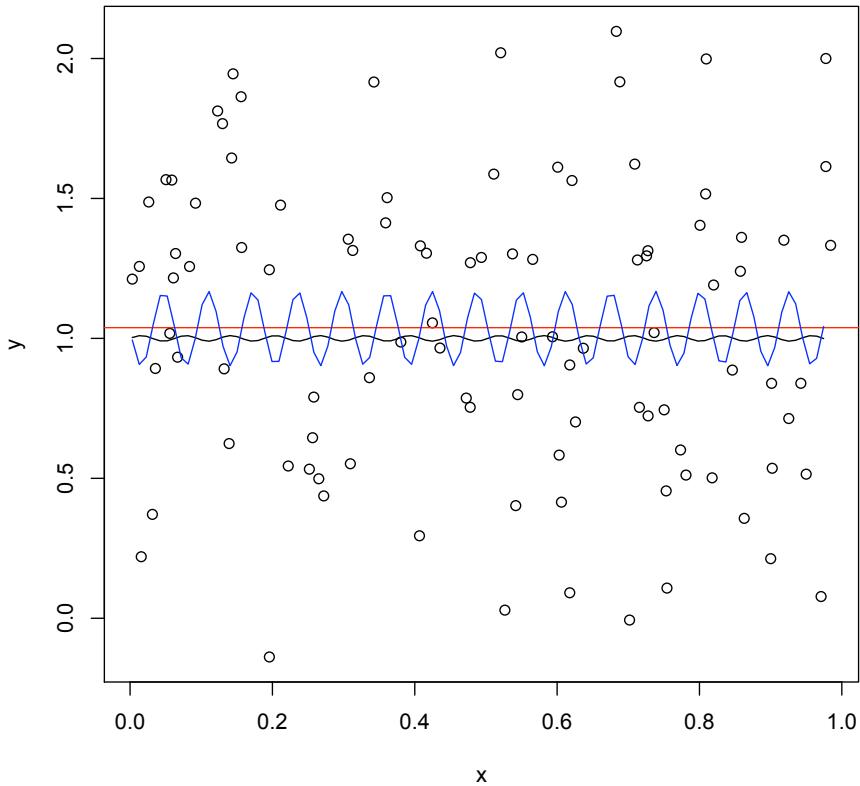
Let's take an extreme example: we could decide to approximate $r(x)$ by a constant r_0 . The implicit smoothing here is very strong, but sometimes appropriate. For instance, it's appropriate when $r(x)$ really is a constant! Then trying to estimate any additional structure in the regression function 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. We'll assume that the predictor variable X is one dimensional, and that both X and Y are

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*r))
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., $r_0 + a \sin 100x$. 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!

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.

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

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

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

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

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:

$$\hat{a} = 0 \quad (1.41)$$

$$\hat{b} = \frac{\sum_i y_i x_i}{\sum_i x_i^2} \quad (1.42)$$

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.

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

$$= x \frac{\sum_i y_i x_i}{\sum_i x_i^2} \quad (1.44)$$

$$= \sum_i y_i \frac{x_i}{\sum_j x_j^2} x \quad (1.45)$$

$$= \sum_i y_i \frac{x_i}{ns_X^2} x \quad (1.46)$$

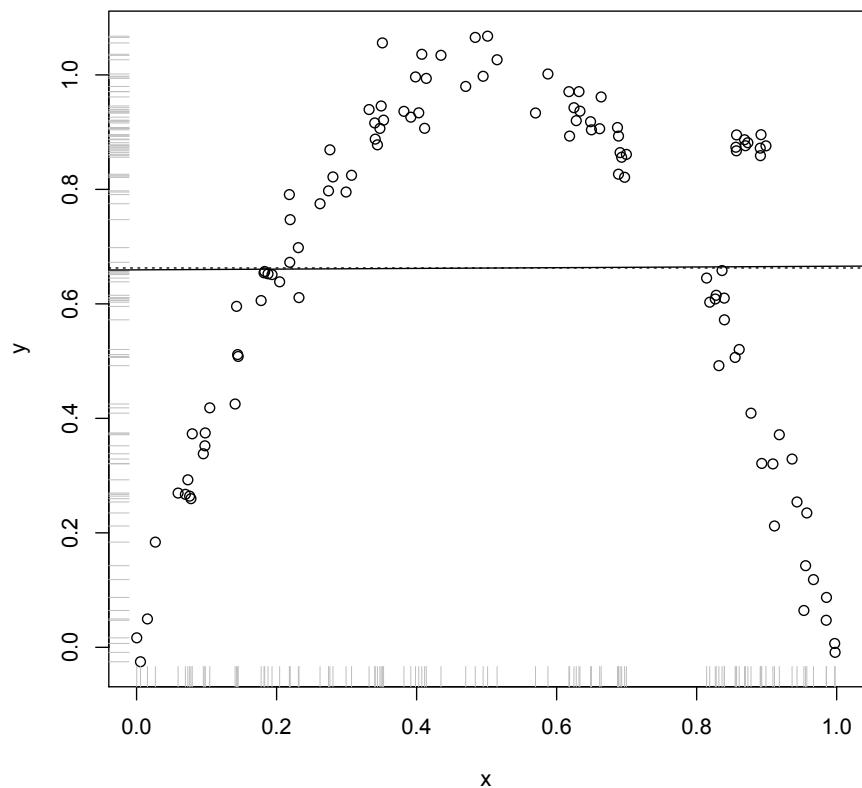
where s_X^2 is the sample variance of X . In words, our prediction is a weighted average of the observed values y_i of the dependent variable, where the weights are proportional to how far x_i is from the center (relative to the variance), and proportional to the magnitude of x . If x_i is on the same 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 and 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 $\hat{w}(x_i, x) = (x_i / ns_X^2)x$ (from Eq. 1.46).

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,

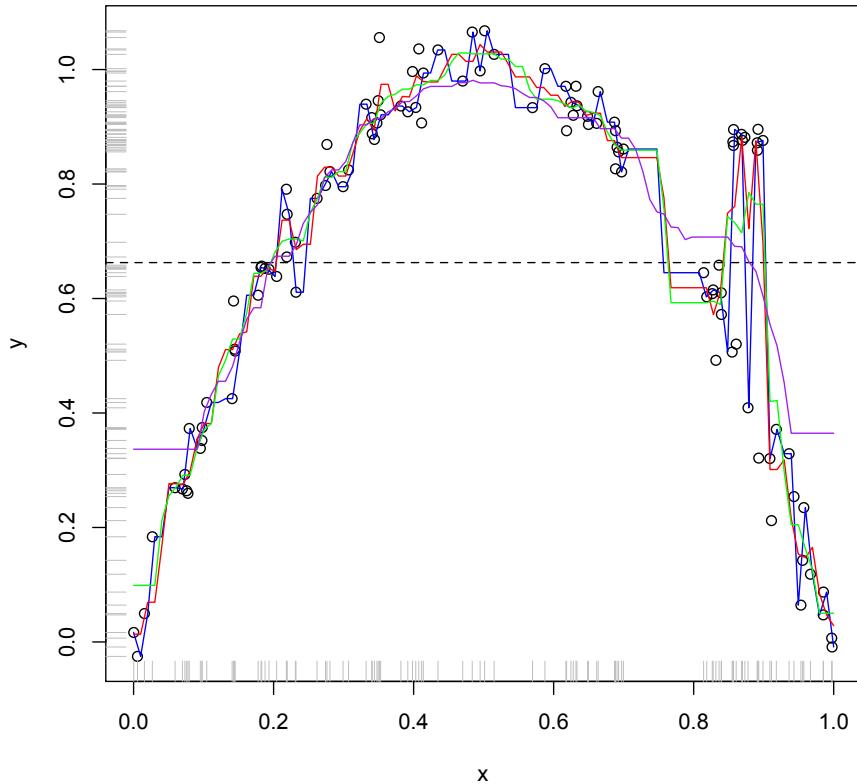
$$\hat{w}(x_i, x) = \begin{cases} 1/k & x_i \text{ one of the } k \text{ nearest neighbors of } x \\ 0 & \text{otherwise} \end{cases} \quad (1.49)$$

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

1.6 Exercises

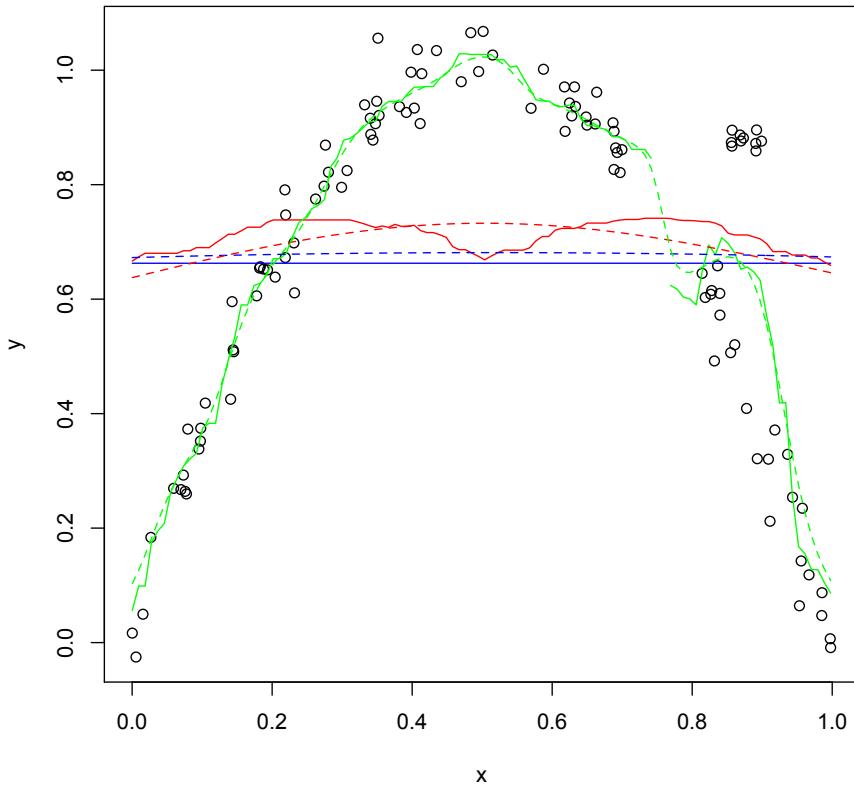
- Suppose we use the **mean absolute error** instead of the mean squared error:

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

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

- Derive Eqs. 1.41 and 1.42 by minimizing Eq. 1.40.
- What does it mean to say that Gaussian kernel regression approaches nearest-neighbor regression as $h \rightarrow 0$? Why does it do so? Is this true for all kinds of 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 , $K(x_j, x) \propto e^{-L^2/2h^2} e^{-(x_j-x_i)L/2h^2} e^{-(x_j-x_i)^2/2h^2} \ll e^{-L^2/2h^2}$. — 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, $h = 2$; red, $h = 1$; green, $h = 0.1$ (per the definition of bandwidth in the `ksmooth` function). Note the abrupt jump around $x = 0.75$ in the box-kernel/ $h = 0.1$ (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.

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 towns in Illinois in the 1950s. [[TODO: Citation!]] 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.
Hints: `diff`, `plot`.
- (d) Linearly regress the number of new adopters against the number of adopters.
Add the regression line to your scatterplot.
- (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).

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,

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

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} :

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

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:

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

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 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 18) 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. We might, for instance, prefer smaller coefficient vectors (all else being equal), or ones where more of the coefficients were exactly zero. Using some quality other than the squared error to pick out a unique solution is called “regularizing” the optimization problem, and a lot of attention has been given to regularized regression, especially in the “high dimensional” setting where the number of coefficients is comparable to, or even greater than, the number of data points. See Appendix E.3.5, and exercise 2 in Chapter 8.

[[Add chapter on ridge and lasso?]]

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 1). 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 comparatively 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 predictor 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 about why $\hat{\beta}$ fluctuates. For the moment, let's fix \mathbf{X} at a particular value \mathbf{x} , but allow \mathbf{Y} to vary randomly (what's called "fixed design" regression).

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,

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

Here ϵ is the noise around the optimal linear predictor; we 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:

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

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

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

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

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

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

Let's write $Var[\epsilon | \mathbf{X} = \mathbf{x}]$ 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

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

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 sample variance themselves, and are more nearly uncorrelated with each other.

If we allow \mathbf{X} to vary, then by the law of total variance,

$$\text{Var}[\hat{\beta}] = \mathbb{E}[\text{Var}[\hat{\beta}|X]] + \text{Var}[\mathbb{E}[\hat{\beta}|X]] = \frac{\sigma^2}{n} \mathbb{E}\left[\left(\frac{1}{n}\mathbf{X}^T\mathbf{X}\right)^{-1}\right] \quad (2.29)$$

As $n \rightarrow \infty$, the sample variance matrix $n^{-1}\mathbf{X}^T\mathbf{X} \rightarrow \mathbf{v}$, and matrix inversion is continuous, so for large n , $\text{Var}[\hat{\beta}] \rightarrow n^{-1}\sigma^2\mathbf{v}^{-1}$, and points (i)–(iii) still hold.

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 variables, 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 $Y = \alpha X + \epsilon$, and we happen to know α exactly. The variance of Y will be $\alpha^2\text{Var}[X] + \text{Var}[\epsilon]$. The amount of variance our regression “explains” — really, the variance of our predictions — will be $\alpha^2\text{Var}[X]$. So $R^2 = \frac{\alpha^2\text{Var}[X]}{\alpha^2\text{Var}[X] + \text{Var}[\epsilon]}$. 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 predictor 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. SHIFTING DISTRIBUTIONS, OMITTED VARIABLES, AND
TRANSFORMATIONS

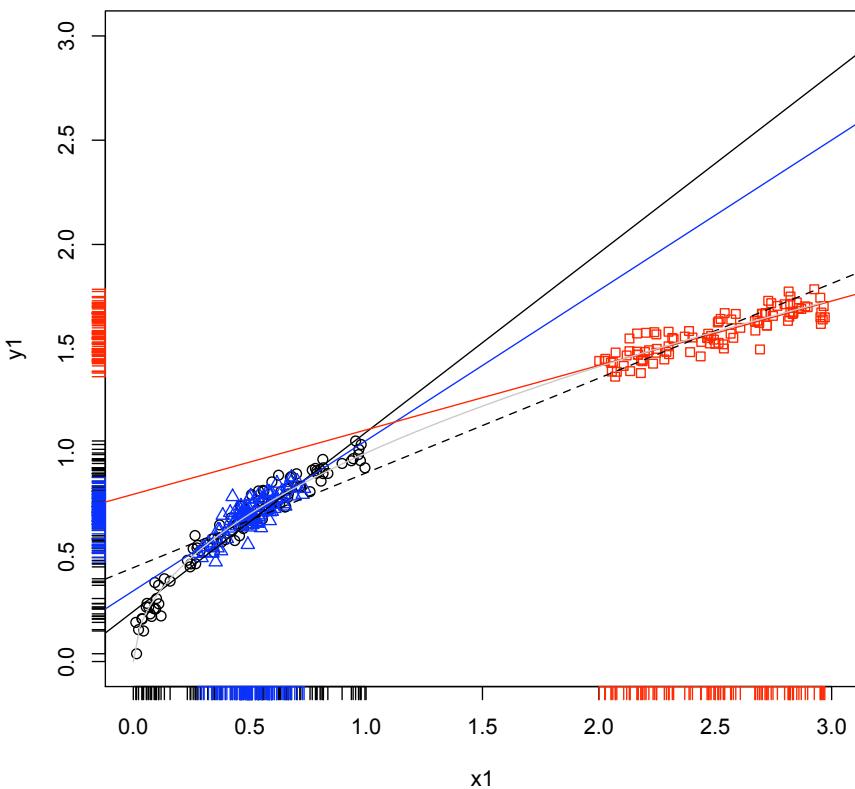


FIGURE 2.1: Behavior of the conditional distribution $Y|X \sim \mathcal{N}(\sqrt{X}, 0.05^2)$ with different distributions of X . Black circles: X is uniformly distributed in the unit interval. Blue triangles: Gaussian with mean 0.5 and standard deviation 0.1. Red squares: uniform between 2 and 3. Axis 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.)

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 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 \vec{X} given Z 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:

$$\vec{X} = \vec{U} + \vec{\eta} \quad (2.31)$$

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

2.2. SHIFTING DISTRIBUTIONS, OMITTED VARIABLES, AND TRANSFORMATIONS

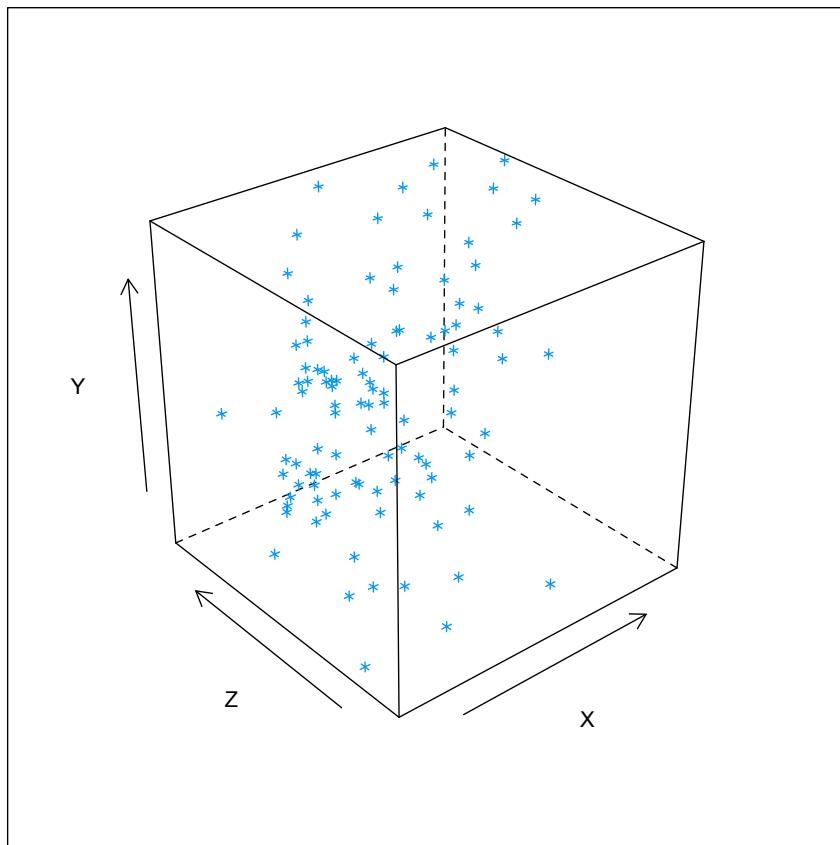


FIGURE 2.2: Scatter-plot of response variable Y (vertical axis) and two variables which influence it (horizontal axes): X , which is included in the regression, and Z , which is omitted. X and Z have a correlation of +0.1. (Figure created using the `cloud` command in the package `lattice`; see accompanying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

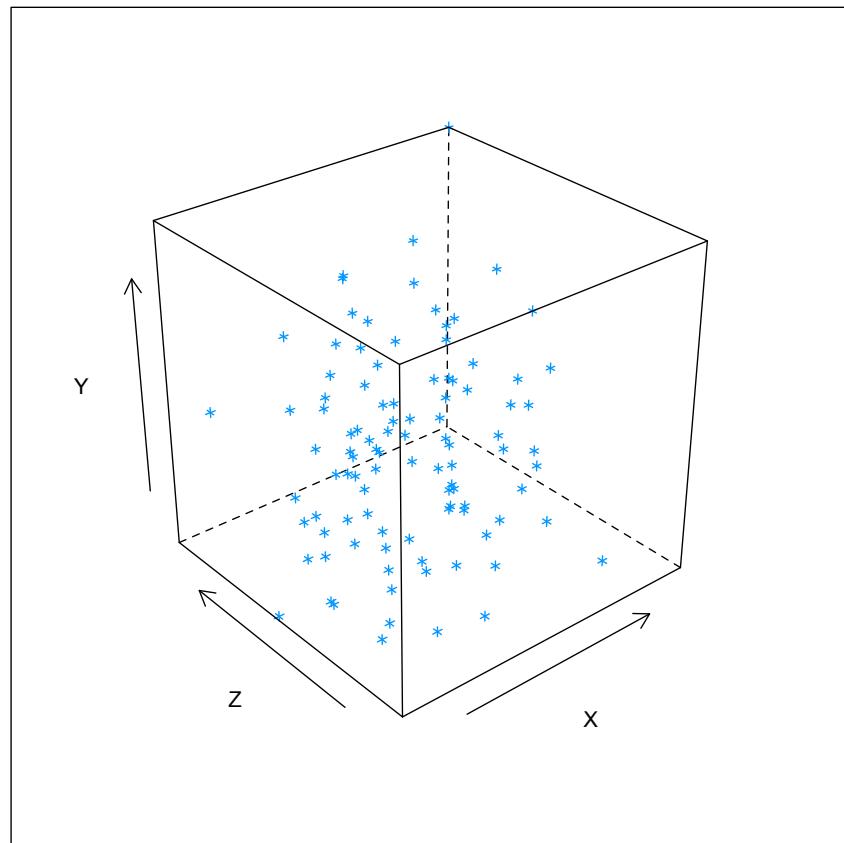


FIGURE 2.3: As in Figure 2.2, but shifting so that the correlation between X and Z is now -0.1 , though the marginal distributions, and the distribution of Y given X and Z , are unchanged. (See accompanying R file for commands used to make this figure.)

2.2. SHIFTING DISTRIBUTIONS, OMITTED VARIABLES, AND TRANSFORMATIONS

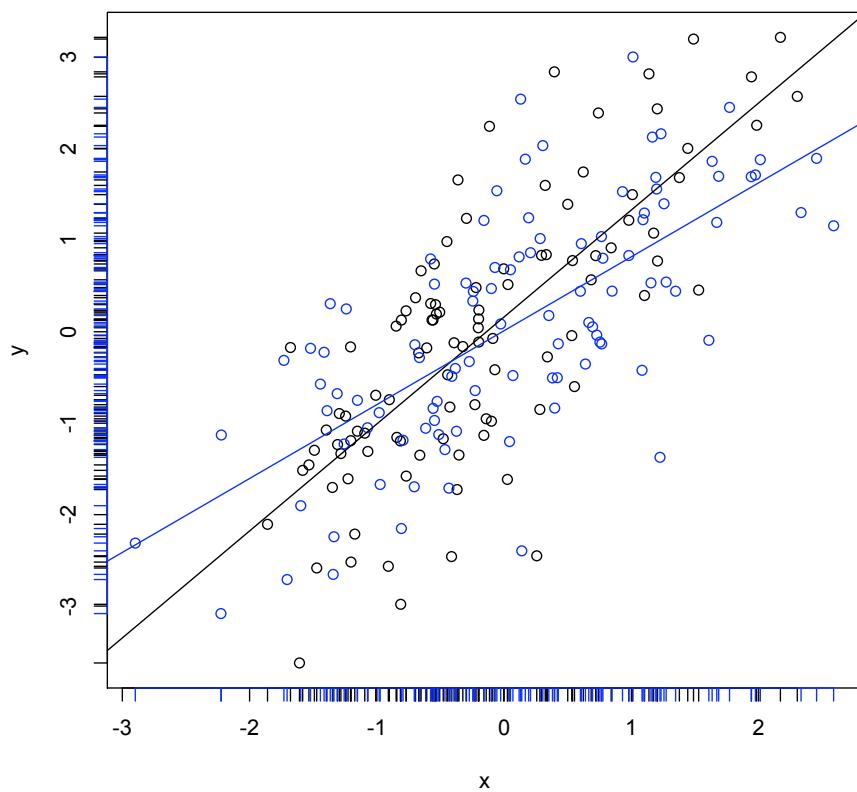


FIGURE 2.4: Joint distribution of X and Y from Figure 2.2 (black, with a positive correlation between X and Z) and from Figure 2.3 (blue, with a negative correlation between X and Z). Tick-marks on the axes show the marginal distributions, which are manifestly little-changed. (See accompanying R file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

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 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 in which errors-in-variables isn't 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 in Ch. 1 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.

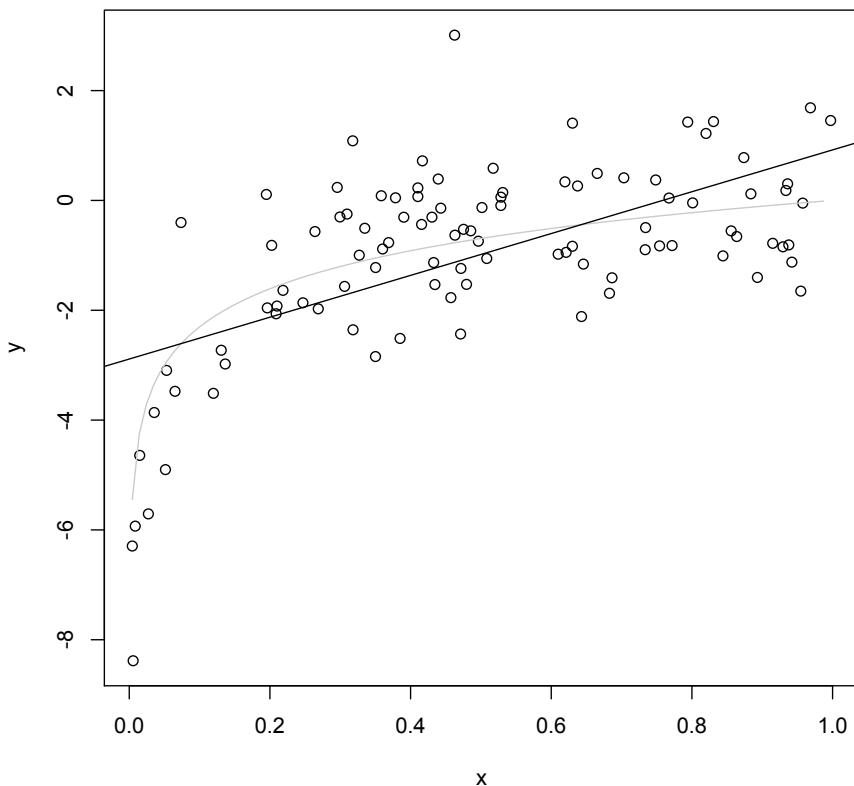
We have two choices: we can transform the response Y , or the predictor X . Here transforming the response would mean regressing $\exp Y$ on X , and transforming the predictor would mean regressing Y on $\log X$. Both kinds of transformations can be worth trying, but transforming the predictors is, in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

1. Mathematically, $E[f(Y)] \neq f(E[Y])$. A mean-squared optimal prediction of

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.)

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

2.2. SHIFTING DISTRIBUTIONS, OMITTED VARIABLES, AND
TRANSFORMATIONS



```

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.

$f(Y)$ 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.
4. Suppose that we are in luck and $Y = r(X) + \epsilon$, with ϵ independent of X , and Gaussian, so all the usual default calculations about statistical inference apply. Then it will generally *not* be the case that $f(Y) = s(X) + \eta$, with η a Gaussian random variable independent of X . In other words, transforming Y completely messes up the noise model. (Consider the simple case where we take the logarithm of Y . Gaussian noise after the transformation implies log-normal noise before the transformation. Conversely, Gaussian noise before the transformation implies a very weird, nameless noise distribution after the transformation.)

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 predictor 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 predictor 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). It is also possible to search for transformations of *both* sides of a regression model; see Breiman and Friedman (1985) and, for an R implementation, Spector *et al.* (2013).

2.2. SHIFTING DISTRIBUTIONS, OMITTED VARIABLES, AND TRANSFORMATIONS

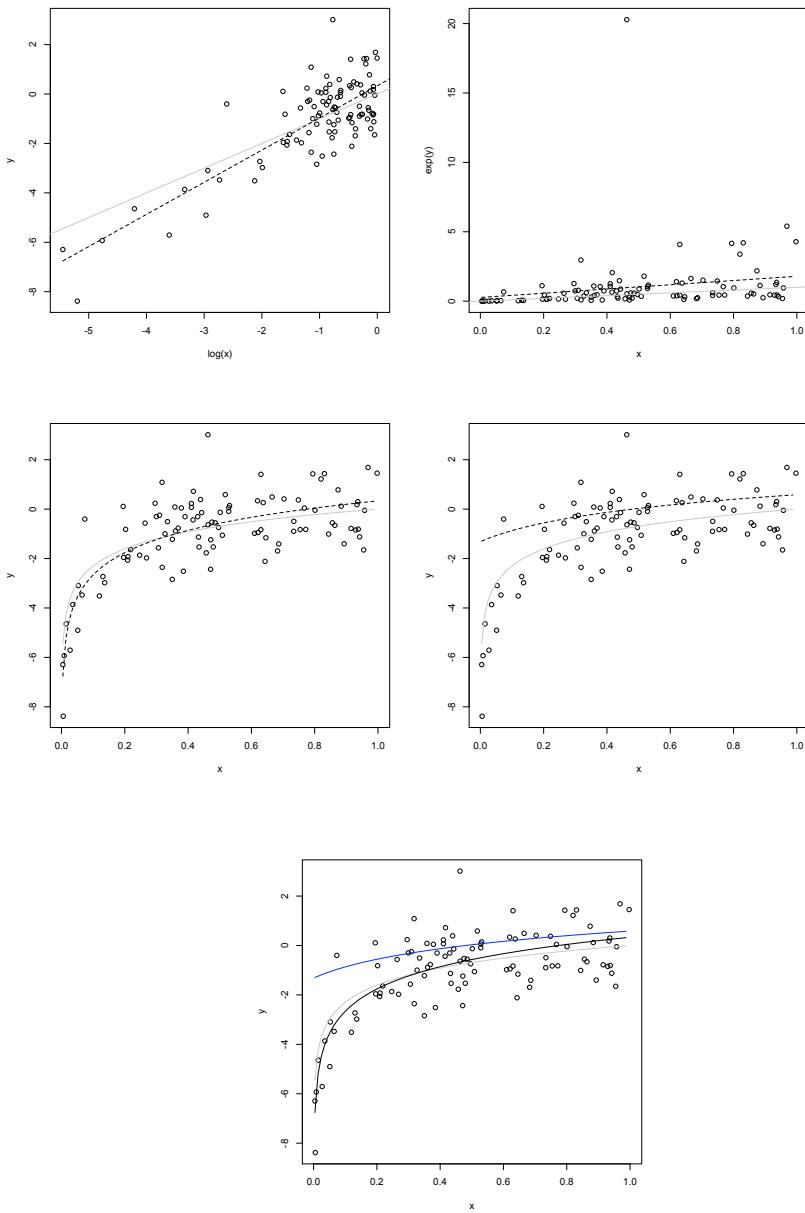


FIGURE 2.6: Transforming the predictor (left column) and the response (right) in the data from Figure 2.5, shown in both the transformed coordinates (top) and the original coordinates (middle). The bottom figure super-imposes the two estimated curves (transformed X in black, transformed Y in blue). The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

2.3 Adding Probabilistic Assumptions

The usual treatment of linear regression adds many more probabilistic assumptions, namely that

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

and that Y values are 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 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 constrained and unconstrained estimates to be converging. 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 predictor 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} .
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, $\vec{X} =$

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 G.

(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-1} , and nothing else”. 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-1} , but nothing else”. 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, and only those other variables — whether it helps prediction “at the margin”. It does not and cannot test 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 in Chapter 3, 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 multiple variables to the model could significantly improve the fit (as checked by, say, a partial F test), even if *none* of the coefficients is significant on its own. 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 $\text{Var}[\hat{\beta} | \mathbf{X} = \mathbf{x}] = \frac{\sigma^2}{n} (\mathbf{n}^{-1} \mathbf{x}^T \mathbf{x})^{-1} \rightarrow n^{-1} \sigma^2 \mathbf{v}^{-1}$. This means that the standard errors will 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 actually *influences* the response a lot.

To sum up, it is *never* the case that statistical significance is the same as scientific, real-world significance. The most important variables are *not* those with the largest-magnitude t statistics or smallest p -values. Statistical significance is always about what “signals” can be picked out clearly from background noise⁶. 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.

⁶In retrospect, it might have been clearer to say “statistically *detectable*” rather than “statistically *significant*”.

Of course, even the limited “does it help linear 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 significant regression coefficient, it must influence the response;
2. because a variable has an insignificant regression coefficient, it must not influence the response;
3. if the input 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 polishing 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 polishing 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 were the claims 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 predictor 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 difference in expected responses (conditional on the given values for the other predictors), assuming that the form of the regression model is right, *and* 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 predictor variables. A **causal** or **counter-factual** 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 inference⁷; 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 extremest agony into the everlasting fire.

2.5 Further Reading

There are many excellent textbooks on linear regression. Among them, I would mention Weisberg (1985) for general statistical good sense, Hastie *et al.* (2009) for emphasizing connections to more advanced methods, and Faraway (2004) for R practicalities. Berk (2004) omits the details those books cover, but is superb on the big picture, and especially on what must be assumed in order to do certain things with linear regression and what cannot be done under any assumption.

2.6 Exercises

1. Show that the expected error of the optimal linear predictor, $E[Y\vec{X} \cdot \beta]$, is zero.
2. 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 predictor variable.

⁷In particular, if our model was estimated from data where Someone *assigned* values of the predictor 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.

3. Run the code from Figure 2.5. Then replicate the plots in Figure 2.6.
4. 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

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

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,

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

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 $\hat{\theta}$ 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:

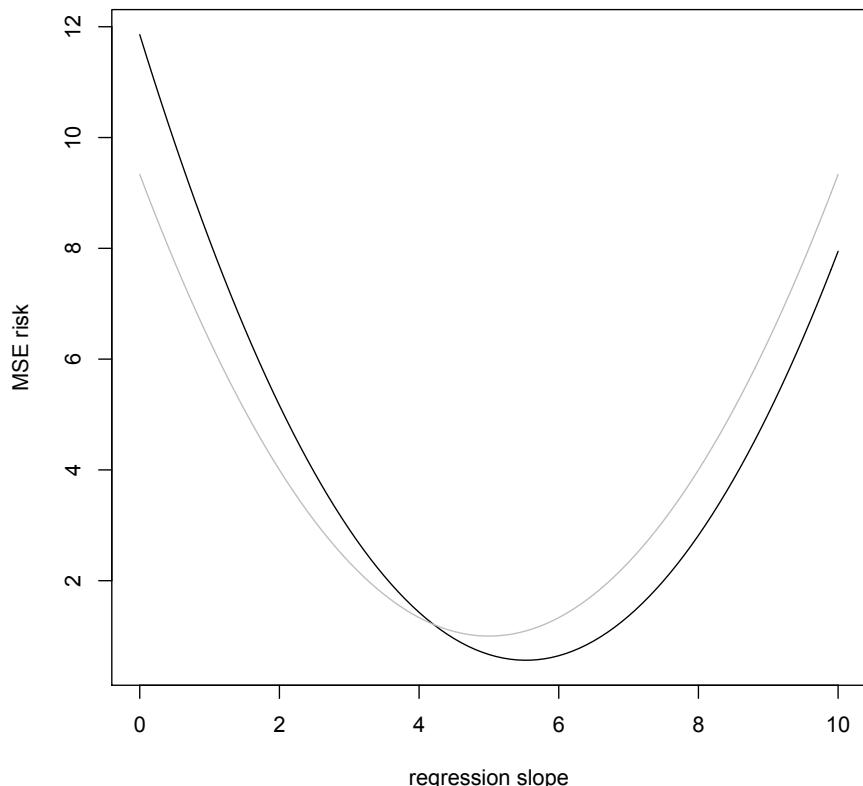
$$\bar{L}(\mathbf{z}_n, \theta) = \mathbb{E}[L(Z, \theta)] + \eta_n(\theta) \quad (3.6)$$

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):

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

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 $\hat{\theta}_n$ 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.emp-b*x.emp)^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 $\theta = 5$, and $X \sim \text{Unif}(0, 1)$. 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 a subtle and involved one. (See §3.6 for references.) 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.

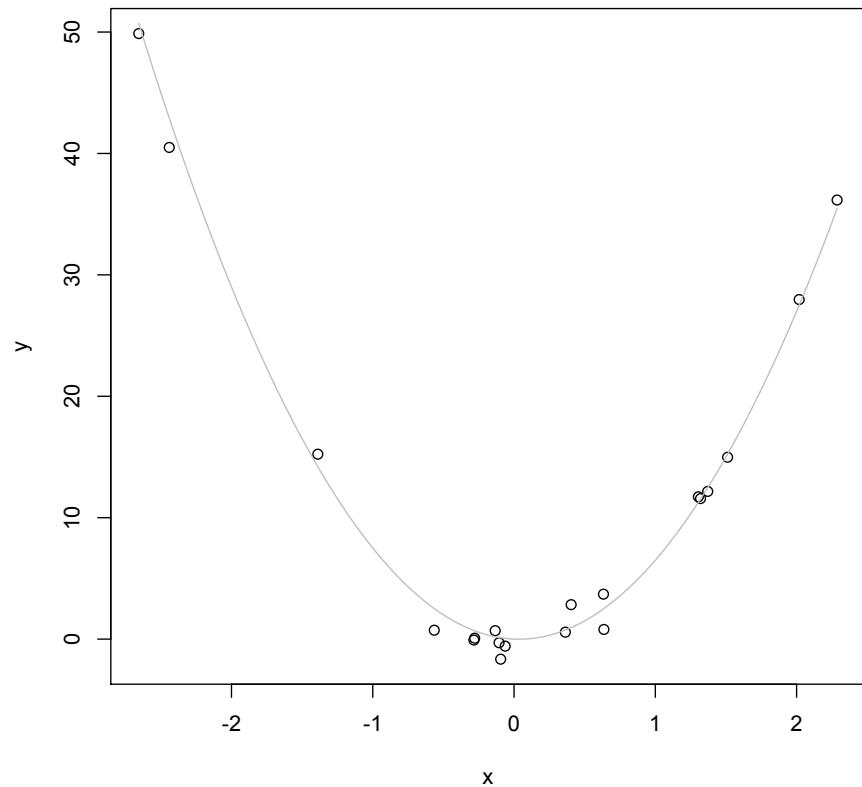
Notice that the in-sample error goes down as the order of the polynomial increases; it has to. Every polynomial of order p is also a polynomial of order $p+1$, so going to a higher-order model can only reduce the in-sample error. Quite generally, in fact, as one uses more and more complex and flexible models, the in-sample error will get smaller and smaller.⁵

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 error⁶. The results are in Figure 3.5.

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,

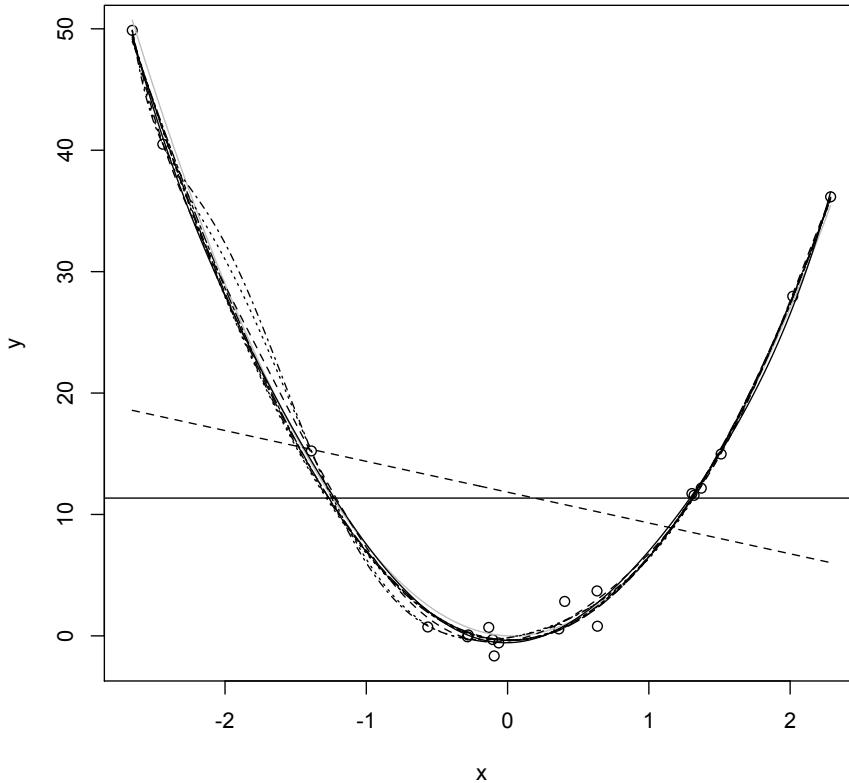
⁵In fact, since there are only 20 data points, they could all be fit exactly if the order of the polynomials went up to 19. (Remember that any two points define a line, any three points a parabola, etc. — $p+1$ points define a polynomial of order p which passes through them.)

⁶This 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).

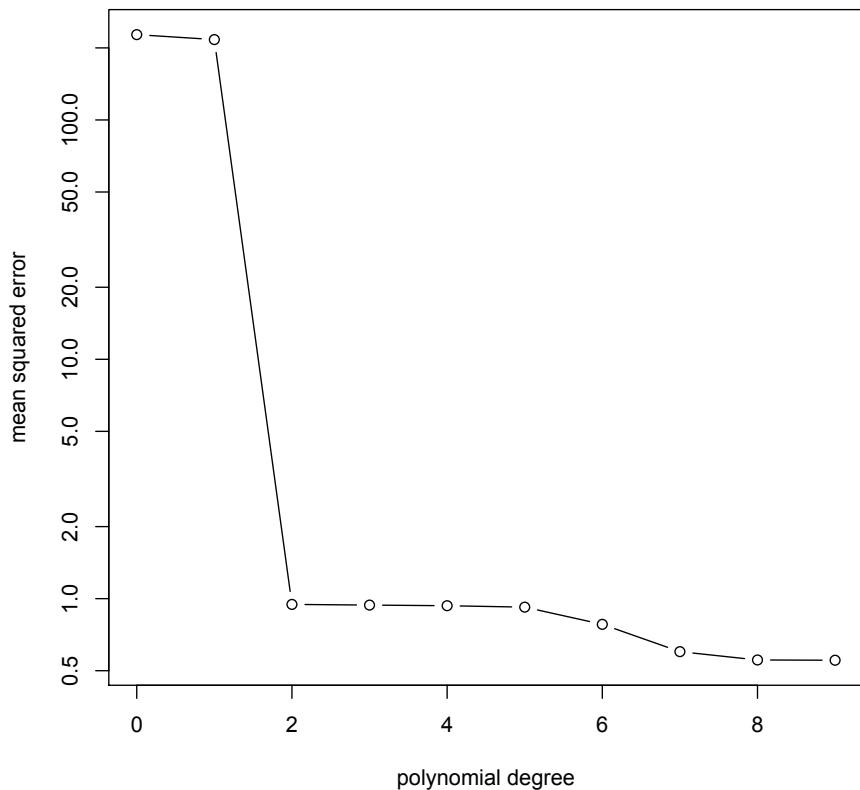


```

poly.formulae <- c("y~1", paste("y ~ poly(x,", 1:9, ")", sep=""))
poly.formulae <- sapply(poly.formulae, as.formula)
df.plot <- data.frame(x=seq(min(x),max(x),length.out=200))
fitted.models <- list(length=length(poly.formulae))
for (model_index in 1:length(poly.formulae)) {
  fm <- lm(formula=poly.formulae[[model_index]])
  lines(df.plot$x, predict(fm,newdata=df.plot),lty=model_index)
  fitted.models[[model_index]] <- fm
}

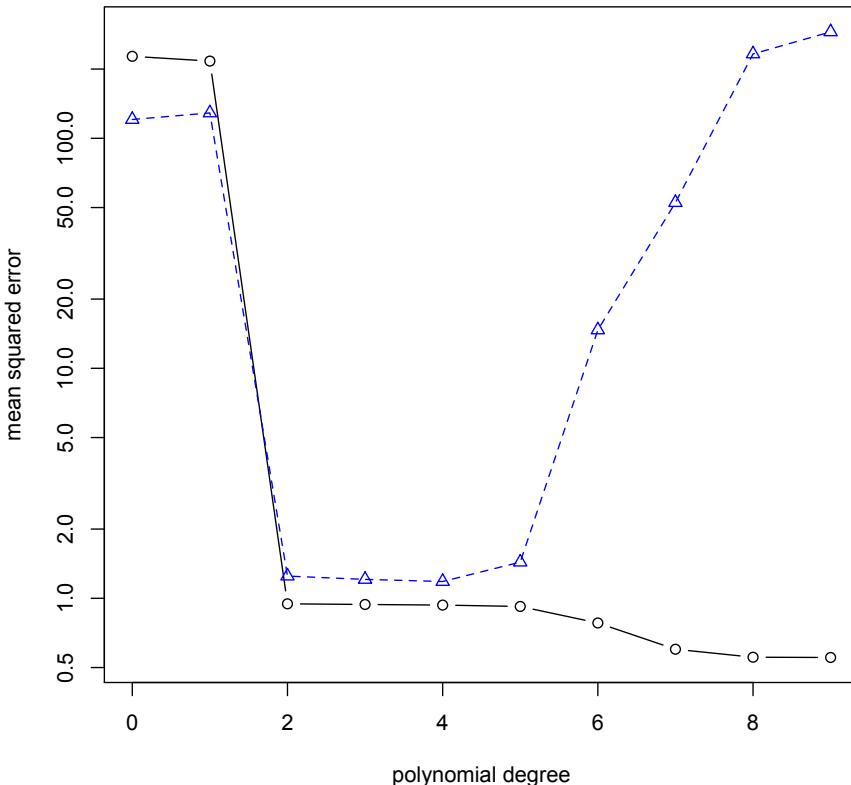
```

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 `?poly`.) This builds a vector of model formulae and then fits each one in turn, storing the fitted models in a new list.



```
mse.q <- sapply(fitted.models, function(mdl) { mean(residuals(mdl)^2) })
plot(0:9,mse.q,type="b",xlab="polynomial degree",ylab="mean squared error",
log="y")
```

FIGURE 3.4: Empirical MSE vs. degree of polynomial for the data from the previous figure. Note the logarithmic scale for the vertical axis.



```

x.new = rnorm(2e4); y.new = 7*x.new^2 - 0.5*x.new + rnorm(2e4)
gmse <- function(mdl) { mean((y.new - predict(mdl, data.frame(x=x.new))^2) }
gmse.q <- sapply(fitted.models, gmse)
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.*

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.

In this example, 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 $\widehat{\theta}_n$. The loss of this on the validation data is

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

where now the sampling noise on the *validation* set, η'_m , is independent of $\widehat{\theta}_n$. So this gives us an unbiased estimate of the generalization error, and, if m is large, a precise

```

A_vs_B <- sample(rep(c("A","B"),length.out=nrow(housing))
half_A <- which(A_vs_B=="A"); half_B <- which(A_vs_B=="B")
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,rows) {
  test <- housing[rows,]
  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.)*

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 §32, and Code Example 1 shows the code used to make Figure 3.6.

	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	
...	
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	
...	
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	
8	439300	59091	4.4	
...	
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	$(3.99 \pm .55) \times 10^4$	4.99 ± 0.8	NA	MSE
Income + Rooms	$(5.040 \pm 0.089) \times 10^5$	7.609 ± 0.079	$(-1.162 \pm 0.020) \times 10^5$	2.59×10^{10}
				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 §32. This is then randomly split into two equally sized parts (tables in the next 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?

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 often 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 same as data-set splitting if $k = 2$.) This is the cross-validated estimate of generalization error for each model. Model selection then picks the model with the smallest estimated risk.⁷ Code Example 2 performs k -fold cross-validation for linear models specified by formulae.

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 testing data is, with high probability, fairly representative of the data-generating process. *Randomly* dividing the data into training and test sets makes 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.⁸

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

⁷A 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.

⁸The 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.

```

cv.lm <- function(data, formulae, nfolds=5) {
  data <- na.omit(data)
  formulae <- sapply(formulae, as.formula)
  response.name <- function(formula) { all.vars(formula)[1] }
  responses <- sapply(formulae, response.name)
  names(responses) <- as.character(formulae)
  fold.labels <- sample(rep(1:nfolds, length.out=nrow(data)))
  mses <- matrix(NA, nrow=nfolds, ncol=length(formulae))
  colnames <- as.character(formulae)
  for (fold in 1:nfolds) {
    test.rows <- which(fold.labels == fold)
    train <- data[-test.rows,]
    test <- data[test.rows,]
    for (form in 1:length(formulae)) {
      current.model <- lm(formula=formulae[[form]], data=train)
      predictions <- predict(current.model, newdata=test)
      test.responses <- test[,responses[form]]
      mses[fold, form] <- mean((test.responses - predictions)^2)
    }
  }
  return(colMeans(mses))
}

```

CODE EXAMPLE 2: Function to do k -fold cross-validation on linear models, given as a vector (or list) of model formulae. Note that this only returns the CV MSE, not the parameter estimates on each fold. See online for comments.

prediction on the left-out point would be by doing calculations on a model fit to the *whole* data (see below). 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.⁹

[[TODO: Appendix on AIC?]]

A Short-cut for Linear Smoothers Suppose the model m is any linear smoother (§1.5). For each of the data points i , then, the predicted value is a linear combination of the observed values of y , $m(x_i) = \sum_j \hat{w}(x_i, x_j)y_j$ (Eq. 1.47). Define the matrix \mathbf{w} by $\hat{w}_{ij} = \hat{w}(x_i, x_j)$. (Because predicted values are often written \hat{y} , this matrix is often called the “hat matrix”.) What happens when we hold back data point i , and then make a prediction at x_i ? Well, observed response at i can't contribute to the prediction, but otherwise the linear smoother should work as before, so

$$m^{(-i)}(x_i) = \frac{(\mathbf{w}\mathbf{y})_i - \hat{w}_{ii}y_i}{1 - \hat{w}_{ii}} \quad (3.9)$$

The numerator just removes the contribution to $m(x_i)$ that came from y_i , and the denominator just re-normalizes the weights in the smoother. Now a little algebra says that

$$y_i - m^{(-i)}(x_i) = \frac{y_i - m(x_i)}{1 - \hat{w}_{ii}} \quad (3.10)$$

The quantity on the left of that equation is what we want to square and average to get the leave-one-out CV score, but everything on the right can be calculated from the fit we did to the whole data. The leave-one-out CV score is thus

$$\frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - m(x_i)}{1 - \hat{w}_{ii}} \right)^2 \quad (3.11)$$

Thus, if we restrict ourselves to leave-one-out *and* to linear smoothers, we can calculate the CV score with just one estimation on the whole data, rather than n re-estimates.

⁹At 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.

An even faster approximation than this is what's called "generalized" cross validation, which is simply the in-sample MSE divided by $(1 - n^{-1} \text{tr} \hat{\mathbf{W}})^2$. That is, rather than dividing each term in Eq. 3.11 by a unique factor that depends on its own diagonal entry in the hat matrix, we use the average of all the diagonal entries $(n^{-1} \text{tr} \hat{\mathbf{W}})$. In addition to speed, this tends to reduce the influence of points with high values of \hat{w}_{ii} ; this may or may not be desirable.

3.5 Warnings

Some caveats are in order.

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 form¹⁰

$$Y_t = \beta_0 + \beta_1 X_t + \beta_2 Z_t + \beta_3 U_t + \epsilon_t \quad (3.12)$$

where Y_t is “the number of violent conflicts initiated among or within continental European countries at time t ”¹¹, X_t is “the number of conflicts in which the Ottoman 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 before¹². The point of this paper is to support this rigorously, and make it precise. That support and precision requires Eq. 3.12 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. 3.12 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. 3.12, Prof. Iyigun looks at a range of slightly different linear-model specifications (e.g., regress the number of intra-Christian conflicts in year t on the number of Ottoman attacks in year $t - 1$), 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*.

¹⁰His Eq. 1 on pp. 1473; I have modified the notation to match mine.

¹¹In one part of the paper; he uses other dependent variables elsewhere.

¹²See §1–2 of Iyigun (2008), and MacCulloch (2004, *passim*).

3.6 Further Reading

Some comparatively easy starting points on statistical learning theory 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) and by Anthony and Bartlett (1999) (the latter is much more general than its title suggests), or read the book by Vapnik (2000) (one of the founders). Hastie *et al.* (2009), while invaluable, is much more oriented towards models and practical methods than towards learning *theory*.

Cross-validation goes back in statistical practice for many decades, though often as a very informal tool. One of the first important papers on the subject was Stone (1974), goes over the earlier history. The tricks for leave-one-out CV in linear smoothers appear to be due to Wahba and collaborators in the 1970s (Wahba, 1990). Arlot and Celisse (2010) is a good recent review.

On model selection in general, the best recent summary is the book by Claeskens and Hjort (2008); it is more theoretically demanding than this book, but includes many real-data examples.

White (1994) is a thorough treatment of parameter estimation in models which may be mis-specified, and some general tests for mis-specification. It also briefly discusses the interpretation of parameters in mis-specified models. That last, very important topic deserves a more in-depth treatment, but I don't know of a good one.

3.7 Exercises

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.

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:

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

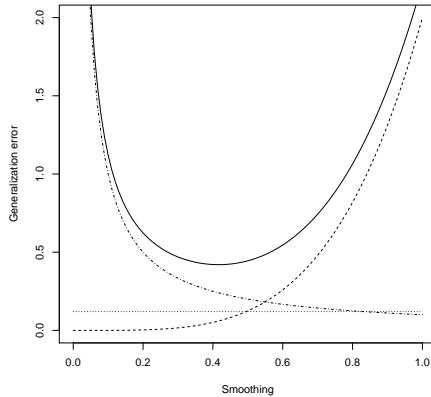
Most smoothing methods have a control setting, here written h , that says *how much* to smooth. With k nearest neighbors, for instance, the weights are $1/k$ if x_i is one of the k -nearest points to x , and $w = 0$ otherwise, so large k means that each prediction is an average over many training points.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

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

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



```
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)
```

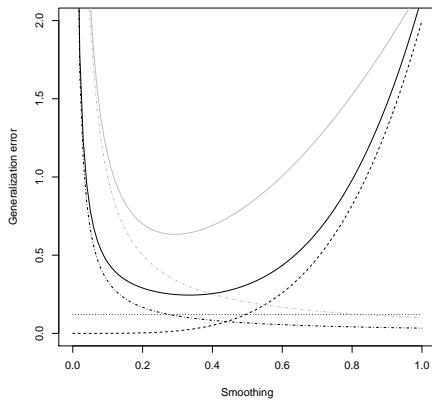
FIGURE 4.1: *Decomposition of the generalization error of smoothing: the total error (solid) equals process noise (dotted) plus approximation error from smoothing (=squared bias, dashed) and estimation variance (dot-and-dash). The numerical values here are arbitrary, but the functional forms ($\text{squared bias} \propto h^4$, $\text{variance} \propto n^{-1}h^{-1}$) are representative of kernel regression (Eq. 4.12).*

(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.

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



```

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).

4.2 Adapting to Unknown Roughness

Figure 4.3, which graphs two functions, f and g . Both are “smooth” functions in the mathematical sense². We could Taylor-expand both functions to approximate their values anywhere, just from knowing enough derivatives at one point x_0 .³ 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 accuracy in the interpolation $f(\cdot)$ needs more, and more closely spaced, training points x_i than does $g(\cdot)$.

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 constant-variance, IID Gaussian noises, say with $\sigma = 0.15$.) The data now look something like Figure 4.4. Can we recover the curves?

As remarked in Chapter 1, if we had many measurements at the same x , then we could find the expectation value by averaging: the regression function $r(x) = E[Y|X = x]$, so with multiple observations at x_i , the mean 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.

Smoothing methods try to use 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 independent terms together also reduces variance. If smoothing gets rid of more variance than it adds 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 C.1), so

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

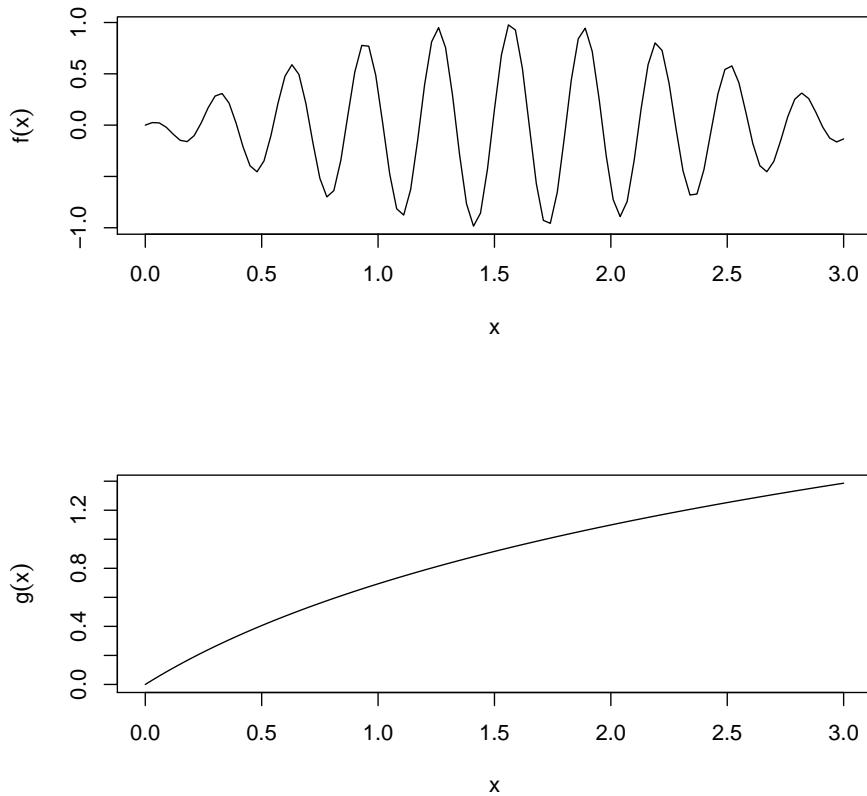
and

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

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

²They are “ C^∞ ”: not just continuous, but with continuous derivatives to all orders.

³See App. C for a refresher on Taylor expansions and Taylor series.

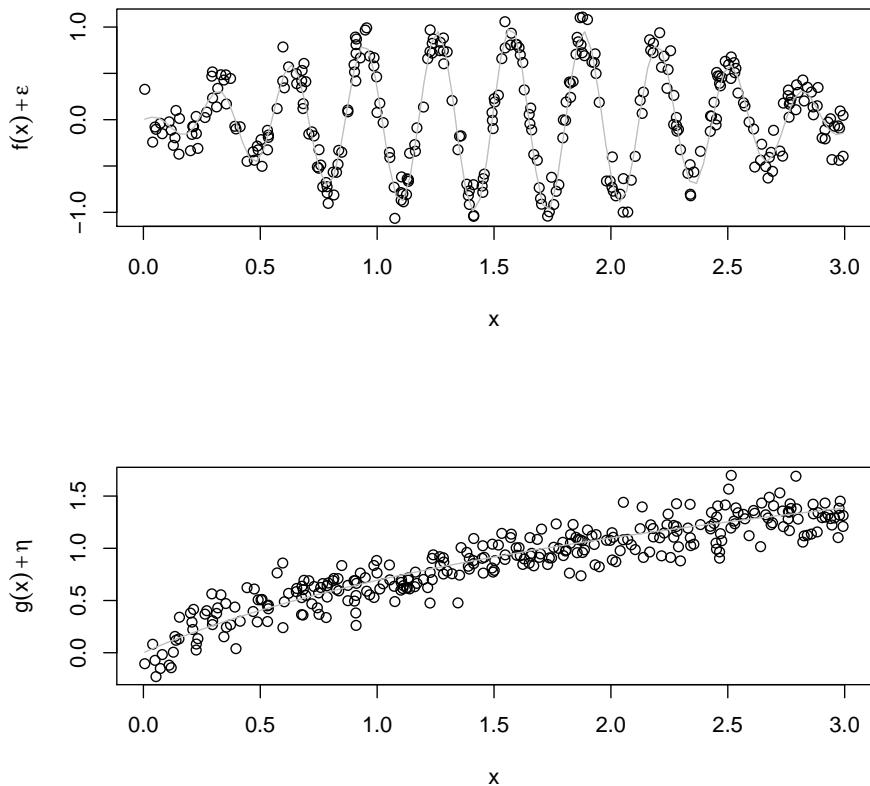


```

par(mfcol=c(2,1))
true.f <- function(x) {sin(x)*cos(20*x)}
true.g <- function(x) {log(x+1)}
curve(true.f(x),from=0,to=3,xlab="x",ylab=expression(f(x)))
curve(true.g(x),from=0,to=3,xlab="x",ylab=expression(g(x)))
par(mfcol=c(1,1))

```

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 = true.f(x)+rnorm(length(x),0,0.15)
yg = true.g(x)+rnorm(length(x),0,0.15)
par(mfcol=c(2,1))
plot(x,yf,xlab="x",ylab=expression(f(x)+epsilon))
curve(true.f(x),col="grey",add=TRUE)
plot(x,yg,xlab="x",ylab=expression(g(x)+eta))
curve(true.g(x),col="grey",add=TRUE)

```

FIGURE 4.4: The curves of Fig. 4.3 (in grey), plus IID Gaussian noise with mean 0 and standard deviation 0.15. The two curves are sampled at the same x values, but with different noise realizations.

just w_i .

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

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

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

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

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

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

The first term on the final right-hand side is an estimation variance, which will tend to shrink as n grows. (If we just do a simple global mean, $w_i = 1/n$ for all i , so we get σ^2/n , just like in baby stats.) The second term, an 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., its growth varies with *how* smooth or wiggly the regression function is. For smoothing 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 $g(x_i) + \eta_i$ for $1.5 < x_i < 1.7$ with equal weights. For f , this gives 0.3, while $f(1.6) = 0.83$. For g , this gives 0.96, with $g(1.6) = 0.96$. (See figure 4.5.) The same window size creates 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.

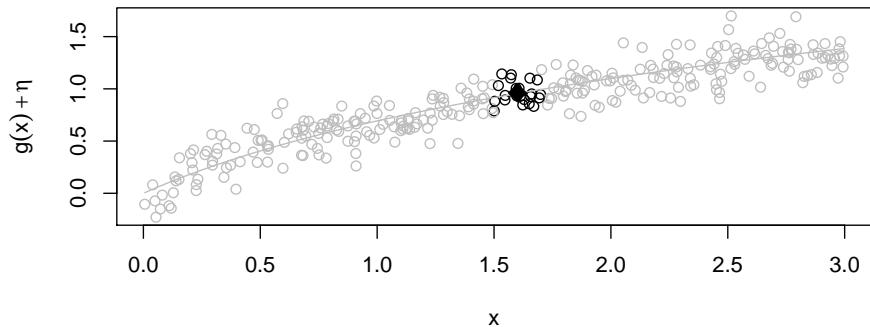
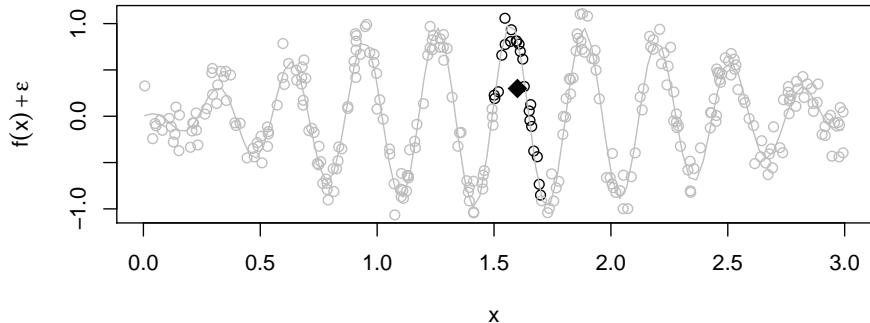
If one does a more careful second-order Taylor expansion like that leading to Eq. 4.9, specifically for kernel regression, one can show that the bias at x is

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

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

⁴The higher derivatives of r also matter, since we should really keep 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.

⁵If you are not familiar with the “order” symbols O and o , see Appendix B.



```

par(mfcol=c(2,1))
x.focus <- 1.6; x.lo <- x.focus-0.1; x.hi <- x.focus+0.1
colors;ifelse((x<x.hi)&(x>x.lo),"black","grey")
plot(x,yf,xlab="x",ylab=expression(f(x)+epsilon),col=colors)
curve(true.f(x),col="grey",add=TRUE)
points(x.focus,mean(yf[(x<x.hi)&(x>x.lo)]),pch=18,cex=2)
plot(x,yg,xlab="x",ylab=expression(g(x)+eta),col=colors)
curve(true.g(x),col="grey",add=TRUE)
points(x.focus,mean(yg[(x<x.hi)&(x>x.lo)]),pch=18,cex=2)
par(mfcol=c(1,1))

```

FIGURE 4.5: Relationship between smoothing and function roughness. In both panels we estimate the value of the regression function at $x = 1.6$ by averaging observations where $1.5 < x_i < 1.7$ (black points, others are “ghosted” in grey). The location of the average is shown by the large black diamond. This 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).

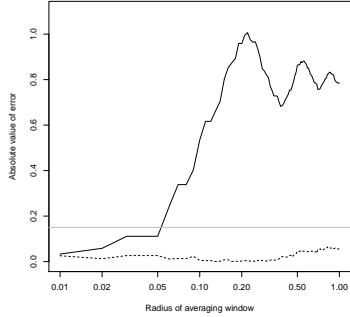


FIGURE 4.6: Error of estimating $f(1.6)$ (solid line) and $g(1.6)$ (dashed) from averaging observed values at $1.6 - h < x < 1.6 + h$, for different radii h . The grey is σ , the standard deviation of the noise — how can the estimation error be smaller than that?

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, $E[w(X_i, x, h)(X_i - x)\hat{r}(x)] \approx 0$. 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 :

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

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

⁶Exercise 1 sketches the demonstration for the special case of the uniform (“boxcar”) kernel.

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 our 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 3 shows how it would work in R, with a one predictor variable, borrowing the `npreg` function from the `np` library (Hayfield and Racine, 2008).¹⁰

⁷You may be wondering why I keep talking about *the* optimal bandwidth, when Eq. 4.14 makes it seem that the bandwidth should vary with x . One can 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.

¹⁰The package has methods for automatically selecting bandwidth by cross-validation — see §4.6 below.

```

cv_bws_npreg <- function(x,y,bandwidths=(1:50)/50,nfolds=10) {
  require(np)
  n <- length(x)
  stopifnot(n > 1, length(y) == n)
  stopifnot(length(bandwidths) > 1)
  stopifnot(nfolds > 0, nfolds==trunc(nfolds))

  fold_MSEs <- matrix(0,nrow=nfolds,ncol=length(bandwidths))
  colnames(fold_MSEs) = bandwidths

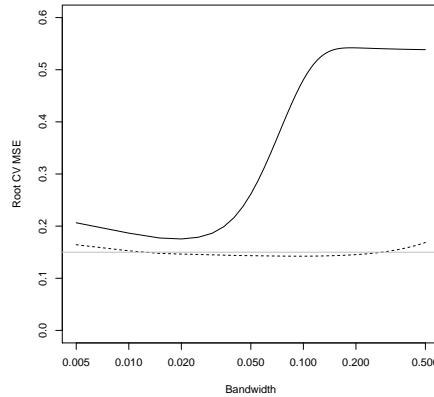
  case.folds <- sample(rep(1:nfolds,length.out=n))
  for (fold in 1:nfolds) {
    train.rows = which(case.folds!=fold)
    x.train = x[train.rows]
    y.train = y[train.rows]
    x.test = x[-train.rows]
    y.test = y[-train.rows]
    for (bw in bandwidths) {
      fit <- npreg(txdat=x.train,tydat=y.train,
                    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 3: *Cross-validation for univariate kernel regression (see online for comments).* 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 only for illustration; it should not be used blindly on data, or in homework.

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 4.7 plots the CV estimate of the (root) mean-squared error versus bandwidth for our two curves. Figure 4.8 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 (which would look like Figure 4.1), so optimizing the CV error is close to optimizing the generalization error.



```

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$CV_MSEs),xlab="Bandwidth",
     ylab="Root CV MSE",type="l",ylim=c(0,0.6),log="x")
lines(1:100/200,sqrt(gbws$CV_MSEs),lty="dashed")
abline(h=0.15,col="grey")

```

FIGURE 4.7: Cross-validated estimate of the (root) mean-squared error as a function of the bandwidth (solid curve, $f(x)$ data; dashed, $g(x)$ data; grey line, true noise 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. CV selects bandwidths of 0.020 for f and 0.055 for g .

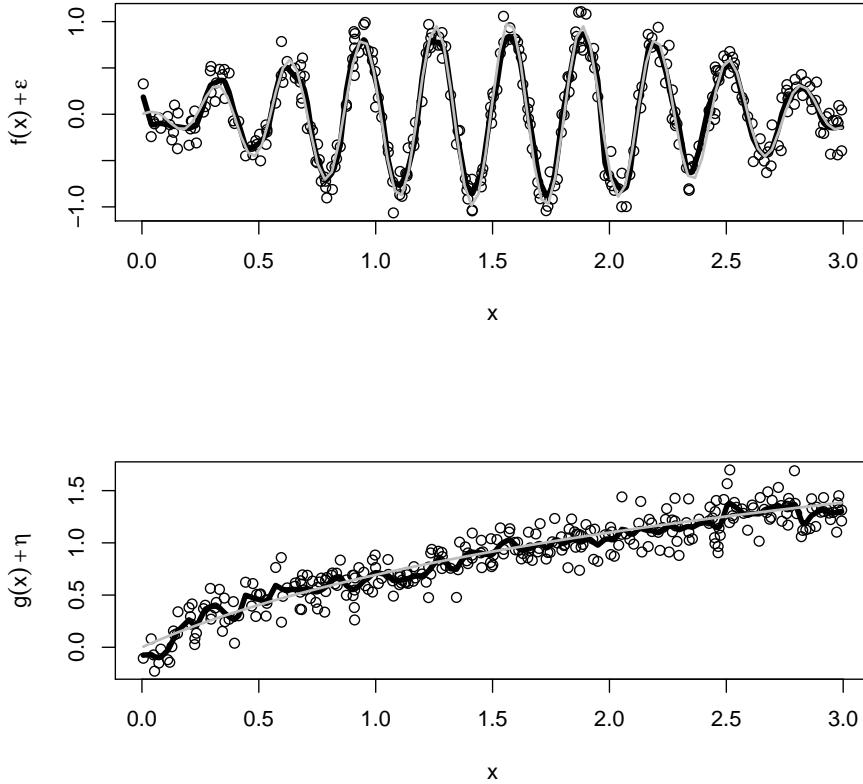
Notice, by the way, in Figure 4.7, that the rougher curve is more sensitive to the choice of bandwidth, and that the smoother curve always has a lower mean-squared error. Also notice that, at the minimum, one of the cross-validation estimates of generalization error is smaller than the true system noise 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 16, 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,

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.



```

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

```

FIGURE 4.8: Data from the running examples (circles), true regression functions (grey) and kernel estimates of regression functions with CV-selected bandwidths (black). R NOTES: The x values aren't sorted, so we need to put them in order before drawing lines connecting the fitted values; then we need to put the fitted values in the same order. Alternately, we could have used predict on the sorted values, as in §4.3.

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 proportional to the product of sample size and bandwidth. If we kept b 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 b change with n — call it b_n . Specifically, suppose $b_n \rightarrow 0$ as $n \rightarrow \infty$, but $nb_n \rightarrow \infty$. Then, by Eq. 4.16, the risk (generalization error) of kernel smoothing is approaching that of the ideal predictor.

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

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

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

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

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 . (For linear regression, θ would be the vector of slopes and the intercept.) The optimal value of the parameter, θ_0 , minimizes the mean-squared error. At θ_0 , the parametric model has MSE

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

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 $\hat{\theta} - \theta_0 = O(1/\sqrt{n})$. Because the error is minimized at θ_0 , the first derivatives of MSE at θ_0 are 0. Doing a second-order Taylor expansion of the parametric model contributes an error $O((\hat{\theta} - \theta_0)^2)$, so altogether

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

This means parametric models converge more quickly (n^{-1} goes to zero faster than $n^{-0.8}$), but they typically converge to the wrong answer ($b^2 > 0$). Kernel smoothing

¹²When the model is wrong, the optimal parameter value θ_0 is often called the **pseudo-truth**.

converges more slowly, but always converges to the right answer¹³.

This doesn't change much if we use cross-validation. Writing \widehat{h}_{CV} for the bandwidth picked by cross-validation, it turns out (Simonoff, 1996, ch. 5) that

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

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 in 1D

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.

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.

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 15.

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 $\vec{b} = (b_1, b_2, \dots, b_d)$. 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.

(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 16, we'll look at formal tests based on this idea.)

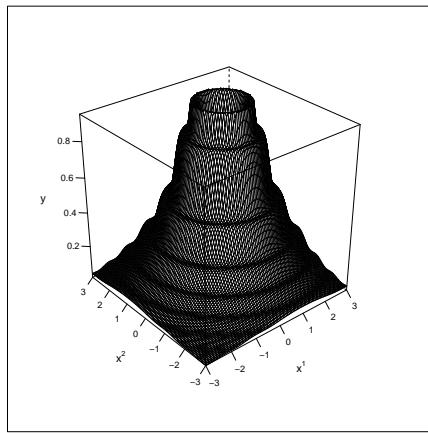
Kernel regression will recover almost any regression function. This is true even when the true regression function involves lots of interactions among the input variables, perhaps in complicated forms that would be very hard to express in linear regression. For instance, Figure 4.9 shows a contour plot 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 4.11 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.

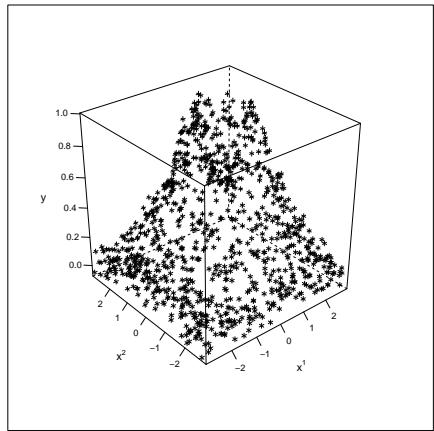


```

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")

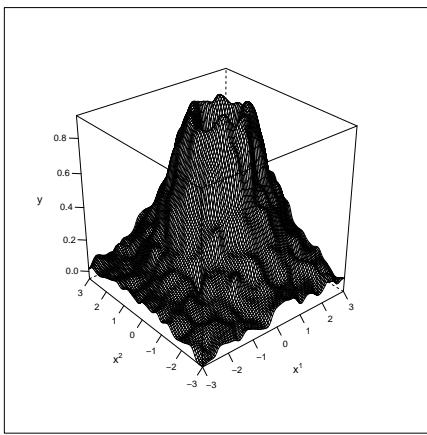
```

FIGURE 4.9: 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(y~x1*x2,data=noise,col="black",scales=list(arrows=FALSE),
      xlab=expression(x^1),ylab=expression(x^2),zlab="y")
```

FIGURE 4.10: 1000 points sampled from the surface in Figure 4.9, 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 4.11: Gaussian kernel regression of the points in Figure 4.10. Notice that the estimated function will make predictions at arbitrary points, not just the places where there was training data.

What about pictures? With only two input variables, we can make wireframe plots like Figure 4.11, or contour or level plots, which will show the predictions for different combinations of the two variables. But what if we want to look at one variable at a time, or there are more than two input variables?

A reasonable way to produce a curve for each input variable is to set all the others to some “typical” value, like their means or medians, and to then plot the predicted response as a function of the one remaining variable of interest (Figure 4.12). Of course, when there are interactions, changing the values of the other inputs will change the response to the input of interest, so it’s a good idea to produce a couple of curves, possibly super-imposed (Figure 4.12 again).

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/or how much 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:

$$Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \epsilon \quad (4.26)$$

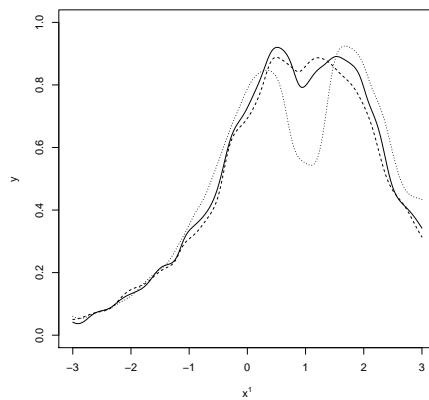
Now the effect of increasing X_1 by 1 is

$$[\beta_1(X_1 + 1) + \beta_2 X_2 + \beta_3(X_1 + 1)X_2] - [\beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2] = \beta_1 + \beta_3 X_2 \quad (4.27)$$

The right answer to “how much does the response change when X_1 is increased by one unit?” depends on the value of X_2 ; it’s certainly not just “ β_1 ”.

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

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



```

new.frame <- data.frame(x1=seq(-3,3,length.out=300),x2=median(y.noise))  

plot(new.frame$x1,predict(noise,np,newdata=new.frame),  

     type="l",xlab=expression(x^1),ylab="y",ylim=c(0,1.0))  

new.frame$x2 <- quantile(y.noise,0.25)  

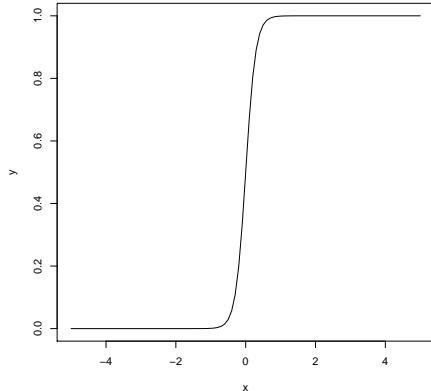
lines(new.frame$x1,predict(noise,np,newdata=new.frame),lty=2)  

new.frame$x2 <- quantile(y.noise,0.75)  

lines(new.frame$x1,predict(noise,np,newdata=new.frame),lty=3)

```

FIGURE 4.12: 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 can make predictions at arbitrary coordinates, whether or not there were any training points there.



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

FIGURE 4.13: *The function of Eq. 4.28, with $\beta = 7$.*

which looks like Figure 4.13, 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 superscripts to a minimum, we'll write $\vec{X} = (\boldsymbol{u}, \vec{V})$, where \boldsymbol{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 \boldsymbol{u} ,

$$\mathbb{E}[Y|\vec{X} = (\boldsymbol{u}^{(2)}, \vec{v})] - \mathbb{E}[Y|\vec{X} = (\boldsymbol{u}^{(1)}, \vec{v})] \quad (4.29)$$

and the change in the response per unit change in \boldsymbol{u} ,

$$\frac{\mathbb{E}[Y|\vec{X} = (\boldsymbol{u}^{(2)}, \vec{v})] - \mathbb{E}[Y|\vec{X} = (\boldsymbol{u}^{(1)}, \vec{v})]}{\boldsymbol{u}^{(2)} - \boldsymbol{u}^{(1)}} \quad (4.30)$$

Both of these, but especially the latter, are called the **predictive comparison**. Note that both of them, as written, depend on $\boldsymbol{u}^{(1)}$ (the starting value for the variable of interest), on $\boldsymbol{u}^{(2)}$ (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, $\boldsymbol{u}^{(1)}$, $\boldsymbol{u}^{(2)}$ and \vec{v} all go away and leave us with the regression coefficient on \boldsymbol{u} . In nonlinear or interacting models, we can't simplify so much.

Once we have estimated a regression model, we can choose our starting point, ending point and context, and just plug in to Eq. 4.29 or Eq. 4.30. (Or problem 9 in problem set 33.) 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** for u

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

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 is a reasonable summary of how rapidly we should expect the response to vary as u changes slightly. But we need to remember that once the model is nonlinear or has interactions, it's just not possible to boil down the whole predictive relationship between u and y into one number. In particular, the value of Eq. 4.31 is going to depend on the distribution of u (and possibly of v), even when the regression function is unchanged. (See Exercise 3.)

4.6 Computational Advice: npreg

The homework will call for you to do nonparametric regression with the `np` package — which we've already looked at a little. It's a powerful bit of software, but it can take a bit of getting used to. This section is not a substitute for reading Hayfield and Racine (2008), but should get you started.

We'll look at a synthetic-data example with four variables: a quantitative response Y , two quantitative predictors X and Z , and a categorical predictor U , which can be either "A" or "B". The true model is

$$Y = \epsilon + 20X^2 + \begin{cases} Z & \text{if } W = A \\ 10e^Z / (1 + e^Z) & \text{if } W = B \end{cases} \quad (4.32)$$

with $\epsilon \sim \mathcal{N}(0, 0.05)$. Code Example 4 generates some data from this model for us.

The basic function for fitting a kernel regression in `np` is `npreg` — conceptually, it's the equivalent of `lm`. Like `lm`, it takes a `formula` argument, which specifies the model, and a `data` argument, which is a data frame containing the variables included in the formula. The basic idea is to do something like this:

```
demo.np1 <- npreg(y ~ x + z, data=demo.df)
```

The variables on the right-hand side of the formula are the predictors; we use `+` to separate them. Kernel regression will automatically include interactions between

¹⁷Actually, they propose something a bit more complicated, which takes into account the uncertainty in our estimate of the regression function, via bootstrapping (Chapter 6).

```

make.demo.df <- function(n) {
  demo.func <- function(x,z,w) {
    20*x^2 + ifelse(w=="A",z,10*exp(z)/(1+exp(z)))
  }
  x <- runif(n,-1,1)
  z <- rnorm(n,0,10)
  w <- sample(c("A","B"),size=n,replace=TRUE)
  y <- demo.func(x,z,w)+rnorm(n,0,0.05)
  return(data.frame(x=x,y=y,z=z,w=w))
}
demo.df <- make.demo.df(100)

```

CODE EXAMPLE 4: *Generating data from Eq. 4.32.*

all variables, so there is no special notation for interactions. Similarly, there is no point in either including or excluding intercepts. If we wanted to transform either a predictor variable or the response, as in `lm`, we can do so. Run like this, `npreg` will try to determine the best bandwidths for the predictor variables, based on a sophisticated combination of cross-validation and optimization.

Let's look at the output of `npreg`:

```

summary(demo.np1)
##
## Regression Data: 100 training points, in 2 variable(s)
##           x         z
## Bandwidth(s): 0.1013 1.541
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 1.797
## R-squared: 0.9587
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 2

```

The main things here are the bandwidths. We also see the root mean squared error on the training data. Note that this is the in-sample root MSE; if we wanted the in-sample MSE, we could do

```

demo.np1$MSE
## [1] 3.229

```

(You can check that this is the square of the residual standard error above.) If we want the cross-validated MSE used to pick the bandwidths, that's

```

demo.np1$bws$fval
## [1] 15.16

```

The `fitted` and `residuals` functions work on these objects just like they do in `lm` objects, while the `coefficients` and `confint` functions do not. (Why?)

The `predict` function also works like it does for `lm`, expecting a data frame containing columns whose names match those in the formula used to fit the model:

```
predict(demo.np1, newdata=data.frame(x=-1,z=5))
## [1] 21.48
```

With two predictor variables, there is a nice three-dimensional default plot (Figure 4.14).

Kernel functions can also be defined for categorical and ordered variables. These can be included in the formula by wrapping the variable in `factor()` or `ordered()`, respectively:

```
demo.np3 <- npreg(y~x+z+factor(w), data=demo.df)
```

Again, there's no point, or need, to indicate interactions. Including the extra variable, not surprisingly, improves the cross-validated MSE:

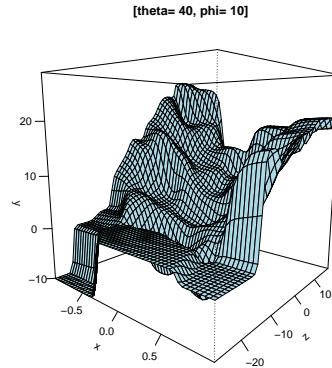
```
demo.np3$bws$fval
## [1] 6.158
```

With three or more predictor variables, we'd need a four-dimensional plot, which is hard. Instead, the default is to plot what happens as we sweep one variable with the others held fixed (by default, at their medians; see `help(nppplot)` for changing that), as in Figure 4.15. We get something parabola-ish as we sweep X (which is right), and something near a step function as we sweep Z (which is right when $W = B$), so we're not doing badly for estimating a fairly complicated function of three variables with only 100 samples. We could also try fixing W at one value or another and making a perspective plot — Figure 4.16.

The default optimization of bandwidths is *extremely* aggressive. It keeps adjusting the bandwidths until the changes in the cross-validated MSE are very small, or the changes in the bandwidths themselves are very small. The “tolerances” for what count as “very small” are controlled by arguments to `npreg` called `tol` (for the bandwidths) and `ftol` (for the MSE), which default to about 10^{-8} and 10^{-7} , respectively. With a lot of data, or a lot of variables, this gets *extremely* slow. One can often make `npreg` run much faster, with no real loss of accuracy, by adjusting these options. A decent rule of thumb is to start with `tol` and `ftol` both at 0.01. One can use the bandwidth found by this initial coarse search to start a more refined one, as follows:

```
bigdemo.df <- make.demo.df(1e3)
demo.np4 <- npreg(y~x+z+factor(w), data=bigdemo.df, tol=0.01, ftol=0.01)
```

This tells us how much time it took R to run `npreg`, dividing that between time spent exclusively on our job and on background system tasks. The result of the run is stored in `demo.np4`:



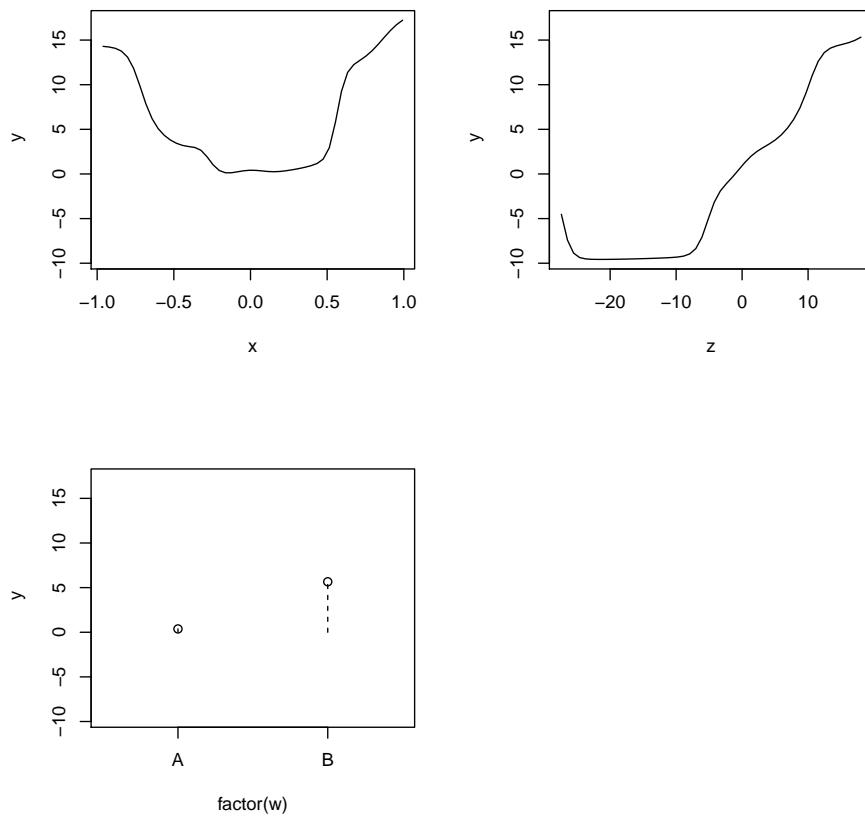
```
plot(demo.np1,theta=40,view="fixed")
```

FIGURE 4.14: Plot of the kernel regression with just two predictor variables. (See `help(npplot)` for plotting options.)

```
demo.np4$bws
##
## Regression Data (1000 observations, 3 variable(s)):
##
##          x      z factor(w)
## Bandwidth(s): 0.04583 1.322 4.214e-07
##
## Regression Type: Local-Constant
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: y ~ x + z + factor(w)
## Bandwidth Type: Fixed
## Objective Function Value: 0.8405 (achieved on multistart 2)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 2
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
```

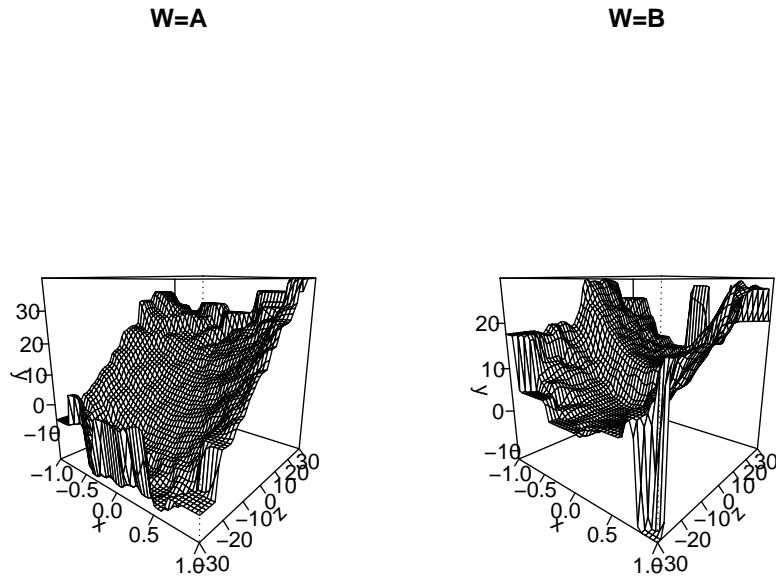
The bandwidths have all shrunk (as they should), and the cross-validated MSE is also much smaller (0.87 versus 6.5 before). Figure 4.16 shows the estimated regression surfaces for both values of the categorical variable.

The package also contains a function, `npregbw`, which takes a formula and a data frame, and just optimizes the bandwidth. This is called automatically by `npreg`, and many of the relevant options are documented in its help page. One can also use the



```
plot(demo.np3)
```

FIGURE 4.15: Predictions of `demo.np3` as each variable is swept over its range, with the others held at their medians.



```

x.seq <- seq(from=-1,to=1,length.out=50)
z.seq <- seq(from=-30,to=30,length.out=50)
grid.A <- expand.grid(x=x.seq,z=z.seq,w="A")
grid.B <- expand.grid(x=x.seq,z=z.seq,w="B")
yhat.A <- predict(demo.np4,newdata=grid.A)
yhat.B <- predict(demo.np4,newdata=grid.B)
par(mfrow=c(1,2))
persp(x=x.seq,y=z.seq,z=matrix(yhat.A,nrow=50),theta=40,main="W=A",
      xlab="x",ylab="z",zlab="y",ticktype="detailed")
persp(x=x.seq,y=z.seq,z=matrix(yhat.B,nrow=50),theta=40,main="W=B",
      xlab="x",ylab="z",zlab="y",ticktype="detailed")

```

FIGURE 4.16: *The regression surfaces learned for the demo function at the two different values of the categorical variable. Note that holding z fixed, we always see a parabolic shape as we move along x (as we should), while whether we see a line or something close to a step function at constant x depends on w, as it should.*

output of `npregbw` as an argument to `npreg`, in place of a formula.

4.7 Further Reading

Simonoff (1996) is a good practical introduction to kernel smoothing and related methods. Wasserman (2006) provides more theory. Li and Racine (2007) is a detailed treatment of nonparametric methods for econometric problems, overwhelming focused on kernel regression and kernel density estimation (which we'll get to in Chapter 16); Racine (2008) summarizes.

Historical Notes Kernel regression was introduced, independently, by Nadaraya (1964) and Watson (1964); both were inspired by kernel density estimation.

4.8 Exercises

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

$$\mathbf{E}[\hat{r}(0)] = \mathbf{E}\left[r(X) \middle| -\frac{h}{2} < X < \frac{h}{2}\right] \quad (4.33)$$

$$= r(0) + r'(0)\mathbf{E}\left[X \middle| -\frac{h}{2} < X < \frac{h}{2}\right] \quad (4.34)$$

$$+ \frac{r''(0)}{2}\mathbf{E}\left[X^2 \middle| -\frac{h}{2} < X < \frac{h}{2}\right] + o(b^2)$$

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

- 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

[[TODO: Insert forward references to detailed simulation examples in other chapters]]

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

¹In your linear models class, you learn about the sampling distribution of regression coefficients when the linear model is true, and the noise is Gaussian, independent of the predictor variables, and has constant variance. As an exercise, try to get parallel results when the noise has a t distribution with 10 degrees of freedom.

basic probability courses. For instance, X_3 is $\mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. These analytical 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 opening the black box at least a little. We'll look at using transformations between distributions, and, in particular, transforming uniform distributions into others (§5.2.2.3). Appendix M explains some more advanced methods, and looks at the issue of how to get uniformly-distributed random numbers in the first place.

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 X , 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 In such situations, we turn to more advanced methods, like those described in Appendix M.

5.2.3 Sampling

A complement to drawing from given distributions is to **sample** from a given collection of objects. This is a common task, so R has a function to do it:

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

Here `x` is a vector which contains the 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`⁴.

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

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 .

⁴If 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`.

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

For example, the code for k -fold cross-validation, Code Example 2, had the lines

```
fold.labels <- sample(rep(1:nfolds, length.out=nrow(data)))
```

Here, `rep` repeats the numbers from 1 to `nfolds` until we have one number for each row of the data frame, say 1,2,3,4,5,1,2,3,4,5,1,2 if there were twelve rows. Then `sample` shuffles 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” process⁵, 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))
}
```

Of course, the labels needn’t be the integers $1 : k$ (exercise 1).

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

⁵A handy term I learned from Gustavo Lacerda.

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 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 when we try to draw inferences about the whole population or process from the sample we happen to have seen.

[[TODO: Check — do we really come back to this?]]

```
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)
```

5.3 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.

5.4 Why Simulate?

There are three major uses for simulation: to understand a model, to check it, and to fit it. We will deal with the first two here, and return to fitting in chapter 30, after we’ve looked at dealing with dependence and hidden variables.

5.4.1 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. (The tilde is a reminder 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

$$\frac{1}{b} \sum_{i=1}^b g(\tilde{Z}_i) \xrightarrow[b \rightarrow \infty]{} \mathbb{E}[g(Z)] \quad (5.7)$$

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**.⁶ If our simulations are independent, we can even use the central limit theorem to say that $\frac{1}{b} \sum_{i=1}^b g(\tilde{Z}_i)$ has approximately the distribution $\mathcal{N}(\mathbb{E}[g(Z)], \text{Var}[g(Z)]/b)$. 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:

$$\frac{1}{b} \sum_{i=1}^b \mathbf{1}_s(g(Z_i)) \xrightarrow[b \rightarrow \infty]{} \Pr(g(Z) \in s) \quad (5.8)$$

The reason this works is of course that $\Pr(g(Z) \in s) = \mathbb{E}[\mathbf{1}_s(g(Z))]$, and we can use the law of large numbers again. 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.

5.4.2 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

⁶The 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.*

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 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.⁷

5.4.2.1 “Exploratory” Analysis of Simulations

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:

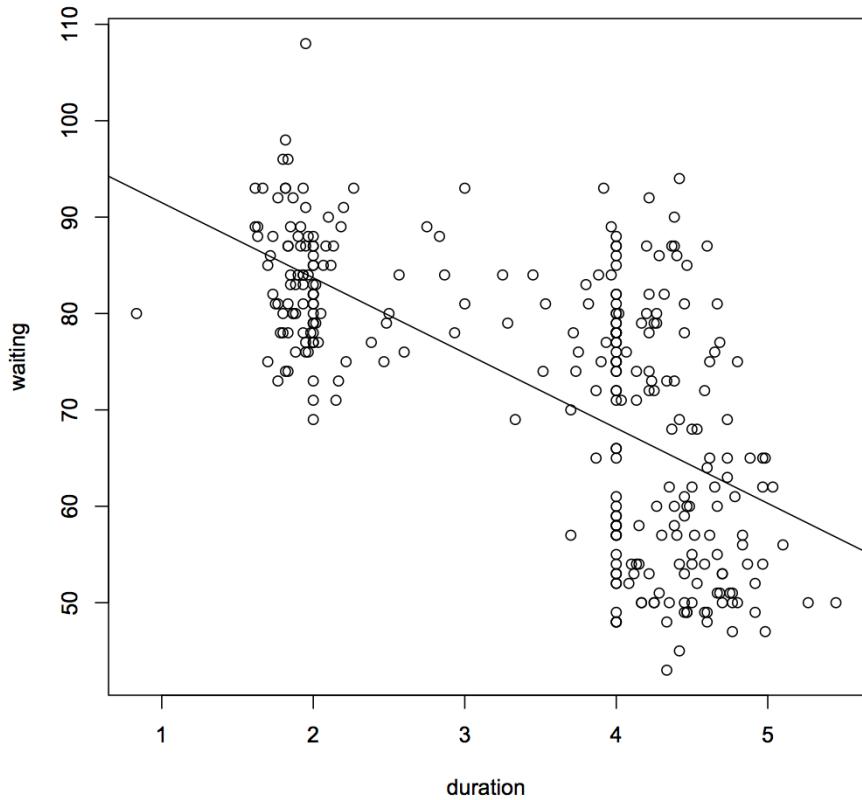
```
library(MASS)
data(geyser)
fit.ols <- lm(waiting~duration,data=geyser)
```

Figure 5.1 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 to the model assumptions.

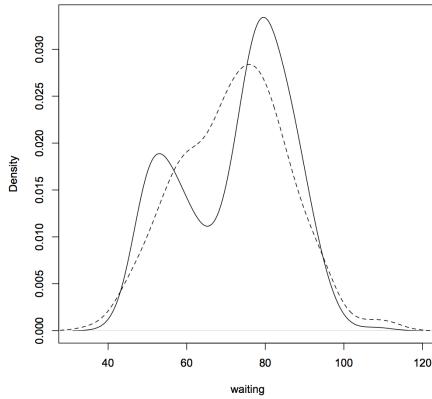
A useful principle for model checking is that if we do some exploratory data analyses of the real data, doing the same analyses to realizations of the model should

⁷“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 5.1: 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 5.2: Actual density of the waiting time between eruptions (solid curve) and that produced by simulating the OLS model (dashed).

give roughly the same results. This is a test the model fails. Figure 5.2 shows the actual density of `waiting`, plus the density produced by simulating — reality is clearly bimodal, but the model is unimodal. Similarly, Figure 5.3 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.

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 so how to fix it. Further simulations, comparing more detailed analyses of the data to analyses of the simulation output, are often very helpful here. Looking at Figure 5.3, 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.

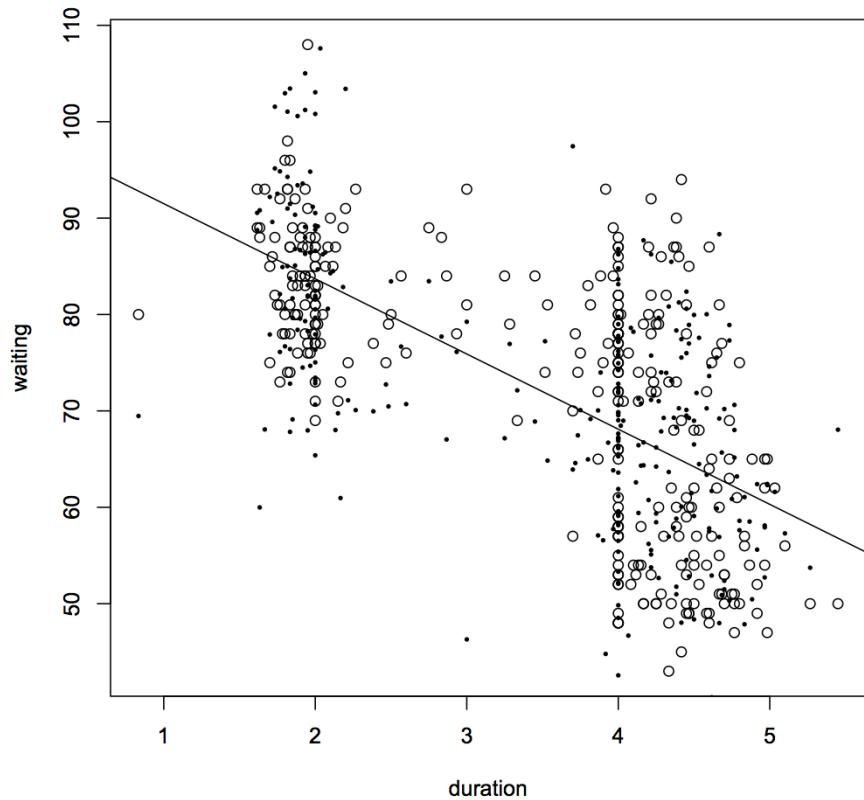
5.4.3 Sensitivity Analysis

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

[[TODO: Citations]]

[[ATTN: Forward ref to density estimation, or replace with histogram?]]

[[TODO: Expand on this]]



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

FIGURE 5.3: As in Figure 5.1, plus one realization of simulating the OLS model (small black dots).

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.

Sections on simulation-based inference moved to new chapter after time series

5.5 Further Reading

On stochastic simulation in general, see [[Ripley]], [[]]. Many references on simulation present it as almost completely disconnected from statistics and data analysis, giving the impression that probability models just fall from the sky. Guttorp (1995) is an excellent exception.

[[CRAN task view on random variable generation]]

For further reading on methods of drawing random variables from a given distribution, on Monte Carlo, and on generating uniform random numbers, see Appendix M.

5.6 Exercises

1. Modify `rmultinoulli` from §5.2.3.2 so that the values in the output are not the integers from 1 to k , but come from a vector of arbitrary labels.

Chapter 6

The Bootstrap

[[TODO: Make sure all code clearly separates (1) simulator, (2) statistic-calculator, and (3) summarizer]]

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.

6.1. STOCHASTIC MODELS, UNCERTAINTY, SAMPLING DISTRIBUTIONS

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 the 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.

[[Cross-ref hypothesis testing chapter, when it's written]]

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 hypoth-

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 or 1960s (Nadaraya, 1964; Watson, 1964; Cover and Hart, 1967) [[TODO: KDE citations]], but didn't begin to be widely used until the late 1970s, or even the 1980s.

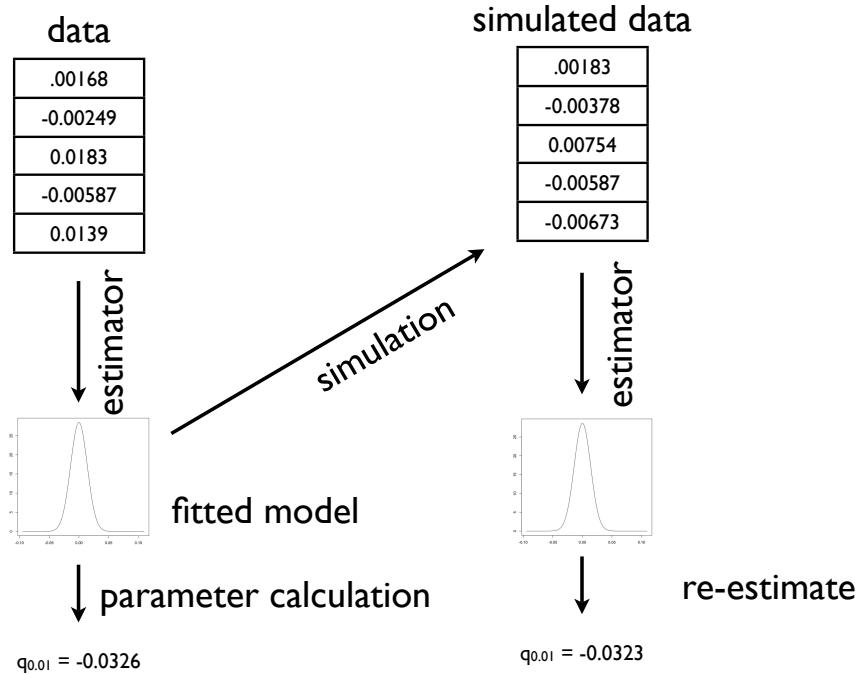


FIGURE 6.1: Schematic for model-based bootstrapping: simulated values are generated from the fitted model, then treated like the original data, yielding a new estimate of the functional of interest, here called $q_{0.01}$.

esis, has the same distribution as the real data. Feeding the simulated data through 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 on his own 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 . This means that we are employing the **parametric bootstrap**. The

```
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 6: 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. IMPORTANT NOTE: This is just a code sketch, because depending on the data structure which the statistic returns, it may not (e.g.) be feasible to just run `sd` on it, and so it might need some modification. See detailed examples below.

true (population or ensemble) values of the functional is likewise t_0 .

6.2.1 Variances and Standard Errors

The simplest thing to do is to get the variance or standard error:

$$\widehat{\text{Var}}[\hat{t}] = \text{Var}[\hat{t}] \quad (6.1)$$

$$\widehat{\text{se}}(\hat{t}) = \text{sd}(\hat{t}) \quad (6.2)$$

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 \hat{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 \hat{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 .

A code sketch is provided in Code Example 6. Note that this may not work *exactly* as given in some circumstances, depending on the syntax details of, say, just what kind of data structure is needed to store \hat{t} .

```
bootstrap.bias <- function(simulator, statistic, B, t.hat) {
  tboots <- rboot(B, statistic, simulator)
  bias <- mean(tboots) - t.hat
  return(bias)
}
```

CODE EXAMPLE 7: Sketch of code for bootstrap bias correction. Arguments are as in Code Example 6, except that `t.hat` is the estimate on the original data. IMPORTANT NOTE: As with Code Example 6, this is just a code sketch, because it won't work with all data types that might be returned by `statistic`, and so might require modification.

6.2.2 Bias Correction

We can use bootstrapping to correct for a biased estimator. Since the sampling distribution of \hat{t} is close to that of \tilde{t} , and \hat{t} itself is close to t_0 ,

$$\mathbb{E}[\hat{t}] - t_0 \approx \mathbb{E}[\tilde{t}] - \hat{t} \quad (6.3)$$

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.

In fact, 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.⁴

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

⁴You might wonder why we'd be unhappy if the coverage level was *greater* than $1 - \alpha$. This is certainly better than if it's *less* than the nominal confidence level, but it usually means we could have used a smaller set, and so been more precise about t_0 , without any more real risk. Confidence intervals whose coverage is greater than the nominal level are called "conservative"; those with less than nominal coverage are "anti-conservative" (and not, say, liberal).

```
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 8: Sketch of code for calculating the basic bootstrap confidence interval. See Code Examples 7 and 6 for `rboot`, and cautions about blindly applying this to arbitrary data-types.

$\tilde{t} - \hat{t}$ 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} .

Specifically, let $q_{\alpha/2}$ and $q_{1-\alpha/2}$ be the $\alpha/2$ and $1 - \alpha/2$ quantiles of \tilde{t} . Then

$$1 - \alpha = \Pr(q_{\alpha/2} \leq \tilde{T} \leq q_{1-\alpha/2}) \quad (6.5)$$

$$= \Pr(q_{\alpha/2} - \hat{T} \leq \tilde{T} - \hat{T} \leq q_{1-\alpha/2} - \hat{T}) \quad (6.6)$$

$$\approx \Pr(q_{\alpha/2} - \hat{T} \leq \tilde{T} - t_0 \leq q_{1-\alpha/2} - \hat{T}) \quad (6.7)$$

$$= \Pr(q_{\alpha/2} - 2\hat{T} \leq -t_0 \leq q_{1-\alpha/2} - 2\hat{T}) \quad (6.8)$$

$$= \Pr(2\hat{T} - q_{1-\alpha/2} \leq t_0 \leq 2\hat{T} - q_{\alpha/2}) \quad (6.9)$$

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.

[[TODO: Add subsection specifically on confidence bands]]

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

$$\tau = \frac{\hat{t} - t_0}{\widehat{\text{se}}(\hat{t})} \quad (6.10)$$

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 (§6.8).

```
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 9: *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, presuming 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 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 9 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.

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 gave us so much trouble in Chapter 3.)

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 10).

```

doubleboot.pvalue <- function(test,simulator,B1,B2,
  estimator, thetahat, testhat) {
  for (i in 1:B1) {
    xboot <- simulator(theta=thetahat, ...)
    thetaboot <- estimator(xboot)
    testboot[i] <- test(xboot)
    pboot[i] <- boot.pvalue(test,simulator,B2,
      testhat=testboot[i],theta=thetaboot)
  }
  p <- (sum(testboot >= testhat)+1)/(B1+1)
  p.adj <- (sum(pboot <= p)+1)/(B1+1)
}

```

CODE EXAMPLE 10: *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.*

6.2.5 Parametric Bootstrapping Example: Pareto’s Law of Wealth Inequality

The Pareto distribution⁵, 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 1). The Pareto is highly right-skewed, with the mean being much larger than the median.

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 consistent⁶, 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

⁵Named after Vilfredo Pareto (1848–1923), the highly influential economist, political scientist, and proto-Fascist.

⁶Because the sample mean of $\log X$ converges, under the law of large numbers

```
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 11: *Standard error and bias calculation for the Pareto distribution, using parametric bootstrapping.*

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 richest⁷. 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
```

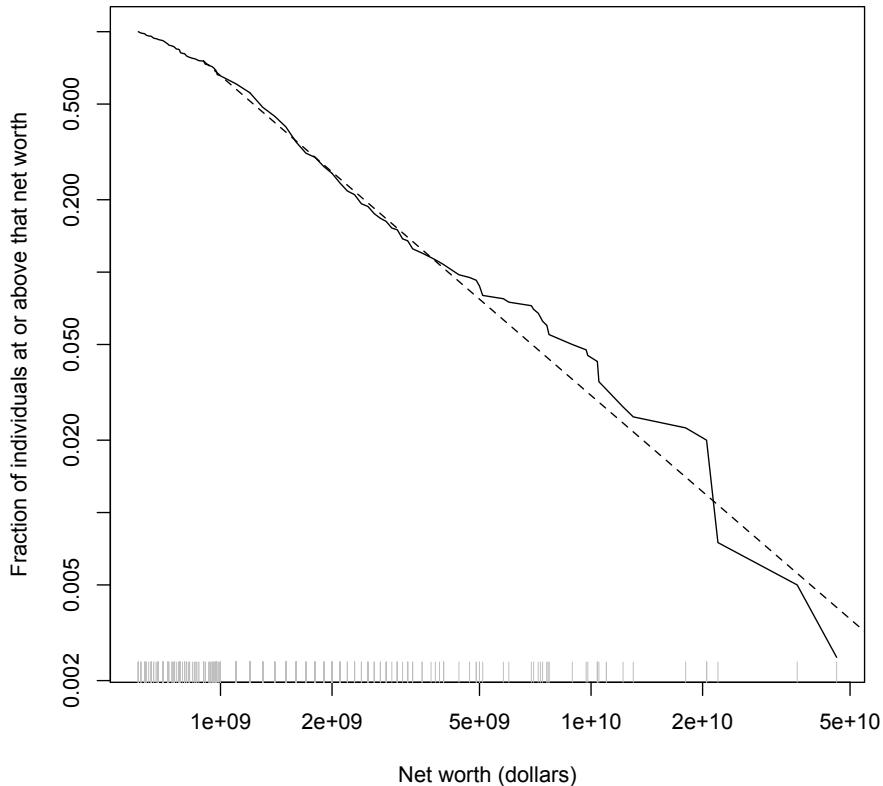
How much uncertainty is there in this estimate of the exponent? Naturally, we'll bootstrap. We need a function to generate Pareto-distributed random variables; this, along with some related functions, is part of the file `pareto.R` on the course website. With that tool, parametric bootstrapping proceeds as in Code Example 11.

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 well⁸, 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.

⁷Most of the distribution conforms to a log-normal, at least roughly.

⁸"In Asymptopia", 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.



```

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 $(302/400)$.*

```

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 12: *Parametric bootstrap confidence interval for the Pareto scaling exponent.*

We can also get the confidence interval (Code Example 12). 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.⁹ Code Example 13 calculates the *p*-value. With ten thousand bootstrap replications,

```

> ks.pvalue.pareto(1e4,wealth,2.34,9e8)
[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 17, we'll look at techniques which can begin to tell us something about *how* it fails.

[[ATTN: Revisit this example in that chapter, cross-ref]]

6.3 Non-parametric Bootstrapping by Resampling

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

⁹The `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 $x_0 = 9 \times 10^8$.

```

ks.stat.pareto <- function(x, exponent, x0) {
  x <- x[x>=x0]
  ks <- ks.test(x, ppareto, exponent=exponent,
    threshold=x0)
  return(ks$statistic)
}

ks.pvalue.pareto <- function(B, x, exponent, x0) {
  testthat <- ks.stat.pareto(x, exponent, x0)
  testboot <- vector(length=B)
  for (i in 1:B) {
    xboot <- rpareto(length(x), exponent=exponent,
      threshold=x0)
    exp.boot <- pareto.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 13: 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.

```
resample <- function(x) { sample(x, size=length(x), replace=TRUE) }
```

CODE EXAMPLE 14: A utility function to resample from a vector. This is used by almost all the bootstrapping code in the rest of this chapter, even when the data is not just a vector.

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 pre-conceptions, possibly accurate but also potentially misleading¹⁰. 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, Code Example 14). 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 Gaussian¹¹.

Code Example 15 shows how to use re-sampling to get a 95% confidence interval for the Pareto exponent¹². 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.

¹⁰See §16.6 in Chapter 16.

¹¹We will see in Chapter 16 that this corresponds to sampling from a kernel density estimate

¹²Even 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-truth**; 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.

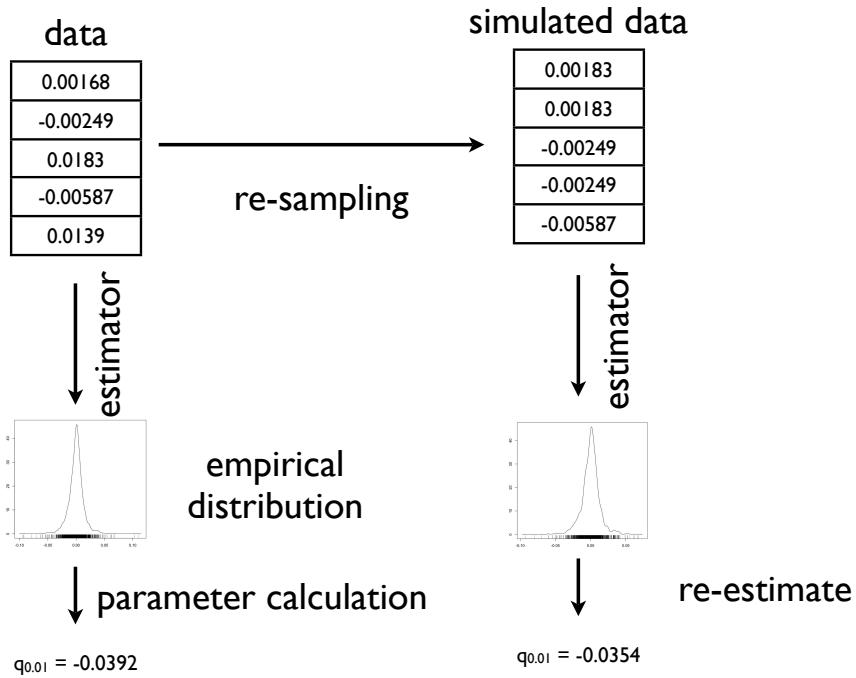


FIGURE 6.3: Schematic for non-parametric bootstrapping. New data is simulated by re-sampling from the original data (with replacement), and parameters are calculated either directly from the empirical distribution, or by applying a model to this surrogate data.

```

resamp.pareto <- function(B,x,x0) {
  replicate(B,pareto.fit(resample(x),threshold=x0)$exponent)
}

resamp.pareto.CI <- function(B,x,alpha,x0) {
  thetahat <- pareto.fit(x,threshold=x0)$exponent
  thetaboot <- resamp.pareto(B,x,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 15: Non-parametric bootstrap confidence intervals for the Pareto scaling exponent.

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, $\epsilon_i = y_i - \hat{y}_i = \hat{r}(x_i)$, 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 saw this data set already in §5.4.2.1, and will see it again in §7.3.2.) 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 linear predictor*, a functional of the distribution, which, as we saw in Chapters 1 and 2, exists

[[TODO: Explain, give example, of bootstrap for generalization error, as alternative to CV]]

[[ATTN: Replace with more modern data example?]]

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.data.frame <- function(data) {
  sample.rows <- resample(1:nrow(data))
  return(data[sample.rows,])
}
```

We can check this by running `summary(resample.data.frame(geyser))`, and seeing that it gives about the same quartiles and mean for both variables as `summary(geyser)`¹³.

Next, we define the estimator:

```
est.waiting.on.duration <- function(data) {
  fit <- lm(waiting ~ duration, data=data)
  return(coefficients(fit))
}
```

We can check that this function works by seeing that `coefficients(geyser.lm)` matches `est.waiting.on.duration(geyser)`, but that `est.waiting.on.duration(resample.data.frame(geyser))` is different every time we run it.

Putting the pieces together according to the basic confidence interval recipe (Code Example 16), 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 here¹⁴.

[[ATTN: Redo code so geyser.lm.cis can optionally take in a matrix of replicates?]]

¹³The minimum and maximum won't match up well — why not?

¹⁴We 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 $\beta_0, \hat{\beta}_1$. [[TODO: Example.]]

```
geyser.lm.cis <- function(B,alpha) {
  tboot <- replicate(B,
    est.waiting.on.duration(resample.data.frame(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 16: *Bootstrapped confidence intervals for the linear model of Old Faithful, based on re-sampling data points. Note: this relies on functions defined in the text.*

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(data,tol=0.1,ftol=0.1) {
  bw <- npregbw(waiting ~ duration, data=data, tol=tol, ftol=ftol)
  fit <- npreg(bw)
  return(fit)
}
geyser.npr <- npr.waiting.on.duration(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 17). Observe that this will go through bandwidth selection again for each bootstrap replicate. 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 laptop¹⁵.

[[ATTN: Talk about the bias issue (here? further reading?), hacks vs. living with it]]

¹⁵Specifically, 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.

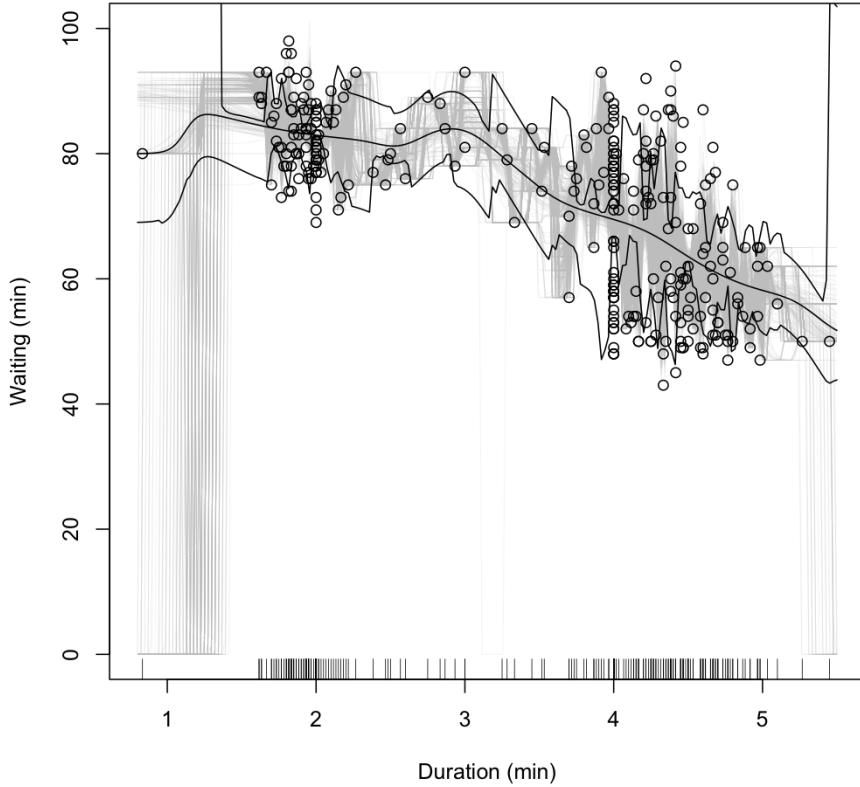
```
evaluation.points <- seq(from=0.8,to=5.5,length.out=200)
evaluation.points <- data.frame(duration=evaluation.points)

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(geyser.waiting.on.duration(resample.data.frame(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 17: *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.

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, based on the data-analysis assignment in §33.¹⁶ We will regress `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 18 shows the new simulator for this set-up (`resample.residuals.penn`)¹⁷, the new estimation function (`penn.estimator`)¹⁸, 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.

¹⁶Note to 2015 students: This is an old problem set related to our homework 2, also on the determinants of economic growth across countries and years, but using a different set of variables. `gdp.growth` and `gdp` are obvious, `pop.growth` is the country's rate of population growth, `invest` is the fraction of GDP devoted to investment, and `trade` is the ratio of imports plus exports to GDP. The data repository is the "Penn World Tables". See pp. 605 of the full draft for details and the assignment.

¹⁷How would you check that this was working right?

¹⁸How 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 18: *Re-sampling the residuals to get confidence intervals in a linear model.*

6.5 Bootstrap with Dependent Data

If the data points 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 28–31.

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 little¹⁹.

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

¹⁹More 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 Which Bootstrap When?

This chapter has introduced a bunch of different bootstraps, and before it closes it’s worth reviewing the general principles, and some of the considerations which go into choosing among them in a particular problem.

When we bootstrap, we try to approximate the sampling distribution of some statistic (mean, median, correlation coefficient, regression coefficients, smoothing curve, difference in MSEs...) by running simulations, and calculating the statistic on the simulation. We’ve seen three major ways of doing this:

- The parametric bootstrap: we estimate the model, and then simulate from the estimated model;
- Resampling residuals: we estimate the model, and then simulate by resampling residuals to that estimate and adding them back to the fitted values;
- Resampling cases or whole data points: we ignore the estimated model completely in our simulation, and just re-sample whole rows from the data frame.

Which kind of bootstrap is appropriate depends on how much trust we have in our model.

The parametric bootstrap trusts the model to be completely correct for *some* parameter value. In, e.g., regression, it trusts that we have the right shape for the regression function *and* that we have the right distribution for the noise. When we trust our model this much, we could in principle work out sampling distributions analytically; the parametric bootstrap replaces hard math with simulation.

Resampling residuals doesn’t trust the model as much. In regression problems, it assumes that the model gets the *shape* of the regression function right, and that the noise around the regression function is independent of the predictor variables, but doesn’t make any assumption about how the fluctuations are distributed. It is therefore more secure than parametric bootstrap.²⁰

Finally, resampling cases assumes nothing at all about either the shape of the regression function or the distribution of the noise, it just assumes that each data point (row in the data frame) is an independent observation. Because it assumes so little, and doesn’t depend on any particular model being correct, it is very safe.

The reason we do not always use the safest bootstrap, which is resampling cases, is that there is, as usual, a bias-variance trade-off. Generally speaking, if we compare three sets of bootstrap confidence intervals on the same data for the same statistic, the parametric bootstrap will give the narrowest intervals, followed by resampling residuals, and resampling cases will give the loosest bounds. If the model really *is*

²⁰You could also imagine simulations where we presume that the noise takes a very particular form (e.g., a *t*-distribution with 10 degrees of freedom), but are agnostic about the shape of the regression function, and learn that non-parametrically. It’s harder to think of situations where this is really plausible, however, except *maybe* Gaussian noise arising from central-limit-theorem considerations.

correct about the shape of the curve, we can get more precise results, without any loss of accuracy, by resampling residuals rather than resampling cases. If the parametric model is also correct about the distribution of noise, we can do even better with a parametric bootstrap.

To sum up: resampling cases is safer than resampling residuals, but gives wider, weaker bounds. If you have good reason to trust a model's guess at the shape of the regression function, then resampling residuals is preferable. If you don't, or it's not a regression problem so there are no residuals, then you prefer to resample cases. The parametric bootstrap works best when the over-all model is correct, and we're just uncertain about the exact parameter values we need.

6.8 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 nowadays²¹. Davison and Hinkley (1997) is both a good textbook, and the reference I consult most often. Efron and Tibshirani (1993), while also very good, is more theoretical. Canty *et al.* (2006) has useful advice for serious applications.

For professional purposes, I strongly recommend using the R package `boot` (Canty and Ripley, 2013), based on Davison and Hinkley (1997). I deliberately do *not* use it in this chapter, or later in the book, for pedagogical purposes; I have found that forcing students to write their own bootstrapping code helps build character, or at least understanding.

6.9 Exercises

1. Show that x_0 is the mode of the Pareto distribution.
2. Derive the maximum likelihood estimator for the Pareto distribution (Eq. 6.15) from the density (Eq. 6.14).
3. Find confidence bands for the linear regression model of §6.4.1 using
 - (a) The usual Gaussian assumptions (*hint*: try the `intervals="confidence"` option to `predict`);
 - (b) Resampling of residuals; and
 - (c) Resampling of cases.

²¹It 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. *Sampling bias.* In many situations, our data comes from a survey, and some members of the population may be more likely to be included in the sample than others. When this happens, the sample is a biased representation of the population. If we want to draw inferences about the population, it can help to give more weight to the kinds of data points which we've under-sampled, and less to those which were over-sampled. In fact, typically the weight put on data point i would be inversely proportional to the probability of i being included in the sample (exercise 1). Strictly speaking, if we are willing to believe that linear model is exactly correct, that there are no omitted variables, and that the inclusion probabilities p_i do not vary with y_i , then this sort of survey weighting is redundant (DuMouchel and Duncan, 1983). When those assumptions are not met — when there're non-linearities, omitted variables, or “selection on the

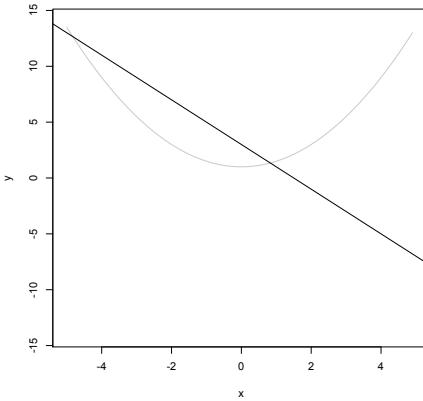


FIGURE 7.1: Black line: Linear response function ($y = 3 - 2x$). Grey curve: standard deviation as a function of x ($\sigma(x) = 1 + x^2/2$).

dependent variable” — survey weighting is advisable, if we know the inclusion probabilities fairly well.

The same trick works under the same conditions when we deal with “covariate shift”, a change in the distribution of X . If the old probability density function was $p(x)$ and the new one is $q(x)$, the weight we’d want to use is $w_i = q(x_i)/p(x_i)$ (Quiñonero-Candela *et al.*, 2009). This can involve estimating both densities, or their ratio (chapter 16).

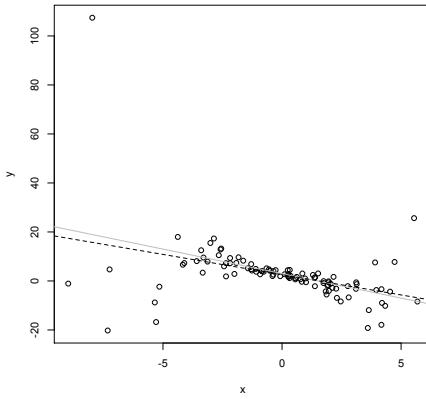
4. *Doing something else.* There are a number of other optimization problems which can be transformed into, or approximated by, weighted least squares. The most important of these arises from generalized linear models, where the mean response is some nonlinear function of a linear predictor; we will 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

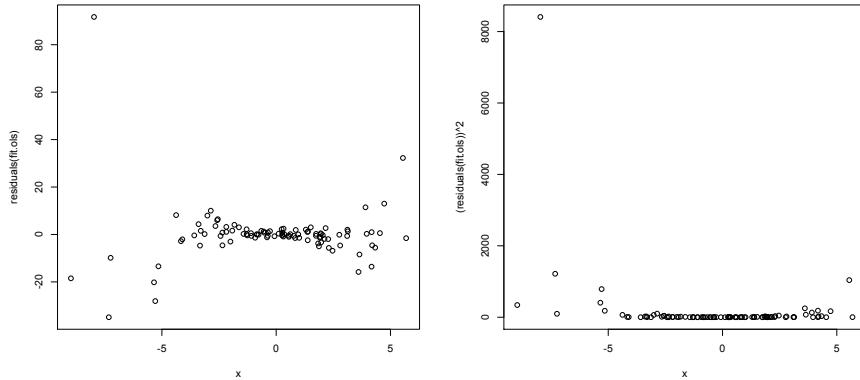
Suppose the noise variance is itself variable. For example, the figure shows a simple linear relationship between the input X and the response Y , but also a nonlinear relationship between X and $\text{Var}[Y]$.

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



```
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.



```
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.

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 19).

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 squares (Figure 7.5).

```

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))
}

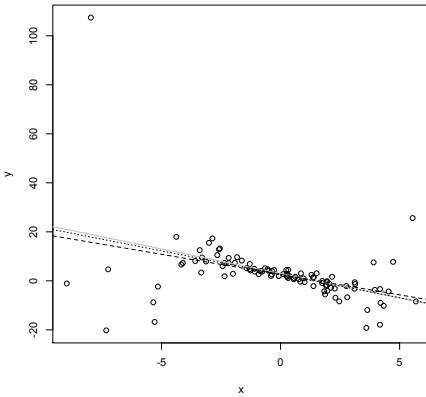
ols.heterosked.error.stats = function(n,m=10000) {
  ols.errors.raw = t(replicate(m,ols.heterosked.example(n)))
  # transpose gives us a matrix with named columns
  intercept.sd = sd(ols.errors.raw[, "(Intercept)"])
  slope.sd = sd(ols.errors.raw[, "x"])
  return(list(intercept.sd=intercept.sd,slope.sd=slope.sd))
}

```

CODE EXAMPLE 19: *Functions to generate heteroskedastic data and fit OLS regression to it, and to collect error statistics on the results.*



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).

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 20.)

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.

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 $w_i \propto 1/\sigma_{x_i}$,

¹Less anthropomorphically, the objective function in Eq. 7.1 has the same derivative with respect to the squared error at each point, $\frac{\partial MSE}{\partial (y_i - \tilde{x}_i \cdot \beta)^2} = \frac{1}{n}$.

```
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 20: *Linear regression of heteroskedastic data, using weighted least-squared regression.*

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 $\text{Var}[\epsilon_i] = \sigma_{x_i}^2$. 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

$$\text{Var}[\hat{\beta}_j] = \sum_{i=1}^n H_{ji}(w) \sigma_{x_i}^2 \quad (7.11)$$



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 “essayen”, <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.)

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 H).

A less general but easier-to-grasp result comes from adding the assumption that the noise around the regression line is Gaussian — that

$$Y = \vec{x} \cdot \beta + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_x^2) \quad (7.12)$$

The log-likelihood is then (Exercise 2)

$$-\frac{n}{2} \ln 2\pi - \frac{1}{2} \sum_{i=1}^n \log \sigma_{x_i}^2 - \frac{1}{2} \sum_{i=1}^n \frac{(y_i - \vec{x}_i \cdot \beta)^2}{\sigma_{x_i}^2} \quad (7.13)$$

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)?

Sometimes 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 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.

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 $\hat{s}(x)$.
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.

²I learned it from Wasserman (2006, pp. 87–88).

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.

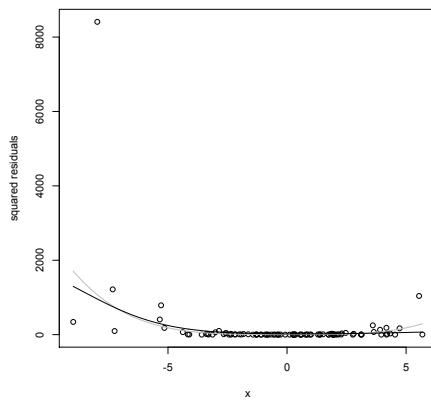
Let's apply this idea to our example. Figure 7.3b already plotted the residuals from OLS. Figure 7.7 shows those squared residuals again, along with the true variance function and the estimated variance function.

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.

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
```

³Occasionally people do things like claiming that gene differences 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 — like Alford *et al.* (2005) — even say this with a straight face.

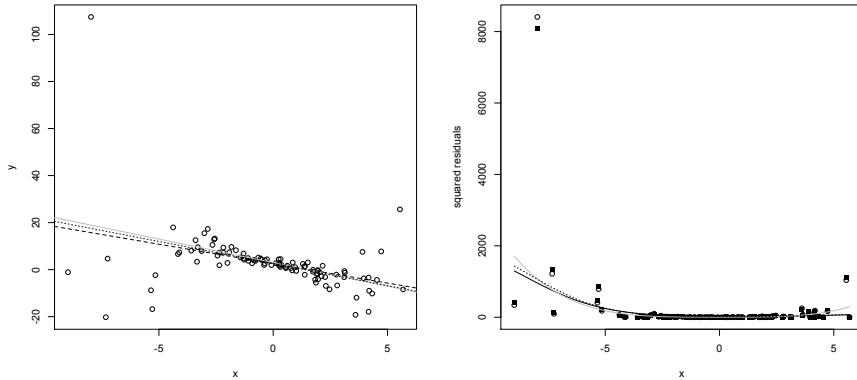


```

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).

```
> 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.

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.

7.3.2 Real Data Example: Old Heteroskedastic

§5.4.2 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 wigginess 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.

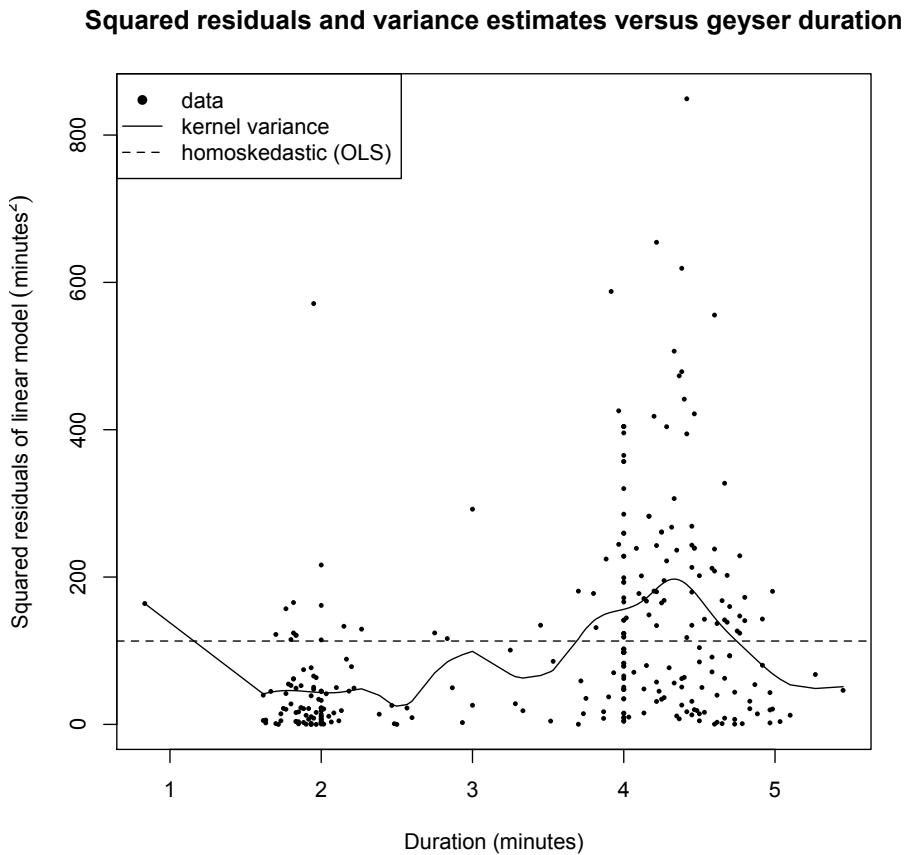
While that sort of qualitative comparison is genuinely informative, one can also be more quantitative. One might measure heteroskedasticity by, say, evaluating the conditional variance at all the data points, and looking at the ratio of the interquartile range to the median. This would be zero for perfect 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 the one made in §5.4.2.1: 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 6.

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 $\hat{\sigma}^2(x)$, 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

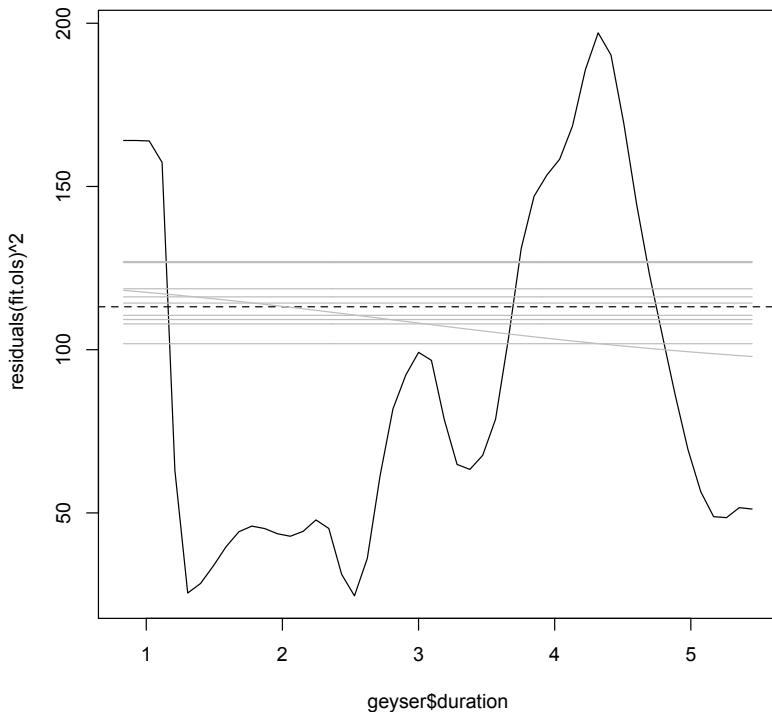


```

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 5.1, 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 §5.4.2 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.*

conditional standard deviation:

$$\eta_i = \epsilon_i / \hat{\sigma}(x_i) \quad (7.16)$$

The η_i should now be all the same magnitude (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 the 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 typically Taylor-expand (App. refapp:taylor) the latter around any point x_0 ,

$$r(x) = r(x_0) + \sum_{k=1}^{\infty} \frac{(x - x_0)^k}{k!} \left. \frac{d^k r}{dx^k} \right|_{x=x_0} \quad (7.17)$$

and similarly with all the partial derivatives in higher dimensions. Truncating the series at first order, $r(x) \approx r(x_0) + (x - x_0)r'(x_0)$, we see the first derivative $r'(x_0)$ is the best linear prediction coefficient, at least if x close enough to x_0 . The snag in this line of argument is that if $r(x)$ is nonlinear, then r' isn't a constant, and the optimal linear predictor changes depending on where we want to make predictions.

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 interest⁵.

The 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 has at least three drawbacks. First, we get weird discontinuities at the boundaries between bins. Second, we pick up an odd sort of bias, where our predictions near the boundaries of a bin depend strongly on data from one 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

⁵Some people say “local linear” and some “locally linear”.

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 to zero.

By now bells may be going off, 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. We set up the problem

$$\operatorname{argmin}_{m(x)} \frac{1}{n} \sum_{i=1}^n w_i(x)(y_i - m(x))^2 \quad (7.18)$$

and get 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 $\mathbf{z}_i = (1, x_i - x)$, i.e., the 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.⁶

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**,

$$K(x_i, x) = \left(1 - \left(\frac{|x_i - x_0|}{h}\right)^3\right)^3 \quad (7.22)$$

⁶Notice 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.

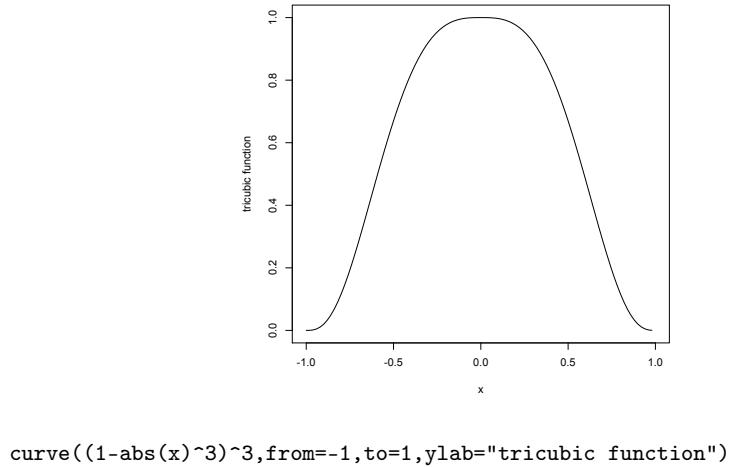


FIGURE 7.11: *The tricubic kernel, with broad plateau where $|x| \approx 0$, and the smooth fall-off to zero at $|x| = 1$.*

if $|x - x_i| < h$, and = 0 otherwise (Figure 7.11).

7.5.1 For and Against Locally Linear Regression

Why would we use locally linear regression, if we already have kernel regression?

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, at least asymptotically. Since locally linear regression has smaller bias, the former is often predictively superior.

[[Drawback: takes more time! Need to solve a linear least squares problem at each point, or for each prediction. This is slower than just taking an average.]]

There are several packages which implement locally linear regression. Since we are already using `np`, one of the simplest is to set the `regtype="ll"` in `npreg`.⁷ 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`.⁸ 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 “robust” loss function,

$$\operatorname{argmin}_{\beta(x)} \frac{1}{n} \sum_{i=1}^n w_i(x) \ell(y - \vec{x}_i \cdot \beta(x)) \quad (7.23)$$

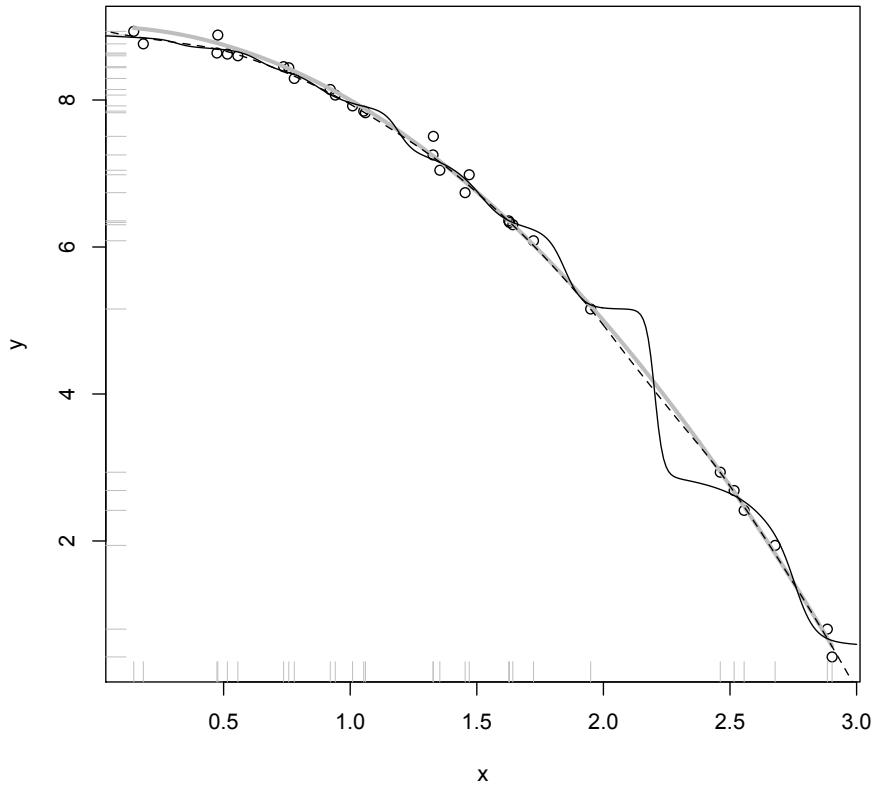
where $\ell(a)$ doesn’t grow as rapidly for large a as a^2 . The idea is to make the fitting less vulnerable to occasional large outliers, which would have very large squared errors, unless the regression curve went far out of its way to accommodate them. For instance, we might have $\ell(a) = a^2$ if $|a| < 1$, and $\ell(a) = 2|a| - 1$ otherwise⁹. 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“ll” stands for “locally linear”, of course; the default is `regtype="lc"`, for “locally constant”.

⁸I have heard this name explained as an acronym for both “locally weighted scatterplot smoothing” and “locally weight sum of squares”.

⁹This 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.



```

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).

7.6 Exercises

1. Imagine we are trying to estimate the mean value of Y from a large population. We observe n members of the population, with individual i being included in our sample with a probability proportional to π_i . Show that the sample mean $n^{-1} \sum_{i=1}^n y_i$ is *not* a consistent estimator of $E[Y]$ unless all the π_i are equal. Show that $(\sum_{i=1}^n y_i / \pi_i) / \sum_{i'=1}^n 1/\pi_{i'}$ is a consistent estimator of $E[Y]$.
2. Show that the model of Eq. 7.12 has the log-likelihood given by Eq. 7.13
3. Do the calculus to verify Eq. 7.4.
4. 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?
5. §7.2.2 showed that 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.
6. §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.
7. 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?

Chapter 8

Splines

8.1 Smoothing by 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 control smoothness itself?

A natural way to do this 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. We will accept changes in m that increase the MSE by 1 unit if they also reduce the average curvature by at least λ .

The curve or function which solves this minimization problem,

$$\hat{r}_\lambda = \operatorname{argmin}_m \mathcal{L}(m, \lambda) \quad (8.2)$$

is called a **smoothing spline**, or **spline curve**. The name “spline” comes from a simple tool used by craftsmen to draw smooth curves, which was a thin strip of a flexible material like a soft wood; you pin it in place at particular points, called **knots**, and let it bend between them. (When the gas company dug up my front yard and my neighbor's driveway, the contractors who put everything back used a plywood board



FIGURE 8.1: *A wooden spline used to create a smooth, curved border for a paved area (Shadyside, Pittsburgh, October 2014).*

to give a smooth, curved edge to the new driveway. That board was a spline, and the knots were pairs of metal stakes on either side of the board. Figure 8.1 shows the spline after concrete was poured on one side of it.) Bending the spline takes energy — the stiffer the material, the more energy has to go into bending it through the same shape, and so the material makes a straighter curve between given points. For smoothing splines, using a stiffer material corresponds to increasing λ .

It is possible to show (§8.6 below) that all solutions to Eq. 8.1, no matter what the data might be, 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 sit at the original data points. By analogy with the craftsman’s spline, the boundary points are called the **knots** of the smoothing spline. The function is continuous beyond the largest and smallest data points, but it is always linear in those regions.¹

I will also assert, without proof, that, with enough pieces, such piecewise cubic polynomials can approximate any well-behaved function arbitrarily closely. Finally, smoothing splines are linear smoothers, in the sense of Chapter 1: predicted values are linear combinations of the training-set response values y_i — see Eq. 8.21 below.

8.1.1 The Meaning of the Splines

Look back to the optimization problem. As $\lambda \rightarrow \infty$, any curvature at all becomes infinitely costly, and only linear functions are allowed. But we know how to minimize mean squared error with linear functions, that’s OLS. So we understand 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.

¹Can you explain why it is linear outside the data range, in terms of the optimization problem?

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.² 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).

[[ATTN: Worth including formulas after all?]]

8.2 Computational Example: Splines for Stock Returns

The default R function for fitting a smoothing spline is `smooth.spline`:

```
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 daily logarithmic returns³ of the S&P 500 stock index, on 5542 consecutive trading days, from 9 February 1993 to 9

²The slightly longer version: Consider minimizing the MSE (not the penalized MSE), but only over functions m where $\int (m''(x))^2 dx$ is at most some maximum level C . λ would then be the Lagrange multiplier enforcing the constraint. The constrained but unpenalized optimization is equivalent to the penalized but unconstrained one. In economics, λ would be called the “shadow price” of average curvature in units of MSE, the rate at which we’d be willing to pay to have the constraint level C marginally increased.

³For a financial asset whose price on day t is p_t and which pays a dividend on that day of d_t , the log-returns on t are $\log(p_t + d_t)/p_{t-1}$. Financiers and other professional gamblers care more about the log returns than about the price change, $p_t - p_{t-1}$, because the log returns give the rate of profit (or loss) on investment. We are using a price series which is adjusted to incorporate dividend (and related) payments.

February 2015⁴.

```
require(pdfetch)
sp <- pdfetch_YAHOO("SPY", fields="adjclose",
  from=as.Date("1993-02-09"), to=as.Date("2015-02-09"))
sp <- diff(log(sp))
# need to drop the initial NA which makes difficulties later
sp <- sp[-1]
```

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 -0.0642 (grey line in the figure). Fitting a smoothing spline with cross-validation selects $\lambda = 0.0127$, and the black curve:

```
> sp.today <- head(sp,-1)
> sp.tomorrow <- tail(sp,-1)
> coefficients(lm(sp.tomorrow ~ sp.today))
  (Intercept)    sp.today
  0.0003708103 -0.0641670375
> sp.spline <- smooth.spline(x=sp.today,y=sp.tomorrow,cv=TRUE)
Warning message:
In smooth.spline(x = sp.today, y = sp.tomorrow, cv = TRUE) :
  cross-validation with non-unique 'x' values seems doubtful
> sp.spline
Call:
smooth.spline(x = sp.today, y = sp.tomorrow, cv = TRUE)

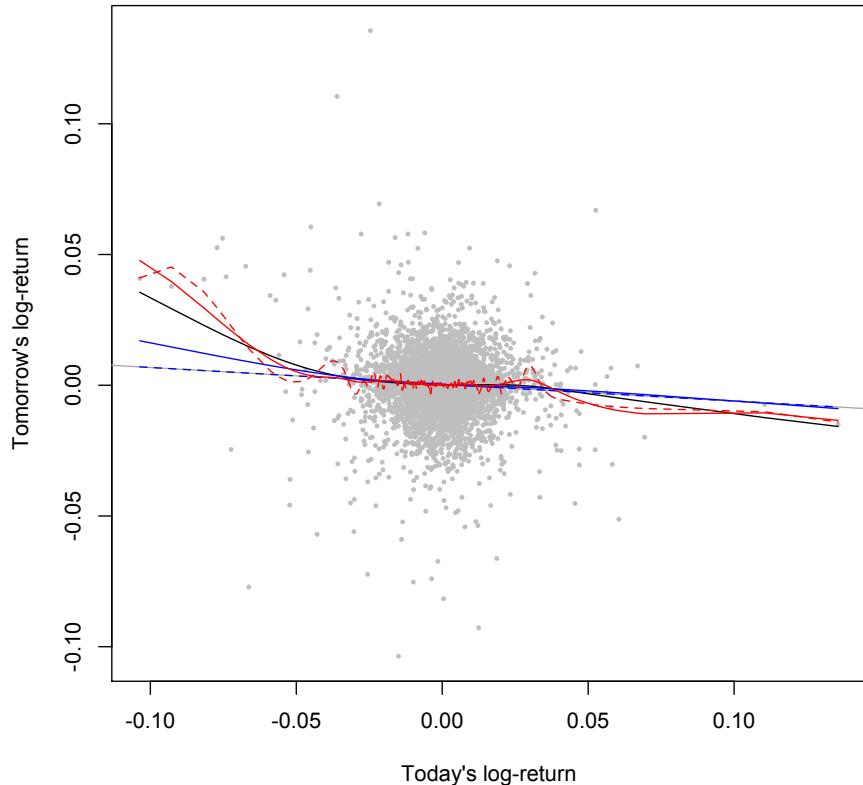
Smoothing Parameter  spar= 1.341553  lambda= 0.0127315 (11 iterations)
Equivalent Degrees of Freedom (Df): 5.883203
Penalized Criterion: 0.7812831
PRESS: 0.000142811
> sp.spline$lambda
[1] 0.0127315
```

(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 blue curves are for large values of λ , and clearly approach the linear regression; the red curves 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)
```

⁴This uses the handy `pdfetch` library, which downloads data from such public domain sources as the Federal Reserve, Yahoo Finance, etc.



```

plot(as.vector(sp.today),as.vector(sp.tomorrow),xlab="Today's log-return",
     ylab="Tomorrow's log-return",pch=16,cex=0.5,col="grey")
abline(lm(sp.tomorrow ~ sp.today),col="darkgrey")
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 (grey dots), with the OLS linear regression (dark grey line), the spline selected by cross-validation (solid black, $\lambda = 0.0127$), some more smoothed splines (blue, $\lambda = 0.178$ and 727) and some less smooth splines (red, $\lambda = 2.88 \times 10^{-4}$ and 1.06×10^{-8}). Inconveniently, `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`.

```
$x
[1] 0.01
$y
[1] 0.0001959963
```

i.e., a very slightly negative log-return.

R Syntax Note: The syntax for predict with `smooth.spline` differs slightly from the syntax for predict with `lm` or `np`. The latter two want a `newdata` argument, which should be a data-frame with column names matching those in the formula used to fit the model. The predict function for `smooth.spline`, though, just wants a vector called `x`. Also, while predict for `lm` or `np` returns a vector of predictions, predict for `smooth.spline` returns a list with an `x` component (in increasing order) and a `y` component, which is the sort of thing that can be put directly into `points` or `lines` for plotting.

[[ATTN: Other spline packages with more uniform interface? Or just use GAM?]]

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 for kernel regression in Chapter 6 and Problem Set 51, problem 5. 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 original⁵. 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) {
```

⁵§28.5 covers more refined ideas about bootstrapping time series.

```

# 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:

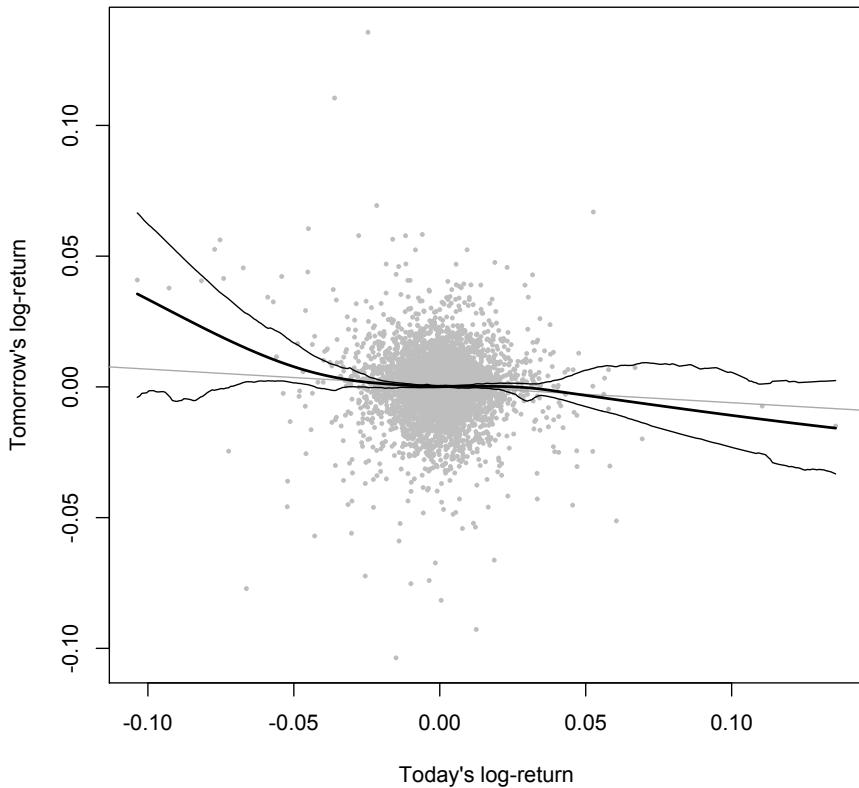
```

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)
  return(list(main.curve=spline.main,lower.ci=cis.lower,upper.ci=cis.upper,
    x=eval.grid))
}

```

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(as.vector(sp.today),as.vector(sp.tomorrow),xlab="Today's log-return",
     ylab="Tomorrow's log-return",pch=16,cex=0.5,col="grey")
abline(lm(sp.tomorrow ~ sp.today),col="darkgrey")
lines(x=sp.cis$x,y=sp.cis$main.curve,lwd=2)
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 independent⁶ 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 :

$$0 = -2\mathbf{B}^T\mathbf{y} + 2\mathbf{B}^T\mathbf{B}\hat{\beta} + 2n\lambda\Omega\hat{\beta} \quad (8.17)$$

$$\mathbf{B}^T\mathbf{y} = (\mathbf{B}^T\mathbf{B} + n\lambda\Omega)\hat{\beta} \quad (8.18)$$

$$\hat{\beta} = (\mathbf{B}^T\mathbf{B} + n\lambda\Omega)^{-1}\mathbf{B}^T\mathbf{y} \quad (8.19)$$

⁶Recall 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 basis⁷, 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

$$\hat{\beta} = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y} \quad (8.22)$$

⁷Or, 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 :

$$\text{tr } \mathbf{h} = \text{tr}(\mathbf{x}(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T) \quad (8.26)$$

$$= \text{tr}(\mathbf{x}^T \mathbf{x}(\mathbf{x}^T \mathbf{x})^{-1}) \quad (8.27)$$

$$= \text{tr} \mathbf{I}_p = p \quad (8.28)$$

[[TODO: Add 2nd way of explaining this]]

since for any matrices \mathbf{a}, \mathbf{b} , $\text{tr}(\mathbf{ab}) = \text{tr}(\mathbf{ba})$, and $\mathbf{x}^T \mathbf{x}$ is a $p \times p$ matrix⁸.

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 .

[[ATTN: Step through the calculations?]]

⁸This 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 symmetrically⁹.

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 do¹⁰. 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 mathematically¹¹, and extend more straightforwardly to multiple variables, and to combinations of discrete and continuous variables.

8.6 Some of the Math Behind Splines

2015 students: This section
definitely optional

Above, I claimed that a solution to the optimization problem Eq. 8.1 exists, and is a continuous, piecewise-cubic polynomial, with continuous first and second deriva-

⁹Generalizations to more than two input variables are conceptually straightforward — just keep adding up more partial derivatives — but the book-keeping gets annoying.

¹⁰In fact, there is a technical sense in which, for large n , splines act like a kernel regression with a specific non-Gaussian kernel, and a bandwidth which varies over the data, being smaller in high-density regions. See Simonoff (1996, §5.6.2), or, for more details, Silverman (1984).

¹¹Most 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.

tives, with pieces at the x_i , and linear outside the range of the x_i . I do not know of any truly elementary way of showing this, but I will sketch here how it's established, if you're interested.

Eq. 8.1 asks us to find the function which minimizes a certain integral. Even the MSE can be brought inside the integral, using Dirac delta functions:

$$\mathcal{L} = \int \left[\lambda(m''(x))^2 + \frac{1}{n} \sum_{i=1}^n (y_i - m(x_i))^2 \delta(x - x_i) \right] dx \quad (8.34)$$

In what follows, without loss of generality, assume that the x_i are ordered, so $x_1 < x_2 < \dots < x_i < x_{i+1} < \dots < x_n$. With some loss of generality but a great gain in simplicity, assume none of the x_i are equal, so we can make those inequalities strict.

The subject which deals with maximizing or minimizing integrals of functions is the calculus of variations¹², and one of its basic tricks is to write the integrand as a function of x , the function, and its derivatives:

$$\mathcal{L} = \int L(x, m, m', m'') dx \quad (8.35)$$

where, in our case,

$$L = \lambda(m''(x))^2 + \frac{1}{n} \sum_{i=1}^n (y_i - m(x_i))^2 \delta(x - x_i) \quad (8.36)$$

This sets us up to use a general theorem of the calculus of variations, to the effect that any function which solves the original "variational" problem must also solve L 's **Euler-Lagrange equation**:

$$\frac{\partial L}{\partial m} - \frac{d}{dx} \frac{\partial L}{\partial m'} + \frac{d^2}{dx^2} \frac{\partial L}{\partial m''} \Big|_{m=\hat{m}} = 0 \quad (8.37)$$

In our case, the Euler-Lagrange equation reads

$$-\frac{2}{n} \sum_{i=1}^n (y_i - \hat{m}(x_i)) \delta(x - x_i) + 2\lambda \frac{d^2}{dx^2} \hat{m}''(x) = 0 \quad (8.38)$$

Remembering that $\hat{m}''(x) = d^2 \hat{m} / dx^2$,

$$\frac{d^4}{dx^4} \hat{m}(x) = \frac{1}{n\lambda} \sum_{i=1}^n (y_i - \hat{m}(x_i)) \delta(x - x_i) \quad (8.39)$$

The right-hand side is zero at any point x other than one of the x_i , so the fourth derivative has to be zero in between the x_i . This in turn means that the function

¹²In addition to its uses in statistics, the calculus of variations also shows up in physics ("what is the path of least action?"), control theory ("what is the cheapest route to the objective?") and stochastic processes ("what is the most probable trajectory?"). Gershenfeld (1999, ch. 4) is a good starting point.

must be piecewise cubic. Now fix an x_i , and pick any two points which bracket it, but are both greater than x_{i-1} and less than x_{i+1} ; call them l and u . Integrate our Euler-Lagrange equation from l to u :

$$\int_l^u \frac{d^4}{dx^4} \hat{m}(x) dx = \int_l^u \frac{1}{n\lambda} \sum_{i=1}^n (y_i - \hat{m}(x_i)) \delta(x - x_i) \quad (8.40)$$

$$\hat{m}'''(u) - \hat{m}'''(l) = \frac{y_i - \hat{m}(x_i)}{n\lambda} \quad (8.41)$$

That is, the third derivative makes a jump when we move across x_i , though (since the fourth derivative is zero), it doesn't matter which pair of points above and below x_i we compare third derivatives at. Integrating the equation again,

$$\hat{m}''(u) - \hat{m}''(l) = (u - l) \frac{y_i - \hat{m}(x_i)}{n\lambda} \quad (8.42)$$

Letting u and l approach x_i from either side, so $u - l \rightarrow 0$, we see that \hat{m}'' makes no jump at x_i . Repeating this trick twice more, we conclude the same about \hat{m}' and \hat{m} itself. In other words, \hat{m} must be continuous, with continuous first and second derivatives, and a third derivative that is constant on each (x_i, x_{i+1}) interval. Since the fourth derivative is zero on those intervals (and undefined at the x_i), the function must be a piecewise cubic, with the piece boundaries at the x_i , and continuity (up to the second derivative) across pieces.

To see that the optimal function must be linear below x_1 and above x_n , suppose that it wasn't. Clearly, though, we could reduce the curvature as much as we want in those regions, without altering the value of the function at the boundary, or even its first derivative there. This would yield a better function, i.e., one with a lower value of \mathcal{L} , since the MSE would be unchanged and the average curvature would be smaller. Taking this to the limit, then, the function must be linear outside the observed data range.

We have now shown¹³ that the optimal function \hat{m} , if it exists, must have all the properties I claimed for it. We have not shown either that there is a solution, or that a solution is unique if it does exist. However, we can use the fact that solutions, if there are any, are piecewise cubics obeying continuity conditions to set up a system of equations to find their coefficients. In fact, we did so already in §8.3.1, where we saw it's a system of n independent linear equations in n unknowns. Such a thing does indeed have a unique solution, here Eq. 8.19.

8.7 Further Reading

There are good discussions of splines in Simonoff (1996, ch. 5), Hastie *et al.* (2009, ch. 5) and Wasserman (2006, §5.5). Wood (2006, ch. 4) includes a thorough practical treatment of splines as a preparation for additive models (see Chapter 9 below) and generalized additive models (see Chapters 12–13). The classic reference, by one of the

[[TODO: Mention alternative of knot selection here?]]

¹³For a very weak value of “shown”, admittedly.

inventors of splines as a useful statistical tool, is Wahba (1990); it's great if you already know what a Hilbert space is and how to navigate one.

Historical notes The first introduction of spline smoothing in the statistical literature seems to be Whittaker (1922). (His “graduation” is more or less our “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.

The general optimization problem, and the use of the word “spline”, seems to have its roots in numerical analysis in the early 1960s; those spline functions were intended as ways of smoothly interpolating between given points. The connection to statistical smoothing was made by Schoenberg (1964) (who knew about Whittaker’s earlier work) and by Reinsch (1967) (who gave code). Splines were then developed as a practical tool in statistics and in applied mathematics in the 1960s and 1970s. Silverman (1985) is a still-readable and insightful summary of this work.

In econometrics, spline smoothing a time series is called 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, as it happens), regardless of the data¹⁴. See Paige and Trindade (2010) for a (polite) discussion, and demonstration of the advantages of cross-validation.

8.8 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. (See §2.1.1.)

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

¹⁴As it were: Hodrick and Prescott re-invented the wheel, and decided that it should be an octagon.

is

$$\hat{\beta}_{OLS} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{x}_i \cdot \beta)^2, \quad (8.43)$$

the regularized or penalized estimate is

$$\hat{\beta}_{RR} = \underset{\beta}{\operatorname{argmin}} \left[\frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{x}_i \cdot \beta)^2 \right] + \lambda \sum_{j=1}^p \beta_j^2 \quad (8.44)$$

- (a) Show that the matrix form of the ridge-regression objective function is

$$n^{-1}(\mathbf{y} - \mathbf{x}\beta)^T(\mathbf{y} - \mathbf{x}\beta) + \lambda\beta^T\beta \quad (8.45)$$

- (b) Show that the optimum is

$$\hat{\beta}_{RR} = (\mathbf{x}^T \mathbf{x} + n\lambda \mathbf{I})^{-1} \mathbf{x}^T \mathbf{y} \quad (8.46)$$

(This is where the name “ridge regression” comes from: we take $\mathbf{x}^T \mathbf{x}$ and add a “ridge” along the diagonal of the matrix.)

- (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.3.5.

[[TODO: Improve connection of this exercise to appendix]]

Chapter 9

Additive Models

9.1 Additive Models

The **additive model** for regression is that the conditional expectation function is a sum of **partial response** functions, one for each predictor variable. Formally, when the vector \vec{X} of predictor variables has p dimensions, x_1, \dots, x_p , the model says that

$$\mathbf{E}[Y|\vec{X} = \vec{x}] = \alpha + \sum_{j=1}^p f_j(x_j) \quad (9.1)$$

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 arbitrary nonlinear functions. The idea is still that each input feature makes a separate contribution to the response, and these just add up (hence “partial response function”), 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 $\mathbf{E}[Y] = \alpha$ and $\mathbf{E}[f_j(X_j)] = 0$.¹

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.

¹To 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 can fix it with this convention.

Of course, none of this would be of any use if we couldn't actually estimate these models, but we can, through a clever computational trick which is worth knowing for its own sake. The use of the trick is also something they share with linear models, so we'll start there.

9.2 Partial Residuals and Back-fitting

9.2.1 Back-fitting for Linear Models

The general form of a linear regression model is

$$\mathbf{E}[Y|\vec{X} = \vec{x}] = \beta_0 + \vec{\beta} \cdot \vec{x} = \sum_{j=0}^p \beta_j x_j \quad (9.2)$$

where 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 ?

$$\mathbf{E}[Y|X_k = x_k] = \mathbf{E}[\mathbf{E}[Y|X_1, X_2, \dots, X_p] | X_k = x_k] \quad (9.3)$$

$$= \mathbf{E}\left[\sum_{j=0}^p \beta_j X_j | X_k = x_k\right] \quad (9.4)$$

$$= \beta_k x_k + \mathbf{E}\left[\sum_{j \neq k} \beta_j X_j | X_k = x_k\right] \quad (9.5)$$

where the first line uses the law of total expectation², and the second line uses Eq. 9.2. Turned around,

$$\beta_k x_k = \mathbf{E}[Y|X_k = x_k] - \mathbf{E}\left[\sum_{j \neq k} \beta_j X_j | X_k = x_k\right] \quad (9.6)$$

$$= \mathbf{E}\left[Y - \left(\sum_{j \neq k} \beta_j X_j\right) | X_k = x_k\right] \quad (9.7)$$

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 $Y^{(k)}$.

$$\beta_k x_k = \mathbf{E}[Y^{(k)} | X_k = x_k] \quad (9.8)$$

²As 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 more variables, provided we then average over those extra variables when we're done.

<p>Given: $n \times (p + 1)$ inputs \mathbf{x} (0^{th} column all 1s) $n \times 1$ responses \mathbf{y} small tolerance $\delta > 0$ center \mathbf{y} and each column of \mathbf{x} $\hat{\beta}_j \leftarrow 0$ 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: $(\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p)$ </p>	
---	--

CODE EXAMPLE 21: *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 ;s; this way the intercept only has to be calculated once, at the end. [[ATTN: Fix horizontal lines]]*

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 ;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 21.

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 obvious³ that this will ever converge, but it (generally) does, and the fixed point on which it converges is the usual least-squares estimate of β .

“One man’s vicious circle is another man’s iterative approximation”

Back-fitting is rarely used to fit linear models these days, because with modern computers and numerical linear algebra it's 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 *linearity*.

³Unless, I suppose, you're Gauss.

9.2.2 Backfitting Additive Models

Defining the partial residuals by analogy with the linear case, as

$$Y^{(k)} = Y - \left(\alpha + \sum_{j \neq k} f_j(x_j) \right) \quad (9.9)$$

a little algebra along the lines of §9.2.1 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 learning 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 22. Once again, while it's not obvious that this converges, it does. 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. §9.4 below illustrates using one of 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 linear regression 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,

$$MSE_{\text{linear}} = \sigma^2 + a_{\text{linear}} + O(n^{-1}) \quad (9.11)$$

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 $k \in 1 : p$ { $y_i^{(k)} = y_i - \sum_{j \neq k} \hat{f}_j(x_{ij})$ $g_k \leftarrow \mathcal{S}(y^{(k)} \sim x_{\cdot k})$ $\hat{g}_k \leftarrow g_k - n^{-1} \sum_{i=1}^n g_k(x_{ik})$ $\hat{f}_k \leftarrow g_k$ } } Return: $(\hat{\alpha}, \hat{f}_1, \dots, \hat{f}_p)$	
--	--

CODE EXAMPLE 22: *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.*

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 nonparametric 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 hand-wave a bit, think of kernel smoothing, 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 the probability that a training point \vec{x}_i falls in the averaging region around \vec{x} gets exponentially smaller as p grows. Turned around, to get the same number of training points per \vec{x} , we need exponentially larger sample sizes. The appearance of the 4s is

⁴See Appendix B if you are not familiar with “big O ” notation.

a little more mysterious, but can be resolved from an error analysis of the kind we did for kernel regression in Chapter 4⁵. This slow rate isn't just a weakness of kernel smoothers, but turns out to be the best any nonparametric estimator can do.

For $p = 1$, the nonparametric 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 makes up for it. But as p grows, the nonparametric rate gets slower and slower, and the fully nonparametric 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. (See Figure 9.1.) 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 nonparametric 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

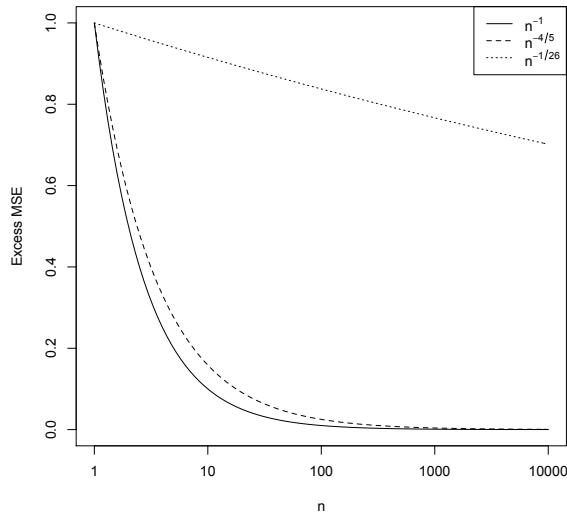
9.4 Example: California House Prices Revisited

As an example, we'll look at data on median house prices across Census tracts from the data-analysis assignment in §32. 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:

⁵Remember that in one dimension, the bias of a kernel smoother with bandwidth h is $O(h^2)$, and the variance is $O(1/nh)$, because only samples falling in an interval about h across contribute to the prediction at any one point, and when h is small, the number of such samples is proportional to nh . Adding bias squared to variance gives an error of $O(h^4) + O(1/nh)$, solving for the best bandwidth gives $h_{\text{opt}} = O(n^{-1/5})$, 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 $O(h^2)$, because the Taylor expansion still goes through. But now only samples falling into a region of volume $O(h^d)$ around x contribute to the prediction at x , so the variance is $O(1/nh^d)$. The best bandwidth is now $h_{\text{opt}} = O(n^{-1/(p+4)})$, 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.



```

curve(x^(-1),from=1,to=1e4,log="x",xlab="n",ylab="Excess MSE")
curve(x^(-4/5),add=TRUE,lty="dashed")
curve(x^(-1/26),add=TRUE,lty="dotted")
legend("topright",legend=c(expression(n^{-1}),
  expression(n^{-4/5}),expression(n^{-1/26})),
  lty=c("solid","dashed","dotted"))

```

FIGURE 9.1: Schematic of rates of convergence of MSEs for parametric models ($O(n^{-1})$), one-dimensional nonparametric regressions or additive models ($O(n^{-4/5})$), and a 100-dimensional nonparametric regression ($O(n^{-1/26})$). Note that the horizontal but not the vertical axis is on a logarithmic scale.

```
housing <- read.csv("http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/data/calif_penn_2011.csv")
housing <- na.omit(housing)
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  
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
```

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 23: *Calculating quick-and-dirty prediction limits from a prediction object (`preds`) containing fitted values and their standard errors, plus an estimate of the noise level. Because those are two (presumably uncorrelated) sources of noise, we combine the standard deviations by “adding in quadrature”.*

```
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 9.2 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% on the original scale), but they do have pretty reasonable coverage; about 96% of actual prices fall within the prediction limits⁸.

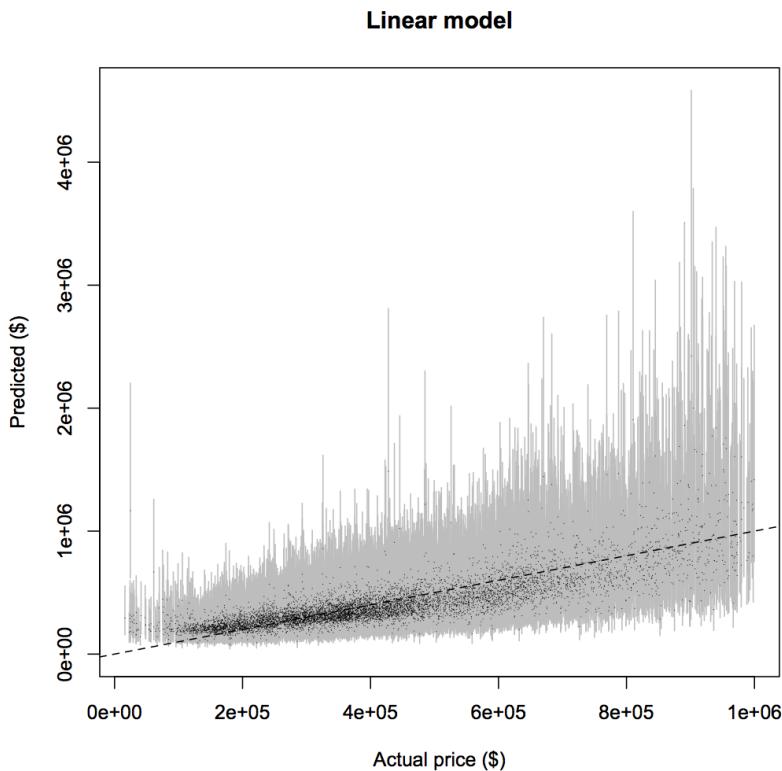
```
sqrt(mean(residuals(calif.lm)^2))
mean((log(calif$Median_house_value) <= predlims.lm[, "upper"])
  & (log(calif$Median_house_value) >= predlims.lm[, "lower"]))
```

On the other hand, the predictions *are* quite precise, with the median of the calculated standard errors being 0.011 on the log scale (i.e., 1.1% in dollars). 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**.

```
require(mgcv)
system.time(calif.gam <- gam(log(Median_house_value)
  ~ s(Median_household_income) + s(Mean_household_income) + s(POPULATION)
  + s(Total_units) + s(Vacant_units) + s(Owners) + s(Median_rooms)
  + s(Mean_household_size_owners) + s(Mean_household_size_renters))
```

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 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 Code Example 23 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 ($)", main="Linear model",
     ylim=c(0,exp(max(predlims.lm))))
segments(calif$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 9.2: *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 are equal. Here predict gives both a fitted value for each point, and a standard error for that prediction. (Without a newdata argument, predict defaults to the data used to estimate calif.lm, which here is what we want.) Predictions are exponentiated so they're comparable to the original values (and because it's easier to grasp dollars than log-dollars).*

```
+ s(LATITUDE) + s(LONGITUDE), data=calif))
  user  system elapsed
 4.652   0.182   5.157
```

(That is, it took about five seconds 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.⁹) The smoothing here is done by splines (hence `s()`), and there are lots of options for controlling the splines, or replacing them by other smoothers, if you know what you're doing.

Figure 9.3 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 9.4 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 together¹⁰:

```
calif.gam2 <- gam(log(Median_house_value)
  ~ s(Median_household_income) + s(Mean_household_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.

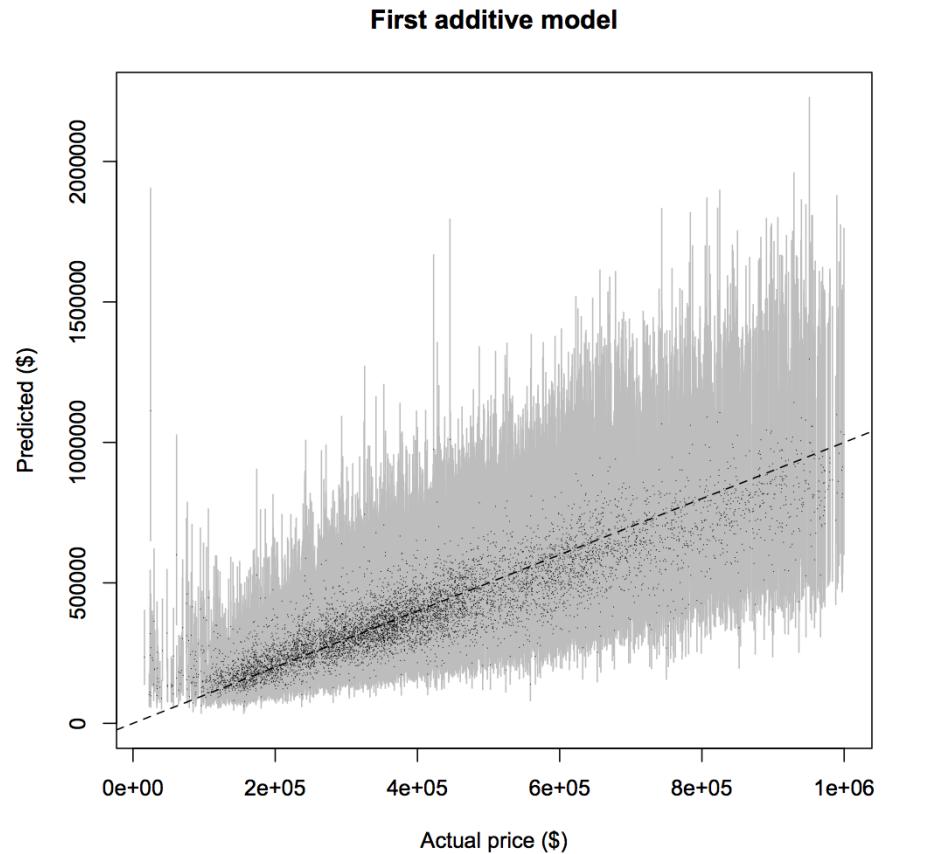
```
preds.gam2 <- predict(calif.gam2, se.fit=TRUE)
predlims.gam2 <- predlims(preds.gam2, sigma=sqrt(calif.gam2$sig2))
mean((log(calif$Median_house_value) <= predlims.gam2[, "upper"]) &
  (log(calif$Median_house_value) >= predlims.gam2[, "lower"]))
```

Figures 9.6 and 9.7 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.

If you worked through §32, you will recall that one of the big things wrong with the linear model is that its errors (the residuals) are highly structured and very far

⁹Alternately, we could use Kevin Gilbert's `formulaTools` functions — see <https://gist.github.com/kgilbert-cmu>.

¹⁰If 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.



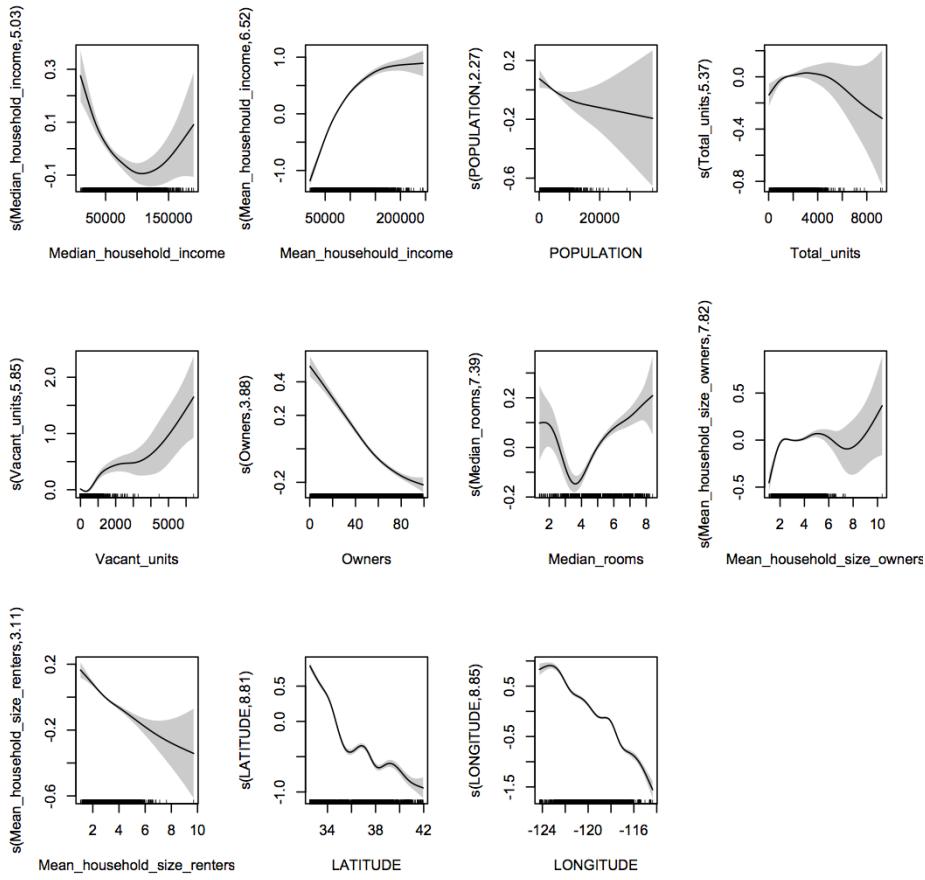
```

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 ($)", main="First additive model",
     ylim=c(0,exp(max(predlims.gam))))
segments(calif$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 9.3: *Actual versus predicted prices for the additive model, as in Figure 9.2. Note that the sig2 attribute of a model returned by gam() is the estimate of the noise variance around the regression surface (σ^2).*

9.4. EXAMPLE: CALIFORNIA HOUSE PRICES REVISITED



```
plot(calif.gam, scale=0, se=2, shade=TRUE, pages=1)
```

FIGURE 9.4: 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.

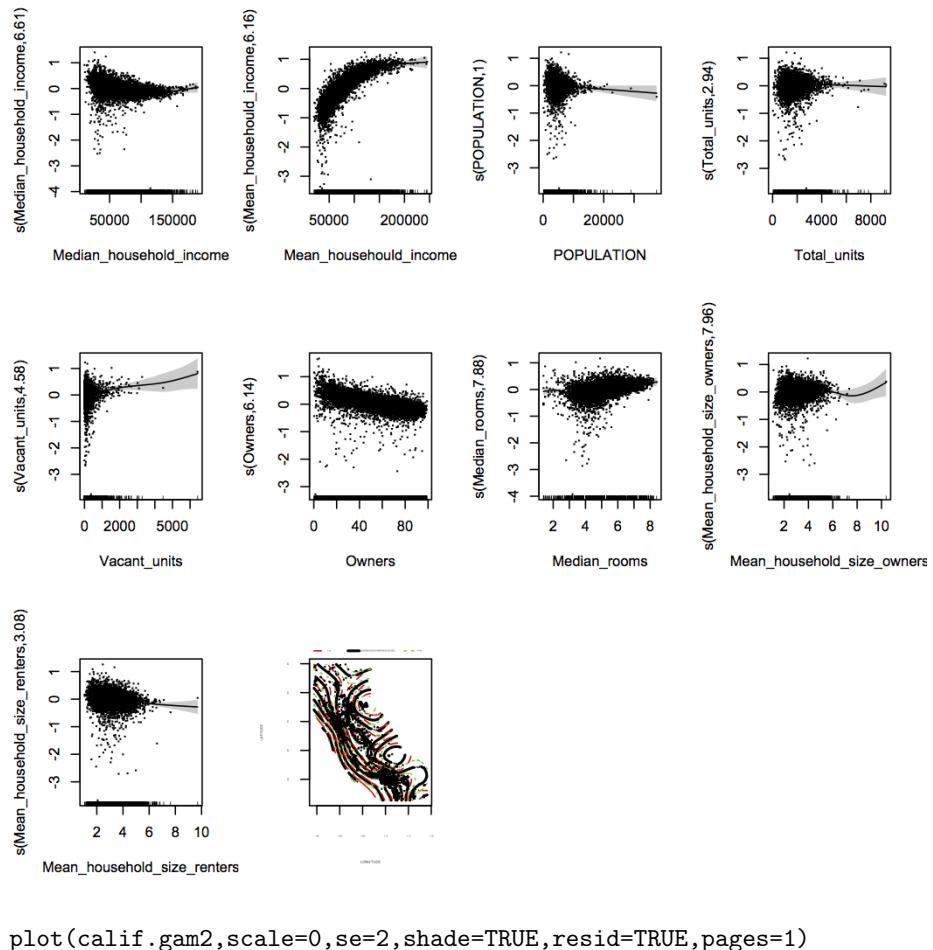
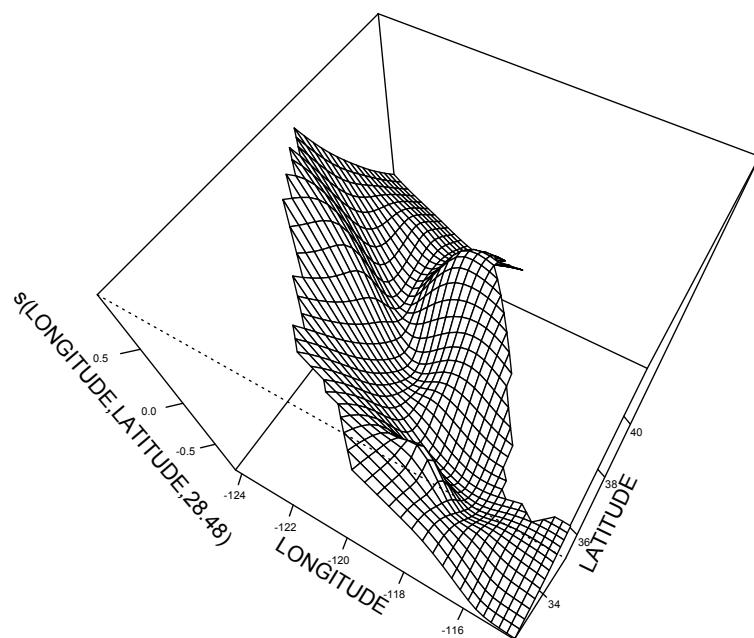
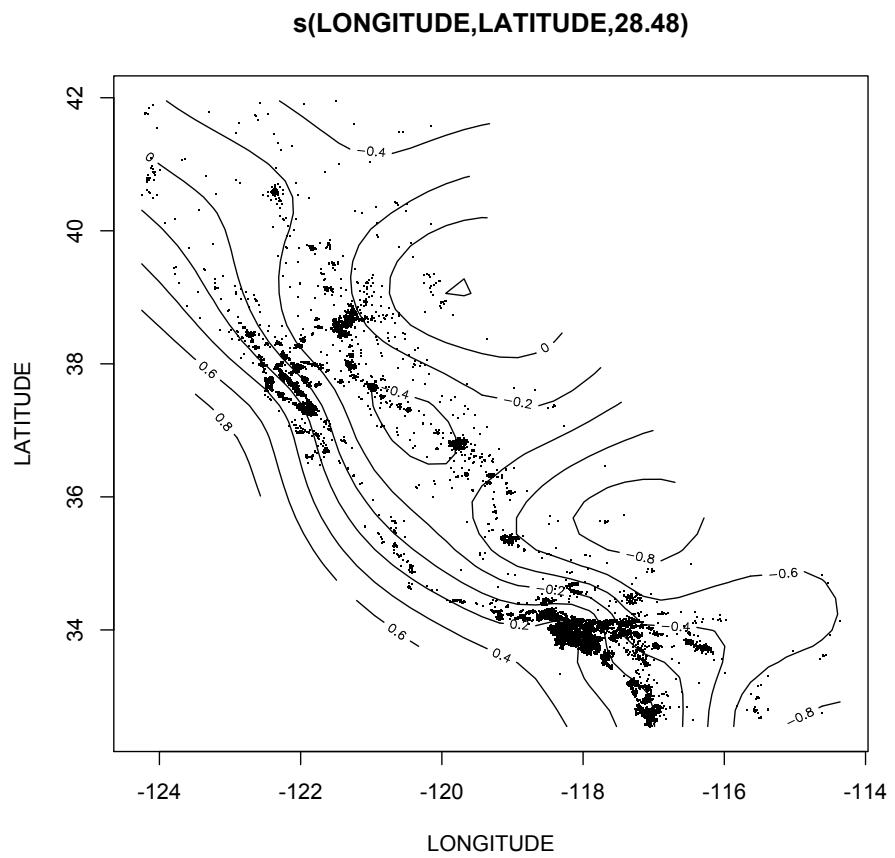


FIGURE 9.5: Partial response functions and partial residuals for addfit2, as in Figure 9.4. See subsequent figures for the joint smoothing of longitude and latitude, which here is an illegible mess. See `help(plot.gam)` for the plotting options used here.



```
plot(calif.gam2,select=10,phi=60,pers=TRUE,ticktype="detailed",cex.axis=0.5)
```

FIGURE 9.6: *The result of the joint smoothing of longitude and latitude.*



```
plot(calif.gam2,select=10,se=FALSE)
```

FIGURE 9.7: 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)
}

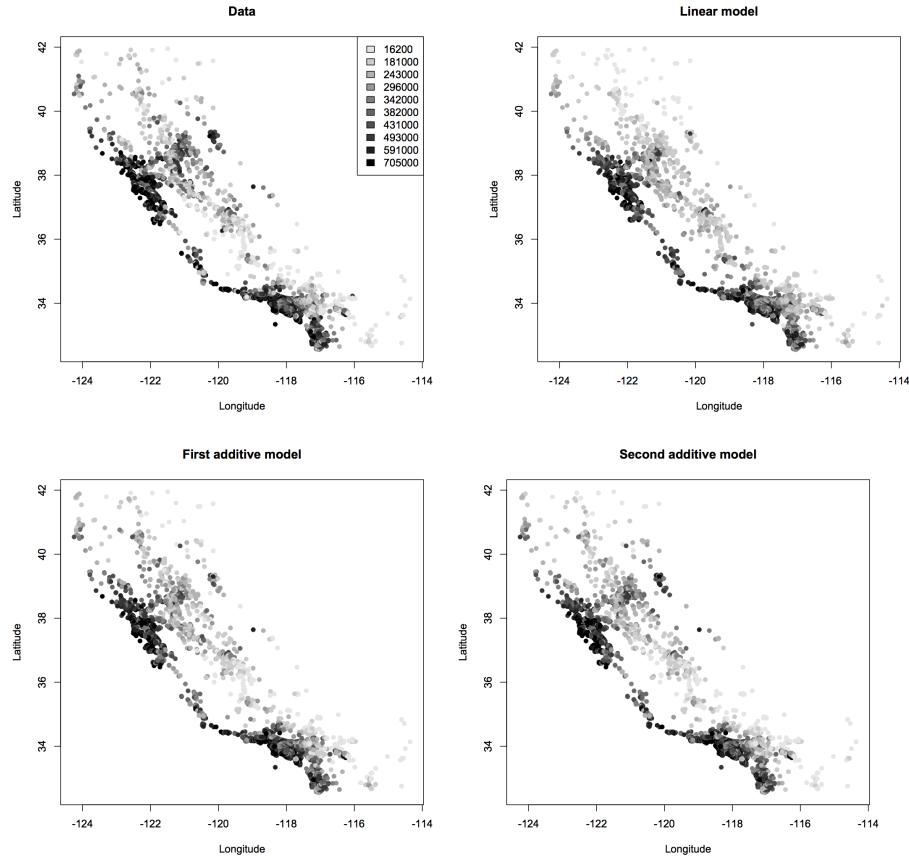
```

CODE EXAMPLE 24: *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.*

from random. In essence, it totally missed the existence of cities, and the fact that houses cost more in cities (because land costs more there). 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 24).

Figures 9.8 and 9.9 show that allowing for the interaction of latitude and longitude (the smoothing term plotted in Figures 9.6–9.7) 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 **LONGITUDE** \times **LATITUDE** term in a linear model.

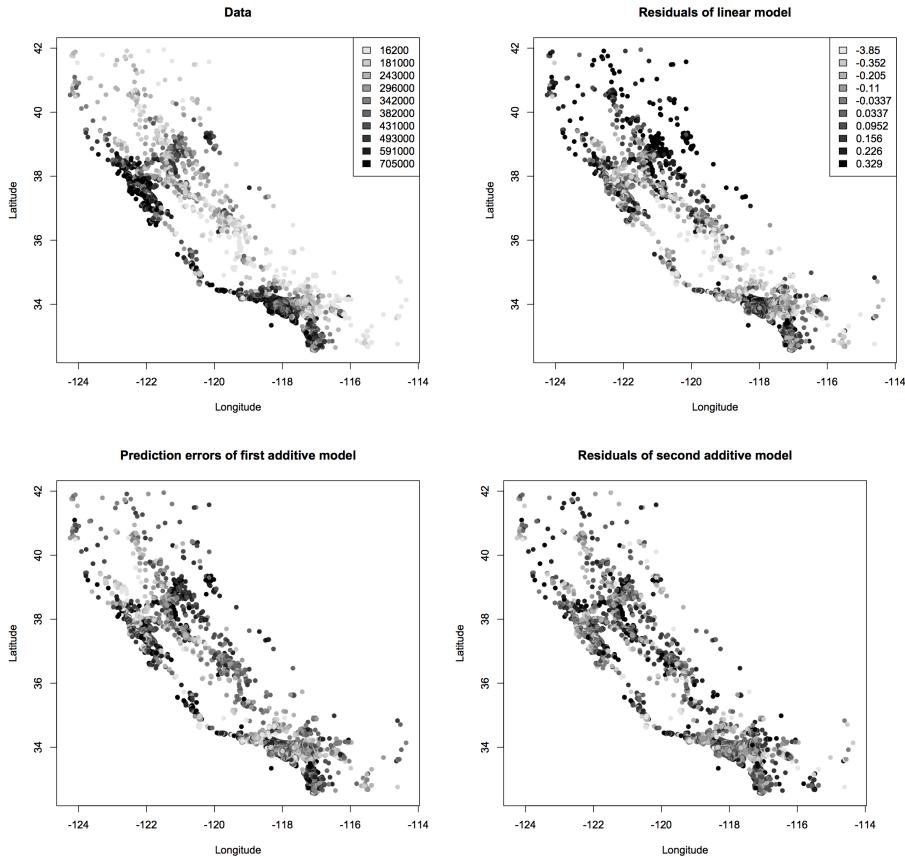
Including an interaction between latitude and longitude in a spatial problem is pretty obvious. There are other potential interactions which might be important



```
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 9.8: 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.

9.4. EXAMPLE: CALIFORNIA HOUSE PRICES REVISITED



```

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 9.9: 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?

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 (§1.5.1) or regression trees (Ch. 14), 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 justifications 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.

Historical notes 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)¹¹. 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.

The Gauss-Seidel or backfitting algorithm was invented by Gauss in the early 1800s during his work on least squares estimation in linear models; he mentioned it in letters to students, but never published it. (Apparently he described it as something one could do “while half asleep”.) Seidel gave the first published version in 1874. (See <https://www.siam.org/meetings/la09/talks/benzi.pdf>.) I am not sure when the connection was made between additive statistical models and back-fitting.

9.7 Exercises

[[TODO: write some]]

¹¹“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

$$Y = f(X; \theta) + \epsilon \tag{10.1}$$

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

$$Y = \theta_1 x_1 + \theta_2 x_2 + \epsilon \tag{10.2}$$

to the model

$$Y = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \epsilon \tag{10.3}$$

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

$$E[Y - f(x; \theta)|X] = E[f(x; \theta) + \epsilon - f(x; \theta)|X] = E[\epsilon|X] = 0 \quad (10.4)$$

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 $\tilde{T} = MSE_p(\tilde{\theta}) - MSE_{np}(\tilde{\theta})$.
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 $g(x) = \log(1 + x)$, 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 examples from §4.2.) 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",pch=16,cex=0.5)
curve(log(1+x),col="grey",add=TRUE,lwd=4)
```

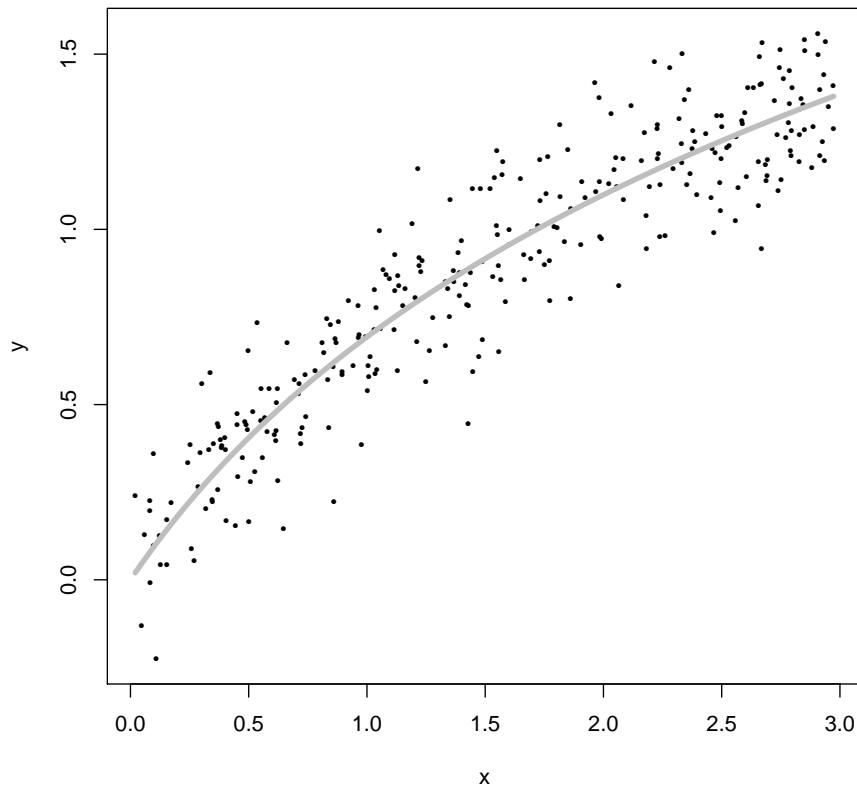


FIGURE 10.1: True regression curve (grey) and data points (circles). The curve $g(x) = \log(1 + x)$.

```

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
## -0.501 -0.103  0.005  0.088  0.439
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.230     0.018     13 <2e-16
## x           0.416     0.010     41 <2e-16
##
## Residual standard error: 0.15 on 298 degrees of freedom
## Multiple R-squared:  0.85, Adjusted R-squared:  0.85
## F-statistic: 1.6e+03 on 1 and 298 DF, p-value: <2e-16

```

R^2 is ridiculously high — the regression line preserves 84.6683 percent 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 fit is⁴

```

mean(residuals(glinfit)^2)
## [1] 0.02379

```

The nonparametric regression has a somewhat smaller MSE⁵

```

library(np)

## Nonparametric Kernel Methods for Mixed Datatypes (version 0.60-2)
## [vignette("np_faq",package="np") provides answers to frequently asked questions]

gnpr <- npreg(y~x,data=gframe)
gnpr$MSE

```

So \hat{t} is

```

(t.hat = mean(glinfit$residual^2) - gnpr$MSE)
## [1] 0.004468

```

⁴If we ask R for the MSE, by doing `summary(glinfit)$sigma2`, we get 0.024. This differs from the mean of the squared residuals by a factor of 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",pch=16,cex=0.5)
curve(log(1+x),col="grey",add=TRUE,lwd=4)
abline(glinfit,lwd=4)
```

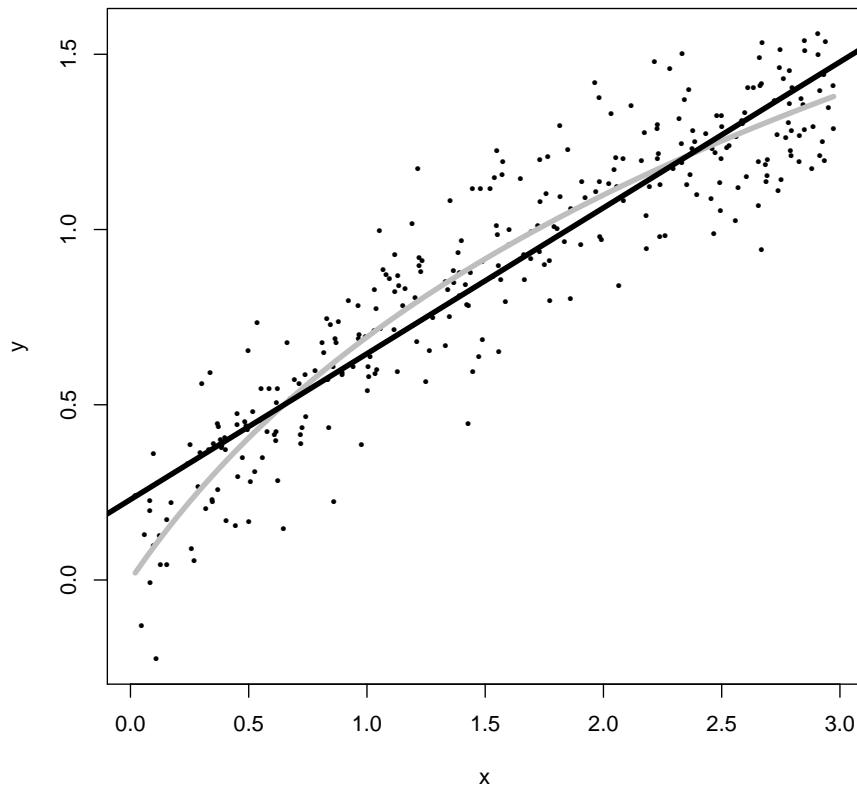


FIGURE 10.2: As previous figure, but adding the least-squares regression line (black). Line widths exaggerated for clarity.

```

# 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 25: 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?

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 25 does this, along the same lines as the examples in 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 26. 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))
```

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

```

# 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 26: Calculate the difference-in-MSEs test statistic.

the specification is vague, like “ x is IID” but with no particular distribution, then resample X .)

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, exactly 0 of the simulated values of the test statistic were that big:

```
sum(null.samples.T > t.hat)
## [1] 0
```

Thus our estimated p -value is ≤ 0.005 . We can 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 $\hat{t} = 9.8 \times 10^{-5}$, together with a slightly different null distribution (Figure 10.5). Now the p -value is 0.88, which it would be quite rash to reject.

10.1.2 Remarks

Other Nonparametric Regressions There is nothing especially magical about using kernel regression here. Any consistent nonparametric estimator (say, your fa-

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)
```

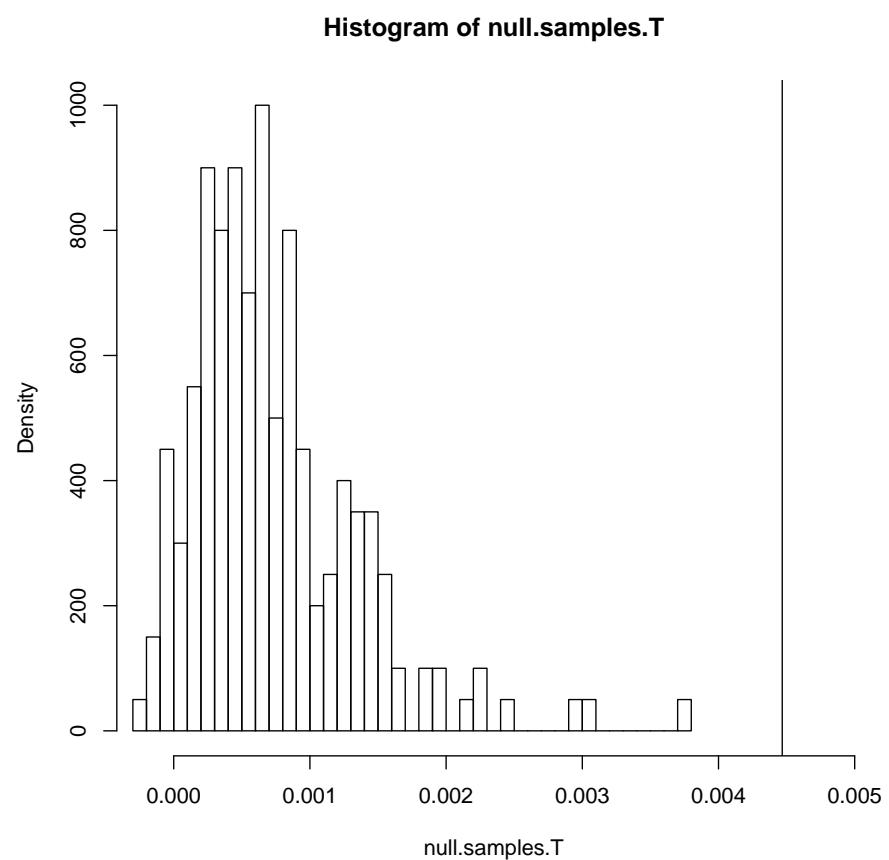


FIGURE 10.3: Histogram of the distribution of $T = MSE_p - MSE_{np}$ for data simulated from the parametric model. The vertical line mark the observed value. Notice that the mode is positive and the distribution is right-skewed; this is typical.

```
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",lwd=2)
```

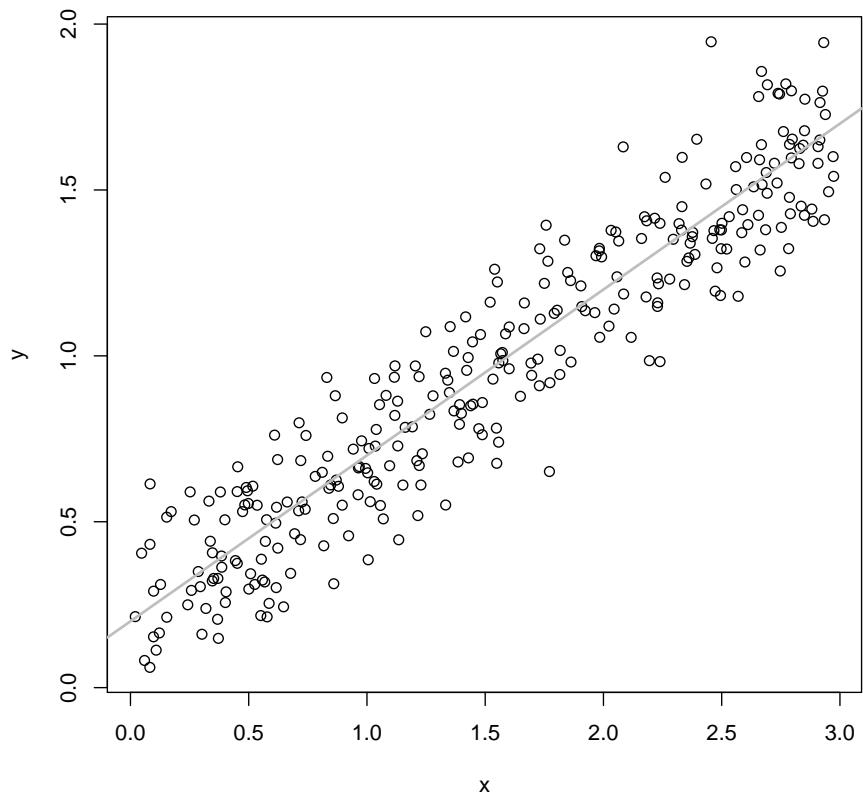


FIGURE 10.4: Data from the linear model (true regression line in grey).

```
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)
```

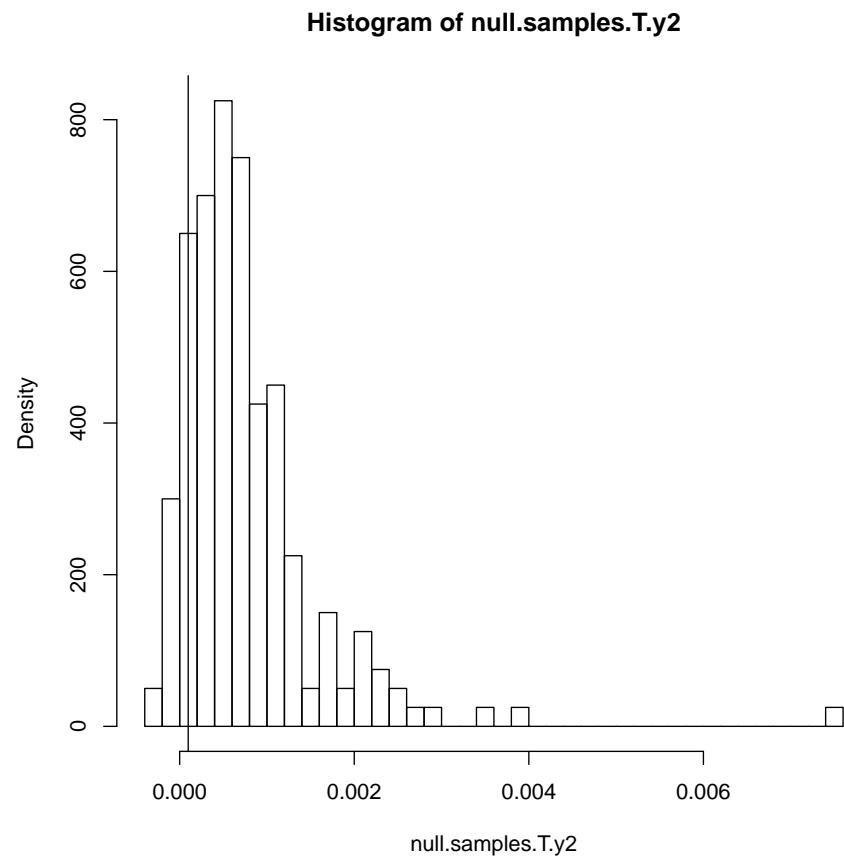


FIGURE 10.5: As in Figure 10.3, but using the data and fits from Figure 10.4.

vorite spline) would work. They may differ somewhat in their answers on particular cases.

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.

Testing $E[\epsilon|X] = 0$ 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.

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 (§8.3.2) of the nonparametric regression.
- Use the log of the ratio in 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.

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,

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.

```

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 27: *Evaluating the out-of-sample error for the nearly-linear problem as a function of n, and evaluating the generalization error by averaging over many samples.*

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

$$\mathbb{E}[Y|X=x] = 0.2 + \frac{1}{2} \left(1 + \frac{\sin x}{10}\right)x \quad (10.6)$$

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 27).

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)
```

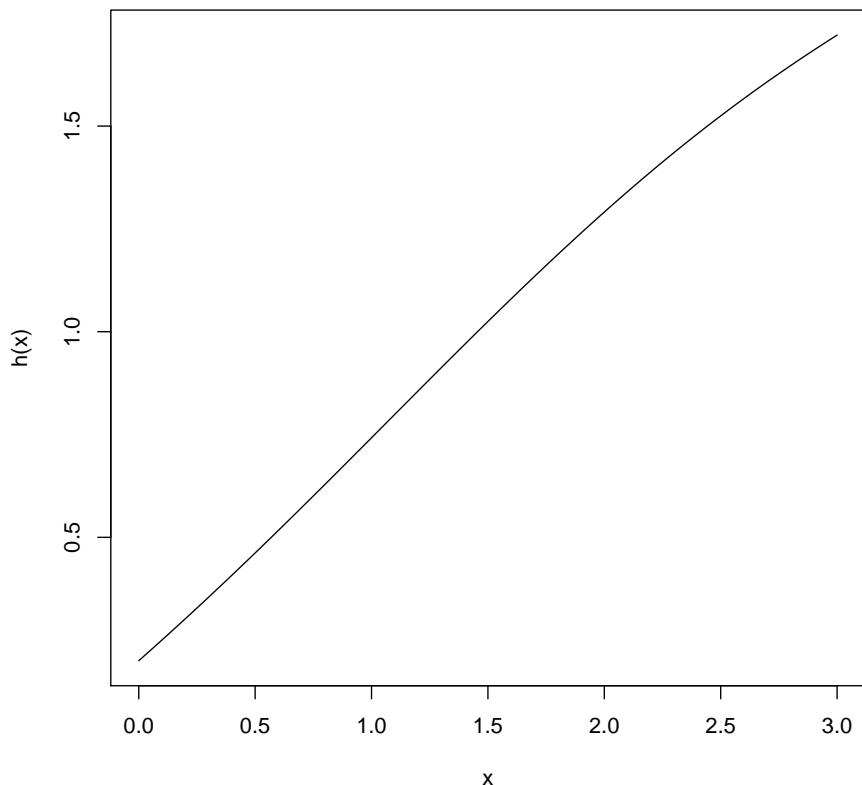


FIGURE 10.6: Graph of $h(x) = 0.2 + \frac{1}{2} \left(1 + \frac{\sin x}{10}\right)x$ over $[0, 3]$.

```

sizes = c(5,10,15,20,25,30,50,100,200,500)
generalizations = sapply(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="dashed")
abline(h=0.15,col="grey")
    
```

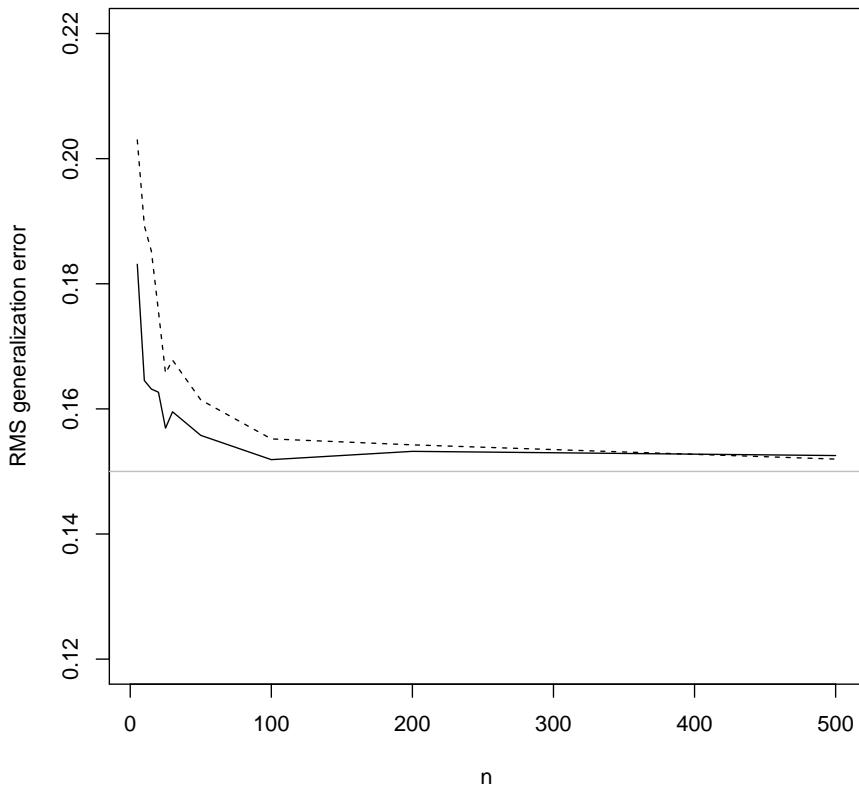


FIGURE 10.7: Root-mean-square generalization error for linear model (solid line) and kernel smoother (dashed line), fit to the same sample of the indicated size. The true regression curve is as in 10.6, and observations are corrupted by IID Gaussian noise with $\sigma = 0.15$ (grey horizontal line). The cross-over after which the nonparametric regressor has better generalization performance happens shortly before $n = 500$.

10.4 Further Reading

In this chapter, we've been looking at specification testing for regression models, and specifically focusing on whether they are correctly specified for the conditional expectation function. I am not aware of any other treatment of this topic at this level, other than the not-wholly-independent Spain *et al.* (2012). If you have somewhat more statistical theory than this book demands, there are very good treatments of related tests in Li and Racine (2007), and of tests based on smoothing residuals in Hart (1997).

Econometrics seems to have more of a tradition of formal specification testing than many other branches of statistics. Godfrey (1988) reviews tests based on looking for parametric extensions of the model, i.e., refinements of the idea of testing whether $\theta_3 = 0$ in Eq. 10.3. White (1994) combines a detailed theory of specification testing within parametric stochastic models, *not* presuming any particular parametric model is correct, with an analysis of when we can and cannot still draw useful inferences from estimates within a mis-specified model. Because of its generality, it, too, is at a higher theoretical level than this book, but is strongly recommend. White was also the co-author of a paper (Hong and White, 1995) presenting a theoretical analysis of the difference-in-MSEs test used in this chapter, albeit for a particular sort of nonparametric regression we've not really touched on.

We will return to specification testing in Chapters 15 and 17, but for models of distributions, rather than regressions.

10.5 Exercises

[[To come]]

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 the sampling distribution changing as the truth changes. Size-power trade-off; ROC curves. Why the likelihood *ratio*, i.e., “economic” interpretation of the Neyman-Pearson lemma. Lack of uniformly most powerful tests in most situations. Ways of increasing power or easing the size-power trade-off: more efficient test statistics (i.e., better filters); increasing (effective) sample size; increasing precision of measurement; focusing power against some alternatives rather than others. Role of modeling assumptions in controlling power. Substantive vs. statistical significance once more. Multiple testing: Bonferroni correction, less conservative ones; false discovery control as an alternative. Higher criticism and related procedures. 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, 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 should be. 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 16, 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 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

$$p(x; \beta) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_0 + x \cdot \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \cdot \beta)}} \quad (12.5)$$

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 $p < 0.5$ (Exercise 1). 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 thermodynamics or physical chemistry, in which case you recognize that this is the Boltzmann distribution for a system with two states, which differ in energy by $\beta_0 + x \cdot \beta$.

```

x <- matrix(runif(n=50*2,min=-1,max=1),ncol=2)
par(mfrow=c(2,2))
plot.logistic.sim(x,beta.0=-0.1,beta=c(-0.2,0.2))
y.1 <- plot.logistic.sim(x,beta.0=-0.5,beta=c(-1,1))
plot.logistic.sim(x,beta.0=-2.5,beta=c(-5,5))
plot.logistic.sim(x,beta.0=-2.5e2,beta=c(-5e2,5e2))

```

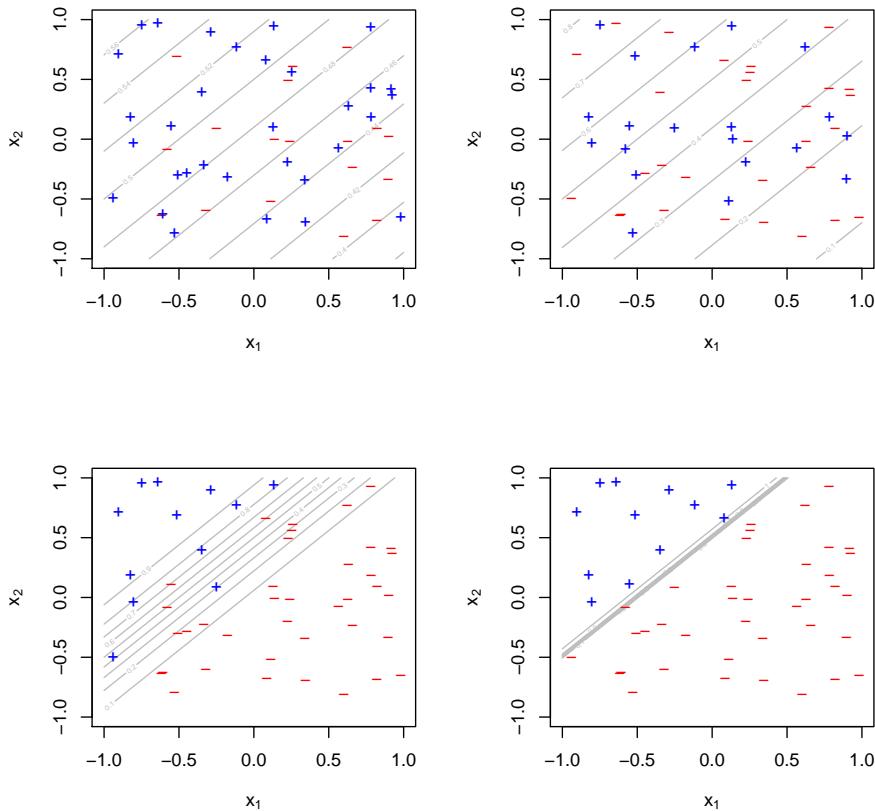


FIGURE 12.1: Effects of scaling logistic regression parameters. Values of x_1 and x_2 are the same in all plots ($\sim \text{Unif}(-1, 1)$ for both coordinates), but labels were generated randomly from logistic regressions with $\beta_0 = -0.1$, $\beta = (-0.2, 0.2)$ (top left); from $\beta_0 = -0.5$, $\beta = (-1, 1)$ (top right); from $\beta_0 = -2.5$, $\beta = (-5, 5)$ (bottom left); and from $\beta_0 = 2.5 \times 10^2$, $\beta = (-5 \times 10^2, 5 \times 10^2)$. Notice how as the parameters get increased in constant ratio to each other, we approach a deterministic relation between Y and x , with a linear boundary between the classes. (We save one set of the random binary responses for use later, as the imaginatively-named $y.1$.)

```

sim.logistic <- function(x, beta.0,beta,bind=FALSE) {
  require(faraway) # For accessible logit and inverse-logit functions
  linear.parts <- beta.0+(x%*%beta)
  y <- rbinom(nrow(x),size=1,prob=ilogit(linear.parts))
  if (bind) { return(cbind(x,y)) } else { return(y) }
}

plot.logistic.sim <- function(x, beta.0, beta, n.grid=50,
                                labcex=0.3, col="grey", ...) {
  grid.seq <- seq(from=-1,to=1,length.out=n.grid)
  plot.grid <- as.matrix(expand.grid(grid.seq,grid.seq))
  require(faraway)
  p <- matrix(ilogit(beta.0 + (plot.grid %*% beta )),nrow=n.grid)
  contour(x=grid.seq,y=grid.seq,z=p, xlab=expression(x[1]),
          ylab=expression(x[2]),main="",labcex=labcex,col=col)
  y <- sim.logistic(x,beta.0,beta,bind=FALSE)
  points(x[,1],x[,2],pch=ifelse(y==1,"+","-"),col=ifelse(y==1,"blue","red"))
  invisible(y)
}

```

CODE EXAMPLE 28: *Code to simulate binary responses from a logistic regression model, and to plot a 2D logistic regression's probability contours and simulated binary values. (How would you modify this to take the responses from a data frame?)*

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.

12.3 Numerical Optimization of the Likelihood

While our likelihood isn't nice enough that we have an explicit expression for the maximum (the way we do in OLS or WLS), it is a pretty well-behaved function,

and one which is amenable to lots of the usual numerical methods for optimization (see Appendix F). In particular, like most log-likelihood functions, it's suitable for an application of Newton's method. Briefly (see Appendix F.1.2 for details), Newton's method starts with an initial guess about the optimal parameters, and then calculates the gradient of the log-likelihood with respect to those parameters. It then adds an amount proportional to the gradient to the parameters, moving up the surface of the log-likelihood function. The size of the step in the gradient direction is dictated by the second derivatives — it takes bigger steps when the second derivatives are small (so the gradient is a good guide to what the function looks like), and small steps when the curvature is large.

12.3.1 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 like a good old fashioned statistics problem.

Logistic regression, after all, is a linear model for a transformation of the probability. Let's call this transformation g :

$$g(p) \equiv \log \frac{p}{1-p} \quad (12.13)$$

So the model is

$$g(p) = \beta_0 + x \cdot \beta \quad (12.14)$$

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 change 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 $p(x)$ or just p , and let's abbreviate that variance as $V(p)$.

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.

$$g(y) \approx g(p) + (y - p)g'(p) \equiv z \quad (12.15)$$

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 $g(p)$ in the definition of z ,

$$z = \beta_0 + x \cdot \beta + (y - p)g'(p) \quad (12.16)$$

and notice that, if we've got the coefficients right, $E[Y|X=x] = p$, so $(y - p)$ 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. (This is our excuse for throwing away the rest of the Taylor expansion, even though we know the discarded terms are infinitely large!) That noise doesn't have constant variance, but we can work it out,

$$\text{Var}[Z|X=x] = \text{Var}[(Y - p)g'(p)|X=x] = (g'(p))^2 V(p), \quad (12.17)$$

and so use that variance in weighted least squares to recover β .

Notice that z and the weights both involve the parameters of our logistic regression, through $p(x)$. 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. This loop — start with a guess about the parameters, use it to calculate the z_i and their weights, regress on the x_i to get new parameters, and repeat — is known as **iterative reweighted least squares** (IRLS or IRWLS), **iterative weighted least squares** (IWLS), etc.

The treatment above is rather heuristic², 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. This takes a reasonable amount of algebra to show, so we'll skip it (but see Exercise 3)³. 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 and 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.⁴ 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 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.

²That is, mathematically incorrect.

³The two key points are as follows. First, the gradient of the log-likelihood turns out to be the sum of the $z_i x_i$. (Cf. Eq. 12.12.) Second, 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. But gradient divided by second derivative is the increment we use in Newton's method, QED.

⁴My experience is that these tend to give similar error rates as classifiers, but have rather different guesses about the underlying probabilities.

Perfect Classification 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. For now, the basic idea is that the iteratively re-weighted least squares procedure of §12.3.1 doesn't really require the model for the log odds to be linear. We get a GAM when we fit an additive model to the z_i ; we could even fit an arbitrary non-parametric model, like a kernel regression, though that's not very common.

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. This is illustrated in §12.6 below.

12.5 Model Checking

The validity of the logistic regression model is no more a fact of mathematics or nature than is the validity of the linear regression model. Both are sometimes convenient assumptions, but neither is guaranteed to be correct, nor even some sort of generally-correct default. In either case, if we want to use the model, the proper scientific (and statistical) procedure is to *check* the validity of the modeling assumptions.

12.5.1 Residuals

In your linear models course, you learned a lot of checks based on the residuals of the model (see Chapter 2). Many of these ideas translates to logistic regression, but we need to re-define residuals. Sometimes people work with the “response” residuals,

$$y_i - p(x_i) \tag{12.18}$$

which should have mean zero (why?), but are heteroskedastic even when the model is true (why?). Others work with standardized or **Pearson** residuals,

$$\frac{y_i - p(x_i)}{\sqrt{V(p(x_i))}} \quad (12.19)$$

and there are yet other notions of residuals for logistic models. Still, both the response and the Pearson residuals should be unpredictable from the covariates, and the latter should have constant variance.

12.5.2 Non-parametric Alternatives

Chapter 10 discussed how non-parametric regression models can be used to check whether parametric regressions are well-specified. The same ideas apply to logistic regressions, with the minor modification that in place of the difference in MSEs, one should use the difference in log-likelihoods, or (what comes to the same thing, up to a factor of 2) the difference in deviances. The use of generalized additive models (§12.4.1) as the alternative model class is illustrated in §12.6 below.

12.5.3 Calibration

Because logistic regression predicts actual *probabilities*, we can check its predictions in a more stringent way than an ordinary regression, which just tells us the mean value of Y , but is otherwise silent about its distribution. If we've got a model which tells us that the probability of rain on a certain class of days is 50%, it had better rain on half of those days, or there model is just *wrong* about the probability of rain. More generally, we'll say that the model is **calibrated** (or **well-calibrated**) when

$$\Pr(Y = 1 | \hat{p}(X) = p) = p \quad (12.20)$$

That is, the actual probabilities should match the predicted probabilities. If we have a large sample, by the law of large numbers, observed relative frequencies will converge on true probabilities. Thus, the observed relative frequencies should be close to the predicted probabilities, or else the model is making systematic mistakes.

In practice, each case may have its own unique predicted probability p , so it might not be possible to accumulate many cases with the same p and check the relative frequency among those cases. When that happens, one option is to look at all the cases where the predicted probability is in some small range $[p, p + \epsilon]$; the observed relative frequency had them better be in that range too. §12.7 below illustrates some of the relevant calculations.

A second option is to use what is called a **proper scoring rule**, which is a function of the outcome variables and the predicted probabilities that attains its minimum when, and only when, the predicted are calibrated. For binary outcomes, one proper scoring rule (historically the oldest) is the **Brier score**,

$$n^{-1} \sum_{i=1}^n (y_i - p_i)^2 \quad (12.21)$$

Another however is simply the (normalized) negative log-likelihood,

$$-\frac{1}{n} \sum_{i=1}^n y_i \log p_i + (1 - y_i) \log 1 - p_i \quad (12.22)$$

Of course, proper scoring rules are better evaluated out-of-sample, or, failing that, through cross-validation, than in-sample. Even an in-sample evaluation is better than nothing, however, which is too often what happens.

12.6 A Toy Example

Here's a worked R example, using the data from the upper right panel of Figure 12.1. The 50×2 matrix \mathbf{x} holds the input variables (the coordinates are independently and uniformly distributed on $[-1, 1]$), and y the corresponding class labels, themselves generated from a logistic regression with $\beta_0 = -0.5$, $\beta = (-1, 1)$.

```
df <- data.frame(y=y.1, x1=x[, 1], x2=x[, 2])
(logr <- glm(y ~ x1 + x2, data=df, family="binomial"))
##
## Call: glm(formula = y ~ x1 + x2, family = "binomial", data = df)
##
## Coefficients:
## (Intercept)          x1          x2
##       -0.397       -0.609        0.378
##
## Degrees of Freedom: 49 Total (i.e. Null); 47 Residual
## Null Deviance: 67.3
## Residual Deviance: 65.3  AIC: 71.3
```

[[TODO: Define deviance earlier]]

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.⁵ Here the saturated model would assign probability 1 to the observed outcomes⁶, 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 β_0 and setting the rest of β to 0. The fitted model definitely improves on that.⁷

⁵The 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. See Appendix G for the connection between log likelihood ratios and χ^2 distributions.

⁶This is not possible when there are multiple observations with the same input features, but different classes.

⁷AIC is of course the Akaike information criterion, $-2\ell + 2p$, with p being the number of parameters (here, $p = 3$). (Some people divide this through by n .) 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; see especially §2.9 for the connection between AIC and leave-one-out. [[TODO: AIC appendix]]

If we're interested in inferential statistics on the estimated model, we can see those with `summary`, as with `lm`:

```
summary(logr,digits=2,signif.stars=FALSE)
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial", data = df)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.361  -1.027  -0.838   1.225   1.652
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.397    0.298   -1.33   0.18
## x1          -0.609    0.511   -1.19   0.23
## x2           0.378    0.556    0.68   0.50
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 67.301 on 49 degrees of freedom
## Residual deviance: 65.348 on 47 degrees of freedom
## AIC: 71.35
##
## Number of Fisher Scoring iterations: 4
```

The fitted values of the logistic regression are the class probabilities; this next line gives us the (in-sample) mis-classification rate.

```
mean(ifelse(fitted(logr)<0.5,0,1) != df$y)
## [1] 0.38
```

An error rate of 38 % may sound bad, but notice from the contour lines in Figure 12.1 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. We'll use the same package for estimating the GAM, `mgcv`, that we used to fit the additive models in Chapter 9.

```
library(mgcv)

## Loading required package: nlme
## This is mgcv 1.7-28. For overview type 'help("mgcv-package")'.

(gam.1 <- gam(y~s(x1)+s(x2),data=df,family="binomial"))
##
## Family: binomial
## Link function: logit
##
```

```

# Simulate a fitted logistic regression and return a new data frame
# Inputs: data frame (df), fitted model (mdl)
# Outputs: new data frame
# Presumes: df contains columns with names for the covariates of mdl
simulate.from.logr <- function(df, mdl) {
  probs <- predict(mdl,newdata=df,type="response")
  df$y <- rbinom(n=nrow(x),size=1,prob=probs)
  return(df)
}

```

CODE EXAMPLE 29: *Code for simulating from an estimated logistic regression model. By default (type="link"), predict for logistic regressions returns predictions for the log odds; changing the type to "response" returns a probability.*

```

## Formula:
## y ~ s(x1) + s(x2)
##
## Estimated degrees of freedom:
## 1.00 6.67 total = 8.67
##
## UBRE score: 0.315

```

This fits a GAM to the same data, using spline smoothing of both input variables. (Figure 12.2 shows the partial response functions.) The (in-sample) deviance is

```

signif(gam.1$deviance,3)
## [1] 48.4

```

which is lower than the logistic regression, so the GAM gives the data higher likelihood. 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.

Now we simulate from our fitted model, and re-fit both the logistic regression and the GAM.

```

# Simulate from an estimated logistic model, and refit both the logistic
# regression and a generalized additive model
# Hard-codes the formula; better code would be more flexible
# Inputs: data frame with covariates (df), fitted logistic model (mdl)
# Output: difference in deviances
# Presumes: df has columns names x.1 and x.2.
delta.deviance.sim <- function (df,mdl) {
  sim.df <- simulate.from.logr(df,mdl)
  GLM.dev <- glm(y~x1+x2,data=sim.df,family="binomial")$deviance
  GAM.dev <- gam(y~s(x1)+s(x2),data=sim.df,family="binomial")$deviance
  return(GLM.dev - GAM.dev)
}

```

```
plot(gam.1,residuals=TRUE,pages=0)
```

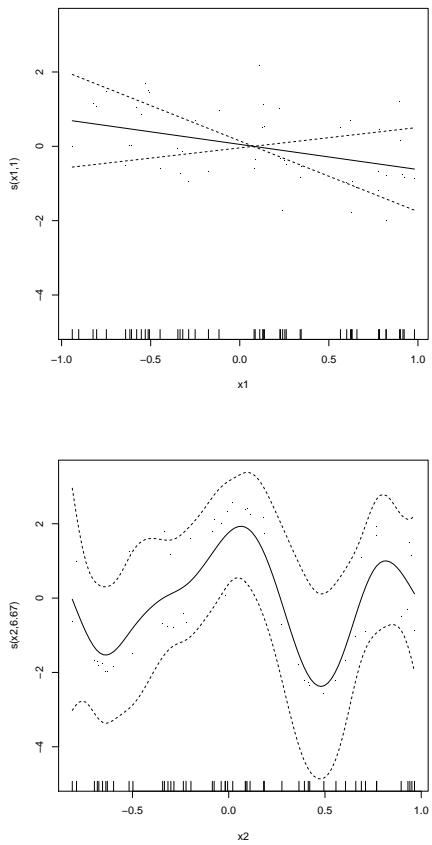


FIGURE 12.2: Partial response functions estimated when we fit a GAM to the data simulated from a logistic regression. Notice that the vertical axes are on the logit scale.

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.observed <- logr$deviance - gam.1$deviance)
## [1] 16.94
delta.dev <- replicate(100,delta.deviance.sim(df,logr))
mean(delta.dev.observed > delta.dev)
## [1] 0.87
```

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.

```
hist(delta.dev, main="",
      xlab="Amount by which GAM fits better than logistic regression")
abline(v=delta.dev.observed,col="grey",lwd=4)
```

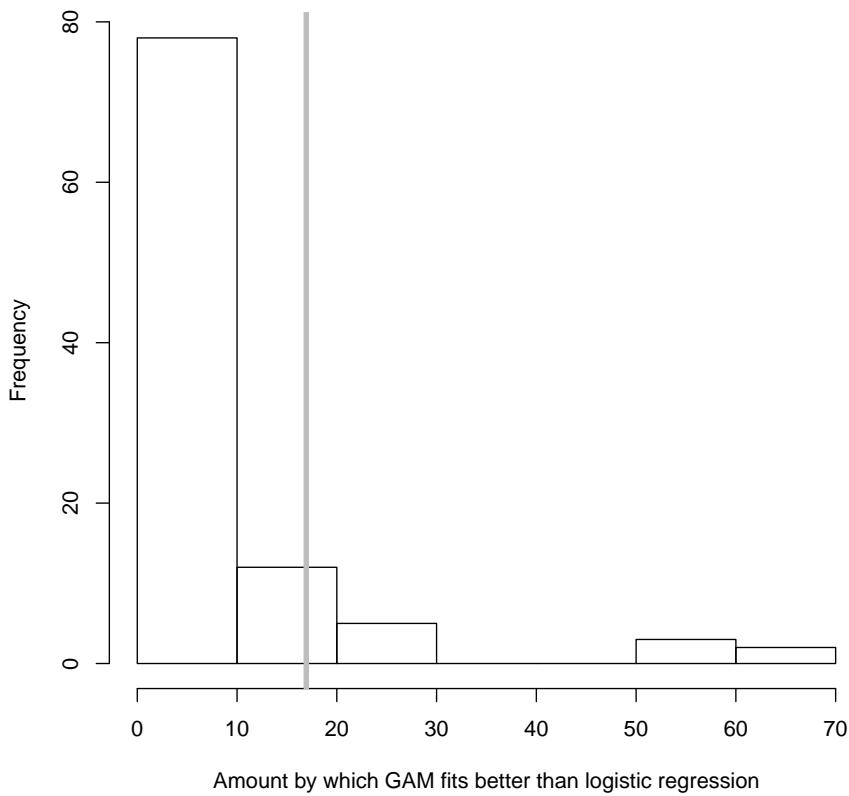


FIGURE 12.3: Sampling distribution for the difference in deviance between a GAM and a logistic regression, on data generated from a logistic regression. The observed difference in deviances is shown by the grey vertical line.

12.7 Weather Forecasting in Snoqualmie Falls

For our worked data example, 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 12.4)⁸. 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.

```
# Read in the whole data set as one big vector, skipping the first line of
# the file (header information)
snoqualmie <- scan("http://www.stat.washington.edu/peter/book.data/set1",skip=1)
# Create a two-column data frame, today's precipitation vs. tomorrow's
snoq <- data.frame(tomorrow=c(tail(snoqualmie,-1),NA),
                     today=snoqualmie)
# Make the data more comprehensible by adding years and days within each year
# First, what years are we talking about?
years <- 1948:1983
# How many days to each year?
days.per.year <- rep(c(366,365,365,365),length.out=length(years))
# Add a "year" column
snoq$year <- rep(years, times=days.per.year)
# Add a day-within-the-year column
snoq$day <- rep(c(1:366,1:365,1:365,1:365),times=length(years)/4)
# Trim the last row to get rid of the NA
snoq <- snoq[-nrow(snoq),]
```

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 one of the local hydroelectric power plants, or the tourist attraction of the Falls themselves. Examining the distribution of the data (Figures 12.5 and 12.6) 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 12.6, as well as common experience, that the coefficient on X should be positive.⁹

The estimation is straightforward:

⁸I learned of this data set from Guttorm (1995); the data file is available from <http://www.stat.washington.edu/peter/stoch.mod.data.html>. Prof. Guttorm formatted it so that each year was a different row, which is rather inconvenient for our purposes; see <http://www.stat.cmu.edu/~cshalizi/ADAFaEPoV/snoqualmie.R> for the commands used to reshape it.

⁹This 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.

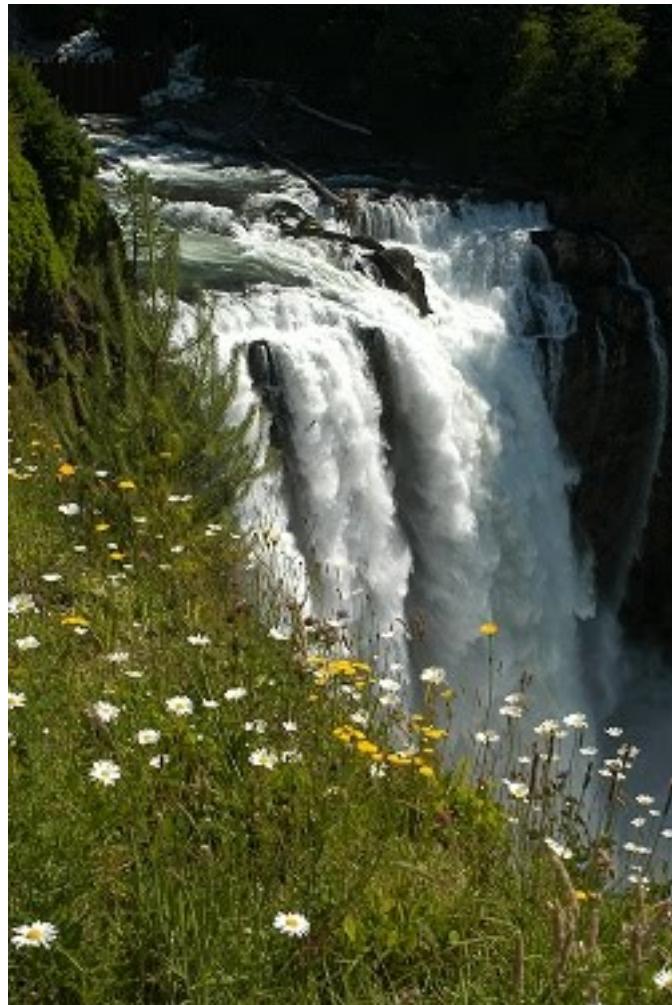


FIGURE 12.4: *Snoqualmie Falls, Washington, on a low-precipitation day.*
Photo by Jeannine Hall Gailey, from <http://myblog.webbish6.com/2011/07/17-years-and-hoping-for-another-17.html>. [[TODO: Get permission for photo use!]]

```
hist(snoqualmie,n=50,probability=TRUE,xlab="Precipitation (1/100 inch)")
rug(snoqualmie,col="grey")
```

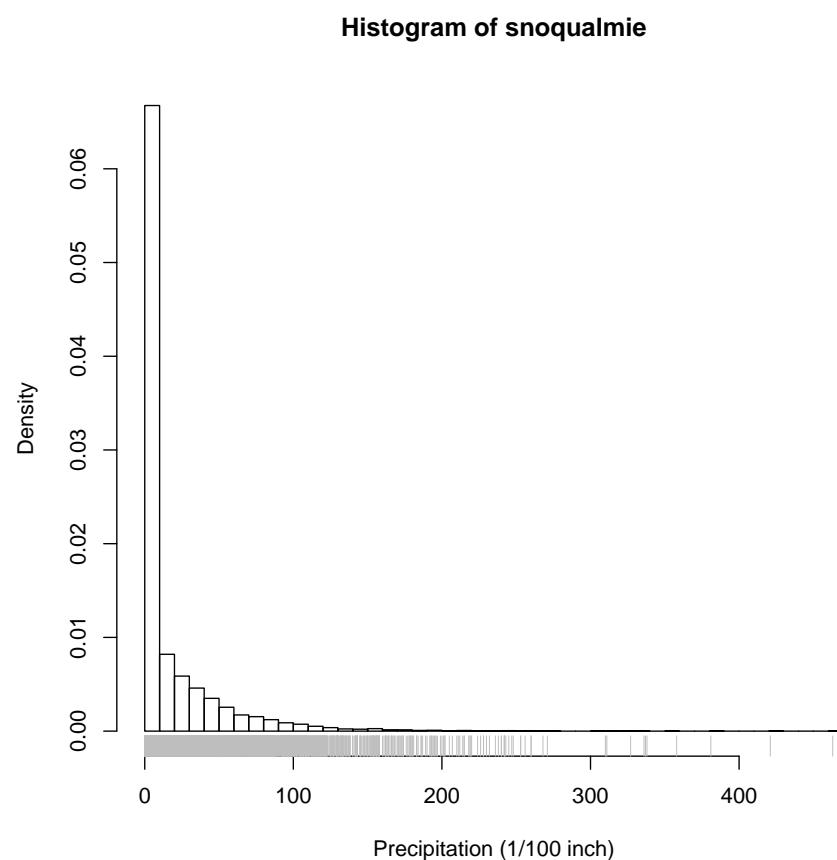


FIGURE 12.5: *Histogram of the amount of daily precipitation at Snoqualmie Falls*

```
plot(tomorrow~today,data=snoq,
      xlab="Precipitation today (1/100 inch)",
      ylab="Precipitation tomorrow (1/100 inch)",cex=0.1)
rug(snoq$today,side=1,col="grey")
rug(snoq$tomorrow,side=2,col="grey")
```

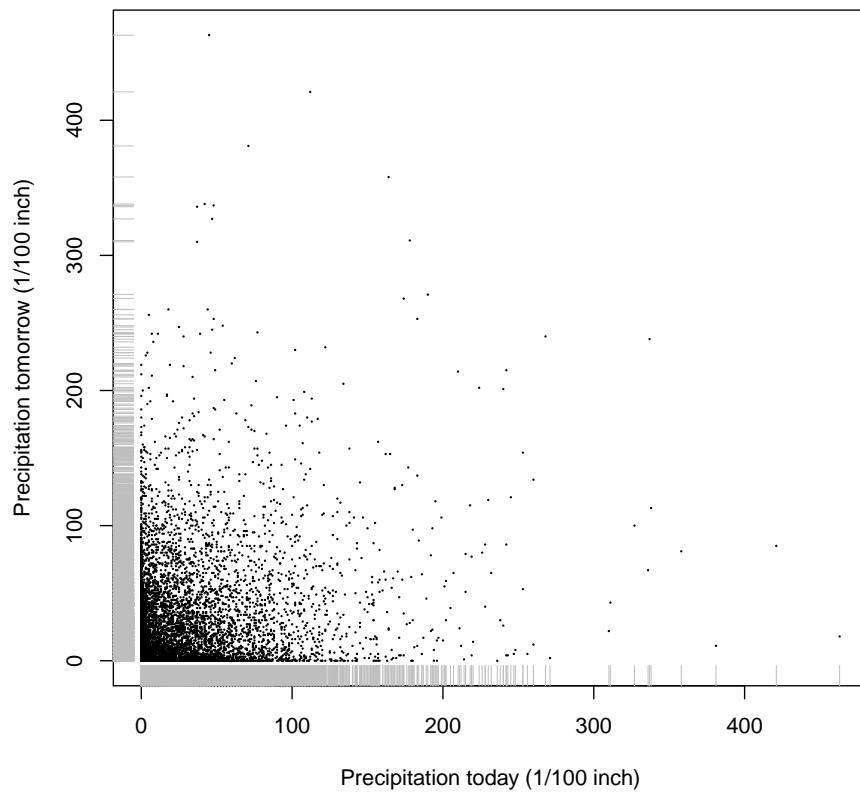


FIGURE 12.6: Scatterplot showing relationship between amount of precipitation on successive days. Notice that days of no precipitation can follow days of any amount of precipitation, but seem to be more common when there is little or no precipitation to start with.

```
snoq.logistic <- glm((tomorrow > 0) ~ today, data=snoq, family=binomial)
```

To see what came from the fitting, run `summary`:

```
print(summary(snoq.logistic), digits=3, signif.stars=FALSE)
##
## Call:
## glm(formula = (tomorrow > 0) ~ today, family = binomial, data = snoq)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.525  -0.999   0.167   1.170   1.367
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.43520  0.02163  -20.1   <2e-16
## today       0.04523  0.00131   34.6   <2e-16
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 18191 on 13147 degrees of freedom
## Residual deviance: 15896 on 13146 degrees of freedom
## AIC: 15900
##
## Number of Fisher Scoring iterations: 5
```

The coefficient on the amount of precipitation today is indeed positive, and (if we can trust R's assumptions) 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 on days of no precipitation. The linear predictor is then just the intercept, -0.435 , and the predicted probability of precipitation is 0.393. That is, even when there is no precipitation today, it's almost as likely as not that there will be some precipitation tomorrow.¹⁰

We can get a more global view of what the model is doing by plotting the data and the predictions (Figure 12.7). 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.

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, $\Pr[Y=1|X=x] = \mathbb{E}[Y|X=x]$, we can use a smoothing spline to estimate $\mathbb{E}[Y|X=x]$ (Figure 12.8). 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.

¹⁰For 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$today,side=1,col="grey")
data.plot <- data.frame(today=(0:500))
pred.bands <- function(mdl,data,col="black",mult=1.96) {
  preds <- predict(mdl,newdata=data,se.fit=TRUE)
  lines(data[,1],ilogit(preds$fit),col=col)
  lines(data[,1],ilogit(preds$fit+mult*preds$se.fit),col=col,lty="dashed")
  lines(data[,1],ilogit(preds$fit-mult*preds$se.fit),col=col,lty="dashed")
}
pred.bands(snoq.logistic,data.plot)

```

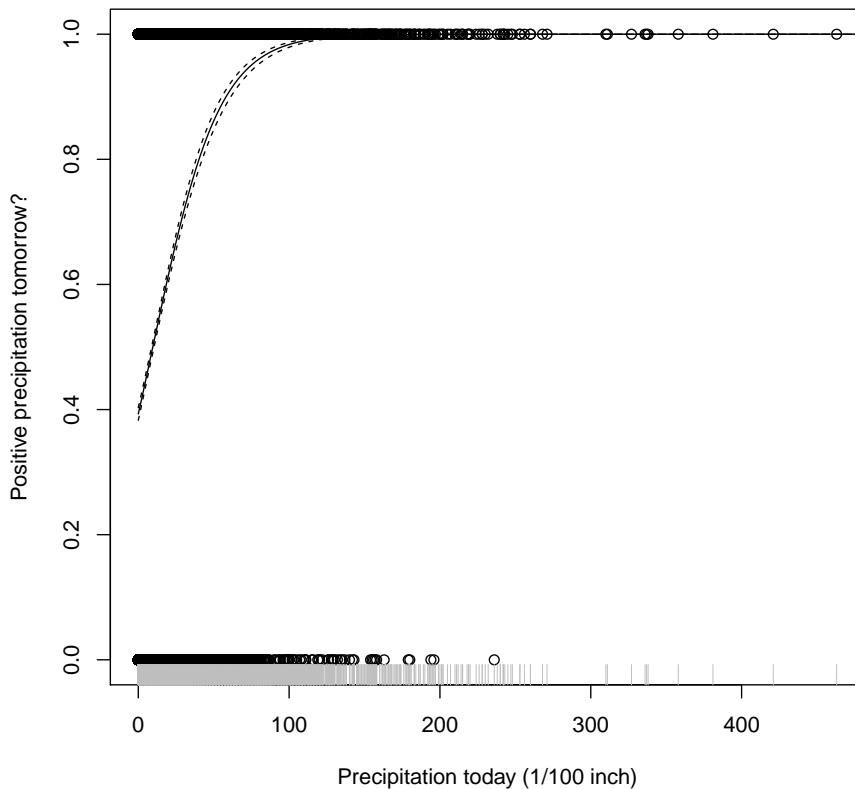


FIGURE 12.7: Data (dots), plus predicted probabilities (solid line) and approximate 95% confidence intervals from the logistic regression model (dashed lines). Note that calculating standard errors for predictions on the logit scale, and then transforming, is better practice than getting standard errors directly on the probability scale.

```
plot((tomorrow>0)~today,data=snoq,xlab="Precipitation today (1/100 inch)",  
      ylab="Positive precipitation tomorrow?")  
rug(snoq$today,side=1,col="grey")  
data.plot <- data.frame(today=(0:500))  
pred.bands(snoq.logistic,data.plot)  
snoq.spline <- smooth.spline(x=snoq$today,y=(snoq$tomorrow>0))  
lines(snoq.spline,col="red")
```

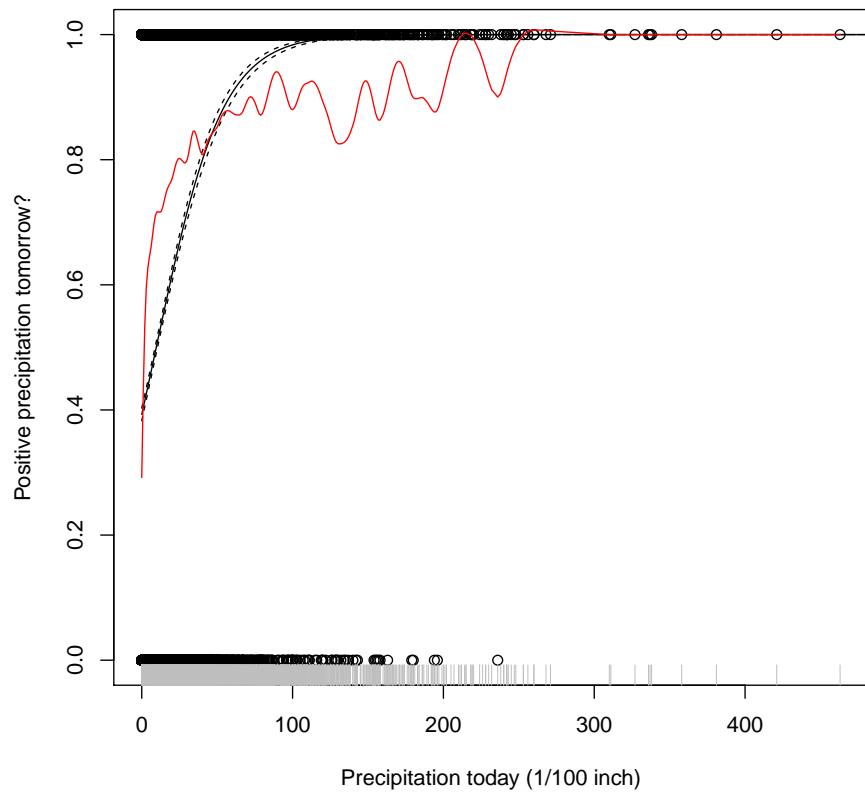


FIGURE 12.8: As Figure 12.7, plus a smoothing spline (red).

The result starts out notably above the logistic regression, then levels out and climbs much more slowly. It also has a bunch of dubious-looking wiggles, despite the cross-validation.

We can try to 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 12.9) is initially similar to the spline, but has some more exaggerated undulations, 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 hardly ever 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 1.5896×10^4 for the logistic regression, and 1.5122×10^4 for the GAM. We can go through the testing procedure outlined in §12.6. 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)
  return(rbinom(n=length(fitted.probs),size=1,prob=fitted.probs))
}
```

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.
## 0.000 0.000 1.000 0.526 1.000 1.000
summary(snoq.sim())
##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## 0.000 0.000 1.000 0.529 1.000 1.000
```

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[,"today"]) {
  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)
}
```

A single run of this takes about 0.6 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:

```
library(mgcv)
plot((tomorrow>0)~today,data=snoq,xlab="Precipitation today (1/100 inch)",
      ylab="Positive precipitation tomorrow?")
rug(snoq$today,side=1,col="grey")
pred.bands(snoq.logistic,data.plot)
lines(snoq.spline,col="red")
snoq.gam <- gam((tomorrow>0)~s(today),data=snoq,family=binomial)
pred.bands(snoq.gam,data.plot,"blue")
```

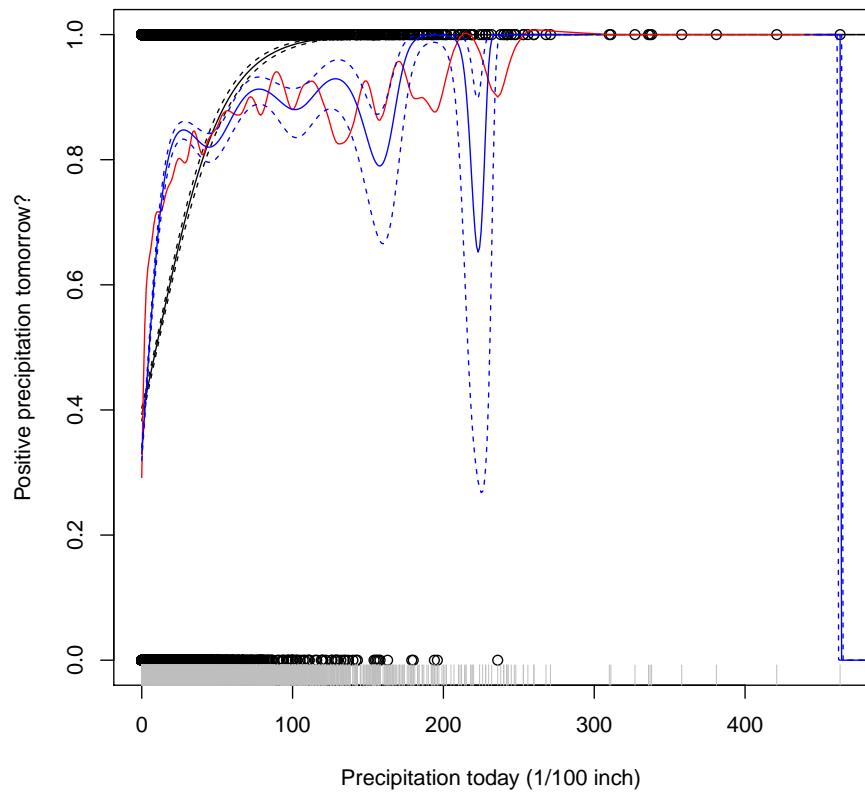


FIGURE 12.9: As Figure 12.8, but with the addition of a generalized additive model (blue line) and its confidence limits (dashed blue lines).

```
diff.dev.obs <- snoq.logistic$deviance - snoq.gam$deviance
null.dist.of.diff.dev <- replicate(100,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 67 seconds, or a bit over a minute; it gives a p -value of $< 1/101$. (A longer run of 1000 replicates, not shown, gives a p -values of $< 10^{-3}$.)

Having detected that there is a problem with the logistic model, 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 12.9 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. The new GLM has a deviance of 1.4955×10^4 , lower than even the GAM before, and the new GAM has a deviance of 1.4842×10^4 . I will leave repeating the specification test as an exercise. Figure 12.10 shows the data and the two new models. These are *extremely* close to each other at low precipitation, and diverge thereafter. The new GAM is the smoothest model we've nonparametric model we've seen yet, which suggests that before the it was being under-smoothed to help capture the special value at zero.

Let's turn now to looking at calibration. The actual fraction of no-precipitation days which are followed by precipitation is

```
signif(mean(snoq$tomorrow[snoq$today==0]>0),3)
## [1] 0.287
```

What does the new logistic model predict?

```
signif(predict(snoq2.logistic,
               newdata=data.frame(today=0,dry=1),type="response"),3)
##      1
## 0.287
```

```
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))  
lines(snoq.spline,col="red")  
pred.bands(snoq2.logistic,data.plot)  
pred.bands(snoq2.gam,data.plot,"blue")
```

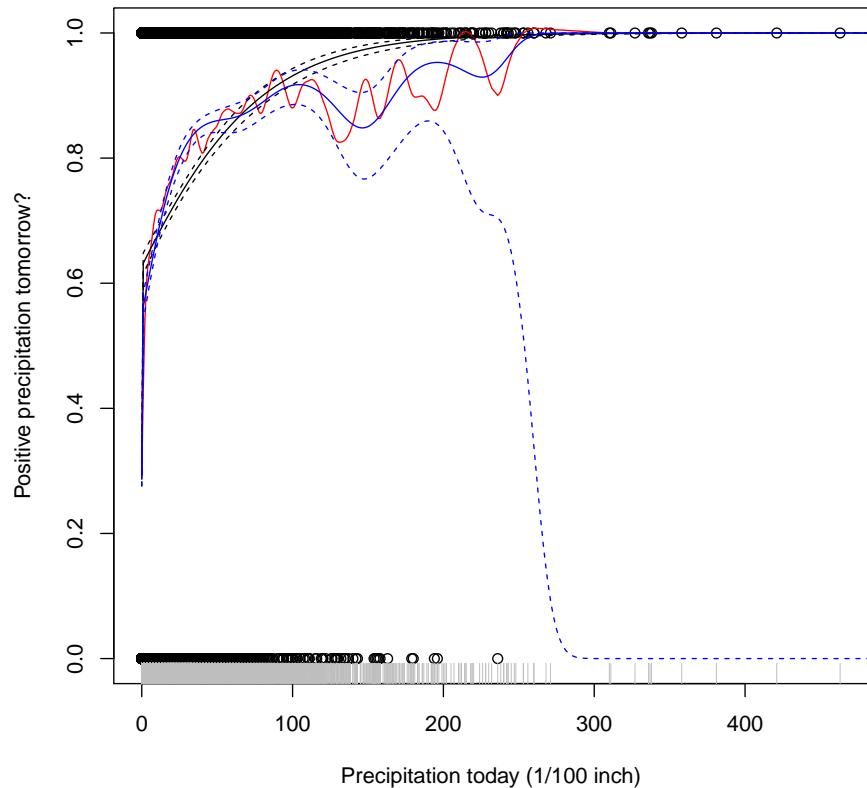


FIGURE 12.10: As Figure 12.9, but allowing the two models to use a dummy variable indicating when today is completely dry ($X = 0$).

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 below 0.63, so we'll look first at days where the predicted probability is between 0.63 and 0.64.

```
signif(mean(snoq$tomorrow[(fitted(snoq2.logistic) >= 0.63) &
                           (fitted(snoq2.logistic) < 0.64)] > 0),3)
## [1] 0.526
```

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))
  return(c(frequency=frequency,ave.prob=ave.prob,se=se))
}
```

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 probabilities¹¹. Try the function out before doing anything rash:

```
frequency.vs.probability(0.63)
## frequency    ave.prob      se
##   0.52603    0.63415   0.01586
```

This agrees with our previous calculation.

Now we can do this for a lot of probability brackets:

```
f.vs.p <- sapply(c(0.28,(63:100)/100),frequency.vs.probability)
```

This comes with some unfortunate R cruft, removable thus

```
f.vs.p <- data.frame(frequency=f.vs.p["frequency",],
                      ave.prob=f.vs.p["ave.prob",],se=f.vs.p["se",])
```

and we're ready to plot (Figure 12.11). The observed frequencies are generally reasonably near the predicted probabilities. While I wouldn't want to say this was the last word in weather forecasting¹², it's surprisingly good for such a simple model. I will leave calibration checking for the GAM as another exercise.

¹¹This could be improved by averaging predicted variances for each point, but using probability ranges of 0.01 makes it hardly worth the effort.

¹²There is an extensive discussion of this data in Guttorp (1995, ch. 2), including many significant refinements, such as dependence across multiple days.

```

plot(frequency~ave.prob,data=f.vs.p,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)

```

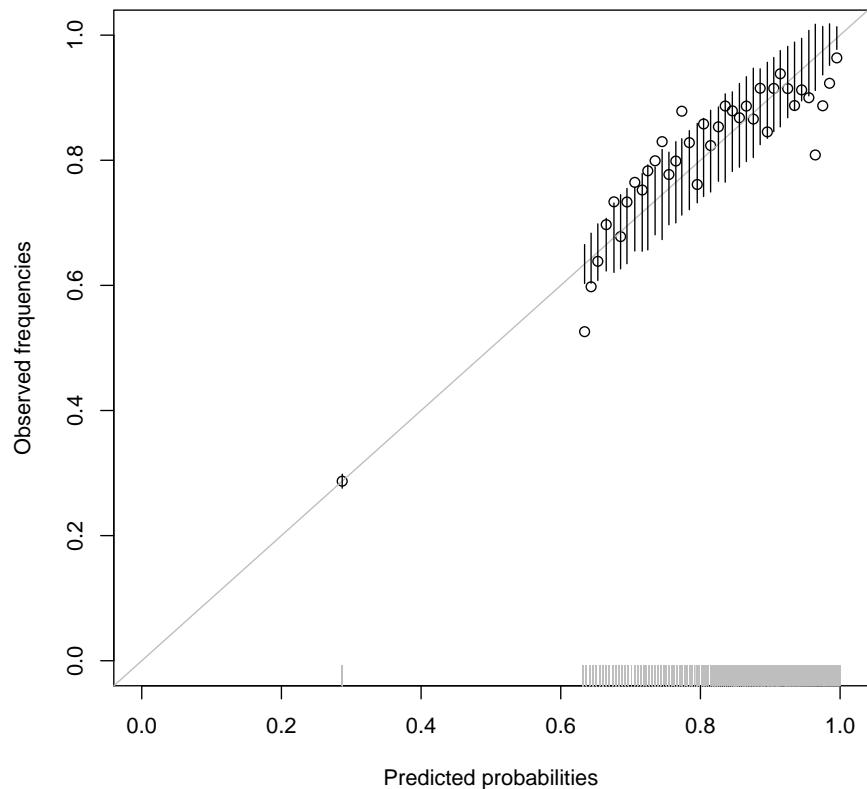


FIGURE 12.11: Calibration plot for the modified logistic regression model `snoq2.logistic`. Points show the actual frequency of precipitation for each level of predicted probability. Vertical lines are (approximate) 95% sampling intervals for the frequency, given the predicted probability and the number of observations.

12.8 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

$$\Pr(Y = c | \vec{X} = x) = \frac{e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_c e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}} \quad (12.23)$$

You can check that when there are only two classes (say, 0 and 1), equation 12.23 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 generality (Exercise 2)¹³.

Calculation of the likelihood now proceeds as before (only with more book-keeping), and so does maximum likelihood estimation.

12.9 Exercises

- “We minimize the mis-classification rate by predicting whichever class is more likely”: Let $\hat{Y}(x)$ be our predicted class, either 0 or 1. Our error rate is then $\Pr(Y \neq \hat{Y})$. Show that $\Pr(Y \neq \hat{Y}) = E[(Y - \hat{Y})^2]$. Further show that $E[(Y - \hat{Y})^2 | X = x] = \Pr(Y = 1 | X = x)(1 - 2\hat{Y}(x)) + \hat{Y}^2(x)$. Conclude by showing that if $\Pr(Y = 1 | X = x) > 0.5$, the risk of mis-classification is minimized by taking $\hat{Y} = 1$, that if $\Pr(Y = 1 | X = x) < 0.5$ the risk is minimized by taking $\hat{Y} = 0$, and that when $\Pr(Y = 1 | X = x) = 0.5$ both predictions are equally risky.
- A multiclass logistic regression, as in Eq. 12.23, 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.
- 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.

¹³Since we can arbitrarily chose which class’s parameters to “zero out” without affecting the predicted probabilities, strictly speaking the model in Eq. 12.23 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.

Chapter 13

Generalized Linear Models and Generalized Additive Models

[[TODO: Merge GLM/GAM and Logistic Regression chapters]]

[[ATTN: Keep as separate chapter, or merge with logistic?]]

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:

$$g(Y) \approx g(r(x)) + (Y - r(x))g'(r(x)) \quad (13.4)$$

$$= \eta(x) + (Y - r(x))g'(r(x)) \equiv z \quad (13.5)$$

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 exactly 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 :

$$\begin{aligned}\text{Var}[Z|X=x] &= \text{Var}[\eta(x)|X=x] + \text{Var}[(Y - r(x))g'(r(x))|X=x] \\ &= 0 + (g'(r(x)))^2 \text{Var}[Y|X=x]\end{aligned}\quad (13.6)$$

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, in logistic regression, 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 0.001 and 0.003!

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**, $\eta(x) = \beta_0 + x \cdot \beta$
- 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 in §13.1.4.

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 $\mathbf{E}[Y] = \text{Var}[Y] = \mu$. 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, $\hat{\epsilon}_i = \frac{y_i - \hat{\mu}(x_i)}{\sqrt{V(\mu(x_i))}}$, resample them to get $\tilde{\epsilon}_i$, and then add $\tilde{\epsilon}_i \sqrt{V(\mu(x_i))}$ to the fitted values. This does not really work when the response is discrete-valued, however.

[[ATTN: Look up if anyone has a good trick for this]]

13.1.4 Modeling Dispersion

When we pick a family for the conditional distribution of Y , we get a predicted conditional variance function, $V(\mu(x))$. The actual conditional variance $\text{Var}[Y|X=x]$ may however not track this. When the variances are larger, the process is **over-dispersed**; when they are smaller, **under-dispersed**. Over-dispersion is more common and more worrisome. In many cases, it arises from some un-modeled aspect of the process — some unobserved heterogeneity, or some missed dependence. For instance, if we observe count data with an upper limit and use a binomial model, we’re assuming that each “trial” within a data point is independent; positive correlation between the trials will give larger variance around the mean than the $mp(1-p)$ we’d expect¹.

The most satisfying solution to over-dispersion is to actually model its origin. Failing that, however, we can fall back on more modeling. One strategy is to say that

$$\text{Var}[Y|X=x] = \phi(x)V(\mu(x)) \quad (13.10)$$

and try to estimate the function ϕ — a modification of the variance-estimation idea we saw in §7.3. In doing so, we need a separate estimate of $\text{Var}[Y|X=x_i]$. This can come from repeated measurements at the same value of x , or from the squared residuals at each data point. Once we have some noisy but independent estimate of $\text{Var}[Y|X=x_i]$, the ratio $\text{Var}[Y|X=x_i]/V(\mu(x_i))$ can be regressed on x_i to estimate ϕ . Some people recommend doing this step, itself, through a generalized linear or generalized additive model, with a gamma distribution for the response, so that the response is guaranteed to be positive.

13.1.5 Likelihood and Deviance

When dealing with GLMs, it is conventional to report not the log-likelihood, but the **deviance**. The deviance of a model with parameters (β_0, β) is defined as

$$D(\beta_0, \beta) = 2[\ell(\text{saturated}) - \ell(\beta_0, \beta)] \quad (13.11)$$

¹If (for simplicity) all the trials have the same covariance ρ , then the variance of their sum is $mp(1-p) + m(m-1)\rho$ (why?).

Here, $\ell(\beta_0, \beta)$ is the log-likelihood of our model, and $\ell(\text{saturated})$ is the log-likelihood of a **saturated** model which has one parameter per data point. Thus, models with high likelihoods will have low deviances, and vice versa. If our model is correct and has $p + 1$ parameters in all (including the intercept), then the deviance will generally approach a χ^2 distribution asymptotically, with $n - (p + 1)$ degrees of freedom (Appendix G); the factor of 2 in the definition is to ensure this.

For discrete response variables, the saturated model can usually ensure that $\Pr(Y = y_i | X = x_i) = 1$, so $\ell(\text{saturated}) = 0$, and deviance is just twice the negative log-likelihood. If there are multiple data points with the same value of x but different values of y , then $\ell(\text{saturated}) < 0$. In any case, even for repeated values of x or even continuous response variables, differences in deviance are just twice differences in log-likelihood: $D(\text{model}_1) - D(\text{model}_2) = \ell(\text{model}_2) - \ell(\text{model}_1)$.

13.1.5.1 Maximum Likelihood and the Choice of Link Function

Having chosen a family of conditional distributions, it may happen that when we write out the log-likelihood, the latter depends on the *both* the response variables y_i and the coefficients only through the product of y_i with some transformation of the conditional mean μ :

$$\ell = \sum_{i=1}^n f(y_i, x_i) + y_i g(\mu_i) + b(\theta) \quad (13.12)$$

In the case of logistic regression, examining Eq. 12.8 (§12.2.1, p. 231) shows that the log-likelihood can be put in this form with $g(\mu_i) = \log \mu_i / (1 - \mu_i)$. In the case of a Gaussian conditional distribution for Y , we would have $f = -y_i^2/2$, $g(\mu_i) = \mu_i$, and $b(\theta) = -\mu_i^2$. When the log-likelihood can be written in this form, $g(\cdot)$ is the “natural” transformation to apply to conditional mean, i.e., the natural link function, and assures us that the solution to iterative least squares will converge on the maximum likelihood estimate.² Of course we are free to nonetheless use other transformations of the conditional expectation.

²To be more technical, we say that a distribution with parameters θ is an **exponential family** if its probability density function at x is $\exp f(x) + T(x) \cdot g(\theta)/z(\theta)$, for some vector of statistics T and some transformation g of the parameters. (To ensure normalization, $z(\theta) = \int \exp(f(x) + T(x) \cdot g(\theta)) dx$. Of course, if the sample space x is discrete, replace this integral with a sum.) We then say that $T(\cdot)$ are the “natural” or “canonical” **sufficient statistics**, and $g(\theta)$ are the “natural” parameters. Eq. 13.12 is picking out the natural parameters, presuming the response variable is itself the natural sufficient statistic. Many of the familiar families of distributions, like Gaussians, exponentials, gammas, Paretos, binomials and Poissons are exponential families. Exponential families are very important in classical statistical theory, and have deep connections to thermodynamics and statistical mechanics (where they’re called “canonical ensembles”, “Boltzmann distributions” or “Gibbs distributions” (Mandelbrot, 1962)), and to information theory (where they’re “maximum entropy distributions”, or “minimax codes” (Grünwald, 2007)). Despite their coolness, they are a rather peripheral topic for our sort of data analysis — though see Guttorp (1995) for examples of using them in modeling discrete processes. Any good book on statistical theory (e.g., Casella and Berger 2002) will have a fairly extensive discussion; Barndorff-Nielsen (1978) and Brown (1986) are comprehensive treatments.

13.1.6 R: `glm`

As with logistic regression, the workhorse R function for all manner of GLMs is, simply, `glm`. The syntax is strongly parallel to that of `lm`, with the addition of a `family` argument that specifies the intended distribution of the response variable (`binomial`, `gaussian`, `poisson`, etc.), and, optionally, a link function appropriate to the family. (See `help(family)` for the details.) With `family="gaussian"` and an identity link function, its intended behavior is the same as `lm`.

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 16.

13.3 Further Reading

At our level of theory, good references on generalized linear and generalized additive models include Faraway (2006) and Wood (2006), both of which include extensive examples in R. Tutz (2012) offers an extensive treatment of GLMs with categorical response distributions, along with comparisons to other models for that task.

Overdispersion is a vast subject, which I have just barely scratched owing to lack of space. All of the references just named discuss methods for it. Lambert and Roeder (1995) is worth mentioning for introducing some simple-to-calculate ways of detecting and describing over-dispersion which give some information about *why* the response is over-dispersed. One of these (the “relative variance curve”) is closely related to the idea sketched above about estimating the dispersion factor.

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. Problem set 35, 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?

2015 students: See the in-class demos for 12 February

Chapter 14

Classification and Regression Trees

[[TODO: Notes taken from another course; integrate]]

Having built up increasingly complicated models for regression, I'll now switch gears and introduce a class of nonlinear predictive model which at first seems too simple to possibly work, namely **prediction trees**. These have two varieties, **regression trees** and **classification trees**.

14.1 Prediction Trees

The basic idea is very simple. We want to predict a response or class Y from inputs X_1, X_2, \dots, X_p . We do this by growing a binary tree. At each internal node in the tree, we apply a test to one of the inputs, say X_i . Depending on the outcome of the test, we go to either the left or the right sub-branch of the tree. Eventually we come to a leaf node, where we make a prediction. This prediction aggregates or averages all the training data points which reach that leaf. Figure 14.1 should help clarify this.

Why do this? Predictors like linear or polynomial regression are **global models**, where a single predictive formula is supposed to hold over the entire data space. When the data has lots of features which interact in complicated, nonlinear ways, assembling a single global model can be very difficult, and hopelessly confusing when you do succeed. Some of the non-parametric smoothers try to fit models **locally** and then paste them together, but again they can be hard to interpret. (Additive models are at least pretty easy to grasp.)

An alternative approach to nonlinear regression is to sub-divide, or **partition**, the space into smaller regions, where the interactions are more manageable. We then partition the sub-divisions again — this is **recursive partitioning**, as in hierarchical clustering — until finally we get to chunks of the space which are so tame that we can fit simple models to them. The global model thus has two parts: one is just the recursive partition, the other is a simple model for each cell of the partition.

Now look back at Figure 14.1 and the description which came before it. Prediction trees use the tree to represent the recursive partition. Each of the **terminal**

Decision Tree: The Obama-Clinton Divide

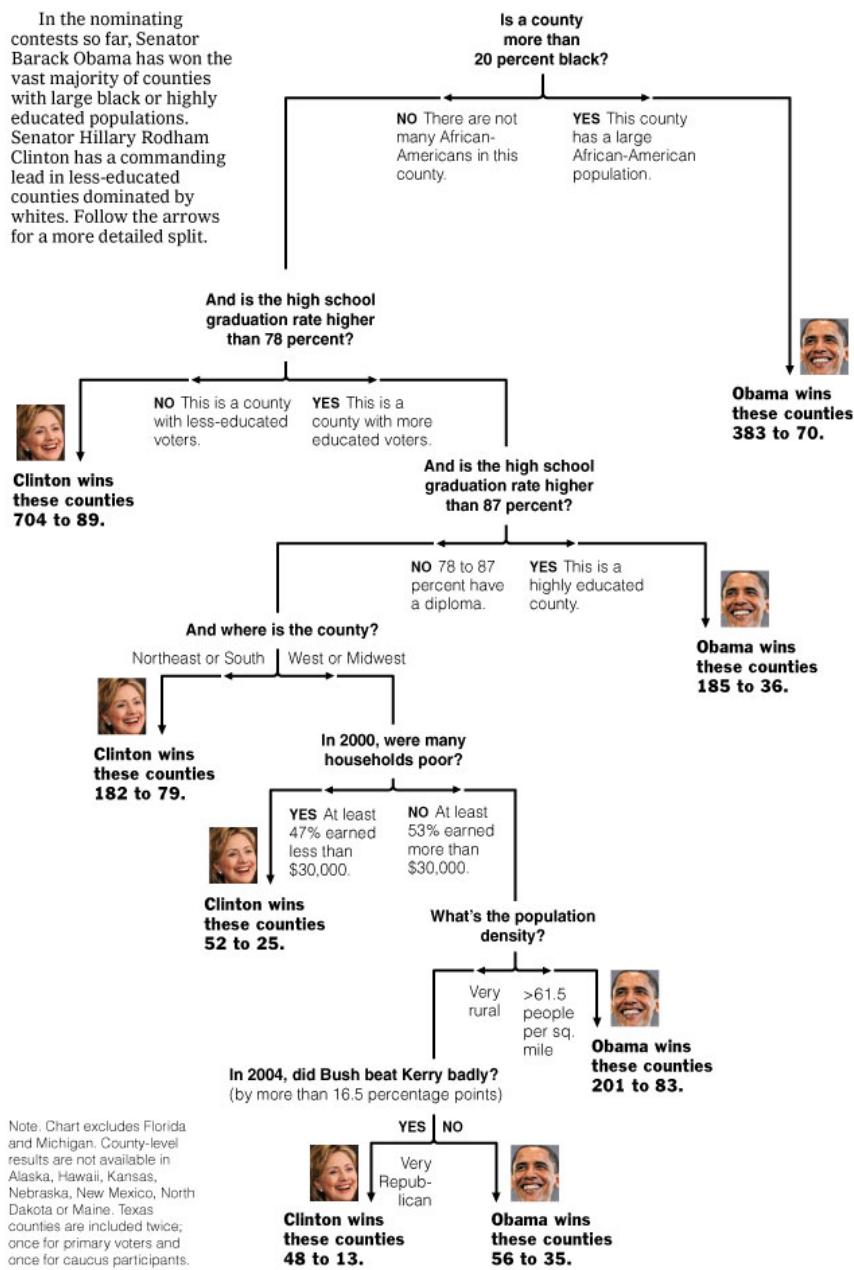


FIGURE 14.1: Classification tree for county-level outcomes in the 2008 Democratic Party primary (as of April 16), by Amanda Cox for the New York Times.

01:17 Wednesday 1st April, 2015

nodes, or **leaves**, of the tree represents a cell of the partition, and has attached to it a simple model which applies in that cell only. A point x **belongs** to a leaf if x falls in the corresponding cell of the partition. To figure out which cell we are in, we start at the **root node** of the tree, and ask a sequence of questions about the features. The interior nodes are labeled with questions, and the edges or branches between them labeled by the answers. Which question we ask next depends on the answers to previous questions. In the classic version, each question refers to only a single attribute, and has a yes or no answer, e.g., “Is HSGrad > 0.78?” or “Is Region == MIDWEST?” The variables can be of any combination of types (continuous, discrete but ordered, categorical, etc.). You could do more-than-binary questions, but that can always be accommodated as a larger binary tree. Asking questions about multiple variables at once is, again, equivalent to asking multiple questions about single variables.

That’s the recursive partition part; what about the simple local models? For classic regression trees, the model in each cell is just a *constant* estimate of Y . That is, suppose the points $(x_1, y_1), (x_2, y_2), \dots (x_c, y_c)$ are all the samples belonging to the leaf-node l . Then our model for l is just $\hat{y} = \frac{1}{c} \sum_{i=1}^c y_i$, the sample mean of the response variable in that cell. This is a piecewise-constant model.¹ There are several advantages to this:

- Making predictions is fast (no complicated calculations, just looking up constants in the tree)
- It’s easy to understand what variables are important in making the prediction (look at the tree)
- If some data is missing, we might not be able to go all the way down the tree to a leaf, but we can still make a prediction by averaging all the leaves in the sub-tree we do reach
- The model gives a jagged response, so it can work when the true regression surface is not smooth. If it is smooth, though, the piecewise-constant surface can approximate it arbitrarily closely (with enough leaves)
- There are fast, reliable algorithms to learn these trees

A last analogy before we go into some of the mechanics. One of the most comprehensible non-parametric methods is k -nearest-neighbors: find the points which are most similar to you, and do what, on average, they do. There are two big drawbacks to it: first, you’re defining “similar” entirely in terms of the inputs, not the response; second, k is constant everywhere, when some points just might have more very-similar neighbors than others. Trees get around both problems: leaves correspond to regions of the input space (a neighborhood), but one where the *responses* are similar, as well as the inputs being nearby; and their size can vary arbitrarily. Prediction trees are *adaptive* nearest-neighbor methods.

¹We could instead fit, say, a different linear regression for the response in each leaf node, using only the data points in that leaf (and using dummy variables for non-quantitative features). This would give a piecewise-linear model, rather than a piecewise-constant one. If we’ve built the tree well, however, all the points in each leaf are pretty similar, so the regression surface would be nearly constant anyway.

14.2 Regression Trees

Let's start with an example.

14.2.1 Example: California Real Estate Again

[[TODO: Replace with the more modern California data]]

After the homework and the last few lectures, you should be more than familiar with the California housing data; we'll try growing a regression tree for it. There are several R packages for regression trees; the easiest one is called, simply, `tree`.

```
calif = read.table("http://www.stat.cmu.edu/~cshalizi/350/hw/06/cadata.dat", header=TRUE)
require(tree) || install.packages("tree", dependencies=TRUE)
treefit = tree(log(MedianHouseValue) ~ Longitude+Latitude, data=calif)
```

This does a tree regression of the log price on longitude and latitude. What does this look like? Figure 14.2 shows the tree itself; Figure 14.3 shows the partition, overlaid on the actual prices in the state. (The ability to show the partition is why I picked only two input variables.)

Qualitatively, this looks like it does a fair job of capturing the interaction between longitude and latitude, and the way prices are higher around the coasts and the big cities. Quantitatively, the error isn't bad:

```
> summary(treefit)

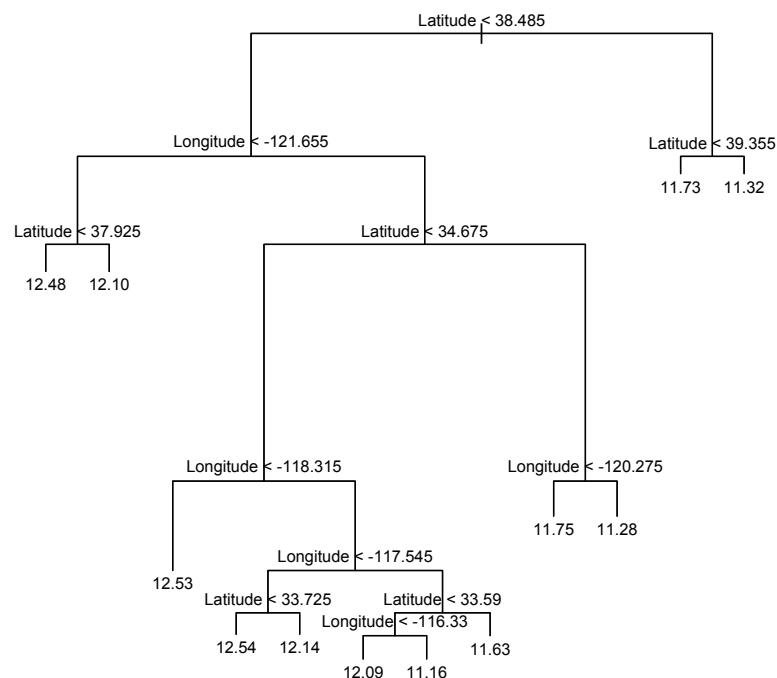
Regression tree:
tree(formula = log(MedianHouseValue) ~ Longitude + Latitude,
     data = calif)
Number of terminal nodes:  12
Residual mean deviance:  0.1662 = 3429 / 20630
Distribution of residuals:
    Min.   1st Qu.   Median   Mean   3rd Qu.   Max. 
-2.759e+00 -2.608e-01 -1.359e-02 -5.050e-15  2.631e-01  1.841e+00
```

Here “deviance” is just mean squared error; this gives us an RMS error of 0.41, which is higher than the models in the last handout, but not shocking since we're using only two variables, and have only twelve nodes.

The flexibility of a tree is basically controlled by how many leaves they have, since that's how many cells they partition things into. The tree fitting function has a number of controls settings which limit how much it will grow — each node has to contain a certain number of points, and adding a node has to reduce the error by at least a certain amount. The default for the latter, `mindev`, is 0.01; let's turn it down and see what happens.

```
treefit2 = tree(log(MedianHouseValue) ~ Longitude+Latitude, data=calif, mindev=0.001)
```

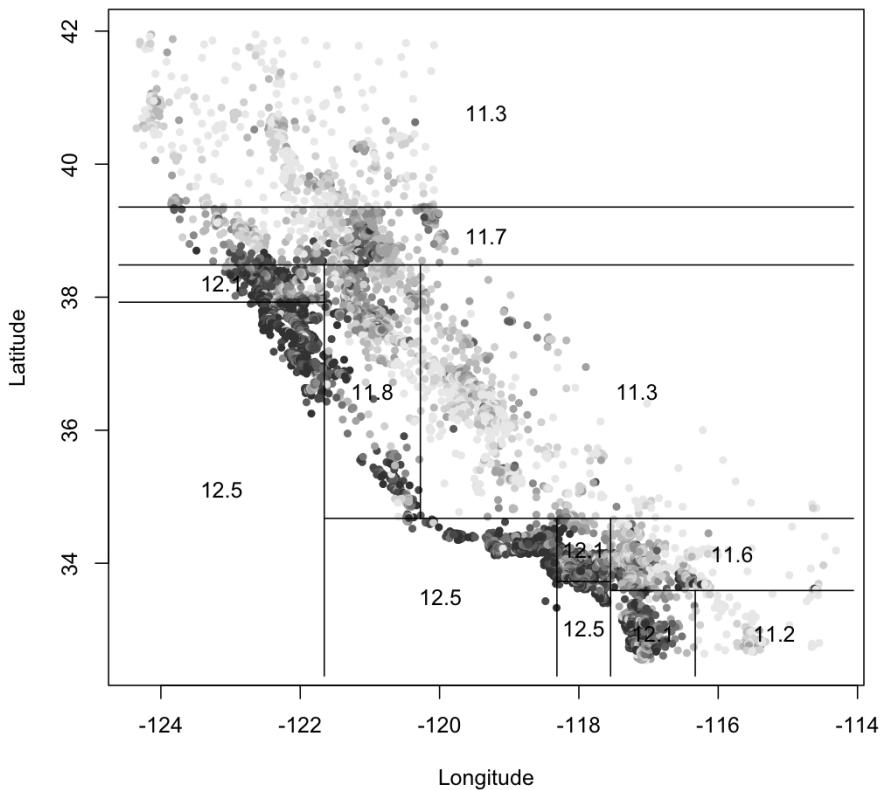
Figure 14.4 shows the tree itself; with 68 nodes, the plot is fairly hard to read, but by zooming in on any part of it, you can check what it's doing. Figure 14.5 shows the corresponding partition. It's obviously much finer-grained than that in Figure



```

plot(treefit)
text(treefit,cex=0.75)
  
```

FIGURE 14.2: Regression tree for predicting California housing prices from geographic coordinates. At each internal node, we ask the associated question, and go to the left child if the answer is “yes”, to the right child if the answer is “no”. Note that leaves are labeled with log prices; the plotting function isn’t flexible enough, unfortunately, to apply transformations to the labels.



```

price.deciles = quantile(calif$MedianHouseValue,0:10/10)
cut.prices = cut(calif$MedianHouseValue,price.deciles,include.lowest=TRUE)
plot(calif$Longitude,calif$Latitude,col=grey(10:2/11)[cut.prices],pch=20,
     xlab="Longitude",ylab="Latitude")
partition.tree(treefit,ordvars=c("Longitude","Latitude"),add=TRUE)

```

FIGURE 14.3: Map of actual median house prices (color-coded by decile, darker being more expensive), and the partition of the `treefit` tree.

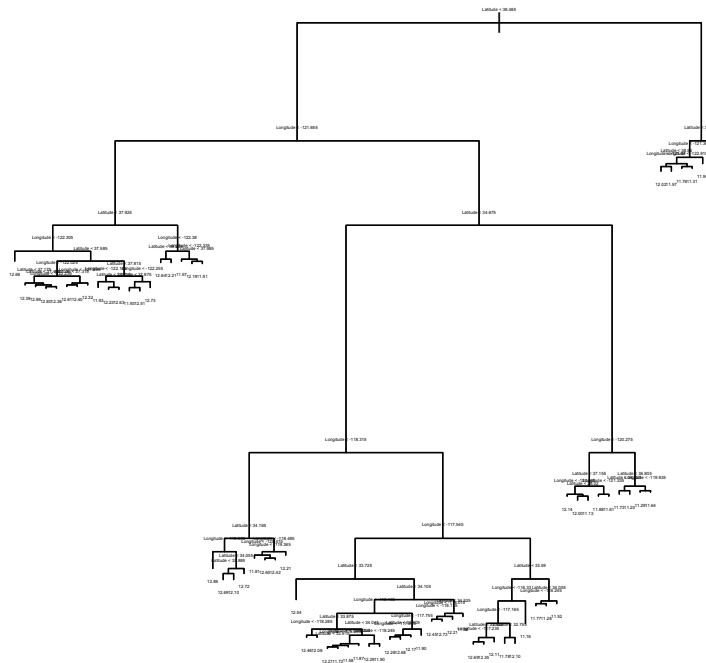


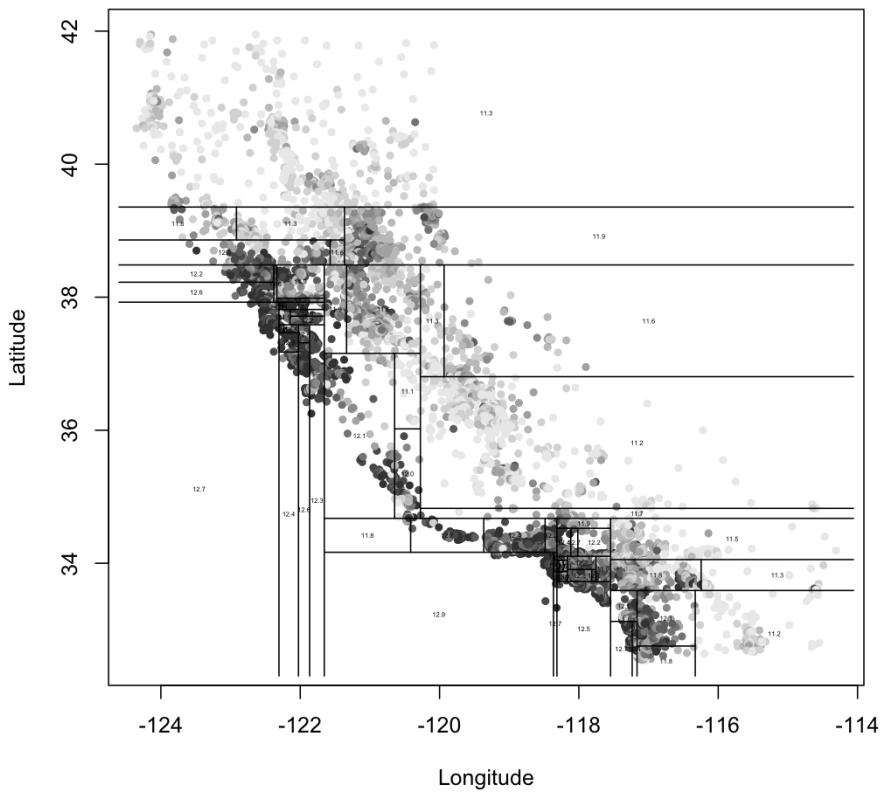
FIGURE 14.4: As Figure 14.2, but allowing splits for smaller reductions in error ($\text{mindev}=0.001$ rather than the default $\text{mindev}=0.01$).

14.3, and does a better job of matching the actual prices (RMS error 0.32). More interestingly, it doesn't just uniformly divide up the big cells from the first partition; some of the new cells are very small, others quite large. The metropolitan areas get a lot more detail than the Mojave.

Of course there's nothing magic about the geographic coordinates, except that they make for pretty plots. We can include all the input features in our model:

```
treefit3 <- tree(log(MedianHouseValue) ~ ., data=calif)
```

with the result shown in Figure 14.6. This model has fifteen leaves, as opposed to sixty-eight for treefit2, but the RMS error is almost as good (0.36). This is highly interactive: latitude and longitude are only used if the income level is sufficiently low. (Unfortunately, this does mean that we don't have a spatial partition to compare to



```
plot(calif$Longitude,calif$Latitude,col=grey(10:2/11)[cut.prices],pch=20,
     xlab="Longitude",ylab="Latitude")
partition.tree(treefit2,ordvars=c("Longitude","Latitude"),add=TRUE,cex=0.3)
```

FIGURE 14.5: Partition for treefit2. Note the high level of detail around the cities, as compared to the much coarser cells covering rural areas where variations in prices are less extreme.

the previous ones, but we *can* map the predictions; Figure 14.7.) Many of the features, while they were available to the tree fit, aren't used at all.

Now let's turn to how we actually grow these trees.

14.2.2 Regression Tree Fitting

Once we fix the tree, the local models are completely determined, and easy to find (we just average), so all the effort should go into finding a good tree, which is to say into finding a good partitioning of the data. We saw some ways of doing this when we did clustering, and will recycle those ideas here.

In clustering, remember, what we would ideally do was maximizing $I[C; X]$, the information the cluster gave us about the features X . With regression trees, what we want to do is maximize $I[C; Y]$, where Y is now the response variable, and C the variable saying which leaf of the tree we end up at. Once again, we can't do a direct maximization, so we again do a greedy search. We start by finding the one binary question which maximizes the information we get about Y ; this gives us our root node and two daughter nodes.² At each daughter node, we repeat our initial procedure, asking which question would give us the maximum information about Y , given where we already are in the tree. We repeat this recursively.

Every recursive algorithm needs to know when it's done, a **stopping criterion**. Here this means when to stop trying to split nodes. Obviously nodes which contain only one data point cannot be split, but giving each observations its own leaf is unlikely to generalize well. A more typical criterion is something like: halt when each child would contain less than five data points, or when splitting increases the information by less than some threshold. Picking the criterion is important to get a good tree, so we'll come back to it presently.

We have only seen entropy and information defined for discrete variables.³ You *can* define them for continuous variables, and sometimes the continuous information is used for building regression trees, but it's more common to do the same thing that we did with clustering, and look not at the mutual information but at the sum of squares. The sum of squared errors for a tree T is

$$S = \sum_{c \in \text{leaves}(T)} \sum_{i \in c} (y_i - m_c)^2$$

where $m_c = \frac{1}{n_c} \sum_{i \in c} y_i$, the prediction for leaf c . Just as with clustering, we can re-write this as

$$S = \sum_{c \in \text{leaves}(T)} n_c V_c$$

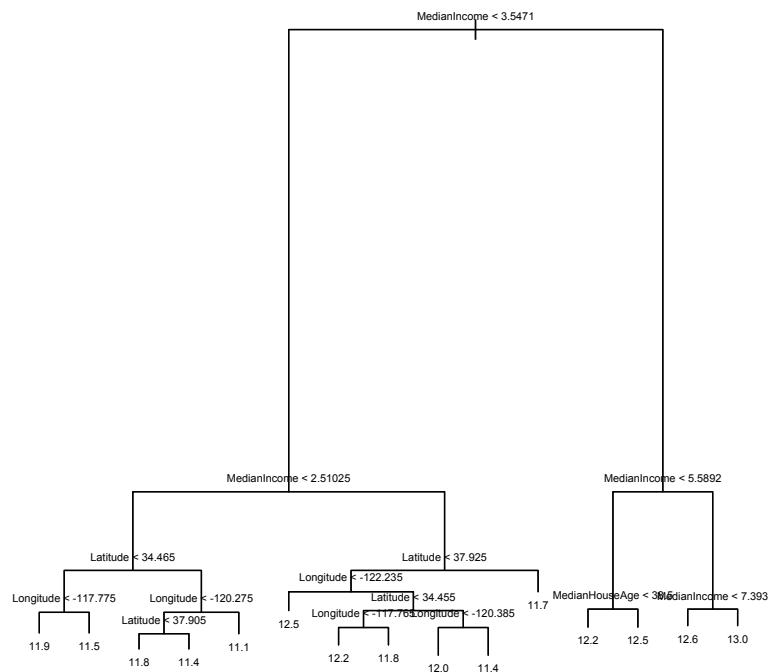
where V_c is the within-leave variance of leaf c . So we will make our splits so as to minimize S .

The basic regression-tree-growing algorithm then is as follows:

1. Start with a single node containing all points. Calculate m_c and S .

²Mixing botanical and genealogical metaphors for trees is ugly, but I can't find a way around it.

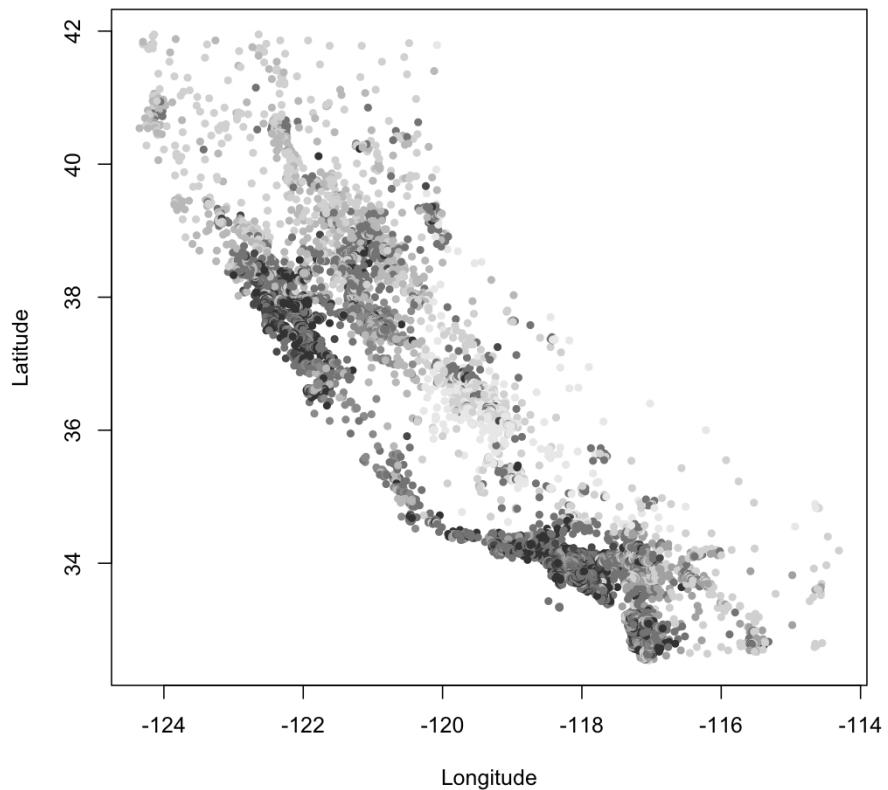
³Unless you read the paper by David Feldman, that is.



```
plot(treefit3)
```

```
text(treefit3,cex=0.5,digits=3)
```

FIGURE 14.6: Regression tree for log price when all other features are included as (potential) inputs. Note that many of the features are not used by the tree.



```
cut.predictions = cut(predict(treefit3), log(price.deciles), include.lowest=TRUE)
plot(calif$Longitude, calif$Latitude, col=grey(10:2/11)[cut.predictions], pch=20,
     xlab="Longitude", ylab="Latitude")
```

FIGURE 14.7: Predicted prices for the `treefit3` model. Same color scale as in previous plots (where dots indicated actual prices).

2. If all the points in the node have the same value for all the input variables, stop. Otherwise, search over all binary splits of all variables for the one which will reduce S as much as possible. If the largest decrease in S would be less than some threshold δ , or one of the resulting nodes would contain less than q points, stop. Otherwise, take that split, creating two new nodes.
3. In each new node, go back to step 1.

Trees use only one feature (input variable) at each step. If multiple features are equally good, which one is chosen is a matter of chance, or arbitrary programming decisions.

One problem with the straight-forward algorithm I've just given is that it can stop too early, in the following sense. There can be variables which are not very informative themselves, but which lead to very informative subsequent splits. (This was the point of all our talk about interactions when we looked at information theory.) This suggests a problem with stopping when the decrease in S becomes less than some δ . Similar problems can arise from arbitrarily setting a minimum number of points q per node.

A more successful approach to finding regression trees uses the idea of cross-validation from last time. We randomly divide our data into a training set and a testing set (say, 50% training and 50% testing). We then apply the basic tree-growing algorithm to the training data *only*, with $q = 1$ and $\delta = 0$ — that is, we grow the largest tree we can. This is generally going to be too large and will over-fit the data. We then use cross-validation to **prune** the tree. At each pair of leaf nodes with a common parent, we evaluate the error on the *testing* data, and see whether the testing sum of squares would shrink if we removed those two nodes and made their parent a leaf. If so, we prune; if not, not. This is repeated until pruning no longer improves the error on the testing data. The reason this is superior to arbitrary stopping criteria, or to rewarding parsimony as such, is that it directly checks whether the extra capacity (nodes in the tree) pays for itself by improving generalization error. If it does, great; if not, get rid of it. This is something we can do with regression trees that we couldn't really do with (say) hierarchical clustering, because trees make predictions we can test on new data, and the clustering techniques we looked at before do not.

There are lots of other cross-validation tricks for trees. One cute one is to alternate growing and pruning. We divide the data into two parts, as before, and first grow and then prune the tree. We then exchange the role of the training and testing sets, and try to grow our pruned tree to fit the second half. We then prune again, on the first half. We keep alternating in this manner until the size of the tree doesn't change.

14.2.2.1 Cross-Validation and Pruning in R

The `tree` package contains functions `prune.tree` and `cv.tree` for pruning trees by cross-validation.

The function `prune.tree` takes a tree you fit by `tree` (see R advice for last homework), and evaluates the error of the tree and various prunings of the tree, all the way down to the stump. The evaluation can be done either on new data, if supplied, or

on the training data (the default). If you ask it for a particular size of tree, it gives you the best pruning of that size⁴. If you don't ask it for the best tree, it gives an object which shows the number of leaves in the pruned trees, and the error of each one. This object can be plotted.

```
my.tree = tree(y ~ x1 + x2, data=my.data) # Fits tree
prune.tree(my.tree,best=5) # Returns best pruned tree with 5 leaves, evaluating
                           # error on training data
prune.tree(my.tree,best=5,newdata=test.set) # Ditto, but evaluates on test.set
my.tree.seq = prune.tree(my.tree) # Sequence of pruned tree sizes/errors
plot(my.tree.seq) # Plots size vs. error
my.tree.seq$dev # Vector of error rates for prunings, in order
opt.trees = which(my.tree.seq$dev == min(my.tree.seq$dev)) # Positions of
               # optimal (with respect to error) trees
min(my.tree.seq$size[opt.trees]) # Size of smallest optimal tree
```

Finally, `prune.tree` has an optional `method` argument. The default is `method="deviance"`, which fits by minimizing the mean squared error (for continuous responses) or the negative log likelihood (for discrete responses; see below).⁵

The function `cv.tree` does k -fold cross-validation (default is 10). It requires as an argument a fitted tree, and a function which will take that tree and new data. By default, this function is `prune.tree`.

```
my.tree.cv = cv.tree(my.tree)
```

The type of output of `cv.tree` is the same as the function it's called on. If I do

```
cv.tree(my.tree,best=19)
```

I get the best tree (per cross-validation) of no more than 19 leaves. If I do

```
cv.tree(my.tree)
```

I get information about the cross-validated performance of the whole *sequence* of pruned trees, e.g., `plot(cv.tree(my.tree))`. Optional arguments to `cv.tree` can include K , and any additional arguments for the function it applies.

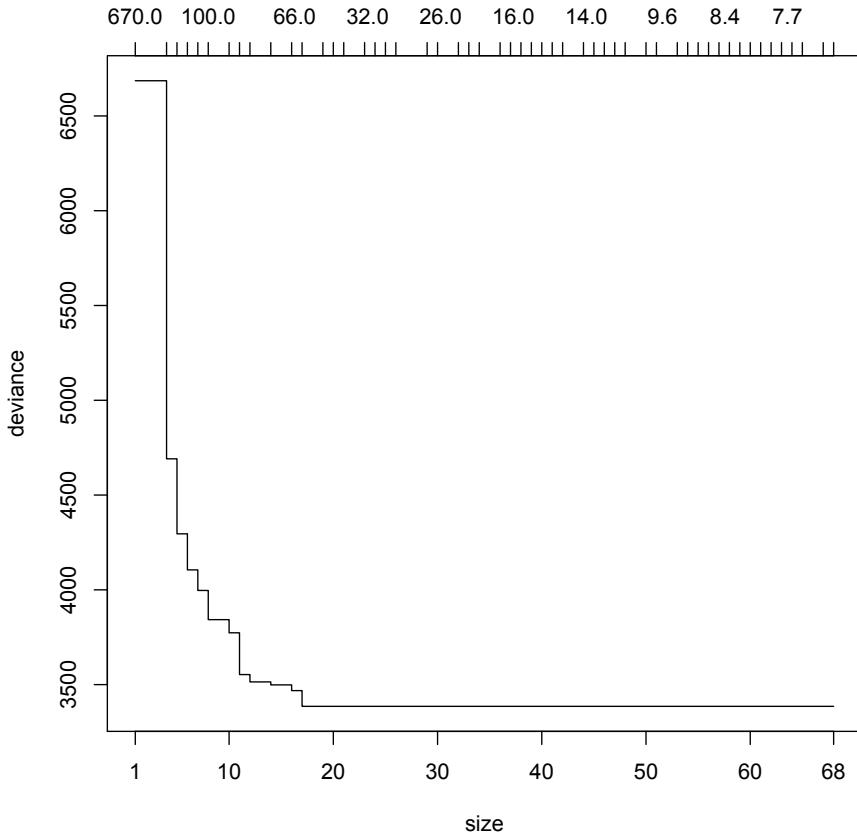
To illustrate, think back to `treefit2`, which predicted predicted California house prices based on geographic coordinates, but had a very large number of nodes because the tree-growing algorithm was told to split on almost any provocation. Figure 14.8 shows the size/performance trade-off. Figures 14.9 and 14.10 show the result of pruning to the smallest size compatible with minimum cross-validated error.

14.2.3 Uncertainty in Regression Trees

Even when we are making point predictions, we have some uncertainty, because we've only seen a finite amount of data, and this is not an *entirely* representative

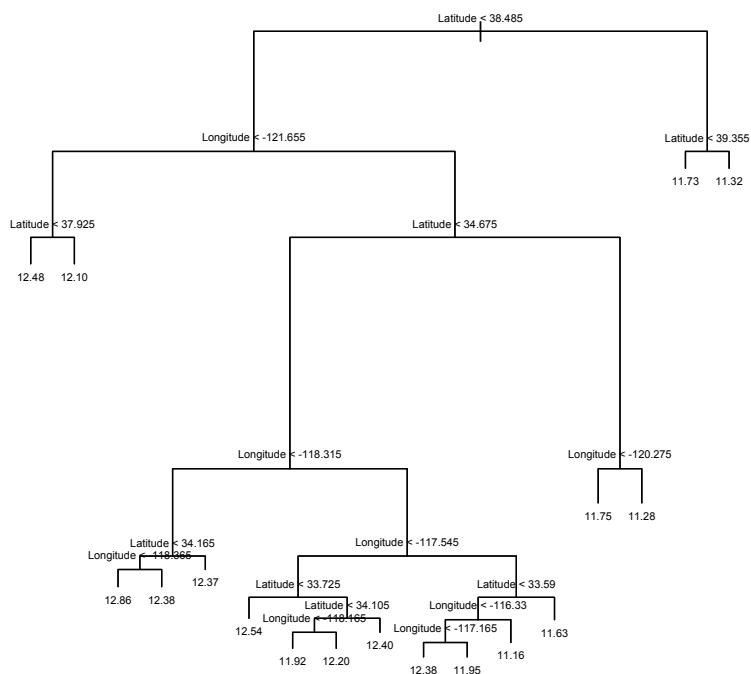
⁴Or, if there is no tree with that many leaves, the smallest number of leaves \geq the requested size.

⁵With discrete responses, you may get better results by saying `method="misclass"`, which looks at the misclassification rate.



```
treefit2.cv <- cv.tree(treefit2)
plot(treefit2.cv)
```

FIGURE 14.8: Size (horizontal axis) versus cross-validated sum of squared errors (vertical axis) for successive prunings of the treefit2 model. (The upper scale on the horizontal axis refers to the “cost/complexity” penalty. The idea is that the pruning minimizes $(\text{total error}) + \lambda(\text{complexity})$ for a certain value of λ , which is what’s shown on that scale. Here complexity is a function of the number of leaves; see Ripley (1996) for details. Or, just ignore the upper scale!)

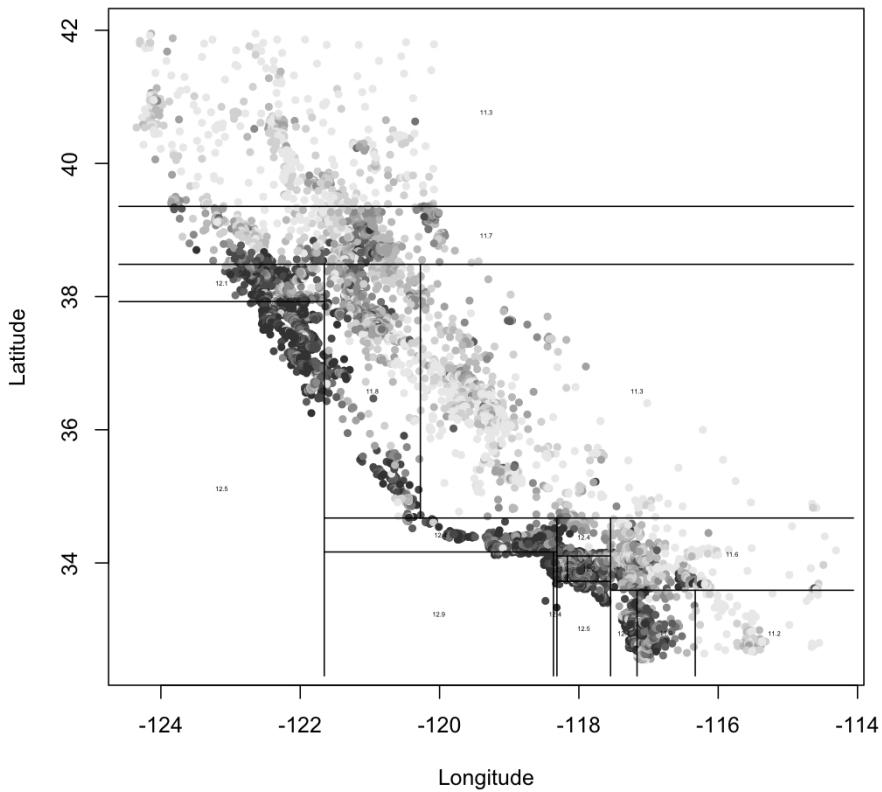


```

opt.trees = which(treefit2.cv$dev == min(treefit2.cv$dev))
best.leaves = min(treefit2.cv$size[opt.trees])
treefit2.pruned = prune.tree(treefit2,best=best.leaves)
plot(treefit2.pruned)

```

FIGURE 14.9: `treefit2`, after being pruned by ten-fold cross-validation.



```
plot(calif$Longitude,calif$Latitude,col=grey(10:2/11)[cut.prices],pch=20,  
     xlab="Longitude",ylab="Latitude")  
partition.tree(treefit2.pruned,ordvars=c("Longitude","Latitude"),  
              add=TRUE,cex=0.3)
```

FIGURE 14.10: `treefit2.pruned`'s partition of California. Compare to Figure 14.5.

sample of the underlying probability distribution. With a regression tree, we can separate the uncertainty in our predictions into two parts. First, we have some uncertainty in what our predictions should be, *assuming* the tree is correct. Second, we may of course be wrong about the tree.

The first source of uncertainty — imprecise estimates of the conditional means within a given partition — is fairly easily dealt with. We can consistently estimate the standard error of the mean for leaf c as $V_c/(n_c - 1)$, just like we would for any other mean of IID samples. The second source is more troublesome; as the response values shift, the tree itself changes, and discontinuously so, tree shape being a discrete variable. What we want is some estimate of how different the tree could have been, had we just drawn a different sample from the same source distribution.

One way to estimate this, from the data at hand, is **non-parametric bootstrap-ping**. Given data $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, we draw a random set of integers J_1, J_2, \dots, J_n , independently and uniformly from the numbers $1 : n$, *with* replacement. Then we set

$$(X'_i, Y'_i) = (x_{J_i}, y_{J_i})$$

Each of the re-sample data points has the same distribution as the whole of the original data sample, and they're independent. This is thus an IID sample of size n from the **empirical distribution**, and as close as we can get to another draw from the original data source without imposing any assumptions about how that's distributed. We can now treat this **bootstrap sample** just like the original data and fit a tree to it. Repeated many times, we get a **bootstrap sampling distribution** of trees, which approximates the actual sampling distribution of regression trees. The spread of the predictions of our bootstrapped trees around that of our original gives us an indication of how our tree's predictions are distributed around the truth.

We will see more uses for bootstrapped trees next time, when we look at how to combine trees into forests.

14.3 Classification Trees

Classification trees work just like regression trees, only they try to predict a discrete category (the class), rather than a numerical value. The variables which go into the classification — the inputs — can be numerical or categorical themselves, the same way they can with a regression tree. They are useful for the same reasons regression trees are — they provide fairly comprehensible predictors in situations where there are many variables which interact in complicated, nonlinear ways.

We find classification trees in almost the same way we found regression trees: we start with a single node, and then look for the binary distinction which gives us the most information about the class. We then take each of the resulting new nodes and repeat the process there, continuing the recursion until we reach some stopping criterion. The resulting tree will often be too large (i.e., over-fit), so we prune it back using (say) cross-validation. The differences from regression-tree growing have to do with (1) how we measure information, (2) what kind of predictions the tree makes, and (3) how we measure predictive error.

14.3.1 Measuring Information

The response variable Y is categorical, so we can use information theory to measure how much we learn about it from knowing the value of another discrete variable A :

$$I[Y;A] = \sum_a \Pr(A=a) I[Y;A=a] \quad (14.1)$$

where

$$I[Y;A=a] = H[Y] - H[Y|A=a] \quad (14.2)$$

and you remember the definitions of entropy $H[Y]$ and conditional entropy $H[Y|A=a]$.

$I[Y;A=a]$ is how much our uncertainty about Y decreases from knowing that $A = a$. (Less subjectively: how much less variable Y becomes when we go from the full population to the sub-population where $A = a$.) $I[Y;A]$ is how much our uncertainty about Y shrinks, on average, from knowing the value of A .

For classification trees, A isn't (necessarily) one of the predictors, but rather the answer to some question, generally binary, about one of the predictors X , i.e., $A = 1_{\mathcal{A}}(X)$ for some set \mathcal{A} . This doesn't change any of the math above, however. So we chose the question in the first, root node of the tree so as to maximize $I[Y;A]$, which we calculate from the formula above, using the relative frequencies in our data to get the probabilities.

When we want to get good questions at subsequent nodes, we have to take into account what we know already at each stage. Computationally, we do this by computing the probabilities and informations using only the cases in that node, rather than the complete data set. (Remember that we're doing *recursive* partitioning, so at each stage the sub-problem looks just like a smaller version of the original problem.) Mathematically, what this means is that if we reach the node when $A = a$ and $B = b$, we look for the question C which maximizes $I[Y;C|A=a,B=b]$, the information *conditional* on $A = a, B = b$. Algebraically,

$$I[Y;C|A=a,B=b] = H[Y|A=a,B=b] - H[Y|A=a,B=b,C] \quad (14.3)$$

Computationally, rather than looking at all the cases in our data set, we just look at the ones where $A = a$ and $B = b$, and calculate as though that were all the data. Also, notice that the first term on the right-hand side, $H[Y|A=a,B=b]$, does not depend on the next question C . So rather than maximizing $I[Y;C|A=a,B=b]$, we can just minimize $H[Y|A=a,B=b,C]$.

14.3.2 Making Predictions

There are two kinds of predictions which a classification tree can make. One is a **point** prediction, a single guess as to the class or category: to say “this is a flower” or “this is a tiger” and nothing more. The other, a **distributional prediction**, gives a *probability* for each class. This is slightly more general, because if we need to extract a point prediction from a probability forecast we can always do so, but we can't go in the other direction.

For probability forecasts, each terminal node in the tree gives us a distribution over the classes. If the terminal node corresponds to the sequence of answers $A = a$, $B = b, \dots Q = q$, then ideally this would give us $\Pr(Y = y | A = a, B = b, \dots Q = q)$ for each possible value y of the response. A simple way to get close to this is to use the empirical relative frequencies of the classes in that node. E.g., if there are 33 cases at a certain leaf, 22 of which are tigers and 11 of which are flowers, the leaf should predict “tiger with probability 2/3, flower with probability 1/3”. This is the **maximum likelihood** estimate of the true probability distribution, and we’ll write it $\hat{\Pr}(\cdot)$.

Incidentally, while the empirical relative frequencies are consistent estimates of the true probabilities under many circumstances, nothing particularly *compells* us to use them. When the number of classes is large relative to the sample size, we may easily fail to see any samples at all of a particular class. The empirical relative frequency of that class is then zero. This is good if the actual probability is zero, not so good otherwise. (In fact, under the negative log-likelihood error discussed below, it’s infinitely bad, because we will eventually see that class, but our model will say it’s impossible.) The empirical relative frequency estimator is in a sense too reckless in following the data, without allowing for the possibility that it the data are wrong; it may under-smooth. Other probability estimators “shrink away” or “back off” from the empirical relative frequencies; Exercise 1 involves one such estimator.

For point forecasts, the best strategy depends on the loss function. If it is just the mis-classification rate, then the best prediction at each leaf is the class with the highest conditional probability in that leaf. With other loss functions, we should make the guess which minimizes the expected loss. But this leads us to the topic of measuring error.

14.3.3 Measuring Error

There are three common ways of measuring error for classification trees, or indeed other classification algorithms: misclassification rate, expected loss, and normalized negative log-likelihood, a.k.a. **cross-entropy**.

14.3.3.1 Misclassification Rate

We’ve already seen this: it’s the fraction of cases assigned to the wrong class.

14.3.3.2 Average Loss

The idea of the average loss is that some errors are more costly than others. For example, we might try classifying cells into “cancerous” or “not cancerous” based on their gene expression profiles⁶. If we think a healthy cell from someone’s biopsy is cancerous, we refer them for further tests, which are frightening and unpleasant, but not, as the saying goes, the end of the world. If we think a cancer cell is healthy, th

⁶Think back to Homework 4, only there *all* the cells were cancerous, and the question was just “which cancer?”

consequences are much more serious! There will be a different cost for each combination of the real class and the guessed class; write L_{ij} for the cost (“loss”) we incur by saying that the class is j when it’s really i .

For an observation x , the classifier gives class probabilities $\Pr(Y = i|X = x)$. Then the expected cost of predicting j is:

$$\text{Loss}(Y = j|X = x) = \sum_i L_{ij} \Pr(Y = i|X = x)$$

A cost matrix might look as follows

		prediction	
		“cancer”	“healthy”
truth	“cancer”	0	100
	“healthy”	1	0

We run an observation through the tree and wind up with class probabilities $(0.4, 0.6)$. The most likely class is “healthy”, but it is not the most cost-effective decision. The expected cost of predicting “cancer” is $0.4 * 0 + 0.6 * 1 = 0.6$, while the expected cost of predicting “healthy” is $0.4 * 100 + 0.6 * 0 = 40$. The probability of $Y = \text{“healthy”}$ must be 100 times higher than that of $Y = \text{“cancer”}$ before “cancer” is a cost-effective prediction.

Notice that if our estimate of the class probabilities is very bad, we can go through the math above correctly, but still come out with the wrong answer. If our estimates were exact, however, we’d always be doing as well as we could, given the data.

You can show (and will, in the homework!) that if the costs are symmetric, we get the mis-classification rate back as our error function, and should always predict the most likely class.

14.3.3.3 Likelihood and Cross-Entropy

The normalized negative log-likelihood is a way of looking not just at whether the model made the wrong call, but whether it made the wrong call with confidence or tentatively. (“Often wrong, never in doubt” is *not* a good idea.) More precisely, this loss function for a model Q is

$$L(\text{data}, Q) = -\frac{1}{n} \sum_{i=1}^n \log Q(Y = y_i | X = x_i)$$

where $Q(Y = y|X = x)$ is the conditional probability the model predicts. If perfect classification were possible, i.e., if Y were a function of X , then the best classifier would give the actual value of Y a probability of 1, and $L = 0$. If there is some irreducible uncertainty in the classification, then the best possible classifier would give $L = H[Y|X]$, the conditional entropy of Y given the inputs X . Less-than-ideal predictors have $L > H[Y|X]$. To see this, try re-write L so we sum over values rather

than data-points:

$$\begin{aligned}
 L &= -\frac{1}{n} \sum_{x,y} N(Y=y, X=x) \log Q(Y=y|X=x) \\
 &= -\sum_{x,y} \hat{\Pr}(Y=y, X=x) \log Q(Y=y|X=x) \\
 &= -\sum_{x,y} \hat{\Pr}(X=x) \hat{\Pr}(Y=y|X=x) \log Q(Y=y|X=x) \\
 &= -\sum_x \hat{\Pr}(X=x) \sum_y \hat{\Pr}(Y=y|X=x) \log Q(Y=y|X=x)
 \end{aligned}$$

If the quantity in the log was $\Pr(Y=y|X=x)$, this would be $H[Y|X]$. Since it's the model's estimated probability, rather than the real probability, it turns out that this is always *larger* than the conditional entropy. L is also called the **cross-entropy** for this reason.

There is a slightly subtle issue here about the difference between the in-sample loss, and the expected generalization error or risk. $N(Y=y, X=x)/n = \hat{\Pr}(Y=y, X=x)$, the empirical relative frequency or empirical probability. The law of large numbers says that this converges to the true probability, $N(Y=y, X=x)/n \rightarrow \Pr(Y=y, X=x)$ as $n \rightarrow \infty$. Consequently, the model which minimizes the cross-entropy in sample may not be the one which minimizes it on future data, though the two ought to converge. Generally, the in-sample cross-entropy is lower than its expected value.

Notice that to compare two models, or the same model on two different data sets, etc., we do not need to know the true conditional entropy $H[Y|X]$. All we need to know is that L is smaller the closer we get to the true class probabilities. If we could get L down to the cross-entropy, we would be exactly reproducing all the class probabilities, and then we could use our model to minimize any loss function we liked (as we saw above).⁷

14.3.3.4 Neyman-Pearson Approach

Using a loss function which assigns different weights to different error types has two noticeable drawbacks. First of all, we have to *pick* the weights, and this is often quite hard to do. Second, whether our classifier will do well in the future depends on getting the *same proportion* of cases in the future. Suppose that we're developing a tree to classify cells as cancerous or not from their gene expression profiles⁸. We will probably want to include lots of cancer cells in our training data, so that we can get a good idea of what cancers look like, biochemically. But, fortunately, most cells are *not* cancerous, so if doctors start applying our test to their patients, they're going to

⁷Technically, if our model gets the class probabilities right, then the model's predictions are just as informative as the original data. We then say that the predictions are a **sufficient statistic** for forecasting the class. In fact, if the model gets the exact probabilities wrong, but has the correct partition of the feature space, then its prediction is still a sufficient statistic. Under any loss function, the optimal strategy can be implemented using *only* a sufficient statistic, rather than needing the full, original data. This is an interesting but much more advanced topic; see, e.g., Blackwell and Girshick (1954) for details.

⁸This is almost like homework 4, except there *all* the cells were from cancers of one sort or another.

find that it massively over-diagnoses cancer — it's been calibrated to a sample where the proportion (cancer):(healthy) is, say, 1:1, rather than, say, 1:20.⁹

There is an alternative to weighting which deals with both of these issues, and deserves to be better known and more widely-used than it is. This was introduced by Scott and Nowak (2005), under the name of the “Neyman-Pearson approach” to statistical learning. The reasoning goes as follows.

When we do a binary classification problem, we're really doing a hypothesis test, and the central issue in hypothesis testing, as first recognized by Neyman and Pearson, is to distinguish between the rates of different *kinds* of errors: false positives and false negatives, false alarms and misses, type I and type II. The Neyman-Pearson approach to designing a hypothesis test is to first fix a limit on the false positive probability, the size of the test, canonically α . Then, among all tests of size α , we want to minimize the false negative rate, or equivalently maximize the power, β .

In the traditional theory of testing, we know the distribution of the data under the null and alternative hypotheses, and so can (in principle) calculate α and β for any given test. This is *not* the case in data mining, but we do generally have very large samples generated under both distributions (depending on the class of the data point). If we fix α , we can ask, for any classifier — say, a tree — whether its false alarm rate is $\leq \alpha$. If so, we keep it for further consideration; if not, we discard it. Among those with acceptable false alarm rates, then, we ask “which classifier has the lowest false negative rate, the highest β ?“ This is the one we select.

Notice that this solves both problems with weighting. We don't have to pick a weight for the two errors; we just have to say what *rate* of false positives α we're willing to accept. There are many situations where this will be easier to do than to fix on a relative cost. Second, the rates α and β are properties of the *conditional* distributions of the features, $\Pr(X|Y)$. If those conditional distributions stay the same but the proportions of the classes change, then the error rates are unaffected. Thus, training the classifier with a different mix of cases than we'll encounter in the future is not an issue.

Unfortunately, I don't know of any R implementation of Neyman-Pearson learning; it wouldn't be hard, I think, but goes beyond one problem set at this level.

14.4 Further Reading

The classic book on prediction trees, which basically introduced them into statistics and data mining, is Breiman *et al.* (1984). Chapter three in Berk (2008) is clear, easy to follow, and draws heavily on Breiman et al. Another very good chapter is the one on trees in Ripley (1996), which is especially useful for us because Ripley wrote the tree package. (The whole book is strongly recommended if you plan to go further in data-mining.) There is another tradition of trying to learn tree-structured models which comes out of artificial intelligence and inductive logic; see Mitchell (1997).

The clearest explanation of the Neyman-Pearson approach to hypothesis testing I have ever read is that in Reid (1982), which is one of the books which made me decide

⁹Cancer is rarer than that, but realistically doctors aren't going to run a test like this unless they have some reason to suspect cancer might be present.

to learn statistics.

14.5 Exercises

1. Repeat the analysis of the California house-price data with the Pennsylvania data.
2. Suppose that we see each of k classes n_i times, with $\sum_{i=1}^k n_i = n$. The maximum likelihood estimate of the probability of the i^{th} class is $\hat{p}_i = n_i/n$. Suppose that instead we use the estimates

$$\tilde{p}_i = \frac{n_i + 1}{\sum_{j=1}^k n_j + 1} \quad (14.4)$$

This estimator goes back to Laplace, who called it the “rule of succession”.

Show that the \tilde{p}_i sum up to one. Show, using the law of large numbers, that $\tilde{p} \rightarrow p$ when $\hat{p} \rightarrow p$. Do these properties still hold if the +1s in the numerator and denominator are replaced by $+d$ for an arbitrary $d > 0$?

3. *Fun with Laplace's rule of succession: will the Sun rise tomorrow?* One illustration Laplace gave of this probability estimator was the following. Suppose we know, from written records, that the Sun has risen in the east every day for the last 4000 years.¹⁰

- (a) Calculate the probability of the event “the Sun will rise in the east tomorrow”, using Eq. 14.4. You may take the year as containing 365.256 days.
- (b) Calculate the probability that the Sun will rise in the east *every day for the next four thousand years*, assuming this is an IID event. Is this a reasonable assumption?
- (c) Calculate the probability of the event “the Sun will rise in the east every day for four thousand years” directly from Eq. 14.4. Does your answer agree with part (b)? Should it?

Laplace did not, of course, base his belief that the Sun will rise in the morning on such calculations; besides everything else, he was the world's expert in celestial mechanics! But this shows the problem with the

4. Show that, when all the off-diagonal elements of L_{ij} are equal (and positive!), the best class to predict is always the most probable class.

¹⁰Laplace was thus ignoring people who live above the Arctic circle, or below the Antarctic circle. The latter seems particularly unfair, because so many of them are scientists.

Part II

Multivariate Data, Distributions, and Latent Structure

Chapter 15

Multivariate Distributions

TODO: a proper opening!

15.1 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

$$F(\vec{a}) = F(a_1, a_2, \dots, a_p) = \Pr(X_1 \leq a_1, X_2 \leq a_2, \dots, X_p \leq a_p) \quad (15.1)$$

Thus

$$F(\vec{b}) - F(\vec{a}) = \Pr(a_1 < X_1 \leq b_1, a_2 < X_2 \leq b_2, \dots, a_p < X_p \leq b_p) \quad (15.2)$$

This is the probability that X is in a (hyper-)rectangle, rather than just in an interval.

The **joint probability density function** is

$$p(\vec{x}) = p(x_1, x_2, \dots, x_p) = \frac{\partial^p F(a_1, \dots, a_p)}{\partial a_1 \dots \partial a_p} \Big|_{\vec{a}=\vec{x}} \quad (15.3)$$

Of course,

$$F(\vec{a}) = \int_{-\infty}^{a_1} \int_{-\infty}^{a_2} \dots \int_{-\infty}^{a_p} p(x_1, x_2, \dots, x_p) dx_p \dots dx_2 dx_1 \quad (15.4)$$

(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 ,

$$p(x_1, x_2, \dots, x_q) = \int p(x_1, x_2, \dots, x_p) dx_{q+1} dx_{q+2} \dots dx_p \quad (15.5)$$

(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:

$$p(x_{q+1}, x_{q+2}, \dots, x_p | X_1 = x_1, \dots, X_q = x_q) = \frac{p(x_1, x_2, \dots, x_p)}{p(x_1, x_2, \dots, x_q)} \quad (15.6)$$

These two tricks can be iterated, so, for instance,

$$p(x_3 | x_1) = \int p(x_3, x_2 | x_1) dx_2 \quad (15.7)$$

[[TODO: Some non-Gaussian examples]]

15.2 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

$$p(x) = (2\pi\sigma^2)^{-1/2} \exp\left\{-\frac{1}{2}\frac{(x - \mu)^2}{\sigma^2}\right\} \quad (15.8)$$

the probability density of the multivariate Gaussian is

$$p(\vec{x}) = (2\pi \det \Sigma)^{-p/2} \exp\left\{-\frac{1}{2}(\vec{x} - \vec{\mu}) \cdot \Sigma^{-1}(\vec{x} - \vec{\mu})\right\} \quad (15.9)$$

Finally, remember that the parameters of a Gaussian change along with linear transformations

$$X \sim \mathcal{N}(\mu, \sigma^2) \Leftrightarrow aX + b \sim \mathcal{N}(a\mu + b, a^2\sigma^2) \quad (15.10)$$

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

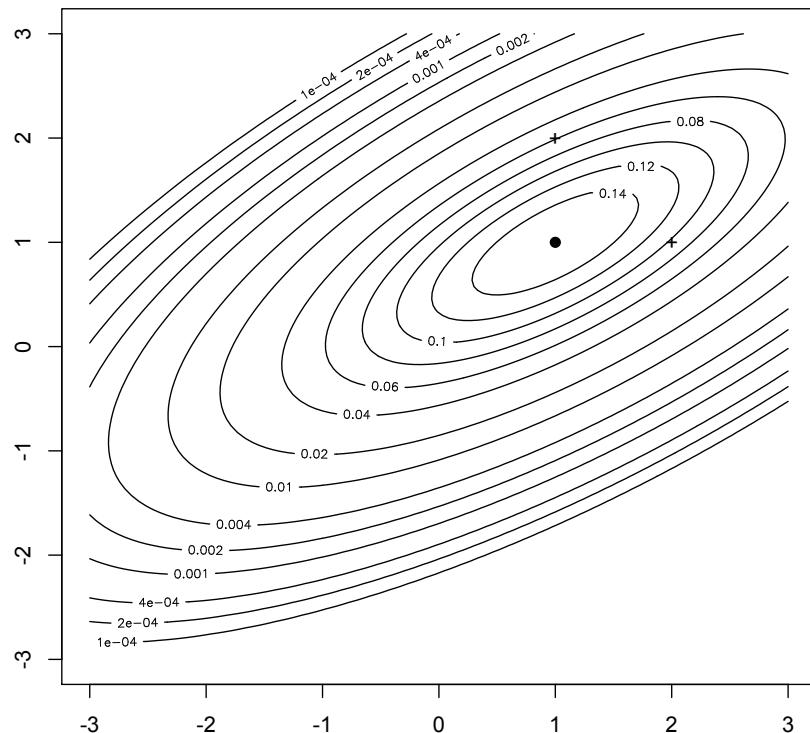
$$\vec{X} \sim \mathcal{MVN}(\vec{\mu}, \Sigma) \quad (15.11)$$

then

$$a\vec{X} + \vec{b} \sim \mathcal{MVN}(a\vec{\mu} + \vec{b}, a\Sigma a^T) \quad (15.12)$$

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 15.1. 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 15.1: 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.

15.2.1 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:

$$\Sigma = \mathbf{v}^T \mathbf{d} \mathbf{v} \quad (15.13)$$

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

$$(\mathbf{v}^T \mathbf{d} \mathbf{v})^{-1} = \mathbf{v}^{-1} \mathbf{d}^{-1} (\mathbf{v}^T)^{-1} = \mathbf{v}^T \mathbf{d}^{-1} \mathbf{v} \quad (15.14)$$

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

$$f_0 = (2\pi \det \Sigma)^{-p/2} \exp \left\{ -\frac{1}{2} (\vec{x} - \vec{\mu}) \cdot \Sigma^{-1} (\vec{x} - \vec{\mu}) \right\} \quad (15.15)$$

$$c = (\vec{x} - \vec{\mu}) \cdot \Sigma^{-1} (\vec{x} - \vec{\mu}) \quad (15.16)$$

$$= (\vec{x} - \vec{\mu})^T \mathbf{v}^T \mathbf{d}^{-1} \mathbf{v} (\vec{x} - \vec{\mu}) \quad (15.17)$$

$$= (\vec{x} - \vec{\mu})^T \mathbf{v}^T \mathbf{d}^{-1/2} \mathbf{d}^{-1/2} \mathbf{v} (\vec{x} - \vec{\mu}) \quad (15.18)$$

$$= (\mathbf{d}^{-1/2} \mathbf{v} (\vec{x} - \vec{\mu}))^T (\mathbf{d}^{-1/2} \mathbf{v} (\vec{x} - \vec{\mu})) \quad (15.19)$$

$$= \|\mathbf{d}^{-1/2} \mathbf{v} (\vec{x} - \vec{\mu})\|^2 \quad (15.20)$$

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 18.

15.2.2 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

$$X_2 | X_1 = x_1 \sim \mathcal{N}(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}) \quad (15.21)$$

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

$$\vec{X}_B | \vec{X}_A = \vec{x}_a \sim \mathcal{MVN}(\vec{\mu}_B + \Sigma_{BA}\Sigma_{AA}^{-1}(\vec{x}_a - \vec{\mu}_A), \Sigma_{BB} - \Sigma_{BA}\Sigma_{AA}^{-1}\Sigma_{AB}) \quad (15.22)$$

(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 .

15.2.3 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.

15.2.4 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 be used in Chapter 21 for mixture models, includes functions for the multivariate Gaussian density and for random-vector generation.

15.3 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 16.

15.3.1 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 choose 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 in 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

$$L(\theta) = \prod_{i=1}^n p(\vec{x}_i; \theta) \quad (15.23)$$

and the maximum likelihood estimate (MLE) is

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} L(\theta) \quad (15.24)$$

Again, as in the univariate case, it is usually simpler and more stable to use the log-likelihood:

$$\ell(\theta) = \sum_{i=1}^n \log p(\vec{x}_i; \theta) \quad (15.25)$$

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.

[[TODO: Move results on maximum likelihood estimation to Appendix E.2]]

making use of the fact that

$$\operatorname{argmax}_{\theta} L(\theta) = \operatorname{argmax}_{\theta} \ell(\theta) \quad (15.26)$$

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

$$\hat{\theta}_{MLE} \rightsquigarrow \mathcal{MVN}(\theta_0, -\mathbf{H}^{-1}(\theta_0)) \quad (15.27)$$

where θ_0 is the true parameter value, and \mathbf{H} is the **Hessian** of the log-likelihood, its matrix of second partial derivatives,

$$H_{jk}(\theta) = \left. \frac{\partial^2 \ell}{\partial \theta_j \partial \theta_k} \right|_{\theta} \quad (15.28)$$

In turn,

$$\frac{1}{n} H_{jk}(\theta_0) \rightarrow \mathbb{E} \left[\frac{\partial^2 \log p(X; \theta_0)}{\partial \theta_j \partial \theta_k} \right] \equiv -J_{jk}(\theta_0) \quad (15.29)$$

which defines the **Fisher information matrix** \mathbf{J} . One can therefore get (approximate) confidence regions by assuming that $\hat{\theta}_{MLE}$ has a Gaussian distribution with covariance matrix $n^{-1}\mathbf{J}^{-1}(\hat{\theta}_{MLE})$ or $-\mathbf{H}^{-1}(\hat{\theta}_{MLE})$. 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. 15.27 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. 15.27, is simply parametric or non-parametric bootstrapping.

15.3.2 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

$$\ell(\hat{\Theta}) - \ell(\hat{\theta}) \geq 0 \quad (15.30)$$

Here we have a clear null hypothesis, which is that the data comes from the narrower, smaller model. Under this null hypothesis, as $n \rightarrow \infty$,

$$2[\ell(\hat{\Theta}) - \ell(\hat{\theta})] \rightsquigarrow \chi^2_{q-r} \quad (15.31)$$

provided that the restriction imposed by the small model doesn't place it on the boundary of the parameter space of Θ . (See Appendix G.)

For instance, suppose that \vec{X} is bivariate, and the larger model is an unrestricted Gaussian, so $\Theta = \left\{ (\mu_1, \mu_2), \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right\}$. A possible narrow model might impose the assumption that the components of \vec{X} are uncorrelated, so $\theta = \left\{ (\mu_1, \mu_2), \begin{bmatrix} \Sigma_{11} & 0 \\ 0 & \Sigma_{22} \end{bmatrix} \right\}$.

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

$$p(\cdot; \theta) = p(\cdot; \psi) \quad (15.32)$$

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:

$$\frac{1}{n} [\ell(\theta_0) - \ell(\psi_0)] \equiv \frac{1}{n} \sum_{i=1}^n \Lambda_i \quad (15.33)$$

$$= \frac{1}{n} \sum_{i=1}^n \log \frac{p(x_i; \theta_0)}{p(x_i; \psi_0)} \quad (15.34)$$

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, $\hat{\theta}_{MLE}$ and $\hat{\psi}_{MLE}$ 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,

$$\frac{\sqrt{n}}{n} \frac{\ell(\hat{\theta}) - \ell(\hat{\psi})}{\sigma_\Lambda^2} \rightsquigarrow \mathcal{N}(E[\Lambda], 1) \quad (15.35)$$

and that we can consistently estimate $E[\Lambda]$ 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,

$$V = \frac{1}{\sqrt{n}} \frac{\ell(\hat{\theta}) - \ell(\hat{\psi})}{\hat{\sigma}_\Lambda} \rightsquigarrow \mathcal{N}(0, 1) \quad (15.36)$$

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

15.3.3 Goodness-of-Fit

For univariate distributions, we often assess goodness-of-fit through the Kolmogorov-Smirnov (KS) test⁶, where the test statistic is

$$d_{KS} = \max_a |\hat{F}_n(a) - F(a)| \quad (15.37)$$

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. 15.31. 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 correction for multiple testing.

[[ATTN: Multiple comparisons needs to be an appendix topic]]

15.4 Exercises

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 §15.2.4.
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 $\mathbf{c}\vec{X}$.

[[TODO: Move review material into an appendix]]

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 16

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 15 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?

16.1 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?

16.2 “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 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

$$\tilde{F}_n(a) \equiv \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{(-\infty, a]}(x_i) \quad (16.1)$$

In words, this is just the fraction of the samples which are $\leq a$. Then the **Glivenko-Cantelli theorem** says

$$\max_a |\tilde{F}_n(a) - F(a)| \rightarrow 0 \quad (16.2)$$

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 . Here’s a sketch, with no details. Fix an $\epsilon > 0$; first show that there is some *finite* set of points on the line, call them $b_1, \dots, b_{m(\epsilon)}$, 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 $m(\epsilon)$ 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. This general strategy — combining pointwise convergence theorems with approximation arguments — forms

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?

[[TODO: Mention DKW inequality and corresponding confidence band for the CDF]]

16.3 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. $\int f(x) \log \frac{f(x)}{\hat{f}(x)} dx$ 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

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 (§16.6), but, like most texts on density estimation, we will give more attention to minimizing (1), because it's mathematically tractable.

Notice that

$$\int (f(x) - \hat{f}(x))^2 dx = \int f^2(x) dx - 2 \int \hat{f}(x)f(x)dx + \int \hat{f}^2(x)dx \quad (16.3)$$

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

$$-\frac{2}{n} \sum_{i=1}^n \hat{f}(x_i) \quad (16.4)$$

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

$$-\frac{2}{n} \sum_{i=1}^n \hat{f}(x_i) + \int \hat{f}^2(x)dx \quad (16.5)$$

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.

16.3.1 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 $x_0 + b$. We want to estimate the density at x , and this is

$$\hat{f}_n(x) = \frac{1}{h} \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{(x_0, x_0+b]}(x_i) \quad (16.6)$$

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, $p = F(x_0 + b) - F(x_0)$. The mean of B is np , and the variance is $np(1-p)$, so

$$\mathbb{E} [\hat{f}_n(x)] = \frac{1}{nb} \mathbb{E}[B] \quad (16.7)$$

$$= \frac{n[F(x_0 + b) - F(x_0)]}{nb} \quad (16.8)$$

$$= \frac{F(x_0 + b) - F(x_0)}{b} \quad (16.9)$$

⁴Admittedly, in high-dimensional spaces, doing the final integral can become numerically challenging.

and the variance is

$$\text{Var} [\hat{f}_n(x)] = \frac{1}{n^2 h^2} \text{Var}[B] \quad (16.10)$$

$$= \frac{n[F(x_0 + h) - F(x_0)][1 - F(x_0 + h) + F(x_0)]}{n^2 h^2} \quad (16.11)$$

$$= E[\hat{f}_n(x)] \frac{1 - F(x_0 + h) + F(x_0)}{nh} \quad (16.12)$$

If we let $h \rightarrow 0$ as $n \rightarrow \infty$, then

$$E[\hat{f}_h(x)] \rightarrow \lim_{h \rightarrow 0} \frac{F(x_0 + h) - F(x_0)}{h} = f(x_0) \quad (16.13)$$

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 $h \rightarrow 0$ but $nh \rightarrow \infty$ as $n \rightarrow \infty$. The bin-width h needs to shrink, but slower than n^{-1} .

At what rate should it shrink? Small h 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 $E[\hat{f}_h(x)]$ 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

$$\int (f(x) - E[\hat{f}_h(x)])^2 dx = \frac{h^2}{12} \int (f'(x))^2 dx + o(h^2) \quad (16.14)$$

We already got the variance at x , and when we integrate that over x we find

$$\int \text{Var}[\hat{f}_h(x)] dx = \frac{1}{nh} + o(n^{-1}) \quad (16.15)$$

So the total integrated squared error is

$$\text{ISE} = \frac{h^2}{12} \int (f'(x))^2 dx + \frac{1}{nh} + o(h^2) + o(n^{-1}) \quad (16.16)$$

Differentiating this with respect to h and setting it equal to zero, we get

$$\frac{h_{\text{opt}}}{6} \int (f'(x))^2 dx = \frac{1}{nh_{\text{opt}}^2} \quad (16.17)$$

⁵You need to use the intermediate value theorem multiple times; see for instance Wasserman (2006, sec. 6.8).

$$h_{\text{opt}} = \left(\frac{6}{\int (f'(x))^2 dx} \right)^{1/3} n^{-1/3} = O(n^{-1/3}) \quad (16.18)$$

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.

16.4 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

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^n \frac{1}{h} K\left(\frac{x - x_i}{h}\right) \quad (16.19)$$

where K is a kernel function such as we encountered when looking at kernel regression. (The factor of $1/h$ 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

16.4.1 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.

$$\mathbb{E} \left[\hat{f}_b(x) \right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E} \left[\frac{1}{b} K \left(\frac{x - X_i}{b} \right) \right] \quad (16.20)$$

$$= \mathbb{E} \left[\frac{1}{b} K \left(\frac{x - X}{b} \right) \right] \quad (16.21)$$

$$= \int \frac{1}{b} K \left(\frac{x - t}{b} \right) f(t) dt \quad (16.22)$$

$$= \int K(u) f(x - bu) du \quad (16.23)$$

$$= \int K(u) \left[f(x) - bu f'(x) + \frac{b^2 u^2}{2} f''(x) + o(b^2) \right] du \quad (16.24)$$

$$= f(x) + \frac{b^2 f''(x)}{2} \int K(u) u^2 du + o(b^2) \quad (16.25)$$

$$(16.26)$$

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

$$\mathbb{E} \left[\hat{f}_b(x) \right] - f(x) = \frac{b^2 \sigma_K^2 f''(x)}{2} + o(b^2) \quad (16.27)$$

So the bias will go to zero if the bandwidth b shrinks to zero. What about the variance? Use Taylor's theorem again:

$$\text{Var} \left[\hat{f}_b(x) \right] = \frac{1}{n} \text{Var} \left[\frac{1}{b} K \left(\frac{x - X}{b} \right) \right] \quad (16.28)$$

$$= \frac{1}{n} \left[\mathbb{E} \left[\frac{1}{b^2} K^2 \left(\frac{x - X}{b} \right) \right] - \left(\mathbb{E} \left[\frac{1}{b} K \left(\frac{x - X}{b} \right) \right] \right)^2 \right] \quad (16.29)$$

$$= \frac{1}{n} \left[\int \frac{1}{b^2} K^2 \left(\frac{x - t}{b} \right) dt - [f(x) + O(b^2)]^2 \right] \quad (16.30)$$

$$= \frac{1}{n} \left[\int \frac{1}{b} K^2(u) f(x - bu) du - f^2(x) + O(b^2) \right] \quad (16.31)$$

$$= \frac{1}{n} \left[\int \frac{1}{b} K^2(u) (f(x) - bu f'(x)) du - f^2(x) + O(b) \right] \quad (16.32)$$

$$= \frac{f(x)}{bn} \int K^2(u) du + O(1/n) \quad (16.33)$$

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,

$$\frac{b^4 \sigma_K^4 (f''(x))^2}{4} + \frac{f(x)}{bn} \int K^2(u) du + \text{small} \quad (16.34)$$

the expected integrated squared error is

$$\text{ISE} \approx \frac{b^4 \sigma_K^4}{4} \int (f''(x))^2 dx + \frac{\int K^2(u) du}{nb} \quad (16.35)$$

Differentiating with respect to b for the optimal bandwidth b_{opt} , we find

$$b_{\text{opt}}^3 \sigma_K^4 \int (f''(x))^2 dx = \frac{\int K^2(u) du}{nb_{\text{opt}}^2} \quad (16.36)$$

$$b_{\text{opt}} = \left(\frac{\int K^2(u) du}{\sigma_K^4 \int (f''(x))^2 dx} \right)^{1/5} n^{-1/5} = O(n^{-1/5}) \quad (16.37)$$

That is, the best bandwidth goes to zero like one over the fifth root of the number of sample points. Plugging this into Eq. 16.35, 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. 16.37 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. 16.37 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. 16.37 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.

16.4.2 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:

$$\hat{f}(\vec{x}) = \frac{1}{n} \sum_{i=1}^n K(\vec{x} - \vec{x}_i) \quad (16.38)$$

⁷Substituting Eq. 16.37 into Eq. 16.35 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,

$$K(\vec{x} - \vec{x}_i) = K_1(x^1 - x_i^1)K_2(x^2 - x_i^2)\dots K_p(x^p - x_i^p), \quad (16.39)$$

Doing this requires a bandwidth for each coordinate, so the over-all form of the joint PDF estimate is

$$\hat{f}(\vec{x}) = \frac{1}{n \prod_{j=1}^p b_j} \sum_{i=1}^n \prod_{j=1}^d K_j \left(\frac{x^j - x_i^j}{b_j} \right) \quad (16.40)$$

Going through a similar analysis for p -dimensional data shows that the ISE goes to zero like $O(n^{-4/(4+p)})$, and again, if we use cross-validation to pick the bandwidths, asymptotically we attain this rate. Unfortunately, if p is large, this rate becomes very slow — for instance, if $p = 24$, the rate is $O(n^{-1/7})$. There is simply no universally good way to learn *arbitrary* high-dimensional distributions. This is the same “curse of dimensionality” we saw in regression (§9.3). The fundamental problem is that in high dimensions, there are just too different possible distributions which are too hard to tell apart.

Evading the curse of dimensionality for density estimation needs some special assumptions. Parametric models make the very strong assumption that we know exactly what the distribution function looks like, and we just need to fill in a few constants. It’s potentially less drastic to hope the distribution has some sort of special structure we can exploit, and most of the rest of Part II will be about searching for various sorts of useful structure⁸. If none of these options sound appealing, or plausible, we’ve got little alternative but to accept a very slow convergence of density estimates.

16.4.3 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,

⁸As Wiener (1956), the reason the ability to do nonparametric estimation doesn’t make scientific theories redundant is that good theories usefully constrain the distributions we’re searching for, and tell us what structures to look for.

then, for $0 \leq b < 1$,

$$K(x_1, x_2) = \begin{cases} 1-b & x_1 = x_2 \\ b/c & x_1 \neq x_2 \end{cases} \quad (16.41)$$

is a valid kernel. For an ordered x ,

$$K(x_1, x_2) = \binom{c}{|x_1 - x_2|} b^{|x_1 - x_2|} (1-b)^{c-|x_1 - x_2|} \quad (16.42)$$

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.

16.4.4 Practicalities

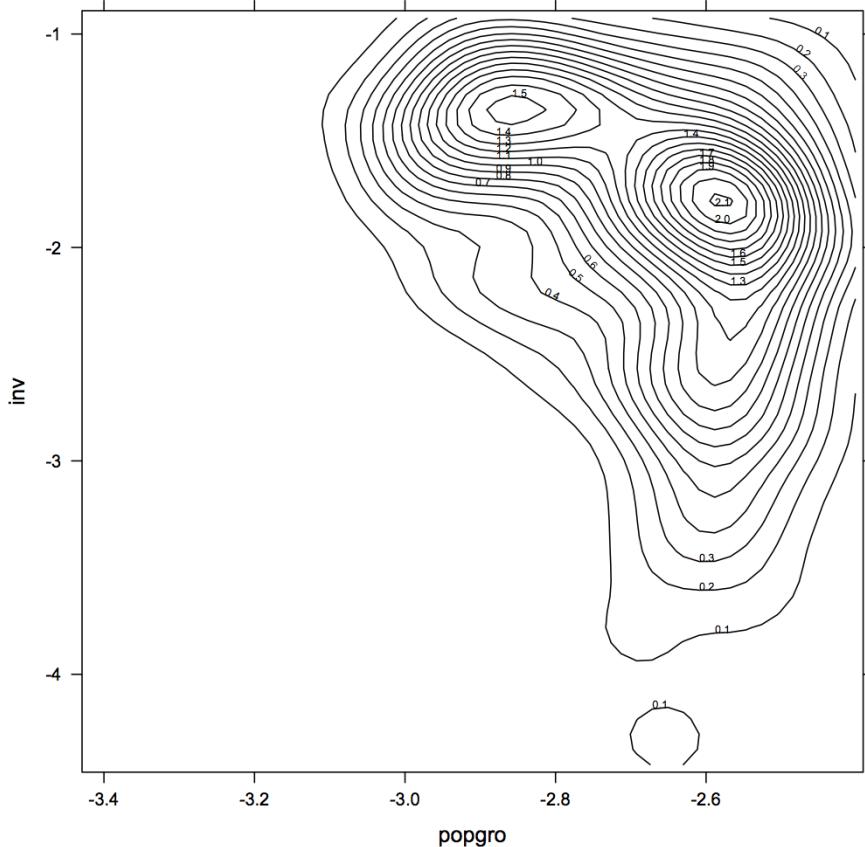
The standard R function `density` implements one-dimensional kernel density estimation, defaulting to Gaussian kernels with the rule-of-thumb bandwidth. There are some options for doing cleverer bandwidth selection, including a plug-in rule. (See the help file.)

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`.

16.4.5 Kernel Density Estimation in R: An Economic Example

The data set `oecdpanel1`, 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(oecdpanel1)` for details.

Here's an example of using `npudens` with variables from the `oecdpanel1` 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 16.1 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 16.1: 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.

16.5 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

$$\hat{f}_{Y|X}(y | x) = \frac{\hat{f}_{X,Y}(x,y)}{\hat{f}_X(x)} \quad (16.43)$$

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:

$$\hat{f}_{X,Y}(x,y) = \frac{1}{nb_X b_Y} \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) K_Y\left(\frac{y-y_i}{b_Y}\right) \quad (16.44)$$

$$\hat{f}_X(x) = \frac{1}{nb_X} \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) \quad (16.45)$$

Thus we need to pick two bandwidths, b_X and b_Y , one for each variable.

This might seem like a solved problem — we just use cross-validation to find b_X and b_Y so as to minimize the integrated squared error for $\hat{f}_{X,Y}$, and then plug in to Equation 16.43. 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 $b_X = \infty$. (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

$$\int dx f_X(x) \int dy \left(f_{Y|X}(y | x) - \hat{f}_{Y|X}(y | x) \right)^2 \quad (16.46)$$

and this is what we want to minimize by picking b_X and b_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, $\hat{b}_U \rightarrow \infty$ 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.

16.5.1 Practicalities and a Second Example

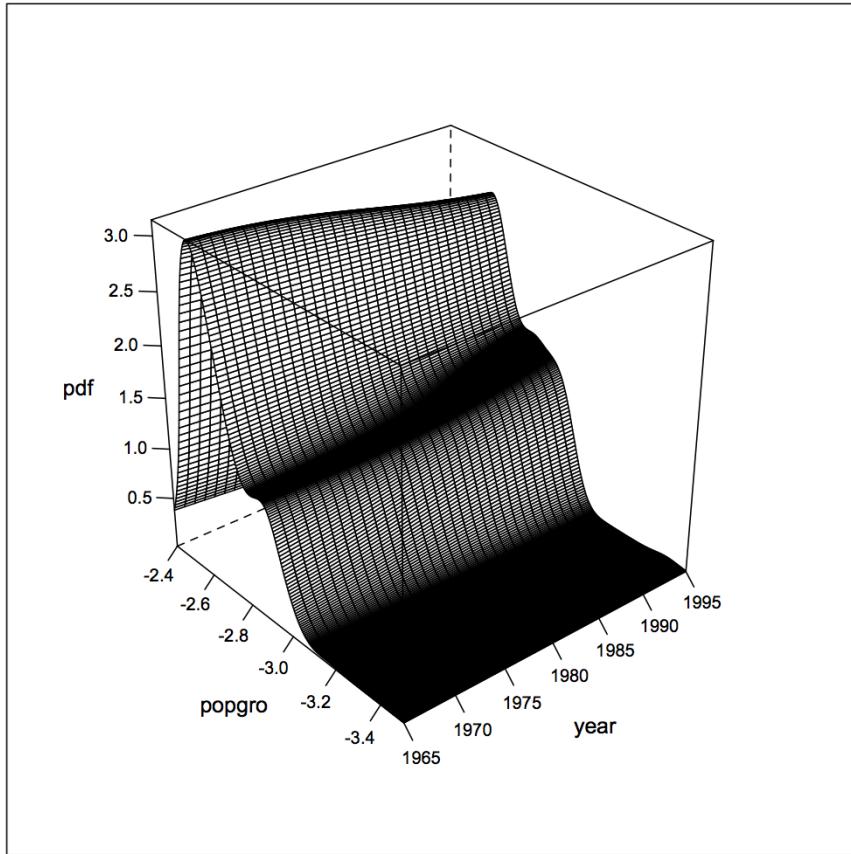
The `np` package implements kernel conditional density estimation through the function `npcdens`. The syntax is pretty much exactly like that of `npreg`, and indeed we can think of estimating the conditional density as a sort of regression, where the dependent variable is actually a distribution.

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 16.2). 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". Over time, Figure 16.2 population growth rates tend to shrink, and the dispersion of growth rates narrows.

Let's expand on this point. One of the variables in `oecdpanel` is `oecd`, 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 16.3).

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.

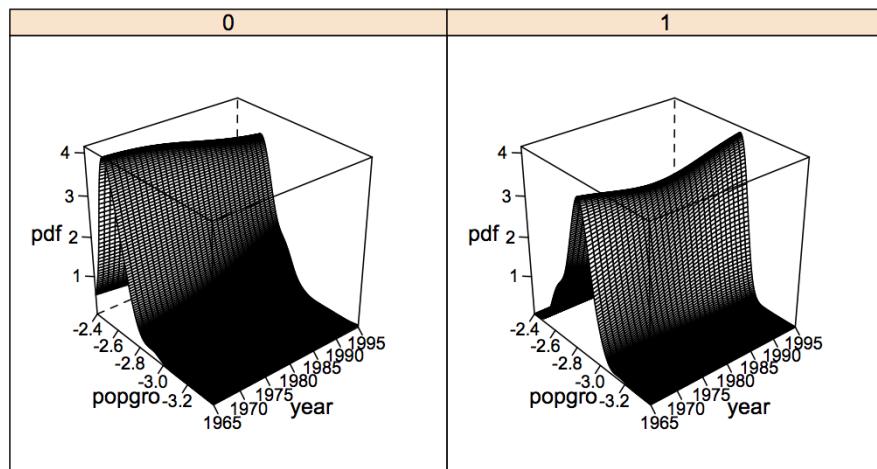


```

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 16.2: 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 16.3: 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.

16.6 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:

$$D(f||\hat{f}) = - \int f(x) \log \hat{f}(x) dx + \int f(x) \log f(x) dx \quad (16.47)$$

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:

$$\int f(x) \log \hat{f}(x) dx \approx \frac{1}{n} \sum_{i=1}^n \log \hat{f}(x_i) \quad (16.48)$$

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:

$$\frac{1}{n} \sum_{i=1}^n \log \left(\frac{1}{nb} \sum_{j=1}^n K\left(\frac{x_j - x_i}{b}\right) \right) = -\log nb + \frac{1}{n} \sum_{i=1}^n \log \left(\sum_{j=1}^n K\left(\frac{x_j - x_i}{b}\right) \right) \quad (16.49)$$

If we take b to be very small, $K\left(\frac{x_j - x_i}{b}\right) \approx 0$ unless $x_j = x_i$, so the over-all likelihood becomes

$$\approx -\log nb + \log K(0) \quad (16.50)$$

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

$$\tilde{f}(x) = \frac{1}{n} \sum_{i=1}^n \delta(x - x_i) \quad (16.51)$$

where δ is the Dirac delta function.⁹ Of course, this is just the same distribution as the empirical CDF.

Why is maximum likelihood failing us here? 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 values of x we *didn't* see can only come at the expense

⁹Recall 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 bona fide function, you're right; strictly speaking it's just a linear operator on 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. But this is where we came in...

of the probability of observed values, so Eq. 16.51 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. 16.48 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").

16.7 Simulating from Density Estimates

16.7.1 Simulating from Kernel Density Estimates

There are times when one wants to draw random values 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 pattern goes like so. 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)`.¹⁰ Using a different kernel, we'd just need to use the random number generator function for the corresponding distribution.

To see that this gives the right distribution needs 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 KDE gives to any set A is just an integral:

$$\hat{F}(A) = \int_A \hat{f}(x) dx \quad (16.52)$$

$$= \int_A \frac{1}{n} \sum_{i=1}^n K(x, x_i, h) dx \quad (16.53)$$

$$= \frac{1}{n} \sum_{i=1}^n \int_A K(x, x_i, h) dx \quad (16.54)$$

$$= \frac{1}{n} \sum_{i=1}^n C(A, x_i, h) \quad (16.55)$$

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

¹⁰In 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

$$\Pr(\tilde{X} \in A) = E[\mathbf{1}_A(\tilde{X})] \quad (16.56)$$

$$= E[E[\mathbf{1}_A(\tilde{X})|J]] \quad (16.57)$$

$$= E[C(A, x_j, b)] \quad (16.58)$$

$$= \frac{1}{n} \sum_{i=1}^n C(A, x_i, b) \quad (16.59)$$

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 .

16.7.1.1 Sampling from a Joint Density

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 draw each coordinate independently from the kernel distribution centered on our random point. (See Code Example 30 below.) The argument for correctness actually goes exactly as before.

16.7.1.2 Sampling from a Conditional Density

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:

$$\hat{F}(Y \in A | X = x) \quad (16.60)$$

$$= \int_A \hat{f}_{Y|X}(y | x) dy \\ = \int_A \frac{\frac{1}{nb_X b_Y} \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) K_Y\left(\frac{y-y_i}{b_Y}\right)}{\hat{f}_X(x)} dy \quad (16.61)$$

$$= \frac{1}{nb_X b_Y \hat{f}_X(x)} \int_A \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) K_Y\left(\frac{y-y_i}{b_Y}\right) dy \quad (16.62)$$

$$= \frac{1}{nb_X b_Y \hat{f}_X(x)} \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) \int_A K_Y\left(\frac{y-y_i}{b_Y}\right) dy \quad (16.63)$$

$$= \frac{1}{nb_X \hat{f}_X(x)} \sum_{i=1}^n K_X\left(\frac{x-x_i}{b_X}\right) C_Y(A, y_i, b_Y) \quad (16.64)$$

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.

16.7.2 Drawing 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.

16.7.3 Examples of Simulating from Kernel Density Estimates

To make all this more concrete, let's continue working with the `oecdpanel` data. Section 16.4.5 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 gives Figure 16.4.

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 rate¹¹. Extracting this by direct calculation from `popinv2` would not be easy; we'd need to do the integral

$$\int_{x=0}^1 \int_{y=0}^1 \frac{y}{x} \hat{f}_{X,Y}(x,y) dy dx \quad (16.65)$$

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:

$$\frac{1}{T} \sum_{i=1}^T \frac{\tilde{Y}_i}{\tilde{X}_i} = \tilde{g}_T \xrightarrow{T \rightarrow \infty} E\left[\frac{Y}{X}\right] \quad (16.66)$$

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:

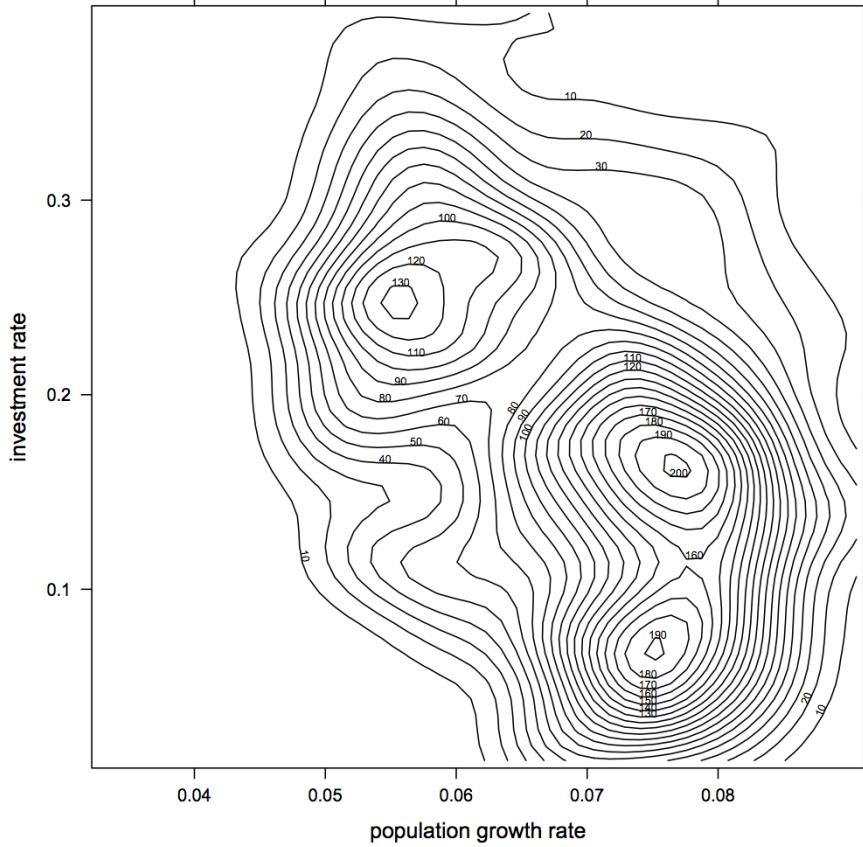
$$\tilde{g}_T \rightsquigarrow \mathcal{N}(E[Y/X], \text{Var}[\tilde{g}_1]/\sqrt{T}) \quad (16.67)$$

How do we find the variance $\text{Var}[\tilde{g}_1]$? We approximate it by simulating.

Code Example 30 is a function which draws 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:

```
signif(mean(exp(oecdpanel$popgro)),3)
## [1] 0.0693
signif(mean(exp(oecdpanel$inv)),3)
## [1] 0.172
signif(colMeans(rpopinv(200)),3)
## pop.growth.rate    invest.rate
##                 0.0698          0.1750
```

¹¹Economically, we might want to know this because it would tell us about how quickly the capital stock per person grows.



```
popinv2 <- npudens(~exp(popgro)+exp(inv), data=oecdpanel)
```

FIGURE 16.4: *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 30: *Simulating from the fitted kernel density estimate popinv2. Can you see how to modify it to draw from other bivariate density estimates produced by npudens? From higher-dimensional distributions? Can you replace the for loop with less iterative code?*

This is pretty satisfactory for only 200 samples, so the simulator seems to be working. Now we just use it:

```

z <- rpopinv(2000)
signif(mean(z[, "invest.rate"] / z[, "pop.growth.rate"]), 3)
## [1] 2.64
signif(sd(z[, "invest.rate"] / z[, "pop.growth.rate"]) / sqrt(2000), 3)
## [1] 0.0351

```

This tells us that $E[Y/X] \approx 2.64 \pm 0.035$.

Suppose we want not the mean of Y/X but the median?

```

signif(median(z[, "invest.rate"] / z[, "pop.growth.rate"]), 3)
## [1] 2.32

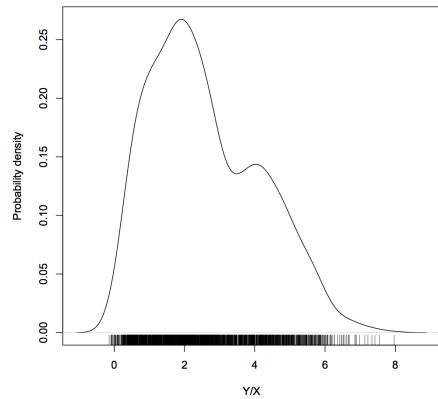
```

Getting the whole distribution of Y/X is not much harder (Figure 16.5). 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 16.6).

Of course, if we are going to do multiple simulations, we could just average them together. Say that $\tilde{g}_T^{(1)}, \tilde{g}_T^{(2)}, \dots, \tilde{g}_T^{(s)}$ are estimates of our statistic of interest from s independent realizations of the model, each of size T . We can just combine them into one grand average:

$$\tilde{g}_{s,T} = \frac{1}{s} \sum_{i=1}^s \tilde{g}_T^{(i)} \quad (16.68)$$

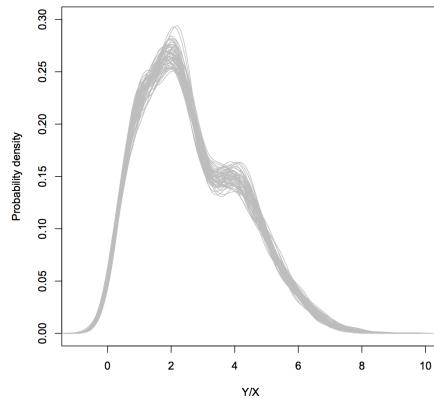
As an average of IID quantities, the variance of $\tilde{g}_{s,T}$ is $1/s$ times the variance of $\tilde{g}_T^{(1)}$.



```
YoverX <- z[, "invest.rate"]/z[, "pop.growth.rate"]
plot(density(YoverX), xlab="Y/X", ylab="Probability density", main="")
rug(YoverX, side=1)
```

FIGURE 16.5: *Distribution of Y/X implied by the joint density estimate popinv2.*

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 16.6: *Showing the sampling variability in the distribution of Y/X by “over-plotting”. Each line is a distribution from an estimated sample of size 2000, as in Figure 16.5; 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.)*

16.8 Exercises

1. Reproduce Figure 16.4?
2. Qualitatively, is this compatible with Figure 16.1?
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?
5. You are given a kernel K which satisfies $K(u) \geq 0$, $\int K(u)du = 1$, $\int uK(u)du = 0$, $\int u^2K(u)du = \sigma_K^2 < \infty$. You are also given a bandwidth $h > 0$, and a collection of n univariate observations x_1, x_2, \dots, x_n . Assume that the data are independent samples from some unknown density f .
 - (a) Give the formula for \hat{f}_h , the kernel density estimate corresponding to these data, this bandwidth, and this kernel.
 - (b) Find the expectation of a random variable whose density is \hat{f}_h , in terms of the sample moments, h , and the properties of the kernel function.
 - (c) Find the variance of a random variable whose density is \hat{f}_h , in terms of the sample moments, h , and the properties of the kernel function.
 - (d) How must h change as n grows to ensure that the expectation and variance of \hat{f}_h will converge on the expectation and variance of f ?
6. Many variables have natural range restrictions, like being non-negative, or lie in some interval. Kernel density estimators don't respect these restrictions, so they can give positive probability density to impossible values. One way around this is the *transformation method* (or "trick"): use an invertible function q to map the limited range of X to the whole real line, find the density of the transformed variable, and then undo the transformation.

In what follows, X is a random variable with pdf f , Y is a random variable with pdf g , and $Y = q(X)$, for a known function q . You may assume that q is continuous, differentiable and monotonically increasing, inverse q^{-1} exists, and is also continuous, differentiable and monotonically increasing.

 - (a) Find $g(y)$ in terms of f and q .
 - (b) Find $f(x)$ in terms of g and q .
 - (c) Suppose X is confined to the unit interval $[0, 1]$ and $q(x) = \log \frac{x}{1-x}$. Find $f(x)$ in terms of g and this particular q .
 - (d) The beta distribution is confined to $[0, 1]$. Draw 1000 random values from the beta distribution with both shape parameters equal to 1/2. Call this sample `x`, and plot its histogram. (Hint: `?rbeta`.)

- (e) Fit a Gaussian kernel density estimate to x , using `density`, `npudens`, or any other existing one-dimensional density estimator you like.
- (f) Find a Gaussian kernel density estimate for `logit(x)`.
- (g) Using your previous results, convert the KDE for `logit(x)` into a density estimate for x .
- (h) Make a plot showing (i) the true beta density, (ii) the “raw” kernel density estimate from 6e, and (iii) the transformed KDE from 6g. Make sure that the plotting region shows all three curves adequately, and that the three curves are visually distinct.

Chapter 17

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).

17.1 Smooth Tests of Goodness of Fit

17.1.1 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,

$$\Pr(F(X) \leq 0) = 0 \tag{17.1}$$

$$\Pr(F(X) \leq 1) = 1 \tag{17.2}$$

Since F is continuous and strictly increasing, it has an inverse, the quantile function Q , which is also continuous and strictly increasing. Then, for $0 \leq a \leq 1$,

$$\Pr(F(X) \leq a) = \Pr(Q(F(X)) \leq Q(a)) \tag{17.3}$$

$$= \Pr(X \leq Q(a)) \tag{17.4}$$

$$= F(Q(a)) = a \tag{17.5}$$

Thus, when F is continuous and strictly-increasing, $F(X)$ is uniformly distributed on the unit interval,

$$F(X) \sim \text{Unif}(0, 1) \quad (17.6)$$

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:

$$\Pr(F_0(X) \leq a) = \Pr(X \leq Q_0(a)) \quad (17.7)$$

$$= F(Q_0(a)) \neq a \quad (17.8)$$

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.

17.1.2 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,

$$d_{KS} = \max_{a \in [0, 1]} |\hat{F}_{n,Y}(a) - a| \quad (17.9)$$

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.

17.1.3 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

$$g(y; \theta) \equiv \begin{cases} \frac{e^{\sum_{j=1}^d \theta_j b_j(y)}}{z(\theta)} & 0 \leq y \leq 1 \\ 0 & \text{elsewhere} \end{cases} \quad (17.10)$$

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:

$$z(\theta) \equiv \int_0^1 e^{\sum_{j=1}^d \theta_j b_j(y)} dy \quad (17.11)$$

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,

$$\int_0^1 b_j(y) dy = 0 \quad (17.12)$$

$$\int_0^1 b_j(y) b_k(y) dy = 0 \quad (17.13)$$

and **normalized** in magnitude,

$$\int_0^1 b_j^2(y) dy = 1 \quad (17.14)$$

Further details, while practically important, do not matter for the general idea of the test, so I'll put them off to §17.1.3.1.

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 G), under the null, as $n \rightarrow \infty$,

$$2(\ell(\hat{\theta}) - \ell(0)) \rightsquigarrow \chi_d^2 \quad (17.15)$$

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 ,

$$\bar{b}_j = \frac{1}{n} \sum_{i=1}^n b_j(y_i) \quad (17.16)$$

and form the test statistic

$$\Psi^2 = n \sum_{j=1}^d \bar{b}_j^2 \quad (17.17)$$

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 .

17.1.3.1 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,

$$\int_0^1 b_j(y) b_k(y) dy = 0 \quad \forall k < j \quad (17.18)$$

including the constant "polynomial" $b_0(y) = 1$, and normalized to size 1,

$$\int_0^1 b_j^2(y) dy = 1 \quad (17.19)$$

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 17.1 shows the first few of these polynomials, and their exponentiated versions (which are what appear in Eq. 17.10).

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 $h_j(y) = c_j \cos 2\pi jy$, where c_j is a normalizing constant⁴.

17.1.3.2 Choice of Number of Basis Functions

As we make d in Eq. 17.10, 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:

$$d^* = \operatorname{argmax}_d \frac{1}{n} \ell(\hat{\theta}^{(d)}) - \frac{d}{2} \frac{\log n}{n} \quad (17.20)$$

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. 17.20, 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.

17.1.3.3 Application: Combining p -Values

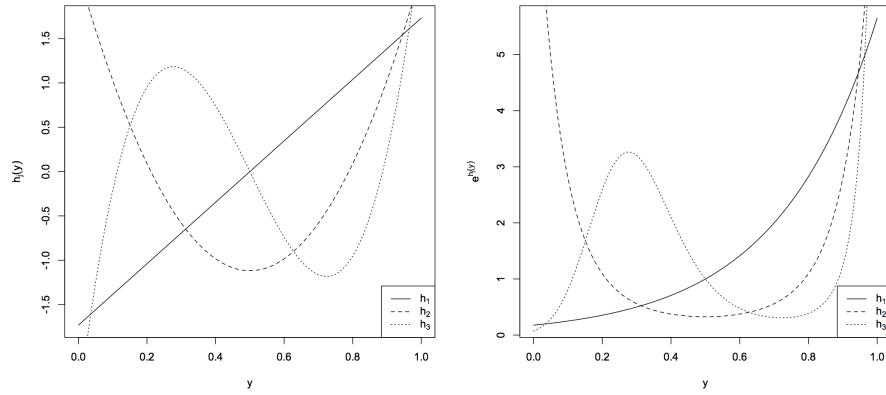
One useful property of p -values is that they are always uniformly distributed on $[0, 1]$ under the null hypothesis⁷. Suppose we have conducted a bunch of tests of the same null hypothesis — these might be different clinical trials of the same drug, or

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.

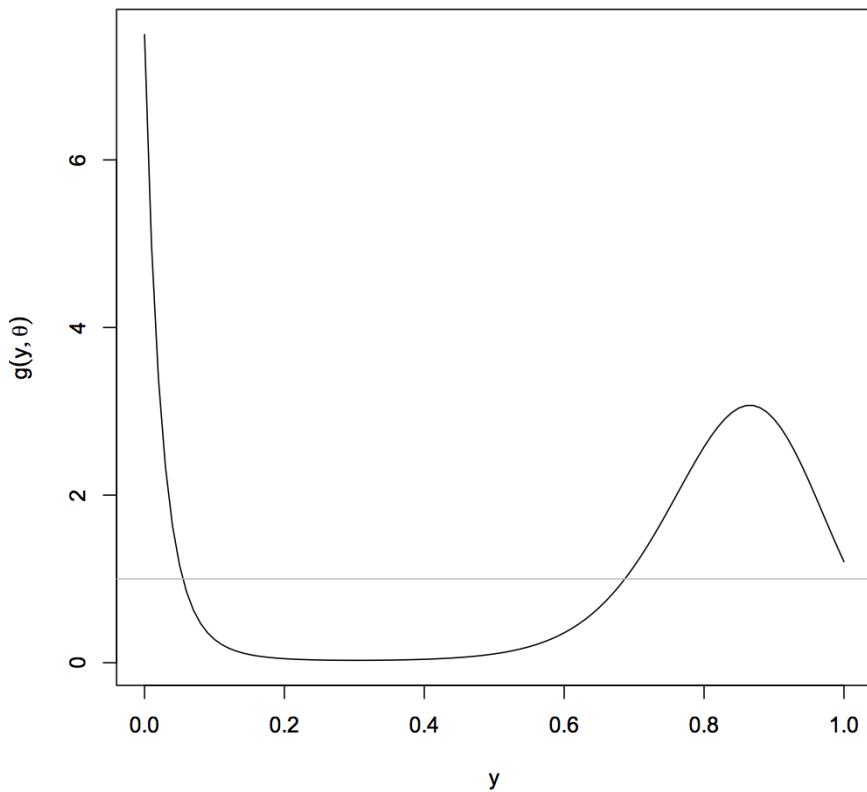


```

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 17.1: *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. 17.10.*



```

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 17.2: 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.

17.1.3.4 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. 17.10 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. 17.20). 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.

17.1.4 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 §17.1.1, 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. 17.10)

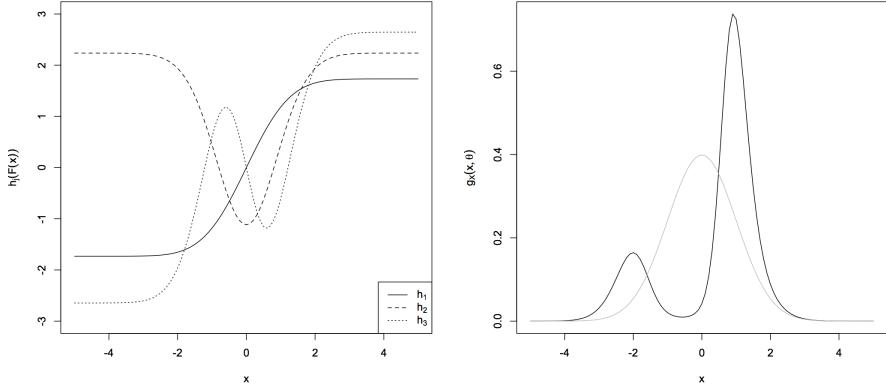
$$g(y; \theta) \equiv \begin{cases} \frac{e^{\sum_{j=1}^d \theta_j b_j(y)}}{z(\theta)} & 0 \leq y \leq 1 \\ 0 & \text{elsewhere} \end{cases} \quad (17.21)$$

Since $X = F^{-1}(Y)$, this implies a density for X :

$$g_X(x; \theta) = \frac{e^{\sum_{j=1}^d \theta_j b_j(F(x))}}{z(\theta)} \frac{dF}{dx} \quad (17.22)$$

$$= \frac{e^{\sum_{j=1}^d \theta_j b_j(F(x))}}{z(\theta)} f(x) \quad (17.23)$$

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 17.3: *Left panel:* the basis functions from Figure 17.1 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 17.2, 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 17.2.

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

$$\int_{-\infty}^{\infty} h_j(F(x))h_k(F(x))f(x)dx = \delta_{jk} \quad (17.24)$$

Figure 17.3 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

$$\bar{b}_j = \frac{1}{n} \sum_{i=1}^n b_j(y_i) = \frac{1}{n} \sum_{i=1}^n h_j(F(x_i)) \quad (17.25)$$

leading as before to the test statistic

$$\Psi^2 = n \sum_{j=1}^n \bar{b}_j^2 \quad (17.26)$$

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. 17.23.) If d is chosen from the data, we still need to bootstrap, but can do so just as before.

17.1.4.1 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. 17.23, they are

$$g_X(x; \beta, \theta) = \frac{e^{\sum_{j=1}^d \theta_j h_j(F(x; \beta))}}{z(\theta)} f(x; \beta) \quad (17.27)$$

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 G). If d is not fixed but data-driven, then, again, we need to bootstrap.

17.1.5 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`).

17.1.5.1 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^2_d distribution, we'd get a p -value of

```
> pchisq(0.6183, df=1, lower.tail=FALSE)
[1] 0.4316797
```

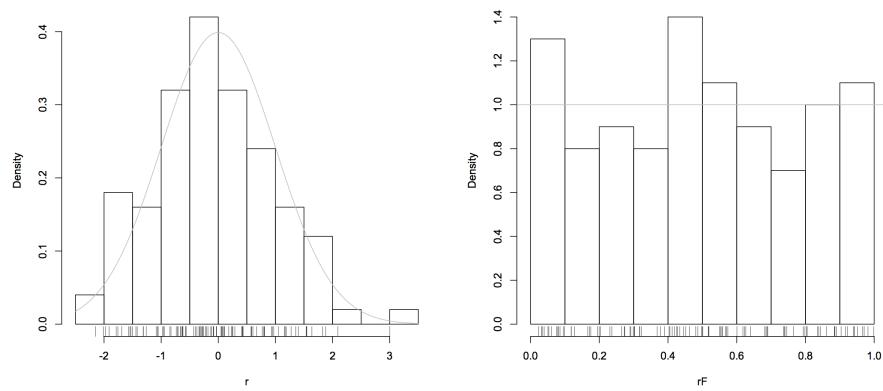
which to say a relative error of about 10%.

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)
```

Data Driven Smooth Test for Normality

```
data: ng, base: ddst.base.legendre, c: 100
WT* = 16.5623, n. coord = 2, p-value = 0.007
```

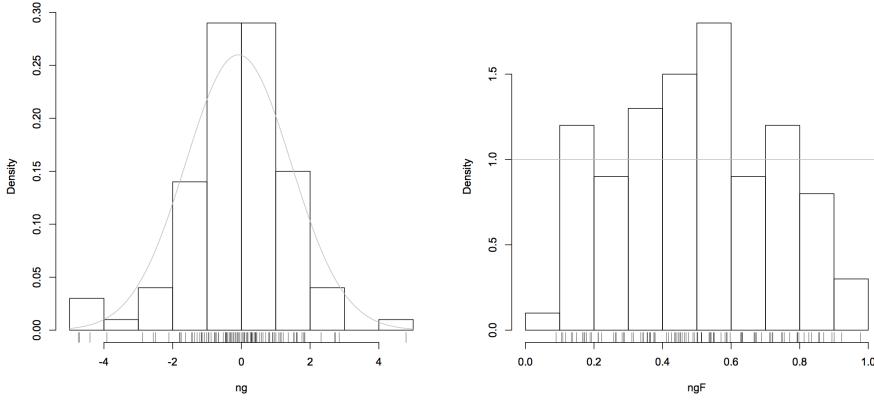


```

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 17.4: 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 17.5: Treating the draw from the t distribution discussed in the text the same as the Gaussian sample in Figure 17.4.

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.

17.1.6 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 §17.1.1, 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.

17.2 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

$$r_i = F_0(x_i) \quad (17.28)$$

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 :

$$G(a) \equiv \Pr(R \leq a) \quad (17.29)$$

$$= \Pr(F_0(X) \leq a) \quad (17.30)$$

$$= \Pr(X \leq Q_0(a)) \quad (17.31)$$

$$= F(Q_0(a)) \quad (17.32)$$

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:

$$g(y) \equiv \left. \frac{dG}{da} \right|_{a=y} \quad (17.33)$$

$$= \left. \frac{dF}{du} \right|_{u=Q_0(y)} \left. \frac{dF_0^{-1}}{da} \right|_{a=y} \quad (17.34)$$

$$= f(Q_0(y)) \frac{1}{f_0(Q_0(y))} = \frac{f(Q_0(y))}{f_0(Q_0(y))} \quad (17.35)$$

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 .

17.2.1 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. 17.10 (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).

17.2.2 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.

17.2.2.1 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.

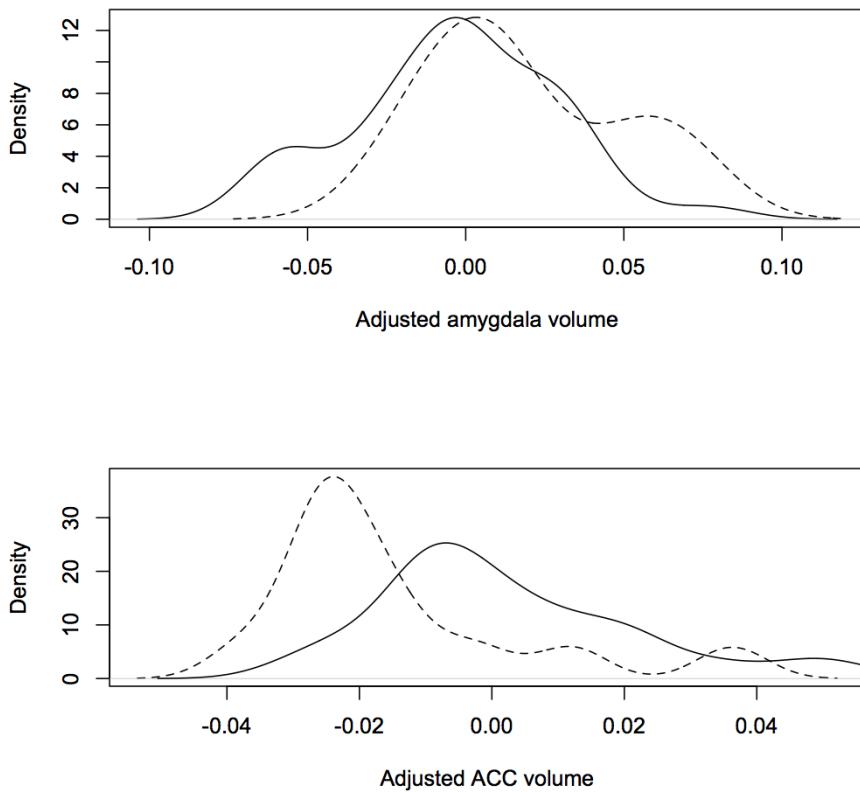
Having loaded the data into the data frame `n90`, we can look at simple density estimates for the two classes and the two variables (Figure 17.6). This indicates that conservative subjects tend to have relatively larger amygdalas and relatively smaller ACCs, though with very considerable overlap. (We are not looking at the uncertainty here at all.)

Enough preliminaries; let's find the relative distribution. Figure 17.7).

```
library(reldist)
acc.rel <- reldist(y=n90$acc[n90$orientation<3] ,
yo=n90$acc[n90$orientation>2] , ci=TRUE,
yolabs=pretty(n90$acc[n90$orientation>2]),
main="Relative density of adjusted ACC volume")
```

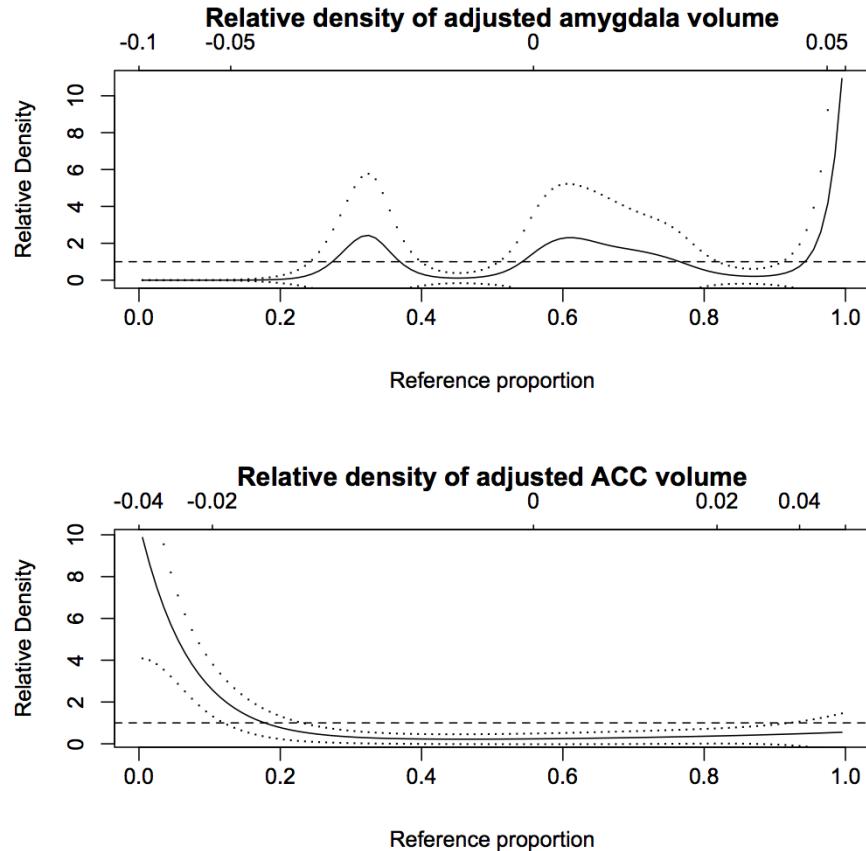
The first argument is the comparison sample; the second is the reference sample. The labeling of the horizontal axis is in terms of the quantiles of the reference distribution; I convert this back to the original units with the optional `yolabs` argument¹⁵. The dots show a pointwise 95%-confidence band, 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 17.6: 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,
        yolabs=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,
        yolabs=pretty(n90$acc[n90$orientation>2]),
        main="Relative density of adjusted ACC volume")
```

FIGURE 17.7: *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$growth[!in.oecd],
        yolabs=pretty(oecdpanel$growth[!in.oecd]),
        ci=TRUE, ylim=c(0,3))
```

FIGURE 17.8: *Relative distribution of the per-capita GDP growth rates of OECD-member countries compared to those of non-OECD countries.*

17.2.2.2 Example: Economic Growth Rates

For a second example, let's return to the OECD data on economic growth featured in Chapter 16. 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 §16.5.1. 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$growth[!in.oecd],
        yolabs=pretty(oecdpanel$growth[!in.oecd]))
```

Examining the resulting plot (Figure 17.8), 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 17.9. The result does not look too different from Figure 17.8. (This does not mean that the nihilistic model of economic growth is right.)

17.2.3 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,

$$f_0(x) = \int f_0(x|z)f_0(z)dz \quad (17.36)$$

If the distribution of the covariate Z is instead taken from the comparison population, we get a *different* distribution for x ,

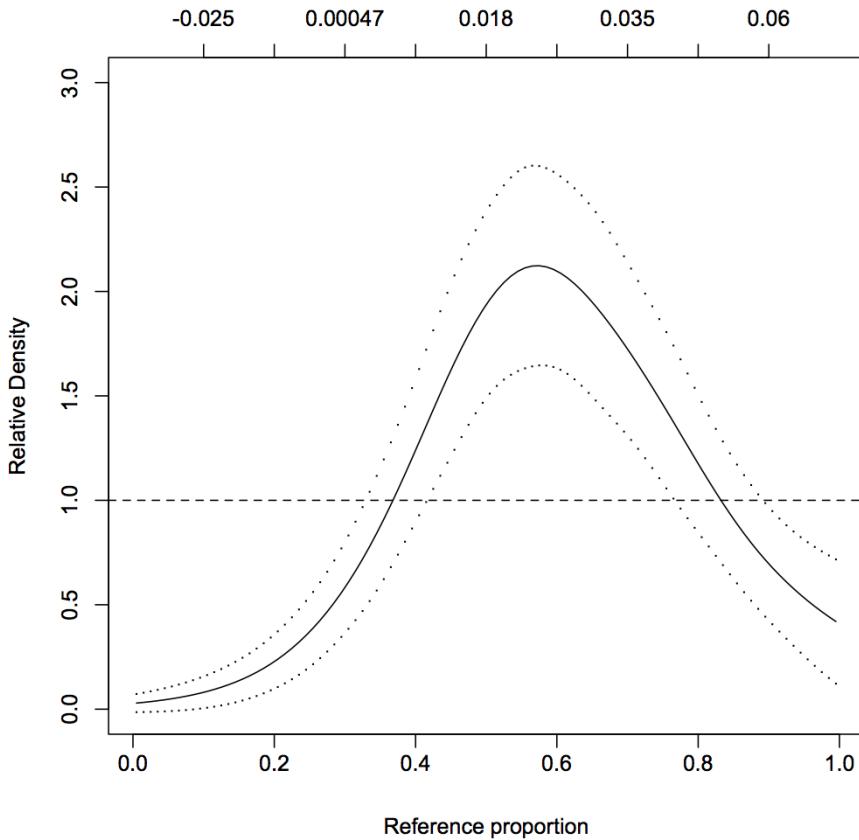
$$f_{0C}(x) = \int f_0(x|z)f_C(z)dz \quad (17.37)$$

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. 17.35) $g(y) = f(Q_0(y))/f_0(Q_0(y))$. Notice that

$$\frac{f(Q_0(y))}{f_0(Q_0(y))} = \frac{f_{0C}(Q_0(y))}{f_0(Q_0(y))} \frac{f(Q_0(y))}{f_{0C}(Q_0(y))} \quad (17.38)$$

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 = pnorm(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 17.9: 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).

17.2.3.1 Example: Adjusting Growth Rates

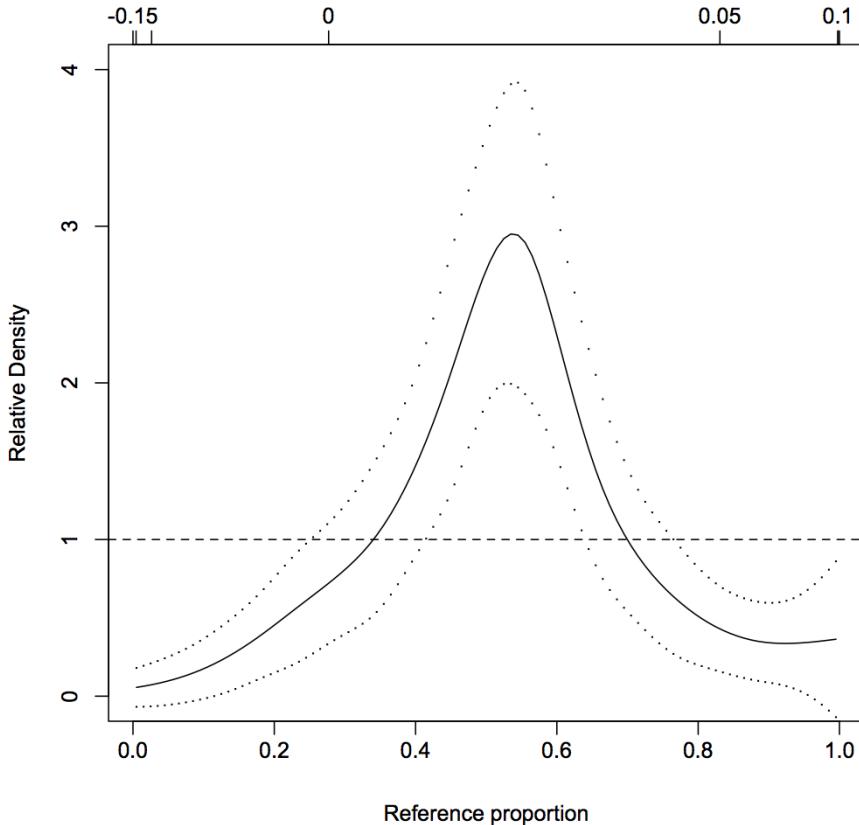
It will be easier to see how this works with an example. The `oecdpanel` data set also includes a variable called `humancap`, which is the log of the average number of years of education of people over the age of fifteen¹⁷. How do the growth rates of developed countries compare to those of undeveloped countries once we adjust for education?

As Figure 17.10 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 17.11) 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.

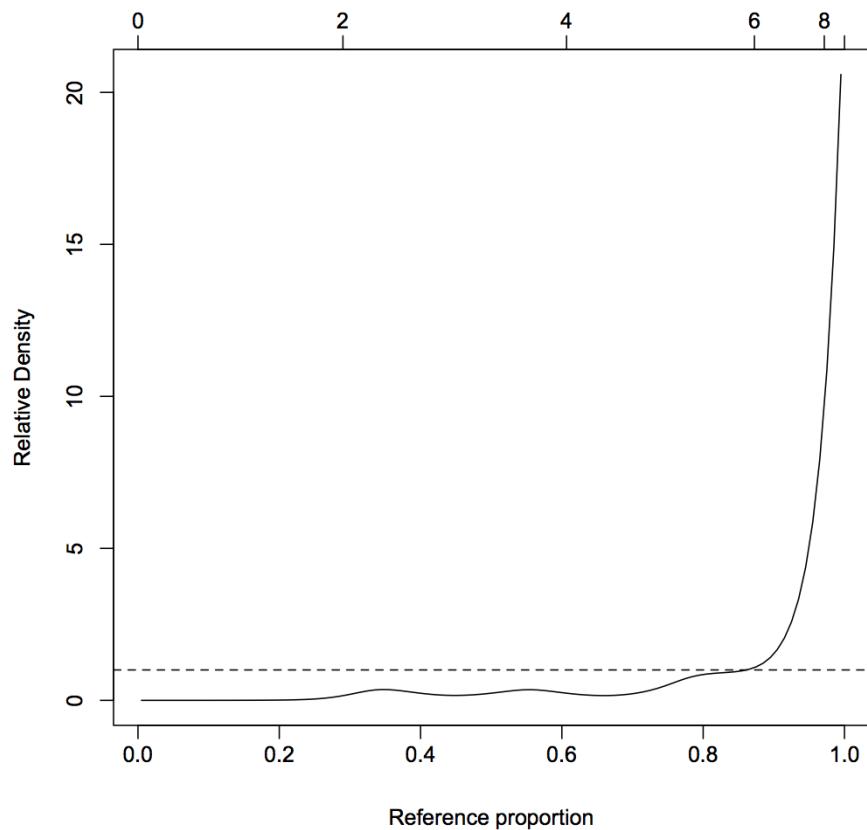


```

reldist(y=oecdpanel$growth[in.oecd],
        yo=oecdpanel$growth[!in.oecd],
        yolabs=pretty(oecdpanel$growth[!in.oecd]),
        z=oecdpanel$humancap[in.oecd],
        zo=oecdpanel$humancap[!in.oecd],
        decomp="covariate",
        ci=TRUE, ylim=c(0,4))

```

FIGURE 17.10: 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 17.11: *Relative distribution of years of education, comparing OECD countries to non-OECD countries.*

17.3 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).

17.4 Exercises

1. §17.1.3.1 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. 17.18 and Eq. 17.19.
2. Prove Eq. 17.24. *Hint:* change of variables. Also, prove that

$$\int_{-\infty}^{\infty} f(x) \exp^{\sum_{j=1}^d \theta_j b_j(F(x))} dx = \int_0^1 \exp^{\sum_{j=1}^d \theta_j b_j(y)} dy = z(\theta) \quad (17.39)$$

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 18

Principal Components Analysis

In Chapter 16, we saw that kernel density estimation gives us, in principle, a consistent way of nonparametrically estimating joint distributions for arbitrarily many variables. We also saw (§16.4.2) that, like regression (§9.3), density estimation suffers from the curse of dimensionality — the amount of data needed grows exponentially with the number of variables. Moreover, this is not a flaw in kernel methods, but reflects the intrinsic difficulty of the problem.

Accordingly, to go forward in multivariate data analysis, we need to somehow lift the curse of dimensionality. One approach is to hope that while we have a large number p of variables, the data is really only q -dimensional, and $q \ll p$. The next few chapters will explore various ways of finding low-dimensional structure. Alternatively, we could hope that while the data really does have lots of dimensions, it also has lots of independent parts. At an extreme, if it had p dimensions but we knew they were all statistically independent, we'd just do p one-dimensional density estimates. Chapter 22 and its sequels are concerned with this second approach, of factoring the joint distribution into independent or conditionally-independent pieces.

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**. The hope with PCA is that the data lie in, or close to, a low-dimensional linear subspace.

18.1 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 vari-

ance. 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 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!)

18.1.1 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:

$$\frac{1}{n} \sum_{i=1}^n (\vec{x}_i \cdot \vec{w})\vec{w} = \left(\left(\frac{1}{n} \sum_{i=1}^n \vec{x}_i \right) \cdot \vec{w} \right) \vec{w} v \quad (18.1)$$

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

$$\|\vec{x}_i - (\vec{w} \cdot \vec{x}_i)\vec{w}\|^2 = (\vec{x}_i - (\vec{w} \cdot \vec{x}_i)\vec{w}) \cdot (\vec{x}_i - (\vec{w} \cdot \vec{x}_i)\vec{w}) \quad (18.2)$$

$$= \vec{x}_i \cdot \vec{x}_i - \vec{x}_i \cdot (\vec{w} \cdot \vec{x}_i)\vec{w} \quad (18.3)$$

$$- (\vec{w} \cdot \vec{x}_i)\vec{w} \cdot \vec{x}_i + (\vec{w} \cdot \vec{x}_i)\vec{w} \cdot (\vec{w} \cdot \vec{x}_i)\vec{w}$$

$$= \|\vec{x}_i\|^2 - 2(\vec{w} \cdot \vec{x}_i)^2 + (\vec{w} \cdot \vec{x}_i)^2 \vec{w} \cdot \vec{w} \quad (18.4)$$

$$= \vec{x}_i \cdot \vec{x}_i - (\vec{w} \cdot \vec{x}_i)^2 \quad (18.5)$$

since $\vec{w} \cdot \vec{w} = \|\vec{w}\|^2 = 1$.

Add those residuals up across all the vectors:

$$MSE(\vec{w}) = \frac{1}{n} \sum_{i=1}^n \|\vec{x}_i\|^2 - (\vec{w} \cdot \vec{x}_i)^2 \quad (18.6)$$

$$= \frac{1}{n} \left(\sum_{i=1}^n \|\vec{x}_i\|^2 - \sum_{i=1}^n (\vec{w} \cdot \vec{x}_i)^2 \right) \quad (18.7)$$

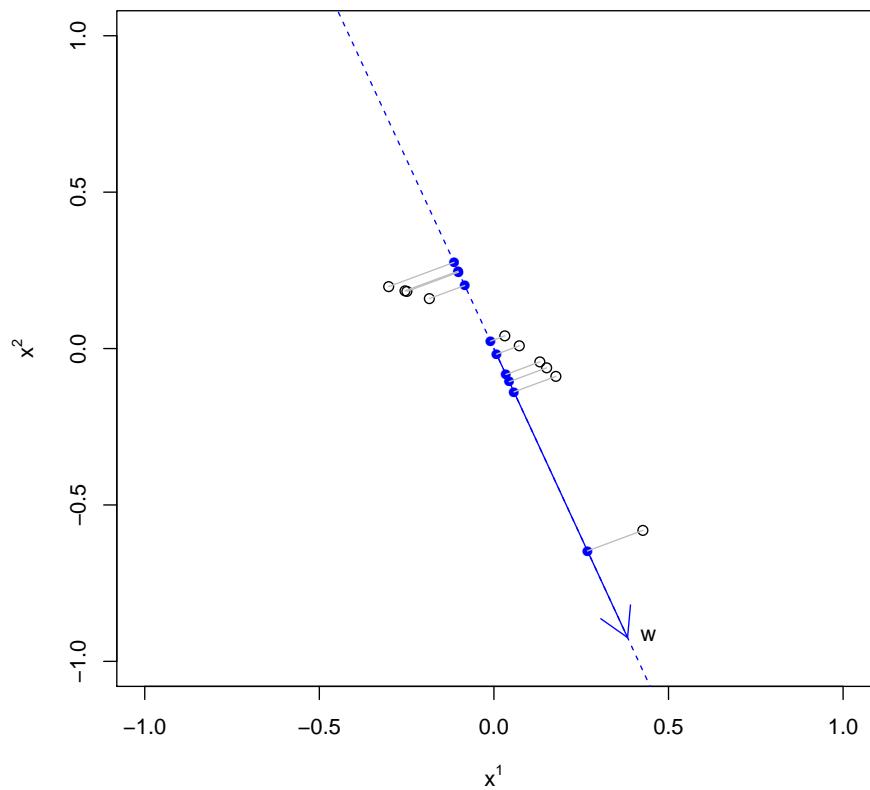


FIGURE 18.1: Illustration of projecting data points \vec{x} (black dots) on to an arbitrary line through the space (blue, dashed), represented by a unit vector \vec{w} along the line (also blue but solid). The blue dots are the projections on to the blue line, $(\vec{x} \cdot \vec{w})\vec{w}$; the gray lines are the vector residuals, $\vec{x} - (\vec{x} \cdot \vec{w})\vec{w}$. These are not the residuals from regressing one of the components of the data vector on the other.

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

$$\frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \vec{x}_i)^2 \quad (18.8)$$

which we can see is the sample mean of $(\vec{w} \cdot \vec{x}_i)^2$. The (sample) mean of a square is always equal to the square of the (sample) mean plus the (sample) variance:

$$\frac{1}{n} \sum_{i=1}^n (\vec{w} \cdot \vec{x}_i)^2 = \left(\frac{1}{n} \sum_{i=1}^n \vec{x}_i \cdot \vec{w} \right)^2 + \widehat{\sigma^2}(\vec{w} \cdot \vec{x}_i) \quad (18.9)$$

Since we've just seen that the mean of the projections is zero, minimizing the residual sum of squares is 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,

$$\sum_{j=1}^k (\vec{x}_i \cdot \vec{w}_j) \vec{w}_j \quad (18.10)$$

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 1] 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.)

18.1.2 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

$$\widehat{\sigma^2}(\vec{w} \cdot \vec{x}_i) = \frac{1}{n} \sum_i (\vec{x}_i \cdot \vec{w})^2 \quad (18.11)$$

$$= \frac{1}{n} (\mathbf{x}\mathbf{w})^T (\mathbf{x}\mathbf{w}) \quad (18.12)$$

$$= \frac{1}{n} \mathbf{w}^T \mathbf{x}^T \mathbf{x} \mathbf{w} \quad (18.13)$$

$$= \mathbf{w}^T \frac{\mathbf{x}^T \mathbf{x}}{n} \mathbf{w} \quad (18.14)$$

$$= \mathbf{w}^T \mathbf{v} \mathbf{w} \quad (18.15)$$

We want to choose a unit vector \vec{w} so as to maximize $\widehat{\sigma^2}(\vec{w} \cdot \vec{x}_i)$. 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:

$$\mathcal{L}(\mathbf{w}, \lambda) \equiv \mathbf{w}^T \mathbf{v} \mathbf{w} - \lambda(\mathbf{w}^T \mathbf{w} - 1) \quad (18.16)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \mathbf{w}^T \mathbf{w} - 1 \quad (18.17)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 2\mathbf{v} \mathbf{w} - 2\lambda \mathbf{w} \quad (18.18)$$

Setting the derivatives to zero at the optimum, we get

$$\mathbf{w}^T \mathbf{w} = 1 \quad (18.19)$$

$$\mathbf{v} \mathbf{w} = \lambda \mathbf{w} \quad (18.20)$$

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 at most p different eigenvectors. 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 $\sum_{i=1}^q \lambda_i$.

18.1.3 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,

$$R^2 \equiv \frac{\sum_{i=1}^q \lambda_i}{\sum_{j=1}^p \lambda_j} \quad (18.21)$$

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.

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 get all the components at once from \mathbf{v} 's eigenvectors, 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.¹

Projections on to the first two or three principal components can be visualized; there is no guarantee, however, that only two or three dimensions really matter. Usually, to get an R^2 of 1, you need to use all p principal components.² How many principal components you should use depends on your data, and how big an R^2 you need. Sometimes, you can get better than 80% of the variance described with just two or three components. Sometimes, however, to keep a lot of the original variance you need to use almost as many components as you had dimensions to start with.

18.1.3.1 Scree Plots

People sometimes like to make plots of the eigenvalues, in decreasing order. Ideally, one starts with a few big eigenvalues, and then sees a clear drop-off to a remainder of

¹Be 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 §18.4.1, 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.

²The 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, as just mentioned, $n < p$.

small, comparatively negligible eigenvalues. These diagrams are called **scree plots**³. (Some people make similar plots, but show $1 - R^2$ versus the number of components, rather than the individual eigenvalues.) Folklore recommends find the “base of the cliff” or “elbow” in the plot, the place where the number eigenvalues decrease dramatically and then level off to the right, and then retaining that number of components. This folklore appears to be based on nothing more than intuition, and offers no recommendation for what to do when there is no clear cliff or elbow in the scree plot.

18.1.4 Statistical Inference, or Not

You may have noticed, and even been troubled by, the fact that I have said nothing at all in this chapter 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 there *is* a larger population or stochastic process. This is part of why §18.1.3 was so wishy-washy about the right number of components to use.

However, we could *add* a statistical assumption and see how PCA behaves under those conditions. The simplest one is to suppose that the data come IIDly from a distribution with covariance matrix v_0 . Then the sample covariance matrix $v \equiv n^{-1}\mathbf{x}^T\mathbf{x}$ will converge on v_0 as $n \rightarrow \infty$. Since the principal components are smooth functions of v (namely its eigenvectors), they will tend to converge as n grows⁴. So, 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.

We could always add stronger statistical assumptions; in fact, Chapter 19 will look at what happens when our assumptions essentially amount to “the data lie on a low-dimensional linear subspace, plus noise”. Even this, however, turns out to make PCA a not-very-attractive estimate of the statistical structure.

18.2 Example 1: Cars

Enough math; let’s work an example. The data⁵ consists of 388 cars from the 2004 model year, with 18 features. Eight features are binary indicators; the other 11 fea-

³The small loose rocks one finds at the base of cliffs or mountains are called “scree”; the metaphor is that one starts with the big eigenvalues at the top of the hill, goes down some slope, and then finds the scree beneath it, which is supposed to be negligible noise. Those who have had to cross scree fields carrying heavy camping backpacks may disagree about whether it can really be ignored.

⁴There is a wrinkle if 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 2.) For instance, if v_0 is the identity matrix, then every vector is an eigenvector, and PCA routines will return an essentially arbitrary collection of mutually perpendicular vectors. Generically, however, any arbitrarily small tweak to v_0 will break the degeneracy.

⁵On the course website; from <http://www.amstat.org/publications/jse/datasets/04cars.txt>, with incomplete records removed.

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 18.1: *Features for the 2004 cars data.*

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, 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, 1, 89765, 79978, 3.2, 6, 290, 17, 24, 3153, 100, 174, 71

TABLE 18.2: *The first few lines of the 2004 cars data set.*

tures are numerical (Table 18.1). All of the features except Type are numerical. Table 18.2 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.⁶ The latter is generally more robust, so we'll just use it.

[[ATTN: Data is now 10 years old; update?]]

```
cars04 = read.csv("http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/data/cars-fixed04.dat")
cars04.pca = prcomp(cars04[,8:18], scale.=TRUE)
```

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

⁶`princomp` actually calculates the covariance matrix and takes its eigenvalues. `prcomp` uses a different technique called “singular value decomposition”.

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.

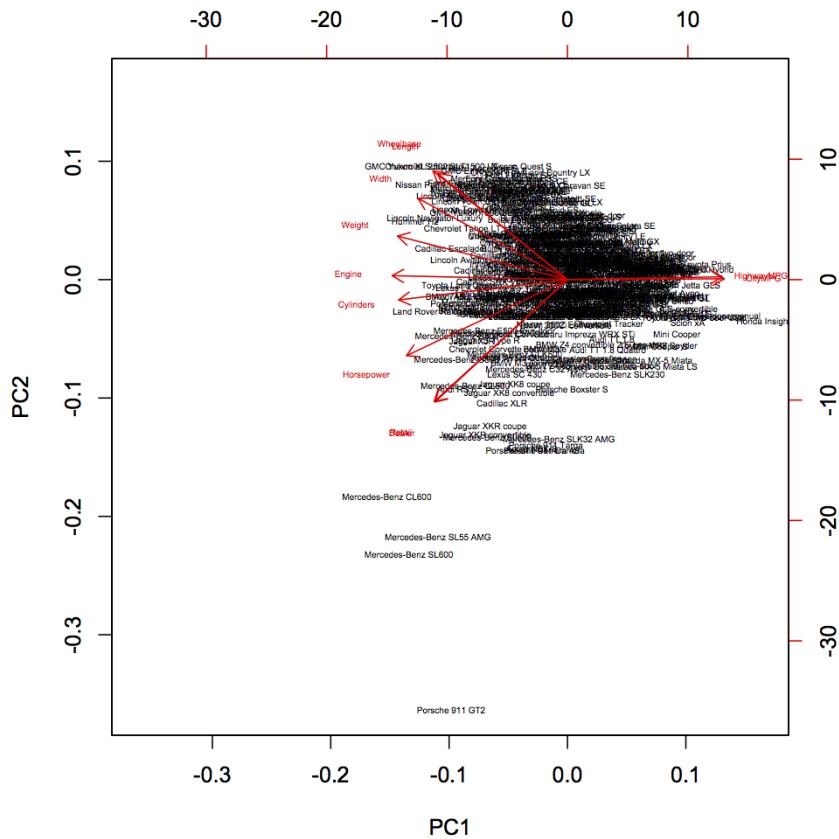
```
round(cars04.pca$rotation[,1:2],2)
##          PC1    PC2
## Retail     -0.26 -0.47
## Dealer     -0.26 -0.47
## Engine     -0.35  0.02
## Cylinders  -0.33 -0.08
## Horsepower -0.32 -0.29
## CityMPG    0.31  0.00
## HighwayMPG 0.31  0.01
## Weight     -0.34  0.17
## Wheelbase   -0.27  0.42
## Length      -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).

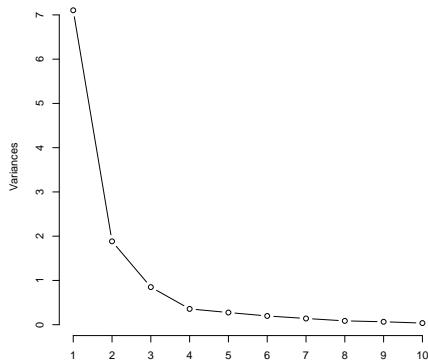
To check this interpretation, we can use a useful tool called a **biplot**, which plots the data, along with the projections of the original variables, on to the first two components (Figure 18.2). Notice that the car with the lowest value of the second component is a Porsche 911, with pick-up trucks and mini-vans at the other end of the scale. Similarly, the highest values of the first component all belong to hybrids.

18.2. EXAMPLE 1: CARS



```
biplot(cars04.pca, cex=0.4)
```

FIGURE 18.2: “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).



```
plot(cars04.pca,type="l",main="")
```

FIGURE 18.3: *Scree plot of the 2004 cars data: the eigenvalues of the principal components, in decreasing order. Each eigenvalue is the variance along that component. Folklore suggests adding components until the plot levels off, or goes past an “elbow”—here this might be 2 or 3 components.*

18.3 Example 2: The United States *circa* 1977

R contains a built-in data file, `state.x77`, with facts and figures for the various states of the USA as of about 1977: population, per-capita income, the adult illiteracy rate, life expectancy, the homicide rate, the proportion of adults with at least a high-school education, the number of days of frost a year, and the state's area. While this data set is almost as old as I am, it still makes a convenient example, so let's step through a principal components analysis of it.

Since the variables all have different, incomparable scales, it's not a bad idea to scale them to unit variance before finding the components⁷:

```
state.pca <- prcomp(state.x77, scale.=TRUE)
```

The biplot and the scree plot (Figure 18.4) look reasonable.

With this reasonable-looking PCA, we might try to interpret the components.

```
signif(state.pca$rotation[,1:2], 2)
##          PC1    PC2
## Population 0.130  0.410
## Income    -0.300  0.520
## Illiteracy 0.470  0.053
## Life Exp   -0.410 -0.082
## Murder     0.440  0.310
## HS Grad   -0.420  0.300
## Frost     -0.360 -0.150
## Area       -0.033  0.590
```

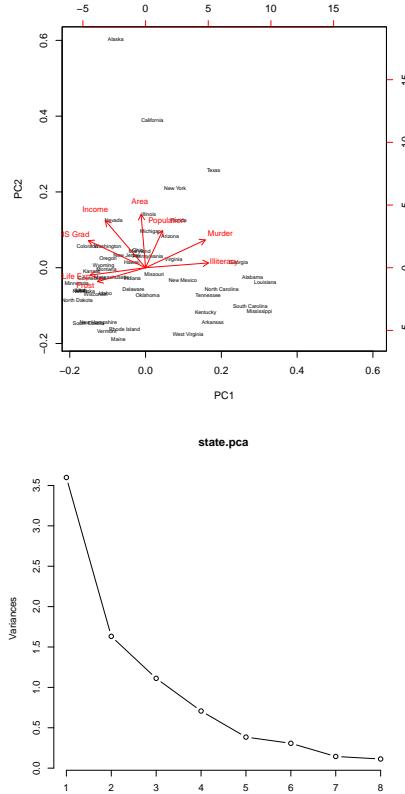
The first component aligns with illiteracy, murder, and (more weakly) population; it's negatively aligned with high school graduation, life expectancy, cold weather, income, and (very weakly) the area of the state. The second component is positively aligned with area, income, population, high school graduation and murder, and negatively aligned, weakly, with cold weather and life expectancy. The first component thus separates short-lived, violent, ill-educated, poor warm states from those with the opposite qualities. The second component separates big, rich, educated, violent states from those which are small (in land or people), poor, less educated, and less violent.

Since each data point has a geographic location, we can make a map, where the sizes of the symbols for each state vary with their projection on to the first principal component. This suggests that the component is something we might call “southernness” — more precisely, the contrast between the South and the rest of the nation⁸. I will leave making a map of the second component as an exercise⁹.

⁷You should try re-running all this with `scale.=FALSE`, and ponder what the experience tells you about the wisdom of advice like “maximize R^2 , or even “minimize the approximation error”.

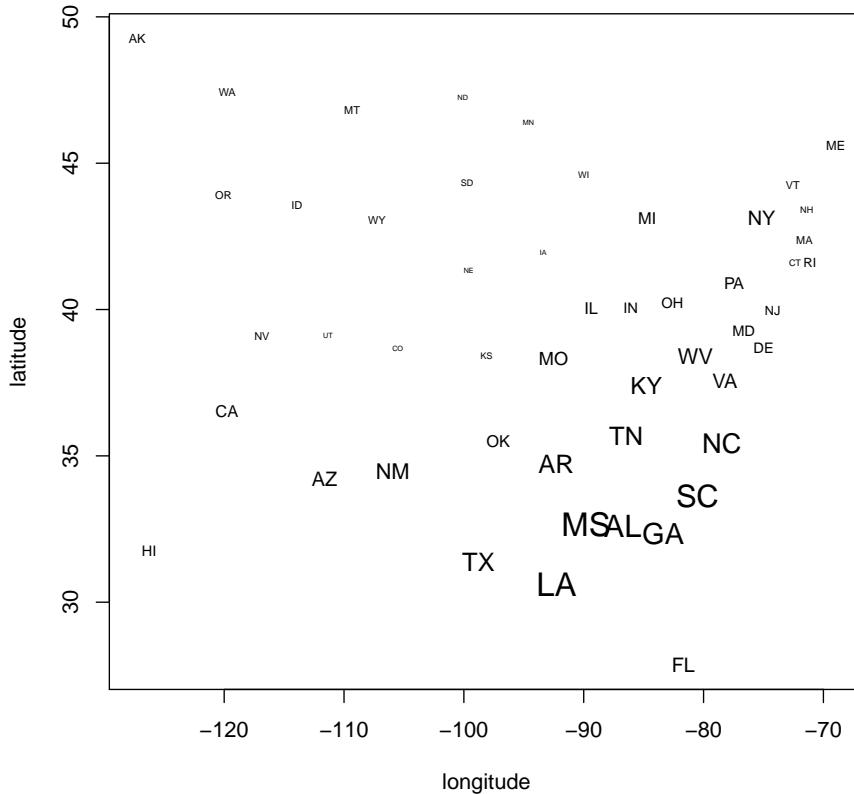
⁸The correlation between the first component and an indicator for being in the Confederacy is 0.8; for being a state which permitted slavery when the Civil War began, 0.78.

⁹§19.9.1 has more on this example.



```
biplot(state.pca,cex=c(0.5,0.75))
plot(state.pca,type="l")
```

FIGURE 18.4: Biplot and scree plot for the PCA of `state.x77`.



```

plot.states_scaled <- function(sizes, min.size = 0.4, max.size = 2, ...) {
  plot(state.center, type = "n", ...)
  out.range = max.size - min.size
  in.range = max(sizes) - min(sizes)
  scaled.sizes = out.range * ((sizes - min(sizes))/in.range)
  text(state.center, state.abb, cex = scaled.sizes + min.size)
  invisible(scaled.sizes)
}
plot.states_scaled(state.pca$x[, 1], min.size = 0.3, max.size = 1.5, xlab = "longitude",
                  ylab = "latitude")

```

FIGURE 18.5: The US states, plotted in their geographic locations, with symbol size varying with the projection of the state on to the first principal component. This suggests the component is something we might call “southernness”.

18.4 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.¹⁰

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, about the meaning of the words) and that between "Kansas" and "farming" (which reflects non-linguistic facts about the world, and probably wouldn't show up in, say, a corpus collected from Australia).

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.

18.4.1 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

¹⁰Or it should anyway; I haven't actually done the experiment with this book.

music.¹¹ 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.¹²

[[ATTN: Why isn't loading from the course website working?]]

The PCA is done as it would be for any other data:

```
load("~/teaching/ADAfaEPoV/data/pca-examples.Rdata")
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.¹³ The words with positive projections are mostly associated

¹¹Actually, 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.

¹²If we were trying to work with the complete corpus, we should expect at least 50000 words, and perhaps more.

¹³Which direction is positive and which negative is of course arbitrary; basically it depends on internal choices in the algorithm.

with music, those with negative components with the visual arts. The letters “m” and “p” 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.¹⁴

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.

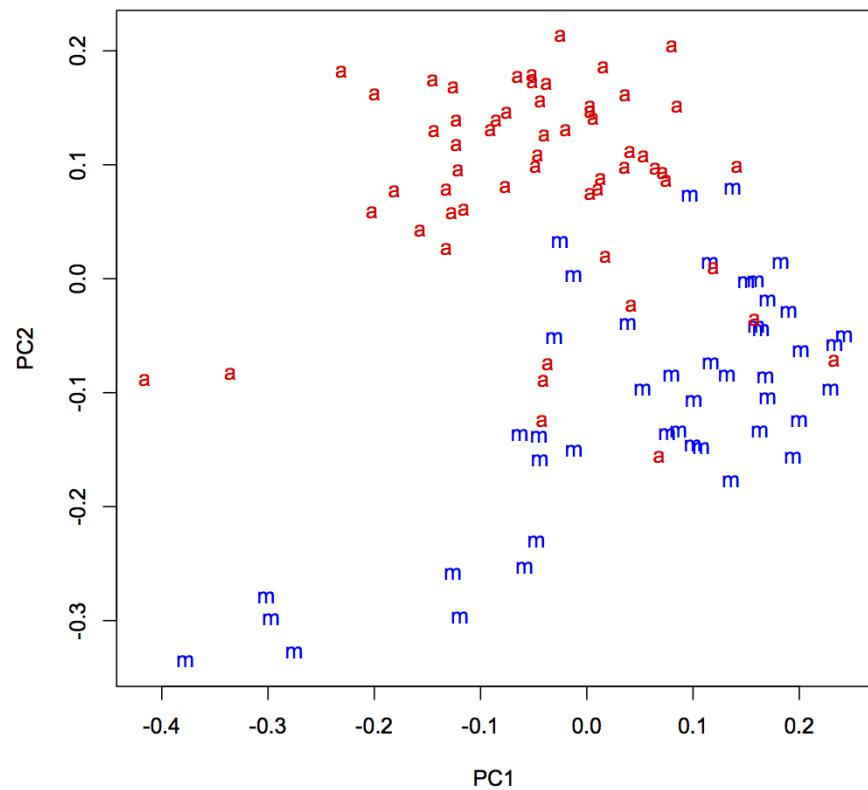
I could go on, but by this point you get the idea.

18.5 PCA for Visualization

Let’s try displaying the *Times* stories using the principal components (Figure 18.6).

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

¹⁴You should check out these explanations for yourself. The raw stories are part of the R workspace.



```
plot(nyt.pca$x[,1:2],
      pch=ifelse(nyt.frame[,"class.labels"]=="music","m","a"),
      col=ifelse(nyt.frame[,"class.labels"]=="music","blue","red"))
```

FIGURE 18.6: *Projection of the Times stories on to the first two principal components. Music stories are marked with a blue “m”, art stories with a red “a”.*

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:

$$\sum_i \sum_{j \neq i} (\|\vec{y}_i - \vec{y}_j\| - \|\vec{x}_i - \vec{x}_j\|)^2 \quad (18.22)$$

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).

18.6 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 18.7. 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.¹⁵ They simulated genetic diffusion processes, without any migration or population expansion, and got results that looked very like the real maps (Figure 18.8). 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.

18.7 Further Reading

Principal components goes back at least to Hotelling in the 1930s, if not to Karl Pearson around 1900. It has been rediscovered many times in many fields, so it is also known as the Karhunen-Loëve transformation, the Hotelling transformation,

¹⁵Remember that PCA re-writes the original vectors as a weighted sum of new, orthogonal vectors, just 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.

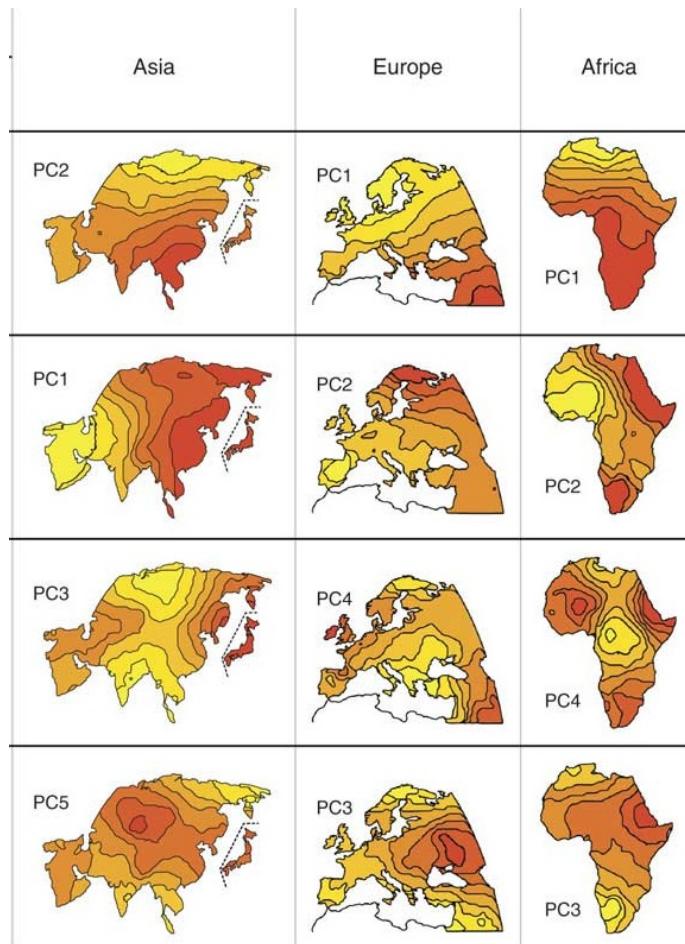


FIGURE 18.7: Principal components of genetic variation in the old world, according to Cavalli-Sforza et al. (1994), as re-drawn by Novembre and Stephens (2008).

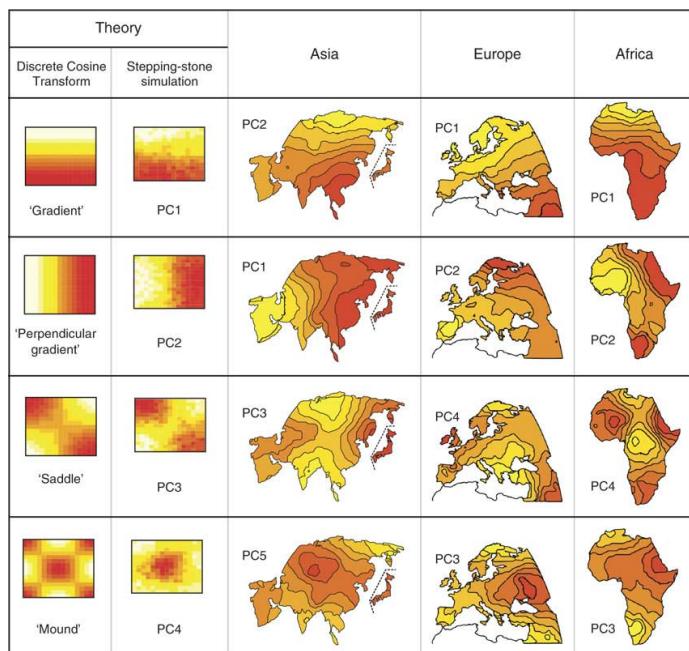


FIGURE 18.8: How the PCA patterns can arise as numerical artifacts (far left column) or through simple genetic diffusion (next column). From Novembre and Stephens (2008).

the method of empirical orthogonal functions, and singular value decomposition¹⁶. Many statistical presentations start with the idea of maximizing the variance kept; this seems less well-motivated to me than trying to find the best-approximating linear subspace.

As I said above, PCA is an example of a data analysis method which involves only approximation, with no statistical inference or underlying probabilistic model. Chapters 19 and 21 describe two (rather-different looking) statistical models which both imply that the data should lie in a linear subspace plus noise. Alternatively, chapter 20 introduces methods for approximating data with low-dimensional curved manifolds, rather than linear subspaces.

Latent semantic analysis, or latent semantic indexing, goes back to Deerwester *et al.* (1990). Hand *et al.* (2001) has a good discussion, setting it in the context of other data-analytic methods, and avoiding some of the more extravagant claims made on its behalf (Landauer and Dumais, 1997).

§18.5 just scratches the surface of the vast literature on multidimensional scaling, the general goal of which is to find low-dimensional, easily-visualized representations which are somehow faithful to the geometry of high-dimensional spaces. Much of this literature, including the name “multidimensional scaling”, comes from psychology. For a brief introduction with references, I recommend Hand *et al.* (2001).

Concerns about interpretation and reification¹⁷ are rarely very far away whenever people start using methods for finding hidden structure, whether they’re just approximation methods or they attempt proper statistical inference. We will touch on them again in chapters 19 and 21. In general, statisticians seem to find it easier to say what it wrong or dubious about other analysts’ interpretations of their components or latent constructs, than to explain what’s right about their own interpretations; certainly I do.

[[TODO: Expand on references to MDS.]]

18.8 Exercises

1. 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 $q = 1$.

¹⁶Strictly speaking, singular value decomposition is a matrix algebra trick which is used in the most common algorithm for PCA.

¹⁷I have not been able to find out where the term “reification” comes from; I have seen claims that it is Marxist in origin, but it’s used by the decidedly non-Marxist, and early, Thomson (1939), so that seems implausible.

- (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. 18.5, and use Exercise 1a.
- (e) Explain in what sense minimizing projection residuals is equivalent to maximizing the sum of variances along the different directions.
2. Suppose that \mathbf{u} has two eigenvectors, \vec{w}_1 and \vec{w}_2 , with the same eigenvalue α . Prove that any linear combination of \vec{w}_1 and \vec{w}_2 is also an eigenvector of \mathbf{u} , and also has eigenvalue α .

Chapter 19

Factor Analysis

19.1 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

$$\begin{aligned} \mathbf{X} &= \mathbf{F}\mathbf{w} \\ [n \times p] &= [n \times p][p \times p] \end{aligned} \tag{19.1}$$

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

$$\begin{aligned} \mathbf{X} &\approx \mathbf{F}_q \mathbf{w}_q \\ [n \times p] &= [n \times q][q \times p] \end{aligned} \tag{19.2}$$

The error of approximation — the difference between the left- and right-hand-sides of Eq. 19.2 — 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:

$$\mathbf{X} = \mathbf{F}\mathbf{w} + \epsilon \tag{19.3}$$

where ϵ has to be an $n \times p$ matrix.

Now, Eq. 19.3 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:

$$\mathbf{X} = \mathbf{F}\mathbf{w} + \epsilon \quad (19.4)$$

All of the systematic patterns in the observations \mathbf{X} should come from the first term on the right-hand side. The residual term ϵ 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.

19.1.1 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.

19.2 The Graphical Model

It's common to represent factor models visually, as in Figure 19.1. 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 actually 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).

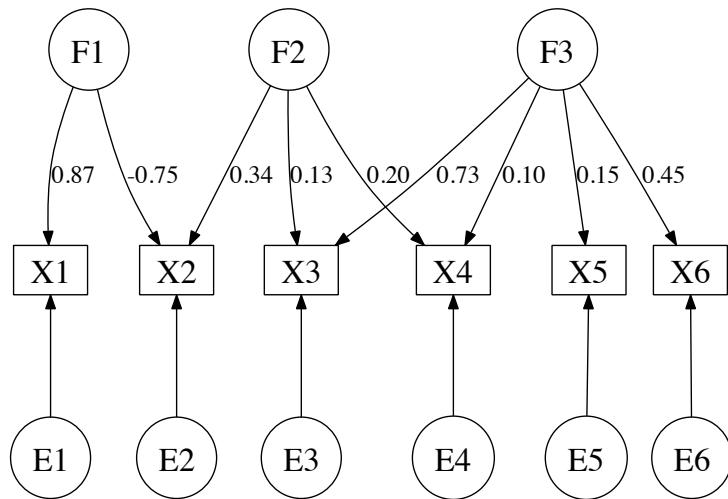


FIGURE 19.1: Graphical model form of a factor model. Circles stand for the unobserved variables (factors above, noises below), boxes for the observed features. Edges indicate non-zero coefficients — entries in the factor loading matrix \mathbf{w} , or specific variances ψ_i . Arrows representing entries 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.

19.2.1 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.)

$$\text{Cov}[X_1, X_2] = \mathbf{E}[X_1 X_2] - \mathbf{E}[X_1] \mathbf{E}[X_2] \quad (19.5)$$

$$= \mathbf{E}[X_1 X_2] \quad (19.6)$$

$$= \mathbf{E}[(F_1 w_{11} + F_2 w_{21} + \epsilon_1)(F_1 w_{12} + F_2 w_{22} + \epsilon_2)] \quad (19.7)$$

$$\begin{aligned} &= \mathbf{E}\left[F_1^2 w_{11} w_{12} + F_1 F_2 (w_{11} w_{22} + w_{21} w_{12}) + F_2^2 w_{21} w_{22}\right] \\ &\quad + \mathbf{E}[\epsilon_1 \epsilon_2] + \mathbf{E}[\epsilon_1 (F_1 w_{12} + F_2 w_{22})] \\ &\quad + \mathbf{E}[\epsilon_2 (F_1 w_{11} + F_2 w_{21})] \end{aligned} \quad (19.8)$$

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

$$\text{Cov}[X_1, X_2] = \mathbf{E}\left[F_1^2\right] w_{11} w_{12} + \mathbf{E}\left[F_2^2\right] w_{21} w_{22} \quad (19.9)$$

$$= w_{11} w_{12} + w_{21} w_{22} \quad (19.10)$$

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

$$\text{Cov}[X_i, X_j] = \sum_{k=1}^q w_{ki} w_{kj} \quad (19.11)$$

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.

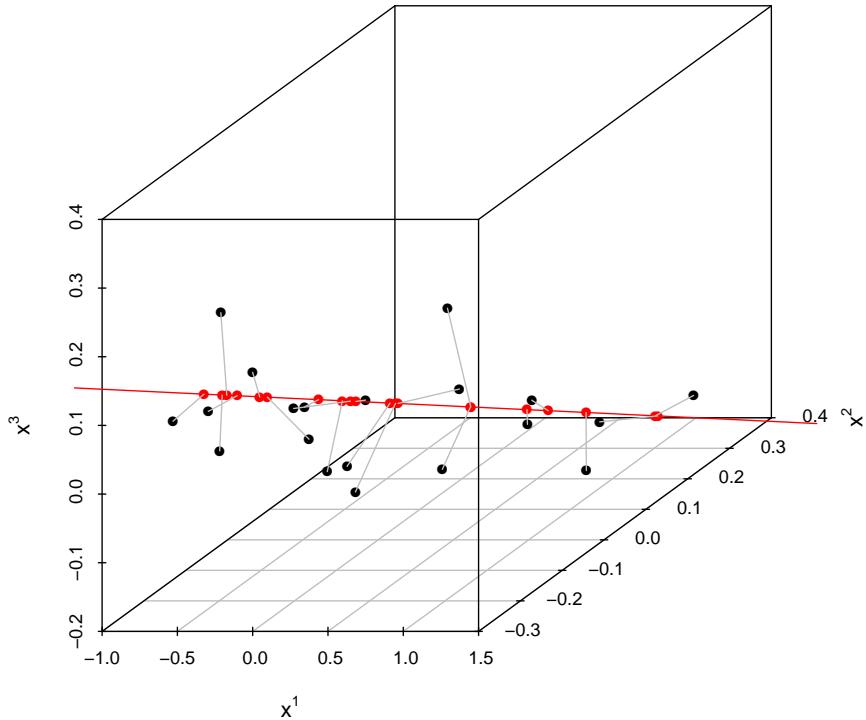
Graphically, we draw an arrow from a factor node to an observable node if and only if the observable loads on the factor. So then we can just *see* that two observables are correlated if they both have in-coming arrows from the same factors. (To find the actual correlation, we multiply the weights on all the edges connecting the two observable nodes to the common factors; that's Eq. 19.11.) 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.

19.2.2 Geometry: Approximation by Linear Subspaces

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 linear subspace. If the factor model is true and we turn off noise, we would find all the data lying *exactly* on this subspace. Of course, *with* noise we expect that the data vectors will be scattered around the subspace; how close depends on the variance of the noise. (Figure 19.2.) 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 linear subspace 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. Chapter 20 introduces two such techniques, but this is a broad and still-growing area.



```

n <- 20; library(scatterplot3d)
f <- matrix(sort(rnorm(n)),ncol=1); w <- matrix(c(0.5,0.2,-0.1),nrow=1)
fw <- f %*% w; x <- fw + matrix(rnorm(n*3, sd=c(.15,.05,.09)),ncol=3,byrow=TRUE)
s3d <- scatterplot3d(x,xlab=expression(x^1),ylab=expression(x^2),
                      zlab=expression(x^3),pch=16)
s3d$points3d(matrix(seq(from=min(f)-1,to=max(f)+1,length.out=2),ncol=1)%*%w,
             col="red",type="l")
s3d$points3d(fw,col="red",pch=16)
for (i in 1:nrow(x)) {
  s3d$points3d(x=c(x[i,1],fw[i,1]),y=c(x[i,2],fw[i,2]),z=c(x[i,3],fw[i,3]),
                col="grey",type="l") }

```

FIGURE 19.2: Geometry of factor models: Black dots are observed vectors \vec{X} in $p = 3$ dimensions. These were generated from the $q = 1$ dimensional factor scores \vec{F} by taking $\vec{F}\mathbf{w}$ (red dots) and adding independent noise (grey lines). The q -dimensional subspace along which all values of $\vec{F}\mathbf{w}$ must fall is also shown in red. (See also exercise 3.)

19.3 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. 19.4, Spearman's model becomes

$$\mathbf{X} = \boldsymbol{\epsilon} + \mathbf{G}\mathbf{w} \quad (19.12)$$

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 $i \neq j$,

$$\mathbf{v}_{ij} \equiv \text{Cov}[X_i, X_j] = w_i w_j \quad (19.13)$$

where I have introduced \mathbf{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 (19.12) is true,

$$\frac{\mathbf{v}_{ij}/\mathbf{v}_{kj}}{\mathbf{v}_{il}/\mathbf{v}_{kl}} = \frac{w_i w_j / w_k w_j}{w_i w_l / w_k w_l} \quad (19.14)$$

$$= \frac{w_i / w_k}{w_i / w_k} \quad (19.15)$$

$$= 1 \quad (19.16)$$

The relationship

$$\mathbf{v}_{ij}\mathbf{v}_{kl} = \mathbf{v}_{il}\mathbf{v}_{kj} \quad (19.17)$$

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

³Actually, the equations didn't hold when music was one of the grades, so Spearman argued musical ability did not load on general intelligence.

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. 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.

19.4 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:

$$\mathbf{v} = \mathbb{E} \left[\frac{1}{n} \mathbf{X}^T \mathbf{X} \right] \quad (19.18)$$

(This is the true, population covariance matrix on the left.) But the factor model tells us that

$$\mathbf{X} = \mathbf{F}\mathbf{w} + \boldsymbol{\epsilon} \quad (19.19)$$

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. 19.19 into Eq. 19.18 and see what happens:

$$\mathbb{E} \left[\frac{1}{n} \mathbf{X}^T \mathbf{X} \right] \quad (19.20)$$

$$= \frac{1}{n} \mathbb{E} [(\boldsymbol{\epsilon}^T + \mathbf{w}^T \mathbf{F}^T)(\mathbf{F}\mathbf{w} + \boldsymbol{\epsilon})] \quad (19.21)$$

$$= \frac{1}{n} (\mathbb{E} [\boldsymbol{\epsilon}^T \boldsymbol{\epsilon}] + \mathbf{w}^T \mathbb{E} [\mathbf{F}^T \boldsymbol{\epsilon}] + \mathbb{E} [\boldsymbol{\epsilon}^T \mathbf{F}] \mathbf{w} + \mathbf{w}^T \mathbb{E} [\mathbf{F}^T \mathbf{F}] \mathbf{w}) \quad (19.22)$$

$$= \psi + 0 + 0 + \frac{1}{n} \mathbf{w}^T n \mathbf{I} \mathbf{w} \quad (19.23)$$

$$= \psi + \mathbf{w}^T \mathbf{w} \quad (19.24)$$

Behold:

$$\mathbf{v} = \psi + \mathbf{w}^T \mathbf{w} \quad (19.25)$$

The individual-specific variables \mathbf{F} have gone away, leaving only population parameters on both sides of the equation.

19.4.1 Degrees of Freedom

It only takes a bit of playing with Eq. 19.25 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).⁴

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.

More unknowns (free parameters) than equations (constraints) This is fairly straightforward: there is no unique solution to Eq. 19.25; 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.

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 $p(p - 1)/2 = 7(7 - 1)/2 = 21 >$

⁴Notice 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.

$7(1) - 1(1-1)/2 = 7 = pq - q(q-1)/2$. 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.⁵

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:

$$\hat{\mathbf{v}} = \hat{\psi} + \hat{\mathbf{w}}^T \hat{\mathbf{w}} \quad (19.26)$$

The book-keeping for degrees of freedom here is the same as for Eq. 19.25. 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.

19.4.2 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:

$$\mathbf{v}_{ij} = w_i w_j \quad (19.27)$$

The exception is that $\mathbf{v}_{ii} = w_i^2 + \psi_i$, rather than w_i^2 . However, if we look at $\mathbf{u} = \mathbf{v} - \psi$, 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:

$$\mathbf{u}_{ij} = \frac{w_i}{w_k} \mathbf{u}_{kj} \quad (19.28)$$

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.⁶ 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,

⁵Actually, 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.

⁶We could also talk about the columns; it wouldn't make any difference.

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

$$\mathbf{u} = \mathbf{c}\mathbf{d}\mathbf{c}^T \quad (19.29)$$

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 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

$$\mathbf{c}_q\mathbf{d}_q\mathbf{c}_q^T = \mathbf{c}_q\mathbf{d}_q^{1/2}\mathbf{d}_q^{1/2}\mathbf{c}_q^T = (\mathbf{c}_q\mathbf{d}_q^{1/2})(\mathbf{c}_q\mathbf{d}_q^{1/2})^T \quad (19.30)$$

So we have

$$\mathbf{u} \approx (\mathbf{c}_q\mathbf{d}_q^{1/2})(\mathbf{c}_q\mathbf{d}_q^{1/2})^T \quad (19.31)$$

but at the same time we know that $\mathbf{u} = \mathbf{w}^T\mathbf{w}$. So we just identify \mathbf{w} with $(\mathbf{c}_q\mathbf{d}_q^{1/2})^T$:

$$\mathbf{w} = (\mathbf{c}_q\mathbf{d}_q^{1/2})^T \quad (19.32)$$

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.

19.4.3 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

$$\hat{\mathbf{u}} = \hat{\mathbf{v}} - \hat{\psi} \quad (19.33)$$

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:

$$\hat{\mathbf{w}} = (\mathbf{c}_q \mathbf{d}_q^{1/2})^T \quad (19.34)$$

$$\hat{\psi}_j = 1 - \sum_{r=1}^k w_{rj}^2 \quad (19.35)$$

$$\tilde{\mathbf{v}} \equiv \hat{\psi} + \hat{\mathbf{w}}^T \hat{\mathbf{w}} \quad (19.36)$$

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.

19.5 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 , why should we expect that $\hat{\mathbf{w}} \rightarrow \mathbf{w}$?) 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 \mathbf{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 $\vec{X}_i \sim \mathcal{N}(\mathbf{0}, \psi + \mathbf{w}^T \mathbf{w})$. This means that the log-likelihood is

$$L = -\frac{np}{2} \log 2\pi - \frac{n}{2} \log |\psi + \mathbf{w}^T \mathbf{w}| - \frac{n}{2} \text{tr}((\psi + \mathbf{w}^T \mathbf{w})^{-1} \hat{\mathbf{v}}) \quad (19.37)$$

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 \mathbf{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.⁷

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.

19.5.1 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.

19.5.2 Estimating Factor Scores

The probably the best method for estimating factor scores is the “regression” or “Thomson” method, which says

$$\hat{F}_{ir} = \sum_j X_{ij} b_{ij} \quad (19.38)$$

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} .

19.6 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

$$\mathbf{r}_\alpha = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} \quad (19.39)$$

⁷The 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.)

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:

$$\mathbf{X} = \epsilon + \mathbf{F}\mathbf{w} \quad (19.40)$$

$$= \epsilon + \mathbf{F}\mathbf{o}\mathbf{o}^T\mathbf{w} \quad (19.41)$$

$$= \epsilon + \mathbf{H}\mathbf{y} \quad (19.42)$$

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.⁸

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.⁹

Mathematically, this should not be surprising at all. The factors 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?

19.7 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,

$$\vec{\mathbf{X}} \sim \mathcal{N}(0, \mathbf{w}^T\mathbf{w} + \psi) \quad (19.43)$$

⁸Notice 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 $\vec{\mathbf{F}} \sim \mathcal{N}(0, \mathbf{I})$, then $\vec{\mathbf{F}}\mathbf{o} \sim \mathcal{N}(0\mathbf{o}, \mathbf{o}^T\mathbf{I}\mathbf{o}) = \mathcal{N}(0, \mathbf{I})$ — in other words, the rotated factor scores still satisfy our distributional assumptions.

⁹Remember 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.

if we make the usual distributional assumptions. Of course it might seem like it makes a more refined, conditional prediction,

$$\vec{X}|\vec{F} \sim \mathcal{N}(F\mathbf{w}, \psi) \quad (19.44)$$

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. 19.43.¹⁰

Now, without going through the trouble of factor analysis, one could always just postulate that

$$\vec{X} \sim \mathcal{N}(0, \mathbf{v}) \quad (19.45)$$

and estimate \mathbf{v} ; the maximum likelihood estimate of it is the observed covariance matrix, but really we could use any consistent estimator of the covariance matrix. The closer our \mathbf{v} 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* 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 estimation 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.)

19.7.1 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

¹⁰A 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, $p(F|X_1)$, 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.

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 G, 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

$$2[\ell(\hat{\Theta}) - \ell(\hat{\theta})] \rightsquigarrow \chi^2_{r-s} \quad (19.46)$$

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.¹¹ 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 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. 19.43 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.

19.7.1.1 R^2 and Goodness of Fit

For PCA, we saw that R^2 depends on the sum of the eigenvalues 18.1.3. For factor models, the natural notion of R^2 is the sum of squared factor loadings:

$$R^2 = \frac{\sum_{j=1}^q \sum_{k=1}^p w_{jk}^2}{p} \quad (19.47)$$

¹¹Suppose q is really 1, but by chance that gets rejected. Whether $q = 2$ gets rejected in turn is not an independent event!

(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 §19.4.2, $\mathbf{w} = (\mathbf{c}_q \mathbf{d}_q^{1/2})^T$, 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 higher.¹² 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 coincidence¹³.

Instead of looking at R^2 , it is much better to check goodness-of-fit by actually doing goodness-of-fit tests. We looked at some tests of multivariate goodness-of-fit in Chapter 15. 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 \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.

19.8 Factor Models versus PCA Once More

We began this chapter by seeking to add some noise, and some probabilistic assumptions, into PCA. The factor models we came up with are closely related to principal components, but are *not* the same. Many of the differences have been mentioned as we went, but it’s worth collecting some of the most important ones here.

1. Factor models assume that the data comes from a certain distribution, IID across data points. PCA assumes nothing about distributions at all. Moreover, factor models can be used *generatively*, to say how the latent factors cause the observable variables. PCA has nothing to say about the data-generating process.

¹²See also <http://bactra.org/weblog/523.html> for a similar experiment, with (not very elegant) R code.

¹³Peterson (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.

2. Factor models can be tested by their predictions on new data points; PCA cannot.
3. Factor models assume that the variance matrix of the data takes a special form, $\mathbf{w}^T \mathbf{w} + \psi$, where \mathbf{w} is $q \times p$ and ψ is diagonal. That is, the variance matrix must be “low rank plus noise”. PCA works no matter what sample variance matrix the data might have.
4. If the factor model *is* true, then the principal components are (or approach with enough data) the eigenvectors of $\mathbf{w}^T \mathbf{w} + \psi$. They do not approach the eigenvectors of $\mathbf{w}^T \mathbf{w}$, which would be the principal factors. If the noise is small, the difference may also be small, but the factor model can be correct, if perhaps not so useful, while ψ is as big as $\mathbf{w}^T \mathbf{w}$.
5. Factor models are subject to the rotation problem; PCA is not. Which one has the advantage here is unclear.
6. Similarly, a principal component is just a linear combination of the observable variables. A latent factor is another, distinct random variable. Differences in factor scores imply differences in the *expected* values of observables. Differences in projections on to principal components imply differences in realized values of observables. (It’s a little like the distinction between the predicted value for the response in a linear regression, which is a combination of the covariates, and the actual value of the response.)

19.9 Examples in R

19.9.1 Example 1: Back to the US *circa* 1977

We resume looking at the properties of the US states around 1977. In §18.3, we did a principal components analysis, finding a first component that seemed to mark the distinction between the South and the rest of the country, and a second that seemed to separate big, rich states from smaller, poorer ones. Let’s now subject the data to factor analysis. We begin with one factor, using the base R function `factanal`.

```
(state.fa1 <- factanal(state.x77,factors=1,scores="regression"))
##
## Call:
## factanal(x = state.x77, factors = 1, scores = "regression")
##
## Uniquenesses:
## Population      Income Illiteracy     Life Exp      Murder     HS Grad
##       0.957      0.791      0.235      0.437      0.308      0.496
##       Frost        Area
##       0.600      0.998
##
## Loadings:
##             Factor1
```

```

## Population -0.208
## Income      0.458
## Illiteracy   -0.875
## Life Exp    0.750
## Murder      -0.832
## HS Grad     0.710
## Frost       0.632
## Area
##
##          Factor1
## SS loadings   3.178
## Proportion Var 0.397
##
## Test of the hypothesis that 1 factor is sufficient.
## The chi square statistic is 91.97 on 20 degrees of freedom.
## The p-value is 3.34e-11

```

The output here tells us what fraction of the variance in each observable comes from its own noise (= the diagonal entries in $\hat{\psi}$ = “uniquenesses”). It also gives us the factor loadings, i.e., the rows of \hat{w} . Here there’s only one loading vector, since we set `factors = q = 1`. As a courtesy, the default printing method for the loadings leaves blanks where the loadings would be very small (here, for `Area`); this can be controlled through options (see `help(loadings)`). The last option picks between different methods of estimating the factor scores.

For comparison, here is the first principal component:

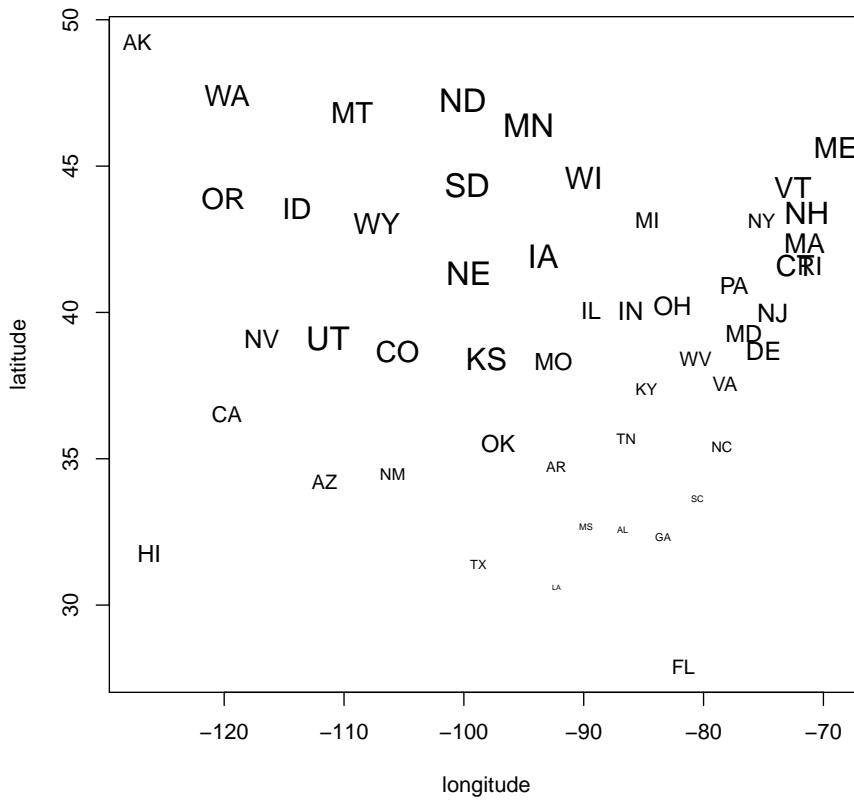
```

##
## Loadings:
##          PC1   PC2   PC3   PC4   PC5   PC6   PC7   PC8
## Population 0.126  0.411 -0.656 -0.409  0.406           -0.219
## Income     -0.299  0.519 -0.100          -0.638  0.462
## Illiteracy  0.468          0.353          0.387 -0.620 -0.339
## Life Exp   -0.412          -0.360  0.443  0.327  0.219 -0.256  0.527
## Murder      0.444  0.307  0.108 -0.166 -0.128 -0.325 -0.295  0.678
## HS Grad    -0.425  0.299          0.232          -0.645 -0.393 -0.307
## Frost      -0.357 -0.154  0.387 -0.619  0.217  0.213 -0.472
## Area        0.588  0.510  0.201  0.499  0.148  0.286
##
##          PC1   PC2   PC3   PC4   PC5   PC6   PC7   PC8
## SS loadings 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
## Proportion Var 0.125 0.125 0.125 0.125 0.125 0.125 0.125 0.125
## Cumulative Var 0.125 0.250 0.375 0.500 0.625 0.750 0.875 1.000

```

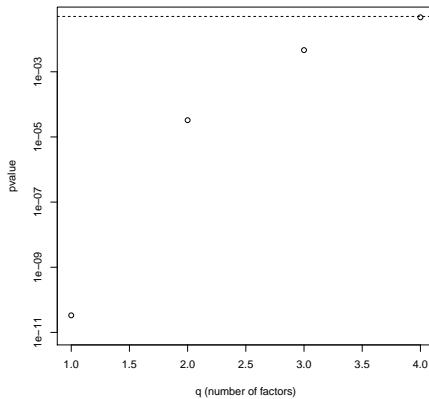
The first principal component is clearly not the same as the single common factor we extracted, even after a sign change, but it’s not shockingly dissimilar either, as a map shows.

Of course, why use just one factor? Given the number of observables, we can fit up to four factors before the problem becomes totally unidentified and `factanal`



```
plot.states_scaled(state.fa1$score[,1],min.size=0.3,max.size=1.5,
                  xlab="longitude",ylab="latitude")
```

FIGURE 19.3: *The US states, plotted in position with symbols scaled by their factor scores in a one-factor model. Compare to Figure 18.5, which is where the `plot.states_scaled` function comes from. (Try plotting the negative of the factor scores to make the maps look more similar.)*



```
plot(1:4,pvalues,xlab="q (number of factors)", ylab="pvalue",
     log="y",ylim=c(1e-11,0.04))
abline(h=0.05,lty="dashed")
```

FIGURE 19.4: *Gaussian likelihood ratio test p-value for models with various numbers of latent factors, fit to the US-in-1977 data.*

refuses to work. That function automatically runs the likelihood ratio test every time it fits a model, assuming Gaussian distributions for the observables. As remarked, this can work reasonably well for non-Gaussian distributions if they're not too non-Gaussian, especially if n is much larger than the number of parameters; of course $n = 50$ is pretty modest. Still, let's try it:

```
pvalues <- sapply(1:4,function(q){factanal(state.x77,factors=q)$PVAL})
signif(pvalues,2)
## objective objective objective objective
##   3.3e-11   3.3e-05   4.6e-03   4.7e-02
```

(Figure 19.4 plots the results.) None of the models has a p -value crossing the conventional 0.05 level, meaning all of them show systematic, detectable departures from what the data should look like if the factor model were true. Still, the four-factor model comes close.

Notice that the first factor's loadings do not stay the same when we add more factors, unlike the first principal component:

```
print(factanal(state.x77, factors=4)$loadings)
##
## Loadings:
##           Factor1 Factor2 Factor3 Factor4
## Population          0.636
```

```

## Income      0.313  0.281  0.561  0.189
## Illiteracy -0.466 -0.878
## Life Exp   0.891  0.191
## Murder     -0.792 -0.384  0.109  0.405
## HS Grad    0.517  0.418  0.581
## Frost      0.128  0.679  0.105 -0.460
## Area       -0.174          0.796
##
##           Factor1 Factor2 Factor3 Factor4
## SS loadings  2.054   1.680   1.321   0.821
## Proportion Var 0.257   0.210   0.165   0.103
## Cumulative Var 0.257   0.467   0.632   0.734

```

19.9.2 Example 2: Stocks

Classical financial theory suggests that the log-returns of corporate stocks should be IID Gaussian random variables, but allows for the possibility that different stocks might be correlated with each other. In fact, theory suggests that the returns to any given stock should be the sum of two components: one which is specific to that firm, and one which is common to all firms. (More specifically, the common component is one which couldn't be eliminated even in a perfectly diversified portfolio.) This in turn implies that stock returns should match a one-factor model.

19.10 Reification, and Alternatives to Factor Models

A natural impulse, when looking at something like Figure 19.1, 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?

19.10.1 The Rotation Problem Again

Consider the following matrix, call it r :

$$\begin{bmatrix} \cos 30 & -\sin 30 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (19.48)$$

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 19.5. Now instead of X_1 being correlated with

The rest of this example is deliberately omitted, since it will be a homework assignment for 2015.

[[ATTN: Get population coding data from neuro., and use that as an example?]]

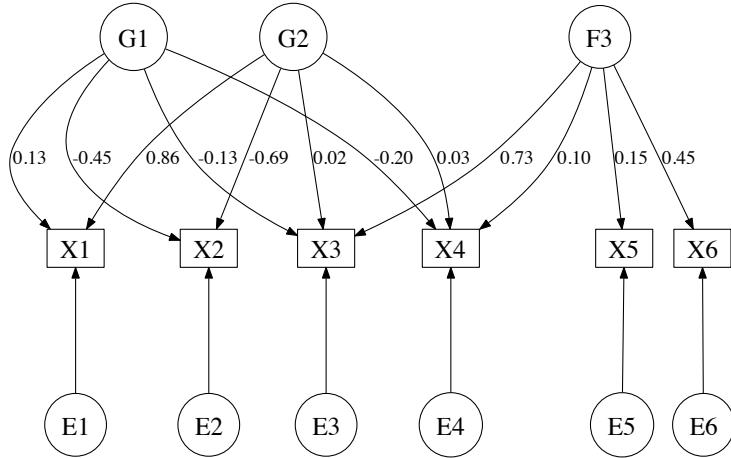


FIGURE 19.5: The model from Figure 19.1, after rotating the first two factors by 30 degrees around the third factor's axis. The new factor loadings are rounded to two decimal places.

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.

19.10.2 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 and $1 - \alpha$). Mixture models are a very flexible and useful way of representing complicated

probability distributions¹⁴, and we will look at them in detail in Chapter 21.

I bring up mixture models here because there is a very remarkable result: any linear factor model with q factors is equivalent to some mixture model with $q + 1$ 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 drawing from $q + 1$ clusters, then a model with q 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.

All of which suggests that factor analysis can't alone really tell us whether we have q continuous latent variables, or one discrete hidden variable taking $q + 1$ values.

19.10.3 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:

$$X_{ij} = \eta_{ij} + \sum_{k=1}^q A_{ik} T_{kj} = \eta_{ij} + \vec{A}_i \cdot \vec{T}_j \quad (19.49)$$

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

¹⁴They are also a probabilistic, predictive alternative to the kind of clustering techniques you may have seen in data mining: 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.

features:

$$\mathbf{E}[X_{ia}X_{ib}] \quad (19.50)$$

$$= \mathbf{E}[(\eta_{ia} + \vec{A}_i \cdot \vec{T}_a)(\eta_{ib} + \vec{A}_i \cdot \vec{T}_b)] \quad (19.51)$$

$$= \mathbf{E}[\eta_{ia}\eta_{ib}] + \mathbf{E}[\eta_{ia}\vec{A}_i \cdot \vec{T}_b] + \mathbf{E}[\eta_{ib}\vec{A}_i \cdot \vec{T}_a] + \mathbf{E}[(\vec{A}_i \cdot \vec{T}_a)(\vec{A}_i \cdot \vec{T}_b)] \quad (19.52)$$

$$= 0 + 0 + 0 + \mathbf{E}\left[\left(\sum_{k=1}^q A_{ik} T_{ka}\right)\left(\sum_{l=1}^q A_{il} T_{lb}\right)\right] \quad (19.53)$$

$$= \mathbf{E}\left[\sum_{k,l} A_{ik} A_{il} T_{ka} T_{lb}\right] \quad (19.54)$$

$$= \sum_{k,l} \mathbf{E}[A_{ik} A_{il}] T_{ka} T_{lb} \quad (19.55)$$

$$= \sum_{k,l} \mathbf{E}[A_{ik} A_{il}] \mathbf{E}[T_{ka} T_{lb}] \quad (19.56)$$

$$= \sum_{k=1}^q \mathbf{E}[T_{ka} T_{kb}] \quad (19.57)$$

where to get the last line I use the fact that $\mathbf{E}[A_{ik} A_{il}] = 1$ 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

$$\mathbf{E}[T_{ka} T_{kb}] = \mathbf{E}[T_{ka}] \mathbf{E}[T_{kb}] = z_a z_b \quad (19.58)$$

and

$$\mathbf{E}[X_{ia}X_{ib}] = q z_a z_b \quad (19.59)$$

Of course, in the one-factor model,

$$\mathbf{E}[X_{ia}X_{ib}] = w_a w_b \quad (19.60)$$

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. 19.49. 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 $q z_a z_b$. 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.

```
rthomson <- function(n, d, a, q, ability.mean = 0, ability.sd = 1) {
  require(MASS)
  general.per.test = sample(1:a, size = d, replace = TRUE)
  specifics.per.test = sample(1:q, size = d, replace = TRUE)
  general.to.tests = matrix(0, a, d)
  for (i in 1:d) {
    abilities = sample(1:a, size = general.per.test[i], replace = FALSE)
    general.to.tests[abilities, i] = 1
  }
  sigma = matrix(0, a, a)
  diag(sigma) = (ability.sd)^2
  mu = rep(ability.mean, a)
  x = mvrnorm(n, mu, sigma)
  general.tests = x %*% general.to.tests
  specific.tests = matrix(0, n, d)
  noisy.tests = matrix(0, n, d)
  for (i in 1:d) {
    j = specifics.per.test[i]
    specifics = rnorm(n, mean = ability.mean * j, sd = ability.sd * sqrt(j))
    specific.tests[, i] = general.tests[, i] + specifics
    noises = rnorm(n, mean = 0, sd = ability.sd)
    noisy.tests[, i] = specific.tests[, i] + noises
  }
  tm = list(data = noisy.tests, general.ability.pattern = general.to.tests,
            numbers.of.specifcics = specifics.per.test, ability.matrix = x, specific.tests = specific.tests)
  return(tm)
}
```

CODE EXAMPLE 31: Function for simulating the Thomson latent-sampling model.

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.¹⁵ 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.

Code Example 31 simulates the Thomson model.

¹⁵When 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.

```

tm <- rthomson(50,11,500,50)
factanal(tm$data,1)
##
## Call:
## factanal(x = tm$data, factors = 1)
##
## Uniquenesses:
## [1] 0.169 0.816 0.368 0.458 0.537 0.332 0.378 0.758 0.757 0.999 0.844
##
## Loadings:
##      Factor1
## [1,] 0.911
## [2,] 0.428
## [3,] 0.795
## [4,] 0.737
## [5,] 0.680
## [6,] 0.817
## [7,] 0.789
## [8,] 0.492
## [9,] 0.493
## [10,]
## [11,] 0.394
##
##           Factor1
## SS loadings    4.582
## Proportion Var  0.417
##
## Test of the hypothesis that 1 factor is sufficient.
## The chi square statistic is 50.17 on 44 degrees of freedom.
## The p-value is 0.242

```

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; this attempt gave the proportion of variance associated with the factor as 0.42, and the p -value as 0.24. 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 19.6). 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.

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:

$$X_{ij} = \eta_{ij} + \sum_{k=1}^{q_1} A_{ik} T_{kj} + \sum_{k=1}^{q_2} B_{ik} R_{kj} + \dots \quad (19.61)$$

where the T_{kj} coefficients are uncorrelated with the R_{kj} , and so forth. In expectation,

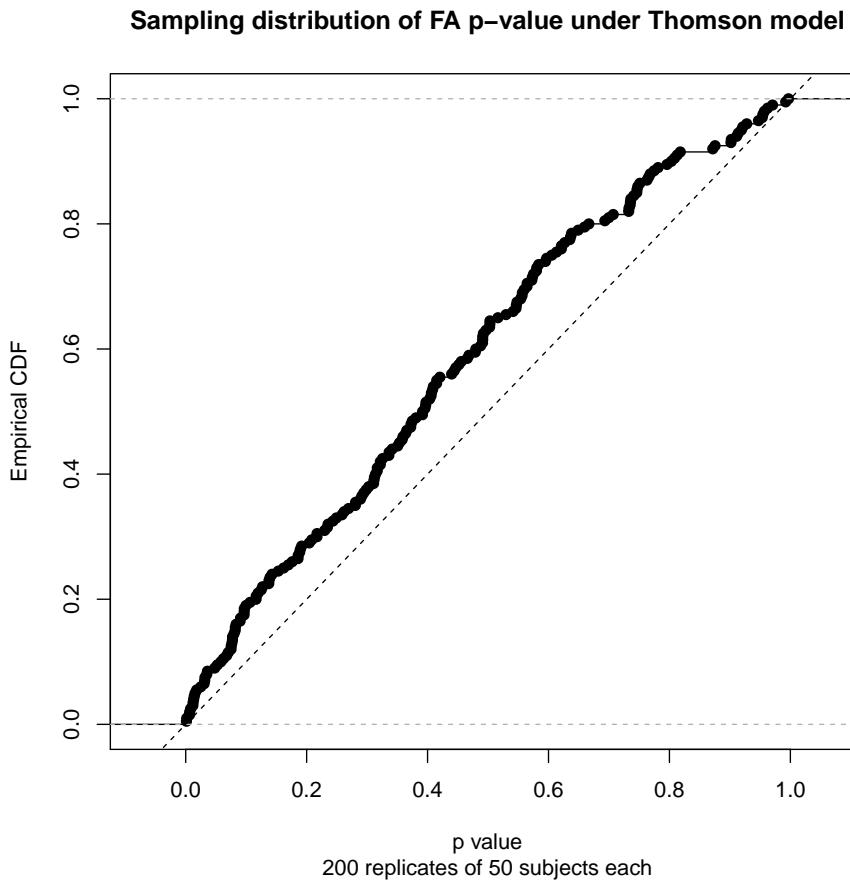


FIGURE 19.6: Mimicry of the one-factor model by the Thomson model. The Thomson model was simulated 200 times with the parameters given above; each time, the simulated data was then fit to a factor model with one factor, and the p-value of the goodness-of-fit test extracted. The plot shows the empirical 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).

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.¹⁶

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.

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, motivation, etc.), which overlap partially from question to question.

19.11 Further Reading

The classical papers by Spearman (1904) and Thurstone (1934) are readily available online, and very much worth reading for getting a sense of the problems which motivated the introduction of factor analysis, and the skill with which the founders grappled with them. Loehlin (1992) is a decent textbook intended for psychologists; the presumably mathematical and statistical level is decidedly lower than that of this book, but it's still useful. Thomson (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 the method. While out of print, used copies are reasonably plentiful and cheap, and at least one edition is free online.

On purely statistical issues related to factor analysis, Bartholomew (1987) is by far the best reference I have found; it quite properly sets it in the broader context of latent variable models, including the sort of latent class models we will explore in Chapter 21. The computational advice of that edition is, necessarily, now quite obsolete; there is an updated edition from 2011, which I have not been able to consult by the time of writing.

The use of factor analysis in psychological testing has given rise to a large controversial literature, full of claims, counter-claims, counter-counter-claims, and so on *ad nauseam*. Without, here, going into that, I will just note that to the extent the

¹⁶A recent paper on the Thomson model (Bartholomew *et al.*, 2009) proposes just this modification to multiple factors and to Bernoulli sampling. However, I proposed this independently, in the fall 2008 version of these notes, about a year before their paper.

strongest arguments against (say) reifying the general factor extracted from intelligence tests as “general intelligence” are good arguments, they do not just apply to intelligence, but also to personality tests, and indeed to many procedures outside psychology. In other words, if there’s a problem, it’s not *just* a problem for intelligence testing alone, or even for psychology alone. On this, see Glymour (1998) and Borsboom (2005, 2006).

Exercises

1. Prove Eq. 19.13.
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. Consider Figure 19.2 and its code. What is \mathbf{w} here? What is ψ ? What is the implied covariance matrix of \vec{X} ?
4. Show that the correlation between the j^{th} feature and G , in the one-factor model, is w_j .
5. Check that Eq. 19.11 and Eq. 19.25 are compatible.
6. Find the weights b_{ij} for the Thomson estimator of factor scores (Eq. 19.38), assuming you know \mathbf{w} . Do you need to assume a Gaussian distribution?

Chapter 20

Nonlinear Dimensionality Reduction: Local Linear Embedding and Diffusion Maps

Re-purposed data mining notes
[[TODO: Fix cross-references]]

[[TODO: Fix notation]]

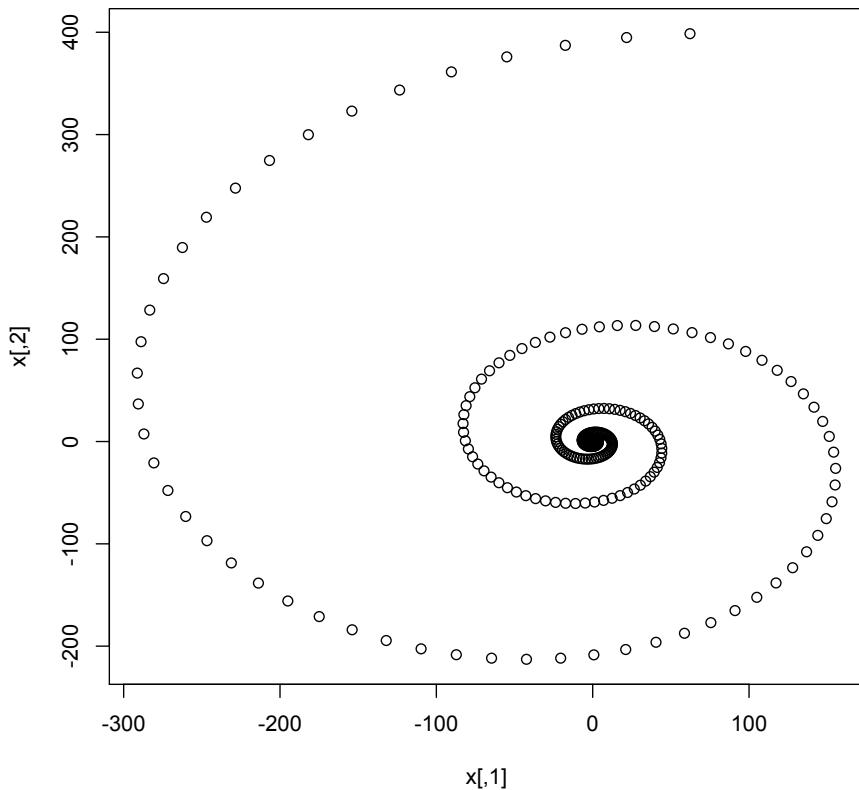
20.1 Why We Need Nonlinear Dimensionality Reduction

Consider the points shown in Figure 20.1. Even though there are two features, a.k.a. coordinates, all of the points fall on a one-dimensional curve (as it happens, a logarithmic spiral). This is exactly the kind of constraint which it would be good to recognize and exploit — rather than using two separate coordinates, we could just say how far along the curve a data-point is.

PCA will do poorly with data like this. Remember that to get a one-dimensional representation out of it, we need to take the first principal component, which is the *straight line* along which the data's projections have the most variance. If this works for capturing structure along the spiral, then projections on to the first PC should have the same order that the points have along the spiral.¹ Since, fortuitously, the data are already in that order, we can just plot the first PC against the row index (Figure 20.2). The results are — there is really no other word for it — *screwy*.

So, PCA with one principal component fails to capture the one-dimensional structure of the spiral. We could add another principal component, but then we've just rotated our two-dimensional data. In fact, *any* linear dimensionality-reduction method is going to fail here, simply because the spiral is not even approximately a one-dimensional *linear subspace*.

¹It wouldn't matter if the coordinate increased as we went out along the spiral or decreased, just so long as it was monotonic.



```

x=matrix(c(exp(-0.2*(-(1:300)/10))*cos(-(1:300)/10),
           exp(-0.2*(-(1:300)/10))*sin(-(1:300)/10)),
         ncol=2)
plot(x)

```

FIGURE 20.1: Two-dimensional data constrained to a smooth one-dimensional region, namely the logarithmic spiral, $r = e^{-0.2\theta}$ in polar coordinates.

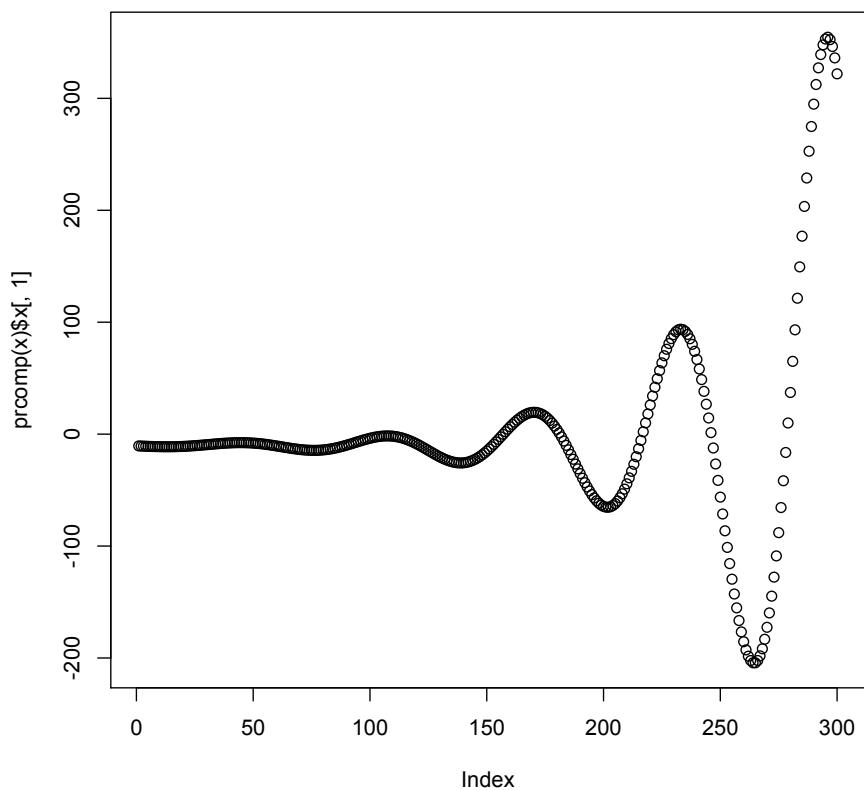
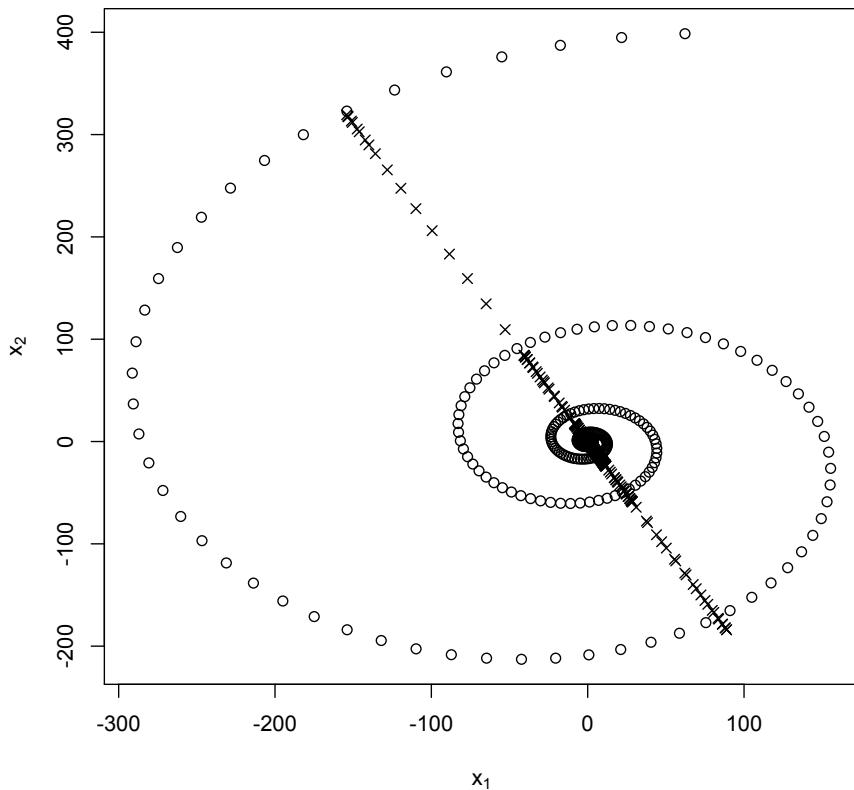


FIGURE 20.2: Projections of the spiral points on to their first principal component.



```

fit.all = prcomp(x)
approx.all=fit.all$x[,1] %*% t(fit.all$rotation[,1])
plot(x,xlab=expression(x[1]),ylab=expression(x[2]))
points(approx.all,pch=4)

```

FIGURE 20.3: *Spiral data (circles) replotted with their one-dimensional PCA approximations (crosses).*

What then are we to do?

1. Stick to not-too-nonlinear structures.
2. Somehow decompose nonlinear structures into linear subspaces.
3. Generalize the eigenvalue problem of minimizing distortion.

There's not a great deal to be said about (1). Some curves can be approximated by linear subspaces without too much heartbreak. (For instance, see Figure 20.4.) We can use things like PCA on them, and so long as we remember that we're just seeing an approximation, we won't go too far wrong. But fundamentally this is *weak*. (2) is hoping that we can somehow build a strong method out of this weak one; as it happens we can, and it's called locally linear embedding (and its variants). The last is diffusion maps, which we'll cover next lecture.

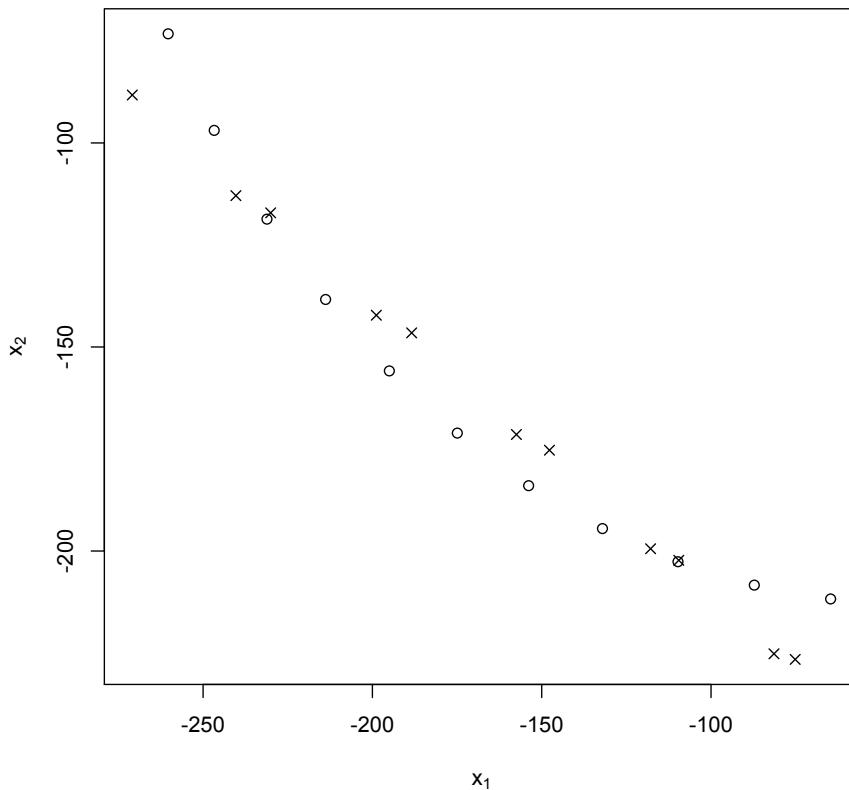
20.2 Local Linearity and Manifolds

Let's look again at Figure 20.4. A one-dimensional linear subspace is, in plain words, a straight line. By doing PCA on this part of the data alone, we are approximating a segment of the spiral curve by a straight line. Since the segment is not *very* curved, the approximation is reasonably good. (Or rather, the segment was chosen so the approximation would be good, consequently it had to have low curvature.) Notice that this error is not a random scatter of points around the line, but rather a systematic mis-match between the true curve and the line — a **bias** which would not go away no matter how much data we had from the spiral. The size of the bias depends on how big a region we are using, and how much the tangent direction to the curve changes across that region — the **average curvature**. By using small regions when the curvature is high and big regions when the curvature is low, we can maintain any desired degree of approximation.

If we shifted to a different part of the curve, we could do PCA on the data there, too, getting a different principal component and a different linear approximations to the data. Generally, as we move around the degree of curvature will change, so the size of the region we'd use would also need to grow or shrink.

This suggests that we could make some progress towards learning nonlinear structures in the data by patching together lots of linear structures. We could, for example, divide up the whole data space into regions, and do a separate PCA in each region. Here we'd hope that in each region we needed only a single principal component. Such hopes would generally be dashed, however, because this is a bit too simple-minded to really work.

1. We'd need to chose the number of regions, introducing a trade-off between having many points in each region (so as to deal with noise) and having small regions (to keep the linear approximation good).
2. Ideally, the regions should be of different sizes, depending on average curvature, but we don't know the curvature.



```

fit = prcomp(x[270:280,])
pca.approx = fit$x[,1] %*% t(fit$rotation[,1]) + colMeans(x[270:280,])
plot(rbind(x[270:280,],pca.approx),type="n",
      xlab=expression(x[1]),ylab=expression(x[2]))
points(x[270:280,])
points(pca.approx,pch=4)

```

FIGURE 20.4: Portion of the spiral data (circles) together with its one-dimensional PCA approximation (crosses).

3. What happens at the boundaries between regions? The principal components of adjacent regions could be pointing in totally different directions.

Nonetheless, this is the core of a good idea. To make it work, we need to say just a little about **differential geometry**, specifically the idea of a **manifold**.² For our purposes, a manifold is a smooth, curved subset of a Euclidean space, in which it is **embedded**. The spiral curve (not the isolated points I plotted) is a one-dimensional manifold in the plane, just as are lines, circles, ellipses and parabolas. The surface of a sphere or a torus is a two-dimensional manifold, like a plane. The essential fact about a q -dimensional manifold is that it can be arbitrarily well-approximated by a q -dimensional linear subspace, the **tangent space**, by taking a sufficiently small region about any point.³ (This generalizes the fact any sufficiently small part of a curve about any point looks very much like a straight line, the tangent line to the curve at that point.) Moreover, as we move from point to point, the local linear approximations change continuously, too. The more rapid the change in the tangent space, the bigger the curvature of the manifold. (Again, this generalizes the relation between curves and their tangent lines.) So if our data come from a manifold, we should be able to do a local linear approximation around every part of the manifold, and then smoothly interpolate them together into a single global system. To do dimensionality reduction — to learn the manifold — we want to find these global low-dimensional coordinates.⁴

20.3 Locally Linear Embedding (LLE)

Locally linear embedding (or: local linear embedding, you see both) is a clever scheme for finding low-dimensional global coordinates when the data lie on (or very near to) a manifold embedded in a high-dimensional space. The trick is to do a different linear dimensionality reduction at each point (because locally a manifold looks linear) and then combine these with minimal discrepancy. It was introduced by Roweis and Saul (2000), though Saul and Roweis (2003) has a fuller explanation. I don't think it uses

²Differential geometry is a very beautiful and important branch of mathematics, with its roots in the needs of geographers in the 1800s to understand the curved surface of the Earth in detail (**geodesy**). The theory of curved spaces they developed for this purpose generalized the ordinary vector calculus and Euclidean geometry, and turned out to provide the mathematical language for describing space, time and gravity (Einstein's general theory of relativity; Lawrie (1990)), the other fundamental forces of nature (gauge field theory; Lawrie (1990)), dynamical systems Arnol'd (1973); Guckenheimer and Holmes (1983), and indeed statistical inference (information geometry; Kass and Vos (1997); Amari and Nagaoka (1993/2000)). Good introductions are Spivak (1965) and Schutz (1980) (which confines the physics to one (long) chapter on applications).

³If it makes you happier: every point has an open neighborhood which is homeomorphic to \mathbb{R}^q , and the transition from neighborhood to neighborhood is continuous and differentiable.

⁴There are technicalities here which I am going to gloss over, because this is not a class in differential geometry. (Take one, it's good for you!) The biggest one is that most manifolds don't admit of a truly *global* coordinate system, one which is good everywhere without exception. But the places where it breaks down are usually isolated point and easily identified. For instance, if you take a sphere, almost every point can be identified by latitude and longitude — except for the poles, where longitude becomes ill-defined. Handling this in a mathematically precise way is tricky, but since these are probability-zero cases, we can ignore them in a statistics class.

any elements which were unknown, mathematically, since the 1950s. Rather than diminishing what Roweis and Saul did, this should make the rest of us feel humble...

The LLE procedure has three steps: it builds a neighborhood for each point in the data; finds the weights for linearly approximating the data in that neighborhood; and finally finds the low-dimensional coordinates best reconstructed by those weights. This low-dimensional coordinates are then returned.

To be more precise, the LLE algorithm is given as inputs an $n \times p$ data matrix \mathbf{X} , with rows \vec{x}_i ; a desired number of dimensions $q < p$; and an integer k for finding local neighborhoods, where $k \geq q + 1$. The output is supposed to be an $n \times q$ matrix \mathbf{Y} , with rows \vec{y}_i .

1. For each \vec{x}_i , find the k nearest neighbors.
2. Find the weight matrix \mathbf{w} which minimizes the residual sum of squares for reconstructing each \vec{x}_i from its neighbors,

$$RSS(\mathbf{w}) \equiv \sum_{i=1}^n \left\| \vec{x}_i - \sum_{j \neq i} w_{ij} \vec{x}_j \right\|^2 \quad (20.1)$$

where $w_{ij} = 0$ unless \vec{x}_j is one of \vec{x}_i 's k -nearest neighbors, and for each i , $\sum_j w_{ij} = 1$. (I will come back to this constraint below.)

3. Find the coordinates \mathbf{Y} which minimize the reconstruction error using the weights,

$$\Phi(\mathbf{Y}) \equiv \sum_{i=1}^n \left\| \vec{y}_i - \sum_{j \neq i} w_{ij} \vec{y}_j \right\|^2 \quad (20.2)$$

subject to the constraints that $\sum_i Y_{ij} = 0$ for each j , and that $\mathbf{Y}^T \mathbf{Y} = \mathbf{I}$. (I will come back to those constraints below, too.)

20.3.1 Finding Neighborhoods

In step 1, we define local neighborhoods for each point. By defining these in terms of the k nearest neighbors, we make them physically large where the data points are widely separated, and physically small when the density of the data is high. We don't know that the curvature of the manifold is low when the data are sparse, but we do know that, whatever is happening out there, we have very little idea what it is, so it's safer to approximate it crudely. Conversely, if the data are dense, we can capture both high and low curvature. If the actual curvature is low, we might have been able to expand the region without loss, but again, this is playing it safe. So, to summarize, using k -nearest neighborhoods means we take a fine-grained view where there is a lot of data, and a coarse-grained view where there is little data.

It's not strictly necessary to use k -nearest neighbors here; the important thing is to establish *some* neighborhood for each point, and to do so in a way which conforms or adapts to the data.

20.3.2 Finding Weights

Step 2 can be understood in a number of ways. Let's start with the local linearity of a manifold. Suppose that the manifold was *exactly* linear around \vec{x}_i , i.e., that it and its neighbors belonged to a q -dimensional linear subspace. Since $q+1$ points generally define a q -dimensional subspace, there would be *some* combination of the neighbors which reconstructed \vec{x}_i exactly, i.e., some set of weights w_{ij} such that

$$\vec{x}_i = \sum_j w_{ij} \vec{x}_j \quad (20.3)$$

Conversely, if there are such weights, then \vec{x}_i and (some of) its neighbors *do* form a linear subspace. Since every manifold is locally linear, by taking a sufficiently small region around each point we get arbitrarily close to having these equations hold — $n^{-1}RSS(\mathbf{w})$ should shrink to zero as n grows.

Vitally, the *same* weights would work to reconstruct \mathbf{x}_i both in the high-dimensional embedding space and the low-dimensional subspace. This means that it is the weights around a given point which characterize what the manifold looks like there (provided the neighborhood is small enough compared to the curvature). Finding the weights gives us the same information as finding the tangent space. This is why, in the last step, we will only need the weights, not the original vectors.

Now, about the constraints that $\sum_j w_{ij} = 1$. This can be understood in two ways, geometrically and probabilistically. Geometrically, what it gives us is invariance under translation. That is, if we add any vector \vec{c} to \vec{x}_i and all of its neighbors, nothing happens to the function we're minimizing:

$$\vec{x}_i + \vec{c} - \sum_j w_{ij}(\vec{x}_j + \vec{c}) = \vec{x}_i + \vec{c} - \left(\sum_j w_{ij} \vec{x}_j \right) - \vec{c} \quad (20.4)$$

$$= \vec{x}_i - \sum_j w_{ij} \vec{x}_j \quad (20.5)$$

Since we are looking at the same shape of manifold no matter how we move it around in space, translational invariance is a constraint we want to impose.

Probabilistically, forcing the weights to sum to one makes \mathbf{w} a stochastic transition matrix.⁵ This should remind you of page-rank, where we built a Markov chain transition matrix from the graph connecting web-pages. There is a tight connection here, which we'll return to next time under the heading of **diffusion maps**; for now this is just to tantalize.

We will see below how to actually minimize the squared error computationally; as you probably expect by now, it reduces to an eigenvalue problem. Actually it reduces to a bunch (n) of eigenvalue problems: because there are no constraints *across* the rows of \mathbf{w} , we can find the optimal weights for each point separately. Naturally, this simplifies the calculation.

⁵Actually, it really only does that if $w_{ij} \geq 0$. In that case we are approximating \vec{x}_i not just by a linear combination of its neighbors, but by a **convex** combination. Often one gets all positive weights anyway, but it can be helpful to impose this extra constraint.

20.3.2.1 $k > p$

If k , the number of neighbors, is greater than p , the number of features, then (in general) the space spanned by k distinct vectors is the whole space. Then \vec{x}_i can be written *exactly* as a linear combination of its k -nearest neighbors.⁶ In fact, if $k > p$, then not only is there a solution to $\vec{x}_i = \sum_j w_{ij} \vec{x}_j$, there are generally *infinitely many* solutions, because there are more unknowns (k) than equations (p). When this happens, we say that the optimization problem is **ill-posed**, or **irregular**. There are many ways of **regularizing** ill-posed problems. A common one, for this case, is what is called L_2 or **Tikhonov regularization**: instead of minimizing

$$\|\vec{x}_i - \sum_j w_{ij} \vec{x}_j\|^2 \quad (20.6)$$

pick an $\alpha > 0$ and minimize

$$\|\vec{x}_i - \sum_j w_{ij} \vec{x}_j\|^2 + \alpha \sum_j w_{ij}^2 \quad (20.7)$$

This says: pick the weights which minimize a combination of reconstruction error *and* the sum of the squared weights. As $\alpha \rightarrow 0$, this gives us back the least-squares problem. To see what the second, sum-of-squared-weights term does, take the opposite limit, $\alpha \rightarrow \infty$: the squared-error term becomes negligible, and we just want to minimize the Euclidean (“ L_2 ”) norm of the weight vector w_{ij} . Since the weights are constrained to add up to 1, we can best achieve this by making all the weights equal — so some of them can’t be vastly larger than the others, and they stabilize at a definite preferred value. Typically α is set to be small, but not zero, so we allow some variation in the weights if it really helps improve the fit.

We will see how to actually implement this regularization later, when we look at the eigenvalue problems connected with LLE. The L_2 term is an example of a **penalty term**, used to stabilize a problem where just matching the data gives irregular results, and there is an art to optimally picking λ ; in practice, however, LLE results are often fairly insensitive to it, when it’s needed at all⁷. Remember, the whole situation only comes up when $k > p$, and p can easily be very large — [[6380 for the gene-expression data]], much larger for the [[Times corpus]], etc.

20.3.3 Finding Coordinates

As I said above, if the local neighborhoods are small compared to the curvature of the manifold, weights in the embedding space and weights on the manifold should be the same. (More precisely, the two sets of weights are exactly equal for linear subspaces, and for other manifolds they can be brought arbitrarily close to each other by shrinking the neighborhood sufficiently.) In the third and last step of LLE, we have just

⁶This is easiest to see when \vec{x}_i lies inside the body which has its neighbors as vertices, their **convex hull**, but is true more generally.

⁷It’s no accident that the scaling factor for the penalty term is written with a Greek letter; it can also be seen as the Lagrange multiplier enforcing a constraint on the solution (§E.3.3).

calculated the weights in the embedding space, so we take them to be *approximately* equal to the weights on the manifold, and solve for coordinates on the manifold.

So, taking the weight matrix \mathbf{w} as fixed, we ask for the \mathbf{Y} which minimizes

$$\Phi(\mathbf{Y}) = \sum_i \left\| \vec{y}_i - \sum_{j \neq i} \vec{y}_j w_{ij} \right\|^2 \quad (20.8)$$

That is, what should the coordinates \vec{y}_i be on the manifold, that *these* weights reconstruct them?

As mentioned, some constraints are going to be needed. Remember that we saw above that we could add any constant vector \vec{c} to \vec{x}_i and its neighbors without affecting the sum of squares, because $\sum_j w_{ij} = 1$. We could do the same with the \vec{y}_i , so the minimization problem, as posed, has an infinity of equally-good solutions. To fix this — to “break the degeneracy” — we impose the constraint

$$\frac{1}{n} \sum_i \vec{y}_i = 0 \quad (20.9)$$

Since if the mean vector was *not* zero, we could just subtract it from all the \vec{y}_i without changing the quality of the solution, this is just a book-keeping convenience.

Similarly, we also impose the convention that

$$\frac{1}{n} \mathbf{Y}^T \mathbf{Y} = \mathbf{I} \quad (20.10)$$

i.e., that the covariance matrix of \mathbf{Y} be the (q -dimensional) identity matrix. This is not as substantial as it looks. If we found a solution where the covariance matrix of \mathbf{Y} was *not* diagonal, we could use PCA to rotate the new coordinates on the manifold so they were uncorrelated, giving a diagonal covariance matrix. The only bit of this which is not, again, a book-keeping convenience is assuming that all the coordinates have the same variance — that the diagonal covariance matrix is in fact \mathbf{I} .

This optimization problem is like [[multi-dimensional scaling]]: we are asking for low-dimensional vectors which preserve certain relationships (averaging weights) among high-dimensional vectors. We are also asking to do it under constraints, which we will impose through Lagrange multipliers. Once again, it turns into an eigenvalue problem, though one just a bit more subtle than what we saw with PCA in Chapter 18⁸.

Unfortunately, finding the coordinates does *not* break up into n smaller problems, the way finding the weights did, because each row of \mathbf{Y} appears in Φ multiple times, once as the focal vector \vec{y}_i , and then again as one of the neighbors of other vectors.

⁸One reason to suspect the appearance of eigenvalues, in addition to my very heavy-handed foreshadowing, is that eigenvectors are automatically orthogonal to each other and normalized, so making the columns of \mathbf{Y} be the eigenvectors of some matrix would automatically satisfy Eq. 20.10.

20.4 More Fun with Eigenvalues and Eigenvectors

To sum up: for each \vec{x}_i , we want to find the weights w_{ij} which minimize

$$RSS_i(\mathbf{w}) = \left\| \vec{x}_i - \sum_j w_{ij} \vec{x}_j \right\|^2 \quad (20.11)$$

where $w_{ij} = 0$ unless \vec{x}_j is one of the k nearest neighbors of \vec{x}_i , under the constraint that $\sum_j w_{ij} = 1$. Given those weights, we want to find the q -dimensional vectors \vec{y}_i which minimize

$$\Phi(\mathbf{Y}) = \sum_{i=1}^n \left\| \vec{y}_i - \sum_j w_{ij} \vec{y}_j \right\|^2 \quad (20.12)$$

with the constraints that $n^{-1} \sum_i \vec{y}_i = 0$, $n^{-1} \mathbf{Y}^T \mathbf{Y} = \mathbf{I}$.

20.4.1 Finding the Weights

In this subsection, assume that j just runs over the neighbors of \vec{x}_i , so we don't have to worry about the weights (including w_{ii}) which we know are zero.

We saw that RSS_i is invariant if we add an arbitrary \vec{c} to all the vectors. Set $\vec{c} = -\vec{x}_i$, centering the vectors on the focal point \vec{x}_i :

$$RSS_i = \left\| \sum_j w_{ij} (\vec{x}_j - \vec{x}_i) \right\|^2 \quad (20.13)$$

$$= \left\| \sum_j w_{ij} \vec{z}_j \right\|^2 \quad (20.14)$$

defining $\vec{z}_j = \vec{x}_j - \vec{x}_i$. If we correspondingly define the $k \times p$ matrix \mathbf{z} , and set \mathbf{w}_i to be the $k \times 1$ matrix, the vector we get from the sum is just $\mathbf{w}_i^T \mathbf{z}$. The squared magnitude of any vector \vec{r} , considered as a row matrix \mathbf{r} , is $\mathbf{r} \mathbf{r}^T$, so

$$RSS_i = \mathbf{w}_i^T \mathbf{z} \mathbf{z}^T \mathbf{w}_i \quad (20.15)$$

Notice that $\mathbf{z} \mathbf{z}^T$ is a $k \times k$ matrix consisting of all the inner products of the neighbors. This symmetric matrix is called the **Gram matrix** of the set of vectors, and accordingly abbreviated \mathbf{G} — here I'll say \mathbf{G}_i to remind us that it depends on our choice of focal point \vec{x}_i .

$$RSS_i = \mathbf{w}_i^T \mathbf{G}_i \mathbf{w}_i \quad (20.16)$$

Notice that the data matter only in so far as they determine the Gram matrix \mathbf{G}_i ; the problem is invariant under any transformation which leaves all the inner products alone (translation, rotation, mirror-reversal, etc.).

We want to minimize RSS_i , but we have the constraint $\sum_j w_{ij} = 1$. We impose this via a Lagrange multiplier, λ .⁹ To express the constraint in matrix form, introduce

⁹This λ should not be confused with the penalty-term λ used when $k > p$. See next sub-section.

the $k \times 1$ matrix of all 1s, call it $\mathbf{1}$.¹⁰ Then the constraint has the form $\mathbf{1}^T \mathbf{w}_i = 1$, or $\mathbf{1}^T \mathbf{w}_i - 1 = 0$. Now we can write the Lagrangian:

$$\mathcal{L}(\mathbf{w}_i, \lambda) = \mathbf{w}_i^T \mathbf{G}_i \mathbf{w}_i - \lambda(\mathbf{1}^T \mathbf{w}_i - 1) \quad (20.17)$$

Taking derivatives, and remembering that \mathbf{G}_i is symmetric,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} = 2\mathbf{G}_i \mathbf{w}_i - \lambda \mathbf{1} = 0 \quad (20.18)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \mathbf{1}^T \mathbf{w}_i - 1 = 0 \quad (20.19)$$

or

$$\mathbf{G}_i \mathbf{w}_i = \frac{\lambda}{2} \mathbf{1} \quad (20.20)$$

If the Gram matrix is invertible,

$$\mathbf{w}_i = \frac{\lambda}{2} \mathbf{G}_i^{-1} \mathbf{1} \quad (20.21)$$

where λ can be adjusted to ensure that everything sums to 1.

20.4.1.1 $k > p$

If $k > p$, we modify the objective function to be

$$\mathbf{w}_i^T \mathbf{G}_i \mathbf{w}_i + \alpha \mathbf{w}_i^T \mathbf{w}_i \quad (20.22)$$

where $\alpha > 0$ determines the degree of regularization. Proceeding as before to impose the constraint,

$$\mathcal{L} = \mathbf{w}_i^T \mathbf{G}_i \mathbf{w}_i + \alpha \mathbf{w}_i^T \mathbf{w}_i - \lambda(\mathbf{1}^T \mathbf{w}_i - 1) \quad (20.23)$$

where now λ is the Lagrange multiplier. Taking the derivative with respect to \mathbf{w}_i and setting it to zero,

$$2\mathbf{G}_i \mathbf{w}_i + 2\alpha \mathbf{w}_i = \lambda \mathbf{1} \quad (20.24)$$

$$(\mathbf{G}_i + \alpha \mathbf{I}) \mathbf{w}_i = \frac{\lambda}{2} \mathbf{1} \quad (20.25)$$

$$\mathbf{w}_i = \frac{\lambda}{2} (\mathbf{G}_i + \alpha \mathbf{I})^{-1} \mathbf{1} \quad (20.26)$$

where, again, we pick λ to properly normalize the right-hand side.

¹⁰This should not be confused with the identity matrix, \mathbf{I} .

20.4.2 Finding the Coordinates

As with PCA, it's easier to think about the $q = 1$ case first; the general case follows similar lines. So \vec{y}_i is just a single scalar number, y_i , and \mathbf{Y} reduces to an $n \times 1$ column of numbers. We'll revisit $q > 1$ at the end.

The objective function is

$$\Phi(\mathbf{Y}) = \sum_{i=1}^n \left(y_i - \sum_j w_{ij} y_j \right)^2 \quad (20.27)$$

$$= \sum_{i=1}^n y_i^2 - y_i \left(\sum_j w_{ij} y_j \right) - \left(\sum_j w_{ij} y_j \right) y_i + \left(\sum_j w_{ij} y_j \right)^2 \quad (20.28)$$

$$= \mathbf{Y}^T \mathbf{Y} - \mathbf{Y}^T (\mathbf{w} \mathbf{Y}) - (\mathbf{w} \mathbf{Y})^T \mathbf{Y} + (\mathbf{w} \mathbf{Y})^T (\mathbf{w} \mathbf{Y}) \quad (20.29)$$

$$= ((\mathbf{I} - \mathbf{w}) \mathbf{Y})^T ((\mathbf{I} - \mathbf{w}) \mathbf{Y}) \quad (20.30)$$

$$= \mathbf{Y}^T (\mathbf{I} - \mathbf{w})^T (\mathbf{I} - \mathbf{w}) \mathbf{Y} \quad (20.31)$$

Define the $m \times m$ matrix $\mathbf{M} = (\mathbf{I} - \mathbf{w})^T (\mathbf{I} - \mathbf{w})$.

$$\Phi(\mathbf{Y}) = \mathbf{Y}^T \mathbf{M} \mathbf{Y} \quad (20.32)$$

This looks promising — it's the same sort of quadratic form that we maximized in doing PCA.

Now let's use a Lagrange multiplier μ to impose the constraint that $n^{-1} \mathbf{Y}^T \mathbf{Y} = 1$ — but, since $q = 1$, that's the 1×1 identity matrix, i.e., the scalar number 1.

$$\mathcal{L}(\mathbf{Y}, \mu) = \mathbf{Y}^T \mathbf{M} \mathbf{Y} - \mu(n^{-1} \mathbf{Y}^T \mathbf{Y} - 1) \quad (20.33)$$

Note that this μ is not the same as the μ which constrained the weights!

Proceeding as we did with PCA,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Y}} = 2\mathbf{M} \mathbf{Y} - 2\mu n^{-1} \mathbf{Y} = 0 \quad (20.34)$$

or

$$\mathbf{M} \mathbf{Y} = \frac{\mu}{n} \mathbf{Y} \quad (20.35)$$

so \mathbf{Y} must be an eigenvector of \mathbf{M} . Because \mathbf{Y} is defined for each point in the data set, it is a function of the data-points, and we call it an **eigenfunction**, to avoid confusion with things like the eigenvectors of PCA (which are p -dimensional vectors in feature space). Because we are trying to minimize $\mathbf{Y}^T \mathbf{M} \mathbf{Y}$, we want the eigenfunctions going with the smallest eigenvalues — the **bottom** eigenfunctions — unlike the case with PCA, where we wanted the **top** eigenvectors.

\mathbf{M} being an $n \times n$ matrix, it has, in general, n eigenvalues, and n mutually orthogonal eigenfunctions. The eigenvalues are real and non-negative; the smallest of them

is always zero, with eigenfunction $\mathbf{1}$. To see this, notice that $\mathbf{w}\mathbf{1} = \mathbf{1}$.¹¹ Then

$$(\mathbf{I} - \mathbf{w})\mathbf{1} = \mathbf{0} \quad (20.36)$$

$$(\mathbf{I} - \mathbf{w})^T(\mathbf{I} - \mathbf{w})\mathbf{1} = \mathbf{0} \quad (20.37)$$

$$\mathbf{M}\mathbf{1} = \mathbf{0} \quad (20.38)$$

Since this eigenfunction is constant, it doesn't give a useful coordinate on the manifold. To get our first coordinate, then, we need to take the two bottom eigenfunctions, and discard the constant.

Again as with PCA, if we want to use $q > 1$, we just need to take multiple eigenfunctions of M . To get q coordinates, we take the bottom $q + 1$ eigenfunctions, discard the constant eigenfunction with eigenvalue 0, and use the others as our coordinates on the manifold. Because the eigenfunctions are orthogonal, the no-covariance constraint is automatically satisfied. Notice that adding another coordinate just means taking another eigenfunction of the *same* matrix \mathbf{M} — as is the case with PCA, but not with factor analysis.

(What happened to the mean-zero constraint? Well, we can add another Lagrange multiplier v to enforce it, but the constraint is linear in \mathbf{Y} , it's $\mathbf{A}\mathbf{Y} = \mathbf{0}$ for some matrix \mathbf{A} [EXERCISE: write out \mathbf{A}], so when we take partial derivatives we get

$$\frac{\partial \mathcal{L}(\mathbf{Y}, \mu, v)}{\partial \mathbf{Y}} = 2\mathbf{M}\mathbf{Y} - 2\mu\mathbf{Y} - v\mathbf{A} = \mathbf{0} \quad (20.39)$$

and this is the *only* equation in which v appears. So we are actually free to pick any v we like, and may as well set it to be zero. Geometrically, this is the translational invariance yet again. In optimization terms, the size of the Lagrange multiplier tells us about how much better the solution could be if we relaxed the constraint — when it's zero, as here, it means that the constrained optimum is *also* an unconstrained optimum — but we knew that already!)

20.5 Calculation

Let's break this down from the top. The nice thing about doing this is that the over-all function is four lines, one of which is just the return (Example 32).

20.5.1 Finding the Nearest Neighbors

The following approach is straightforward (exploiting an R utility function, `order`), but not recommended for “industrial strength” uses. A *lot* of thought has been given to efficient algorithms for finding nearest neighbors, and this isn't even close to the state of the art [[cites]]. For large n , the difference in efficiency would be quite substantial. For the present, however, this will do.

To find the k nearest neighbors of each point, we first need to calculate the distances between all pairs of points. The neighborhoods only depend on these distances, not the actual points themselves. We just need to find the k smallest entries in each row of the distance matrix (Example 33).

¹¹Each row of $\mathbf{w}\mathbf{1}$ is a weighted average of the other rows of $\mathbf{1}$. But all the rows of $\mathbf{1}$ are the same.

```

# Local linear embedding of data vectors
# Inputs: n*p matrix of vectors, number of dimensions q to find (< p),
#          # number of nearest neighbors per vector, scalar regularization setting
# Calls: find.kNNs, reconstruction.weights, coords.from.weights
# Output: n*q matrix of new coordinates
lle <- function(x,q,k=q+1,alpha=0.01) {
  stopifnot(q>0, q<ncol(x), k>q, alpha>0) # sanity checks
  kNNs = find.kNNs(x,k) # should return an n*k matrix of indices
  w = reconstruction.weights(x,kNNs,alpha) # n*n weight matrix
  coords = coords.from.weights(w,q) # n*q coordinate matrix
  return(coords)
}

```

CODE EXAMPLE 32: *Locally linear embedding in R.* Notice that this top-level function is very simple, and mirrors the math exactly.

```

# Find multiple nearest neighbors in a data frame
# Inputs: n*p matrix of data vectors, number of neighbors to find,
#          # optional arguments to dist function
# Calls: smallest.by.rows
# Output: n*k matrix of the indices of nearest neighbors
find.kNNs <- function(x,k,...) {
  x.distances = dist(x,...) # Uses the built-in distance function
  x.distances = as.matrix(x.distances) # need to make it a matrix
  kNNs = smallest.by.rows(x.distances,k+1) # see text for +1
  return(kNNs[,-1]) # see text for -1
}

```

CODE EXAMPLE 33: *Finding the k nearest neighbors of all the row-vectors in a data frame.*

```
# Find the k smallest entries in each row of an array
# Inputs: n*p array, p >= k, number of smallest entries to find
# Output: n*k array of column indices for smallest entries per row
smallest.by.rows <- function(m,k) {
  stopifnot(ncol(m) >= k) # Otherwise "k smallest" is meaningless
  row.orders = t(apply(m,1,order))
  k.smallest = row.orders[,1:k]
  return(k.smallest)
}
```

CODE EXAMPLE 34: *Finding which columns contain the smallest entries in each row.*

Most of the work is done either by `dist`, a built-in function optimized for calculating distance matrices, or by `smallest.by.rows` (Example 34), which we are about to write. The `+1` and `-1` in the last two lines come from simplifying that. Instead of `dist`, we could have recycled code from the first lecture, but this really is faster, and more flexible.

`smallest.by.rows` uses the utility function `order`. Given a vector, it returns the *permutation* that puts the vector into increasing order, i.e., its return is a vector of integers as long as its input.¹² The first line of `smallest.by.rows` applies `order` to each row of the input matrix `m`. The first column of `row.orders` now gives the column number of the smallest entry in each row of `m`; the second column, the second smallest entry, and so forth. By taking the first k columns, we get the set of the smallest entries in each row. `find.kNNs` applies this function to the distance matrix, giving the indices of the closest points. However, every point is closest to itself, so to get k neighbors, we need the $k + 1$ closest *points*; and we want to discard the first column we get back from `smallest.by.rows`.

Let's check that we're getting sensible results from the parts.

```
> r
 [,1] [,2]
[1,]    7    2
[2,]    3    4
> smallest.by.rows(r,1)
[1] 2 1
> smallest.by.rows(r,2)
 [,1] [,2]
[1,]    2    1
[2,]    1    2
```

Since $7 > 2$ but $3 < 4$, this is correct. Now try a small distance matrix, from the first five points on the spiral:

```
> round(as.matrix(dist(x[1:5,])),2)
```

¹²There is a lot of control over ties, but we don't care about ties. See `help(order)`, though, it's a handy function.

```

      1   2   3   4   5
1 0.00 0.11 0.21 0.32 0.43
2 0.11 0.00 0.11 0.22 0.33
3 0.21 0.11 0.00 0.11 0.22
4 0.32 0.22 0.11 0.00 0.11
5 0.43 0.33 0.22 0.11 0.00
> smallest.by.rows(as.matrix(dist(x[1:5,])),3)
 [,1] [,2] [,3]
1     1     2     3
2     2     1     3
3     3     2     4
4     4     3     5
5     5     4     3

```

Notice that the first column, as asserted above, is saying that every point is closest to itself. But the two nearest neighbors are right.

```

> find.kNNs(x[1:5,],2)
 [,1] [,2]
1     2     3
2     1     3
3     2     4
4     3     5
5     4     3

```

Success!

20.5.2 Calculating the Weights

First, the slow iterative way (Example 35). Aside from sanity-checking the inputs, this just creates a square, $n \times n$ weight-matrix w , initially populated with all zeroes, and then fills each line of it by calling a to-be-written function, `local.weights` (Example 36).

For testing, it would really be better to break `local.weights` up into two sub-parts — one which finds the Gram matrix, and another which solves for the weights — but let's just test it altogether this once.

```

> matrix(mapply("*",local.weights(x[1,],x[2:3,],0.01),x[2:3,]),nrow=2)
      [,1]      [,2]
[1,]  2.014934 -0.4084473
[2,] -0.989357  0.3060440
> colSums(matrix(mapply("*",local.weights(x[1,],x[2:3,],0.01),x[2:3,]),nrow=2))
[1]  1.0255769 -0.1024033
> colSums(matrix(mapply("*",local.weights(x[1,],x[2:3,],0.01),x[2:3,]),nrow=2))
+ - x[1,]
[1]  0.0104723155 -0.0005531495

```

```

# Least-squares weights for linear approx. of data from neighbors
# Inputs: n*p matrix of vectors, n*k matrix of neighbor indices,
# scalar regularization setting
# Calls: local.weights
# Outputs: n*n matrix of weights
reconstruction.weights <- function(x,neighbors,alpha) {
  stopifnot(is.matrix(x),is.matrix(neighbors),alpha>0)
  n=nrow(x)
  stopifnot(nrow(neighbors) == n)
  w = matrix(0,nrow=n,ncol=n)
  for (i in 1:n) {
    i.neighbors = neighbors[i,]
    w[i,i.neighbors] = local.weights(x[i,],x[i.neighbors,],alpha)
  }
  return(w)
}

```

CODE EXAMPLE 35: *Iterative (and so not really recommended) function to find linear least-squares reconstruction weights.*

The `mapply` function is another of the `lapply` family of utility functions. Just as `sapply` sweeps a function along a vector, `mapply` sweeps a multi-argument function (hence the `m`) along multiple argument vectors, recycling as necessary. Here the function is multiplication, so we're getting the products of the reconstruction weights and the vectors. (I re-organize this into a matrix for comprehensibility.) Then I add up the weighted vectors, getting something that looks reasonably close to `x[1,]`. This is confirmed by actually subtract the latter from the approximation, and seeing that the differences are small for both coordinates.

This didn't use the regularization; let's turn it on and see what happens.

```

> colSums(matrix(mapply("*",local.weights(x[1,],x[2:4,],0.01),x[2:4,]),nrow=3))
+ -x[1,]
Error in drop(.Call("La_dgesv", a, as.matrix(b), tol, PACKAGE = "base")) :
  system is computationally singular: reciprocal condition number = 6.73492e-19
[1] 0.01091407 -0.06487090

```

The error message alerts us that the unregularized attempt to solve for the weights failed, since the determinant of the Gram matrix was as close to zero as makes no difference, hence it's uninvertible. (The error message could be suppressed by adding a `silent=TRUE` option to `try`; see `help(try)`.) However, with just a touch of regularization ($\alpha = 0.01$) we get quite reasonable accuracy.

Let's test our iterative solution. Pick $k = 2$, each row of the weight matrix should have two non-zero entries, which should sum to one. (We might expect some small deviation from 1 due to finite-precision arithmetic.) First, of course, the weights should match what the `local.weights` function says.

```
> x.2NNs <- find.kNNs(x,2)
```

```

# Calculate local reconstruction weights from vectors
# Inputs: focal vector (1*p matrix), k*p matrix of neighbors,
# scalar regularization setting
# Outputs: length k vector of weights, summing to 1
local.weights <- function(focal,neighbors,alpha) {
  # basic matrix-shape sanity checks
  stopifnot(nrow(focal)==1,ncol(focal)==ncol(neighbors))
  # Should really sanity-check the rest (is.numeric, etc.)
  k = nrow(neighbors)
  # Center on the focal vector
  neighbors=t(t(neighbors)-focal) # exploits recycling rule, which
    # has a weird preference for columns
  gram = neighbors %*% t(neighbors)
  # Try to solve the problem without regularization
  weights = try(solve(gram,rep(1,k)))
    # The try function tries to evaluate its argument and returns
    # the value if successful; otherwise it returns an error
    # message of class "try-error"
  if (identical(class(weights),"try-error")) {
    # Un-regularized solution failed, try to regularize
    # TODO: look at the error, check if it's something
    # regularization could fix!
    weights = solve(gram+alpha*diag(k),rep(1,k))
  }
  # Enforce the unit-sum constraint
  weights = weights/sum(weights)
  return(weights)
}

```

CODE EXAMPLE 36: *Find the weights for approximating a vector as a linear combination of the rows of a matrix.*

```

> x.2NNs[1,]
[1] 2 3
> local.weights(x[1,],x[x.2NNs[1,],],0.01)
[1] 1.9753018 -0.9753018
> wts<-reconstruction.weights(x,x.2NNs,0.01)
> wts[1,1:6]
[1] 0.0000000 1.9753018 -0.9753018 0.0000000 0.0000000 0.0000000
> sum(wts[1,] != 0)
[1] 2
> all(rowSums(wts != 0)==2)
[1] TRUE
> all(rowSums(wts) == 1)
[1] FALSE
> summary(rowSums(wts))
   Min. 1st Qu. Median      Mean 3rd Qu.      Max.
      1        1        1        1        1        1

```

Why does `summary` say that all the rows sum to 1, when directly testing that says otherwise? Because some rows don't *quite* sum to 1, just closer-than-display tolerance to 1.

```

> sum(wts[1,]) == 1
[1] FALSE
> sum(wts[1,])
[1] 1
> sum(wts[1,]) - 1
[1] -1.110223e-16
> summary(rowSums(wts)-1)
   Min. 1st Qu. Median      Mean 3rd Qu.      Max.
-2.220e-16 0.000e+00 0.000e+00 -1.406e-17 0.000e+00 2.220e-16

```

So the constraint is satisfied to $\pm 2 \cdot 10^{-16}$, which is good enough for all practical purposes. It does, however, mean that we have to be careful about testing the constraint!

```

> all(abs(rowSums(wts)-1) < 1e-7)
[1] TRUE

```

Of course, iteration is usually Not the Way We Do It in R — especially here, where there's no dependence between the rows of the weight matrix.¹³ What makes this a bit tricky is that we need to combine information from two matrices — the data frame and the matrix giving the neighborhood of each point. We could try using something like `mapply` or `Map`, but it's cleaner to just write a function to do the calculation for each row (Example 37), and then apply it to the rows.

As always, check the new function:

¹³Remember what makes loops slow in R is that every time we change an object, we actually create a new copy with the modified values and then destroy the old one. If n is large, then the weight matrix, with n^2 entries, is very large, and we are wasting a lot of time creating and destroying big matrices to make small changes.

```

# Get approximation weights from indices of point and neighbors
# Inputs: index of focal point, n*p matrix of vectors, n*k matrix
# of nearest neighbor indices, scalar regularization setting
# Calls: local.weights
# Output: vector of n reconstruction weights
local.weights.for.index <- function(focal,x,NNs,alpha) {
  n = nrow(x)
  stopifnot(n> 0, 0 < focal, focal <= n, nrow(NNs)==n)
  w = rep(0,n)
  neighbors = NNs[focal,]
  wts = local.weights(x[focal,],x[neighbors,],alpha)
  w[neighbors] = wts
  return(w)
}

```

CODE EXAMPLE 37: *Finding the weights for the linear approximation of a point given its index, the data-frame, and the matrix of neighbors.*

```

> w.1 = local.weights.from.indices(1,x,x.2NNs,0.01)
> w.1[w.1 != 0]
[1] 1.9753018 -0.9753018
> which(w.1 != 0)
[1] 2 3

```

So (at least for the first row!) it has the right values in the right positions.

Now the final function is simple (Example 38), and passes the check:

```

> wts.2 = reconstruction.weights.2(x,x.2NNs,0.01)
> identical(wts.2,wts)
[1] TRUE

```

20.5.3 Calculating the Coordinates

Having gone through all the eigen-manipulation, this is a straightforward calculation (Example 39).

Notice that **w** will in general be a very **sparse** matrix — it has only k non-zero entries per row, and typically $k \ll n$. There are special techniques for rapidly solving eigenvalue problems for sparse matrices, which are not being used here — another way in which this is not an industrial-strength version.

Let's try this out: make the coordinate (with $q = 1$), plot it (Figure 20.5), and check that it really is monotonically increasing, as the figure suggests.

```

> spiral.lle = coords.from.weights(wts,1)
> plot(spiral.lle,ylab="Coordinate on manifold")
> all(diff(spiral.lle) > 0)
[1] TRUE

```

```

# Local linear approximation weights, without iteration
# Inputs: n*p matrix of vectors, n*k matrix of neighbor indices,
# scalar regularization setting
# Calls: local.weights.for.index
# Outputs: n*n matrix of reconstruction weights
reconstruction.weights.2 <- function(x,neighbors,alpha) {
  # Sanity-checking should go here
  n = nrow(x)
  w = sapply(1:n,local.weights.for.index,x=x,NNs=neighbors,
             alpha=alpha)
  w = t(w) # sapply returns the transpose of the matrix we want
  return(w)
}

```

CODE EXAMPLE 38: *Non-iterative calculation of the weight matrix.*

```

# Find intrinsic coordinates from local linear approximation weights
# Inputs: n*n matrix of weights, number of dimensions q, numerical
# tolerance for checking the row-sum constraint on the weights
# Output: n*q matrix of new coordinates on the manifold
coords.from.weights <- function(w,q,tol=1e-7) {
  n=nrow(w)
  stopifnot(ncol(w)==n) # Needs to be square
  # Check that the weights are normalized
  # to within tol > 0 to handle round-off error
  stopifnot(all(abs(rowSums(w)-1) < tol))
  # Make the Laplacian
  M = t(diag(n)-w)%*%(diag(n)-w)
  # diag(n) is n*n identity matrix
  soln = eigen(M) # eigenvalues and eigenvectors (here,
  # eigenfunctions), in order of decreasing eigenvalue
  coords = soln$vectors[,((n-q):(n-1))] # bottom eigenfunctions
  # except for the trivial one
  return(coords)
}

```

CODE EXAMPLE 39: *Getting manifold coordinates from approximation weights by finding eigenfunctions.*

So the coordinate we got through LLE increases along the spiral, just as it should, and we have successfully recovered the underlying structure of the data. To verify this in a more visually pleasing way, Figure 20.6 plots the original data again, but now with points colored so that their color in the rainbow corresponds to their inferred coordinate on the manifold.

Before celebrating our final victory, test that everything works when we put it together:

```
> all(lle(x,1,2)==spiral.lle)
```

```
[1] TRUE
```

□

20.6 Diffusion Maps

Let's re-cap what we did with locally linear embedding. We start p -dimensional data ($\vec{x}_1, \dots, \vec{x}_n$ in vector form, stacked into a data-frame \mathbf{X}), which we suspect are or are near a q -dimensional manifold embedded in the feature space. As an exercise in least squares, we got weights w_{ij} saying how much of data-point \vec{x}_j we needed to use to reconstruct \vec{x}_i . Since $\sum_j w_{ij} = 1$, we can think of w_{ij} as saying how similar i is to j . Next, we asked for the new vectors \vec{y}_i which were best reconstructed by those weights, i.e., which minimized

$$\sum_{i=1}^n \left\| \vec{y}_i - \sum_{j=1}^n w_{ij} \vec{y}_j \right\|^2 \quad (20.40)$$

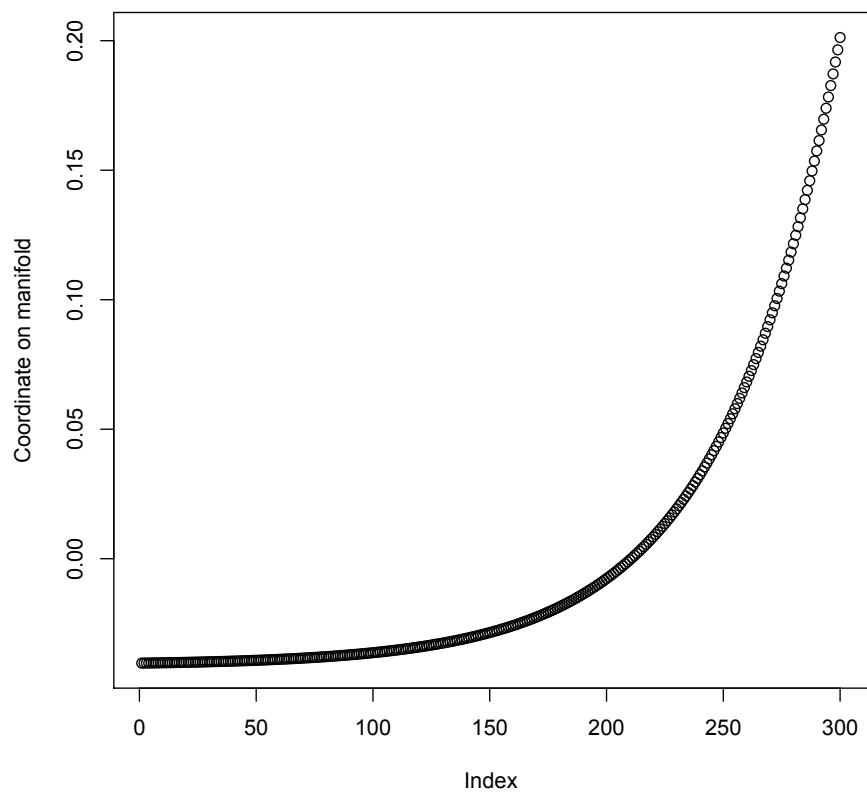
under the constraint that $n^{-1}\mathbf{Y}^T \mathbf{Y} = \mathbf{I}$. We turned this into an eigenvalue problem, that of finding the eigenvectors of

$$\mathbf{M} = (\mathbf{I} - \mathbf{w})^T (\mathbf{I} - \mathbf{w}) \quad (20.41)$$

It was nice to do this, because finding eigenvectors is something our computers are very good at. These eigenvectors were then the new coordinates of our data points on the low-dimensional manifold.

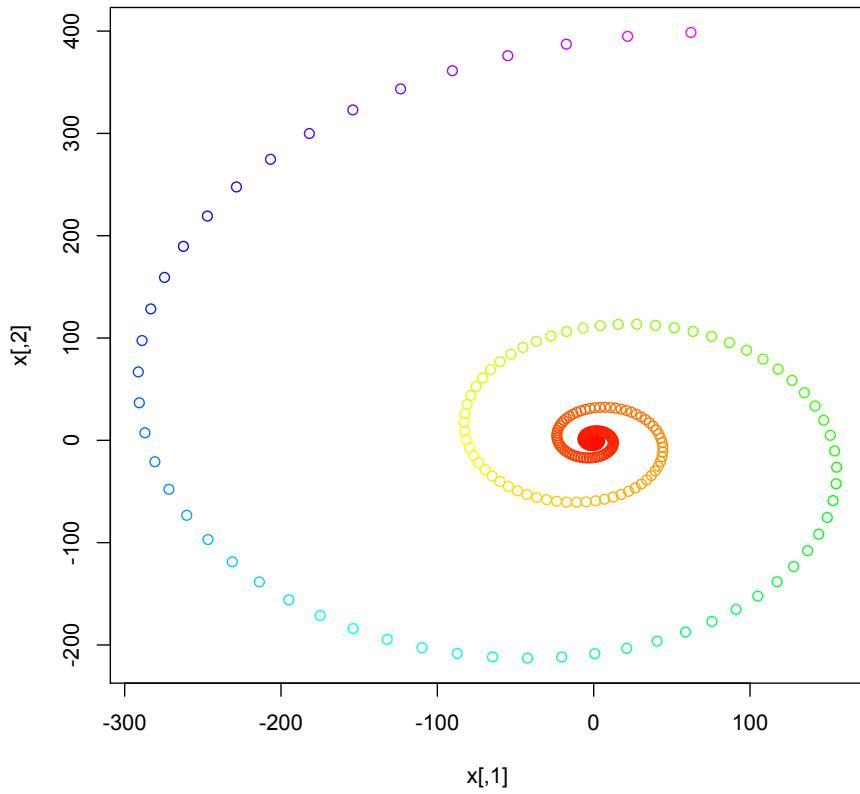
To introduce diffusion maps, we need to change tack a little from the geometric/optimization approach we've been taking, though we'll come back to that. Notice that when we do LLE, we identify a neighborhood of k points for each point in our data set. We can imagine drawing a graph or network which connects a point to its neighbors. We can also attach weights to the edges in this graph, since we get them from our \mathbf{w} matrix. Now, we've seen networks with weighted edges before in this class, when we looked at page-rank: there the edge-weights reflected how many links there were between pages, and they were normalized to sum to one for each page. It is not at all a coincidence that each row in \mathbf{w} also sums to one. If the weights are all non-negative, we can use them to define a **random walk on the data**, just as in page-rank we had a **random walk on the web**.¹⁴

¹⁴We didn't *require* the weights to be non-negative, but they often will be spontaneously, and it's not, generally, a hard requirement to impose. See Saul and Roweis (2003), if you want the details.



```
plot(coords.from.weights(wts,1),ylab="Coordinate on manifold")
```

FIGURE 20.5: *Coordinate on the manifold estimated by locally-linear embedding for the spiral data. Notice that it increases monotonically along the spiral, as it should.*



```
plot(x,col=rainbow(300,end=5/6)[cut(spiral.lle,300,labels=FALSE)])
```

FIGURE 20.6: *The original spiral data, but with color advancing smoothly along the spectrum according to the intrinsic coordinate found by LLE.*

Let's be a little more formal, and more general. Suppose that we have a function K which takes two data-points \vec{x}_1, \vec{x}_2 as inputs, and gives us a non-negative real number as an output:

$$K(\vec{x}_1, \vec{x}_2) \geq 0 \quad (20.42)$$

Further suppose that it's symmetric

$$K(\vec{x}_2, \vec{x}_1) = K(\vec{x}_1, \vec{x}_2) \quad (20.43)$$

We'll form a matrix \mathbf{K} for our data, where

$$K_{ij} = K(\vec{x}_i, \vec{x}_j) \quad (20.44)$$

and as a last requirement, insist that the *matrix* be non-negative, that for any vector \vec{v}

$$\vec{v}^T \mathbf{K} \vec{v} \geq 0 \quad (20.45)$$

Tradition says that we should call this the **kernel matrix** or **Gram matrix** corresponding to the **kernel**¹⁵ K . The point of this matrix is, once again, to give us a weighted graph, which tells us which points are tied to which other points, and how similar those neighbors are — it's going to play much the same role in the generalized case that \mathbf{w} did for LLE.¹⁶ The advantage of abstracting away from the least-squares procedure of LLE to “some kernel function or other” is that it lets us deal with cases where least-squares does not apply — because it's not the right metric, because some of the data are categorical, etc.; I will return to the “kernel trick” below, and we'll come back to it again later in the course.

Given this \mathbf{K} , we need one more step to make a random walk on the data graph, which is to turn it into a stochastic transition matrix, where each row adds up to one:

$$\mathbf{D} = \text{diag}(\text{rowSums}(\mathbf{K})) \quad (20.46)$$

that is, \mathbf{D} is the diagonal matrix whose i^{th} entry is the sum of the i^{th} row of \mathbf{K} .¹⁷ Now

$$\mathbf{D}^{-1} \mathbf{K} \quad (20.47)$$

defines the same graph as \mathbf{K} , but it's properly row-normalized. Let's call this matrix \mathbf{A} . It's the transition matrix for a Markov chain — specifically or a random walk on the graph of the data. The random walk is biased towards making transitions that take it between *similar* points. If we look at the behavior of the random walk, then, it should tell us about the *geometry* of the data (how can we move around the data without ever making a big, discontinuous step?), and about *groups* of similar points (i.e., what are the clusters?). We'll take these two in order.

¹⁵A name whose origins are lost in the mists of 19th century algebra.

¹⁶If you are getting suspicious because \mathbf{w} is not symmetric, even if we force its entries to be non-negative, good; but notice that what ultimately shows up in finding the coordinates is $\mathbf{w}^T + \mathbf{w} + \mathbf{w}^T \mathbf{w}$, which *is* symmetric.

¹⁷If you don't like my mix of R and matrix notation, you try writing it out in proper matrix form.

20.6.1 Diffusion-Map Coordinates

Let's say that we want to assign a single coordinate to every point on the data, which we'll write as f , with $f(i)$ being the coordinate for i . Because there are n data points, instead of treating f as a function, we can also treat it as an $n \times 1$ matrix, which for simplicity I'll also write f . (Here, as with PCA and with LLE, starting with a single coordinate, the $q = 1$ case, is just to simplify the initial algebra.) Let's say that we want to minimize

$$\Phi(f) \equiv \sum_{i,j} A_{ij}(f_i - f_j)^2 \quad (20.48)$$

This should force the coordinates of points which are very similar ($A_{ij} \approx 1$) to be close to each other, but let the coordinates of deeply dissimilar points ($A_{ij} \approx 0$) vary freely.¹⁸ Now that we know what to do with quadratic forms, let's try and turn this into one.

$$\sum_{i,j} A_{ij}(f_i - f_j)^2 = \sum_{ij} A_{ij}(f_i^2 - 2f_i f_j + f_j^2) \quad (20.49)$$

$$= \sum_i f_i^2 \sum_j A_{ij} - 2 \sum_i f_i \sum_j A_{ij} f_j + \sum_j A_{ij} f_j^2 \quad (20.50)$$

$$= \sum_i f_i^2 - 2 \sum_i f_i \sum_j A_{ij} f_j + \sum_j f_j^2 \quad (20.51)$$

$$= 2f^T f - 2f^T \mathbf{A}f \quad (20.52)$$

using the fact that $A_{ij} = A_{ji}$. So minimizing Φ is the same as minimizing

$$f^T (\mathbf{I} - \mathbf{A}) f \equiv f^T \mathbf{L} f \quad (20.53)$$

where of course $\mathbf{L} \equiv \mathbf{I} - \mathbf{A}$. The matrix \mathbf{L} is called the **Laplacian** of the graph. Now we impose the constraint that $f^T f = 1$, to avoid the uninteresting minimum at $f = 0$, and as before we get an eigenvalue problem:

$$\mathbf{L}f = \lambda f \quad (20.54)$$

is the solution. Substituting back in to the expression for Φ , we see that the value of Φ is 2λ , so we want the bottom eigenvectors, as we did with LLE.

For exactly that same reasons that $\mathbf{w} \mathbf{1} = 1$ for LLE, we know that $\mathbf{A} \mathbf{1} = \mathbf{1}$ — i.e., the vector of all 1s is an eigenvector of \mathbf{A} with eigenvalue 1. It turns out (see below) that this is the *largest* eigenvalue. Now, every eigenvector of \mathbf{A} is also an eigenvector of \mathbf{L} , but the eigenvalues change:

$$\mathbf{L}f = \lambda f \quad (20.55)$$

$$(\mathbf{I} - \mathbf{A})f = \lambda f \quad (20.56)$$

$$f - \mathbf{A}f = \lambda f \quad (20.57)$$

$$\mathbf{A}f = (1 - \lambda)f = \mu f \quad (20.58)$$

¹⁸This is not *quite* the same as the minimization problem for LLE.

where of course $\mu = 1 - \lambda$. Thus, $\mathbf{1}$ is an eigenvector of the Laplacian with eigenvalue 0. Again as with LLE, we discard this solution as useless.

The next-to-bottom eigenvector of \mathbf{L} has to be orthogonal to $\mathbf{1}$, because all the eigenvectors are orthogonal. This means it must have both negative and positive entries (otherwise $\mathbf{1}^T f = \sum_i f_i > 0$). We will see later how to use the signs of the entries in this eigenvector. What matters for right now is that it gives us a non-trivial coordinate, in which similar data points are close to each other.

If we want q coordinates, then (yet again as with LLE) we just take the bottom $q+1$ eigenvectors¹⁹ $f^{(n)}, f^{(n-1)}, \dots, f^{(n-q)}$ of \mathbf{L} , and their eigenvalues $0 = \lambda_n, \lambda_{n-1}, \dots, \lambda_{n-q}$. We discard the bottom one as uninteresting. The **diffusion map** Ψ uses the remaining eigenvectors and eigenvalues to find a point in \mathbb{R}^q corresponding to each of the original data points. Specifically, the image of the point v is

$$\Psi(v) = (\mu_1 f_v^{(n-1)}, \mu_2 f_v^{(n-2)}, \dots, \mu_q f_v^{(n-q)}) \quad (20.59)$$

where I've abbreviated $1 - \lambda_{n-i}$ as μ_i .

The coordinates we get from the diffusion map are related to, but not identical with, the coordinates we would get from LLE.²⁰ They former are, however, in a reasonable sense the optimal coordinates to represent the original matrix of similarities \mathbf{K} (Lee and Wasserman, 2010).

By now you should be asking what any of this has to do with “diffusion”.

20.6.1.1 Fun with Transition Matrices

The matrix \mathbf{A} is a stochastic transition matrix, so the entries in each row are non-negative and add up to one. Suppose we have a function f on the graph; since there are only n data points, we can represent it as an $n \times 1$ matrix, which for simplicity I'll also write as f . What's \mathbf{Af} ? Well, it's another $n \times 1$ matrix:

$$(\mathbf{Af})_i = \sum_{j=1}^n A_{ij} f_j \quad (20.60)$$

In words, the i^{th} entry of \mathbf{Af} is a weighted average of the values of f at i 's neighbors; the weight of j is the likelihood of going from i to j . Suppose I write Z_t for the node occupied by the random walk at time t . Then

$$\mathbf{E}[f(Z_{t+1})|Z_t = i] = (\mathbf{Af})_i \quad (20.61)$$

If f encodes a function, then \mathbf{Af} encodes the expected value of that function after one step of the random walk, $\mathbf{A}^2 f$ is the expected function after two steps, and so forth. So, acting to the right, \mathbf{A} forecasts what the value of a fixed function is going to be later.

¹⁹Eigenvectors are conventionally numbered starting from 1 at the top.

²⁰In fact, in some cases, it can be shown (Belkin and Niyogi, 2003, §5) that the matrix in the LLE minimization problem is related to the Laplacian, because $(\mathbf{I} - \mathbf{w})^T (\mathbf{I} - \mathbf{w}) \approx \frac{1}{2} \mathbf{L}^2$. Since the powers of \mathbf{L} have the same eigenvectors as \mathbf{L} , when this holds the coordinates we get from the diffusion map are *approximately* the same as the LLE coordinates.

This is inside-out from (or, as the mathematicians say, “adjoint to”) asking *where* the random walk will be later. As you know, if ρ is a distribution over the states of a Markov chain, written as a $1 \times n$ matrix, then

$$\rho \mathbf{A} \quad (20.62)$$

is the distribution of the Markov chain at the next time step. So from the left \mathbf{A} updates distributions, and from the right \mathbf{A} forecasts functions. Without getting into the more profound aspects of this duality²¹, what we care about here is how this can help us understand the data.

We know (if only because I asserted it during the lecture on page-rank) that \mathbf{A} has a left eigenvector with eigenvalue 1:

$$\rho^{(1)} \mathbf{A} = \rho^{(1)} \quad (20.63)$$

Moreover, every entry of $\rho^{(1)}$ is > 0 , and it is the top (or “dominant”) eigenvector, the one with the largest eigenvalue. This means that all the other eigenvectors must have both positive and negative entries.

What happens if we let the walk evolve for many steps undisturbed? Ergodicity happens, that’s what. Let’s say that the left eigenvectors of \mathbf{A} are $\rho^{(1)}, \rho^{(2)}, \dots, \rho^{(n)}$, with corresponding eigenvalues $1 = \mu_1 > \mu_2 > \dots > \mu_n > 0$. Since the eigenvectors are orthogonal, and there are n of them, they span the space — any arbitrary vector can be written as a linear combination of eigenvectors. So start with any initial distribution ρ we like; it’s the case that

$$\rho = \sum_{i=1}^n a_i \rho^{(i)} \quad (20.64)$$

How does ρ evolve after t steps of the random walk?

$$\rho \mathbf{A}^t = \left(\sum_{i=1}^n a_i \rho^{(i)} \right) \mathbf{A}^t \quad (20.65)$$

$$= \sum_{i=1}^n a_i \rho^{(i)} \mathbf{A}^t \quad (20.66)$$

$$= \sum_{i=1}^n a_i \rho^{(i)} \mu_i \mathbf{A}^{t-1} \quad (20.67)$$

$$= \sum_{i=1}^n a_i \rho^{(i)} \mu_i^t \quad (20.68)$$

Since all of the $\mu_i < 1$, except for $\mu_1 = 1$, as we step through the random walk, the distribution comes closer and closer to $\rho^{(i)}$, the invariant distribution. This is ergodicity.

I bring this up not only because it’s a cool bit of math, but also because it’s relevant to diffusion maps. The last few paragraphs have been all about *left* eigenvectors,

²¹It’s the difference between the Schrödinger and Heisenberg pictures of quantum mechanics.

but of course there is a close relationship between them and the *right* eigenvectors, which are the coordinates of the diffusion map. Recall what happens when you transpose a matrix product:

$$(\rho \mathbf{A})^T = \mathbf{A}^T \rho^T \quad (20.69)$$

If ρ should happen to be a left eigenvector of \mathbf{A} , then ρ^T is a right eigenvector of \mathbf{A}^T , with the same eigenvalue. Since we have arranged for *our* matrix \mathbf{A} to be symmetric, $\mathbf{A} = \mathbf{A}^T$, the left and right eigenvectors are the same. So we know the right eigenvalues of \mathbf{A} — they're the same as the left eigenvalues, starting at 1 and counting down from there towards a lowest value $\mu_n > 0$. I could write the right eigenvectors as $(\rho^{(i)})^T$, but that's ugly and awkward; let's call them $f^{(i)}$ instead.

We've already seen that $\mathbf{A}\mathbf{1} = \mathbf{1}$, so we've identified the top eigenvector: $f^{(i)} = \mathbf{1}$. In other words, if a function is constant, then its expected value after one step is also constant. Averaging a constant just gives you back the constant. What happens to other functions, especially if we are trying to predict multiple steps ahead (i.e., averaging repeatedly)? We'll pull the expansion-in-eigenvectors trick again.

$$\mathbf{A}^t f = \mathbf{A}^t \left(\sum_{i=1}^n a_i f^{(i)} \right) \quad (20.70)$$

$$= \sum_{i=1}^n a_i \mathbf{A}^t f^{(i)} \quad (20.71)$$

$$= \sum_{i=1}^n a_i \mu_i \mathbf{A}^{t-1} f^{(i)} \quad (20.72)$$

$$= \sum_{i=1}^n a_i \mu_i^t f^{(i)} \quad (20.73)$$

$$\rightarrow a_1 \mathbf{1} \quad (20.74)$$

since all the eigenvalues are ≤ 1 . In other words, projected far enough into the future, every function looks constant. Any irregularities in the initial function f get averaged away.

This is where diffusion enters into it. In physics, diffusion refers to the process of *passive* drift of substances²² away from regions of high concentration and towards regions of low concentration. This is accomplished by particles of the substance taking unbiased random walks. If two adjacent regions of space start with different concentrations of the substance — say it's higher in region 1 than in region 2 — then, even though the individual random walks are unbiased, there will be more random walkers going from region 1 to region 2 than going in the opposite direction; which will tend to reduce the difference in concentrations.

More specifically, if $\rho(\vec{r}, t)$ is the *density* of the substance at the point \vec{r} at the time t , the **diffusion equation** is

$$\frac{\partial \rho}{\partial t} = D \left(\frac{\partial^2 \rho}{\partial x^2} + \frac{\partial^2 \rho}{\partial y^2} + \frac{\partial^2 \rho}{\partial z^2} \right) = D \nabla^2 \rho \quad (20.75)$$

²²Or heat; which for many purposes acts roughly like a subtle fluid.

where the last equation (implicitly) defines the **Laplacian** ∇^2 , which is sometimes also written as Δ , and D is the diffusion constant.²³ Remember that second derivatives are positive at minima and negative at maxima; this is saying that the substance, whose density is ρ , will move away from regions of high concentration and towards ones of low concentration, until it's evenly distributed everywhere.²⁴ On the one hand, it's almost Biblical²⁵; on the other hand, it's the mathematical expression of what we mean when in ordinary language we say "it oozes" (Haldane, 1985, p. 33).

Without going into all the ramifications of the diffusion equation (a subject that fills volumes), the important thing to notice about it is that $\nabla^2 f$ says how much f changes per unit time. It takes the very simple form it does because Euclidean space is **isotropic** — every direction is the same as every other direction — and flat, uncurved. Now consider a diffusion process which is confined to some manifold — say some substance diffusing over the surface of a sphere. The flow can't be given by ∇^2 , since that describes inequalities of concentration in the embedding space, and we don't care about (for instance) the fact that the concentration is > 0 on the manifold and $= 0$ off it — we can't diffuse off the manifold. Rather, each manifold \mathcal{M} has its *own* Laplacian operator, $\nabla_{\mathcal{M}}^2$, which takes into account its curvature and anisotropy.²⁶ In fact, the connection also goes the other way: knowing a manifold's Laplacian basically tells you its shape, because it tells you how the manifold curves.

This is where we connect back, at last, to the random walk on the graph. $\nabla^2 f$ says how rapidly f changes through diffusion in ordinary Euclidean space. $\nabla_{\mathcal{M}}^2 f$ says how rapidly f changes through diffusion on a manifold \mathcal{M} . $\mathcal{L} f$ says how much f changes through one step of the random walk on the graph. Suppose we build the graph by uniformly sampling points from \mathcal{M} ; as we sample more and more densely, our graph looks more and more like a discrete approximation to the continuous manifold, and so \mathbf{L} has to encode the same geometry as $\nabla_{\mathcal{M}}^2$.²⁷ So \mathbf{L} is estimating the natural or intrinsic "shape" of the data. This remains true even if the data don't come from uniform sampling on a manifold, but some other distribution.

20.6.1.2 Multiple Scales

If \mathbf{A} is the transition matrix for a Markov chain, then so is \mathbf{A}^m , for any integer m — it's just taking m steps at a time, rather than one. It's easy to show (see exercises)

²³See the appendix.

²⁴Actually, precisely linear gradients are *also* left alone by the diffusion equation (why?). This reflects a difference between the finite-dimensional operator \mathbf{L} for graphs, and the infinite-dimensional operator ∇^2 for Euclidean space. (If this comment makes no sense, don't worry; it's mostly intended to reassure anyone who may have taken operator theory and started reading these notes by mistake.)

²⁵Isaiah 40:4, "Every valley shall be exalted, and every mountain and hill shall be laid low".

²⁶You might want to try to work out what the Laplacian is for a sphere, say in spherical coordinates.

²⁷The exact sense in which this is true is fairly subtle. On the one hand, \mathbf{L} is an $n \times n$ matrix, so multiplying by it transforms vectors in \mathbb{R}^n into other vectors in \mathbb{R}^n . On the other hand, $\nabla_{\mathcal{M}}^2$ takes functions to functions — points in $\mathbb{R}^{\mathcal{M}}$ to $\mathbb{R}^{\mathcal{M}}$. Similarly, \mathbf{L} represents an *amount* of change in *one time step*, while $\nabla_{\mathcal{M}}^2$ represents a *rate* of change *per unit time*. Roughly speaking, we need to integrate $\nabla_{\mathcal{M}}^2$ over the duration of a time-step, call it b . This gives us an operator \mathcal{L}_b defined through $\mathcal{L}_b f = e^{b\nabla_{\mathcal{M}}^2} f$. If we evaluate $\mathcal{L}_b f$ only at points on the graph, we should have $\mathcal{L}_b f \approx \mathbf{L} f$. See Grimmett and Stirzaker (1992) for an introduction on how discrete-time Markov chains relate to continuous-time Markov processes, and Lee and Wasserman (2010) for a precise statement of what's going on in the present case.

that \mathbf{A}^m has the same eigenvectors as \mathbf{A} , but different eigenvalues — specifically, the eigenvalues of \mathbf{A} , all also raised to the power m . We can therefore look at diffusion maps defined not by one step of the random walk but by m steps. The advantage of doing so is that it tends to reduce the influence of small-scale local noise. Computationally, this is because it shrinks the small eigenvalues rapidly, meaning those coordinates become less influential. Stochastically, it's because, by ergodicity, where the random walk is after m steps depends less on its starting position than where it is after 1 step. Of course if m is very large, all the points blur into each other (again by ergodicity), so m is best seen as a control setting.

20.6.1.3 Choosing q

It can be shown (through a *very* complicated argument; [[Lee and Wasserman]]) that how accurately the diffusion map reconstructs the underlying geometry depends on the ratio

$$\frac{\sum_{i=1}^q \mu_i}{\sum_{j=1}^n \mu_j} \quad (20.76)$$

where the μ are the eigenvalues of \mathbf{A} .²⁸ This suggests that a practical rule for choosing q is to fix a value for this ratio, and use the smallest q which achieves it. This is the default implemented in the `diffuse` function in the CRAN package `diffusionMap`.

20.6.2 What to Do with the Diffusion Map Once You Have It

First, everything you might do with ordinary vector-valued data can be done with the diffusion coordinates. You can do similarity search, you can look for clusters (for instance, k -means; there's a function to do this nicely in `diffusionMap`), classification, etc. We'll be looking at regression after the midterm, and you can use the diffusion-map coordinates as the input variables in a regression.

20.6.2.1 Spectral Clustering

One cute application of diffusion maps is to clustering. Remember that only the top eigenvector of \mathbf{A} is all positive; all the other eigenvectors have both positive and negative entries. What does the difference in signs mean?

To be concrete, let's think about $\rho^{(2)}$, the next-to-top eigenvector. Every point gets either a positive or a negative sign in the eigenvector. Suppose we want to start with a distribution which is concentrated solely on the points with positive sign. We can decompose any such distribution into the eigenvectors:

$$\rho = \rho^{(1)} + \alpha \rho^{(2)} + \sum_{j=2}^n b_j \rho^{(j)} \quad (20.77)$$

²⁸More exactly, the accuracy depends on the ratio of the sums of the population quantities of which these eigenvalues are estimates, and n in the denominator has to go to infinity. I told you it was complicated.

but now α will be large and positive, enhancing the probability of the positive points and lowering that of the negative ones, and the b_j will be small. (The b_j might even manage to be zero.) After one time step, this will become

$$\rho \mathbf{A} = \rho^{(1)} + \alpha \mu_2 \rho^{(2)} + \sum_{j=2}^n b_j \mu_j \rho^{(j)} \quad (20.78)$$

... which looks rather like ρ : while it's closer to being uniform than ρ was, since the eigenvalues are all < 1 , it's still concentrated on the positive points of $\rho^{(2)}$, since the eigenvalues are decreasing and the b_j are small. In other words, if we start with a distribution concentrated on the positive points of $\rho^{(2)}$, it will tend to stay there, though diffusion will disperse it eventually. Things would work exactly the same if we concentrated the initial probability on the negative points — the only difference would be that $\alpha < 0$.

What about the other eigenvectors? Well, we could make a similar argument, but, because eigenvalues are smaller, initial concentrations on the positive points will diffuse away more quickly — how much more quickly will depend on the ratios of the eigenvalues. If we had to split the graph into two parts with the minimum diffusion between them, we'd split it into the points with positive and negative coordinates in $\rho^{(2)}$.

How does this relate to *clustering*? Well, suppose the data fall into two or more clusters, within which all the points are much more similar to each other than they are to outsiders. We would like to call these clusters. We could then re-arrange the kernel matrix \mathbf{K} so it's block-diagonal, or nearly so. Then \mathbf{A} will also be nearly block-diagonal. Thus a random walk which starts inside one of the blocks will tend to stay inside the block for a long time. This means that all the points in a block should have the same sign in at least one of the eigenvectors. The signs of the eigenvectors, in other words, act like indicator functions, or linear combinations of indicator functions, for the clusters.

In the most basic form of spectral clustering, we first divide the data into two clusters by the signs of entries in $\rho^{(2)}$. Having done that, we can further sub-divide the clusters by the signs of $\rho^{(3)}$ and so forth. This is a top-down (**divisive**) hierarchical clustering. At some point it's no longer worth it to keep splitting, and we should instead use the remaining eigenvectors as coordinates within each cluster.

[[TODO: Add pictures of data, block-diagonal-ish transition matrix, eigenvectors.]]

20.6.2.2 Asymmetry

We required the matrix \mathbf{K} we started from to be symmetric; consequently the transition matrix \mathbf{A} was too. In page-rank, which is otherwise similar, the transition matrix is *not* symmetric. How much of what we've done depends on symmetry, and how much would apply to any Markov chain over the data?

Most of it carries through, actually. It's no longer the case that the left and right eigenvectors are the same, but the left and right eigenvalues are. And it's still the case that the top right eigenvector is 1, that all of the others have both positive and negative signs, etc. (Also, the non-dominant left eigenvectors all have both positive

and negative signs.) It's also the case that the signs of the entries in the non-dominant eigenvectors correspond to clusters.

20.6.3 The Kernel Trick

[[TODO: reframe as being about ways of making linear procedures work with nonlinear relations]]

Everything that went before rested on having a kernel matrix K , derived from a kernel function. *Where* that function came from, or what kind of data it took as arguments, was irrelevant. The data could be Euclidean vectors and the kernel the ordinary inner product, but nothing required that. So long as the kernel function was mathematically OK, nothing else mattered. This is a very powerful idea in data mining: we can split our problem into (a) finding an algorithm (for prediction or clustering or whatever) that works in terms of kernels, and (b) finding a kernel function for our representation. Then we can recycle algorithms across problems. We will return to kernel methods repeatedly, but I want to close with a small illustration of how they can be powerful, something often called “the kernel trick”.²⁹

Suppose that we have data points x_1, x_2, \dots, x_n (which may be vectors or something else). Each point is represented by certain features, but we don't think those are really the right ones for our problem. Instead, we have q different functions $\psi_1, \psi_2, \dots, \psi_q$, and we really think those are the appropriate features. So we'd like to work with the vectors

$$\Psi(x) = (\psi_1(x), \psi_2(x), \dots, \psi_q(x)) \quad (20.79)$$

Let us say that $K(x_i, x_j)$ is the inner product of $\Psi(x_i)$ and $\Psi(x_j)$:

$$K(x_i, x_j) = \sum_{k=1}^q \psi_k(x_i) \psi_k(x_j) \quad (20.80)$$

Since this is an inner product, it's got all the properties we need for a kernel function — symmetry, forms a positive-definite matrix, etc. So if I can express my problem in terms of inner products of the transformed features Ψ , I can also write it in terms of the kernel function K on the *original* features. In particular, if I can find a formula for the right-hand side of Eq. 20.80, then I never have to calculate the functions ψ at all. In fact, I can even let q go to infinity!

A concrete and very useful example is the Gaussian kernel:

$$K_b(x_i, x_j) = \frac{1}{\sqrt{2\pi b^2}} \exp(-(x_i - x_j)^2 / 2b^2) \quad (20.81)$$

(Sometimes you see this without the normalizing factor.) The features here are actually *all* the powers of x_i and x_j , though they're not all equally weighted. This is in fact what people typically use with diffusion maps, rather than the Euclidean inner product, because the Gaussian distribution is preserved under diffusion.

[[TODO: Mention kernel PCA and kernel CCA at least]]

²⁹For much more about kernel methods in data mining, see Shawe-Taylor and Cristianini (2004).

20.7 Exercises

- Let \mathbf{B} be any $n \times n$ matrix. Show that if \vec{v} is an eigenvector of \mathbf{B} , then it is also an eigenvector of \mathbf{B}^2 , and of any power of \mathbf{B} . Conclude that \mathbf{B} and \mathbf{B}^2 have the *same* eigenvectors. (*Hint:* how many eigenvectors does each matrix have?) What happens to the eigenvalues?
- In local linear embedding, we obtain an $n \times n$ matrix \mathbf{w} , where w_{ij} is the weight on \vec{x}_j we use to reconstruct \vec{x}_i . Each row of \mathbf{w} sums to one. We then try to find coordinates y_1, y_2, \dots, y_n which minimize

$$\Phi(\mathbf{Y}) = \sum_{i=1}^n \left(y_i - \sum_{j=1}^n w_{ij} y_j \right)^2$$

where \mathbf{Y} is the $n \times 1$ matrix of y_i values (this is the $q = 1$ case, for simplicity). In the notes, we showed that this is the same as minimizing

$$\Phi(\mathbf{Y}) = \mathbf{Y}^T \mathbf{M} \mathbf{Y}$$

where

$$\mathbf{M} = ((\mathbf{I} - \mathbf{w})^T (\mathbf{I} - \mathbf{w}))$$

- (a) Show that \mathbf{M} is a symmetric matrix.
- (b) Show that $\mathbf{1}$ is an eigenvector of \mathbf{M} , and that its eigenvalue is zero.
- (c) Show that $\Phi(\mathbf{Y}) = \Phi(\mathbf{Y} + c\mathbf{1})$, where c is any constant and $\mathbf{1}$ is the $n \times 1$ matrix whose entries are all 1s. (*Hint:* one way is to use the previous two parts.)
- (d) Show that $\Phi(\mathbf{Y})$ is minimized by $\mathbf{Y} = \mathbf{0}$.
- (e) To avoid the trivial solution of setting all the y_i to zero, we impose the constraint that $n^{-1} \sum_{i=1}^n y_i^2 = 1$. We use a Lagrange multiplier to enforce this constraint; write down the Lagrangian (modified objective function) for the constrained minimization problem.
- (f) Show that a solution \mathbf{Y} to the constrained minimization problem must be an eigenvector of \mathbf{M} .

[[TODO: Check notation, opening, as this was originally a separate assignment]]

Chapter 21

Mixture Models

21.1 Two Routes to Mixture Models

21.1.1 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.

21.1.2 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

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.

21.1.3 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

$$f(x) = \sum_{k=1}^K \lambda_k f_k(x) \quad (21.1)$$

with the λ_k being the **mixing weights**, $\lambda_k > 0$, $\sum_k \lambda_k = 1$. Eq. 21.1 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,

$$Z \sim \text{Mult}(\lambda_1, \lambda_2, \dots, \lambda_K) \quad (21.2)$$

$$X|Z \sim f_Z \quad (21.3)$$

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

$$f(x) = \sum_{k=1}^K \lambda_k f(x; \theta_k) \quad (21.4)$$

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

features are 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.

21.1.4 Geometry

In Chapter 18, 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 19, 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.

21.1.5 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

21.1.6 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. 21.1; the hidden variable Z I introduced above is just the cluster label.

One of the very nice things about probabilistic clustering is that Eq. 21.1 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).

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.

21.1.7 Simulation

Simulating from a mixture model works rather like simulating from a kernel density estimate (§16.7.1). 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 .

21.2 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

$$\prod_{i=1}^n f(x_i; \theta) \quad (21.5)$$

for observations x_1, x_2, \dots, x_n . As always, we’ll use the logarithm to turn multiplication into addition:

$$\ell(\theta) = \sum_{i=1}^n \log f(x_i; \theta) \quad (21.6)$$

$$= \sum_{i=1}^n \log \sum_{k=1}^K \lambda_k f(x_i; \theta_k) \quad (21.7)$$

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 .

$$\frac{\partial \ell}{\partial \theta_j} = \sum_{i=1}^n \frac{1}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} \lambda_j \frac{\partial f(x_i; \theta_j)}{\partial \theta_j} \quad (21.8)$$

$$= \sum_{i=1}^n \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} \frac{1}{f(x_i; \theta_j)} \frac{\partial f(x_i; \theta_j)}{\partial \theta_j} \quad (21.9)$$

$$= \sum_{i=1}^n \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} \frac{\partial \log f(x_i; \theta_j)}{\partial \theta_j} \quad (21.10)$$

If we just had an ordinary parametric model, on the other hand, the derivative of the log-likelihood would be

$$\sum_{i=1}^n \frac{\partial \log f(x_i; \theta_j)}{\partial \theta_j} \quad (21.11)$$

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

$$w_{ij} = \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} \quad (21.12)$$

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 $X = x_i$,

$$w_{ij} = \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^K \lambda_k f(x_i; \theta_k)} = p(Z = j | X = x_i; \theta) \quad (21.13)$$

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_{ij} (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.

21.2.1 More about the EM Algorithm

[[TODO: Give explicit update for mixing proportions]]

[[TODO: Discuss continuous case]]

[[TODO: Relate general formulation here to explicit mixture set-up]]

[[TODO: Discuss other missing data approaches, including Monte Carlo EM and Geyer’s Monte Carlo missing data]]

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

$$\ell(\theta) = \log p(d; \theta) = \log \sum_b p(d, h; \theta) \quad (21.14)$$

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.

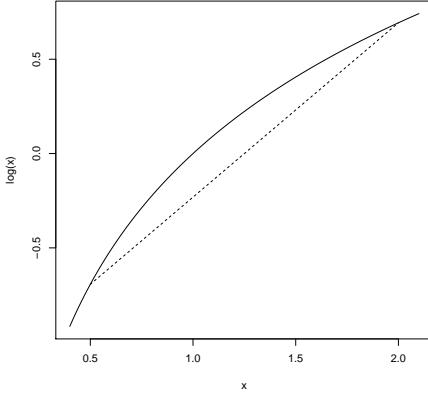
We can introduce an arbitrary⁶ distribution on h , call it $q(h)$, and we’ll

$$\ell(\theta) = \log \sum_b p(d, h; \theta) \quad (21.15)$$

$$= \log \sum_b \frac{q(h)}{q(h)} p(d, h; \theta) \quad (21.16)$$

$$= \log \sum_b q(h) \frac{p(d, h; \theta)}{q(h)} \quad (21.17)$$

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 21.1: *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 21.1). Algebraically, this means that

$$w \log t_1 + (1-w) \log t_2 \leq \log w t_1 + (1-w) t_2 \quad (21.18)$$

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 $\sum_{i=1}^r w_i = 1$,

$$\sum_{i=1}^r w_i \log t_i \leq \log \sum_{i=1}^r w_i t_i \quad (21.19)$$

In words: the log of the average is at least the average of the logs. This is called **Jensen's inequality**. So

$$\log \sum_b q(b) \frac{p(d, b; \theta)}{q(b)} \geq \sum_b q(b) \log \frac{p(d, b; \theta)}{q(b)} \quad (21.20)$$

$$\equiv J(q, \theta) \quad (21.21)$$

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

$$\frac{p(d, h; \theta)}{q(h)} = \frac{p(d, h; \theta)}{p(h|d; \theta)} = \frac{p(d, h; \theta)}{p(h, d; \theta)/p(d; \theta)} = p(d; \theta) \quad (21.22)$$

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,

$$J(q^{(t)}, \theta^{(t)}) \geq J(q^{(t-1)}, \theta^{(t)}) \quad (21.23)$$

and that, after the M step,

$$J(q^{(t)}, \theta^{(t+1)}) \geq J(q^{(t)}, \theta^{(t)}) \quad (21.24)$$

Putting these two inequalities together,

$$J(q^{(t+1)}, \theta^{(t+1)}) \geq J(q^{(t)}, \theta^{(t)}) \quad (21.25)$$

$$\ell(\theta^{(t+1)}) \geq \ell(\theta^{(t)}) \quad (21.26)$$

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?

$$\sum_b q(h) \log \frac{p(d, h; \theta)}{q(h)} = \sum_b q(h) \log p(d, h; \theta) - \sum_b q(h) \log q(h) \quad (21.27)$$

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!

21.2.2 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.

21.2.3 Topic Models and Probabilistic LSA

Mixture models over words provide an alternative to latent semantic indexing (§18.4) 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.

21.3 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 16. (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.

21.4 Worked Computing Example: Snoqualmie Falls Revisited

21.4.1 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.

21.4.2 Fitting a Mixture of Gaussians to Real Data

Let's go back to the Snoqualmie Falls data set, last used in §12.7⁷. 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 <- scan("http://www.stat.washington.edu/peter/book/data/set1", skip=1)
snoq <- snoqualmie[snoqualmie > 0]
```

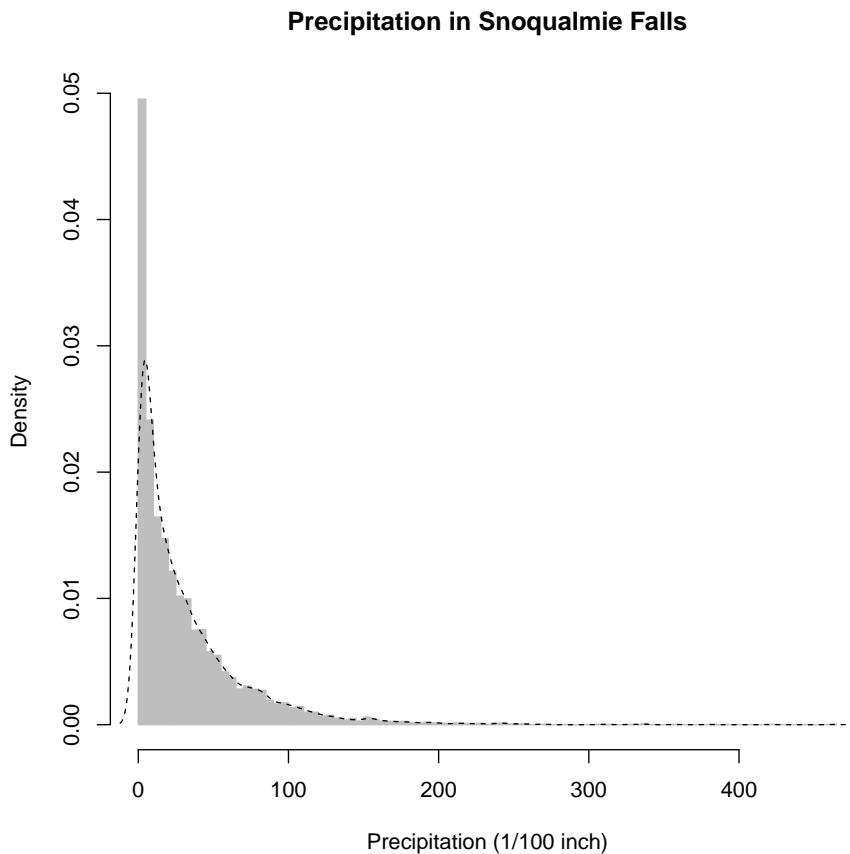
Figure 21.2 shows a histogram (with a fairly large number of bins), together with a simple kernel density estimate. This suggests that the distribution is rather skewed to the right, which is reinforced by the simple summary statistics:

```
summary(snoq)
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.0     6.0    19.0    32.3    44.0   463.0
```

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

⁷See that section for explanations of some of the data manipulation done in this section.



```
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="dashed")
```

FIGURE 21.2: 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)
```

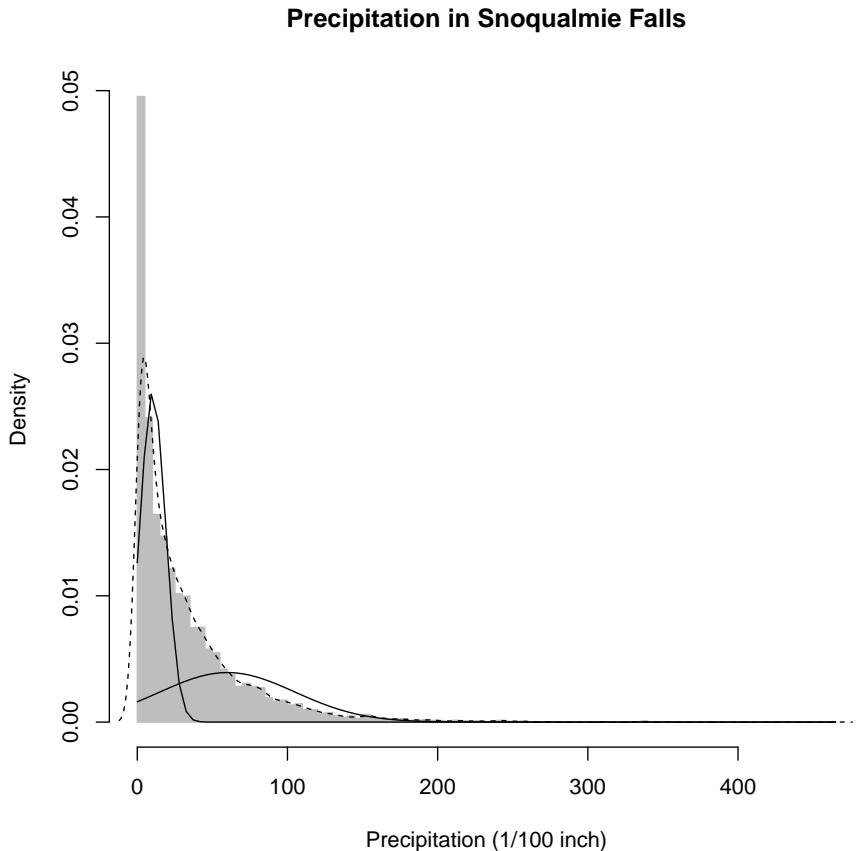
```
summary(snoq.k2)
## summary of normalmixEM object:
##           comp 1     comp 2
## lambda  0.557523 0.442477
## mu      10.266161 60.009440
## sigma   8.510234 44.997233
## loglik at estimate: -32681
```

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)
invisible(sapply(1:2,plot.normal.components,mixture=snoq.k2))
```

FIGURE 21.3: 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.

21.4.3 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.⁸ The CDF of a two-component mixture is

$$F(x) = \lambda_1 F_1(x) + \lambda_2 F_2(x) \quad (21.28)$$

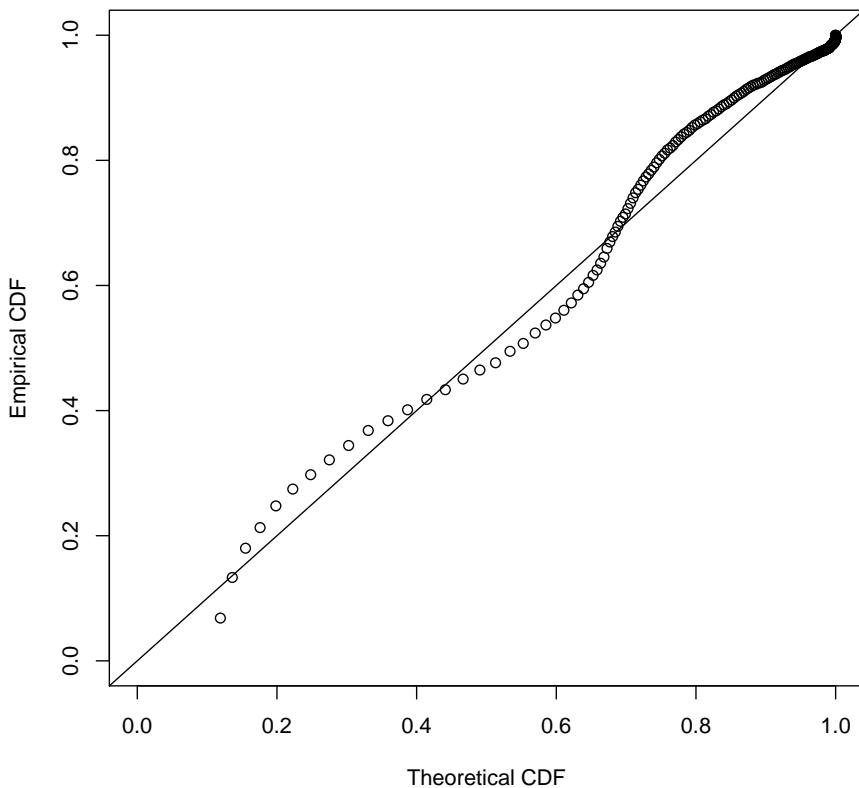
and similarly for more components. A little R experimentation gives a function for computing the CDF of a Gaussian mixture:

```
pnormmix <- 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 21.4. We do not have the tools to assess whether the size of the departure from the main diagonal is significant⁹, but the fact that the errors are so very structured is rather suspicious.

⁸We saw this principle when we looked at generating random variables in Chapter 5.

⁹Though 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 <- pnormmix(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 21.4: 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 $\leq x$.

21.4.4 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
```

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 may 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 16) — 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 lot¹⁰:

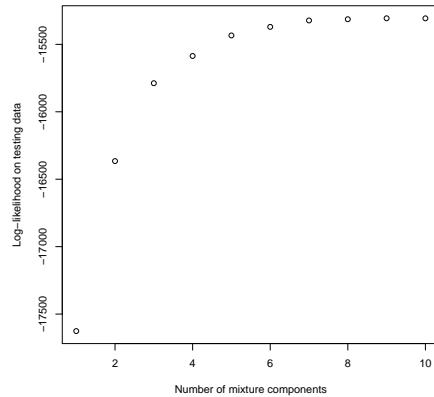
```

loglikes
## [1] -17626 -16366 -15788 -15586 -15434 -15370 -15323 -15314 -15307 -15307

```

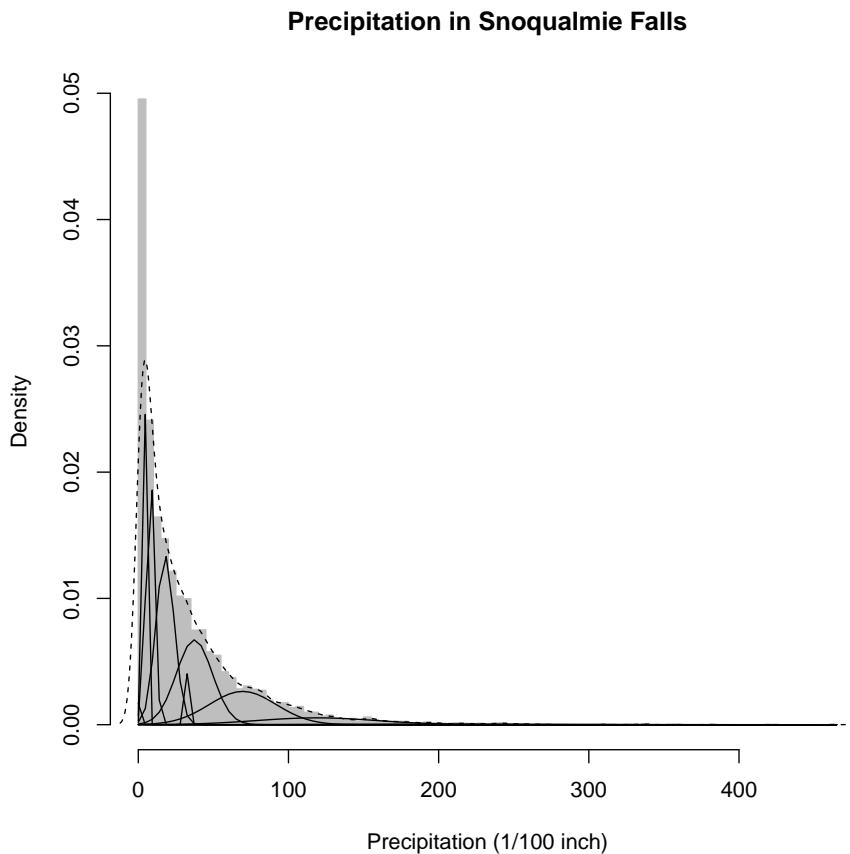
(See also Figure 21.5). This favors nine components to the mixture. It looks like Figure 21.6. The calibration is now nearly perfect, at least on the training data (Figure 21.7).

¹⁰Notice 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.



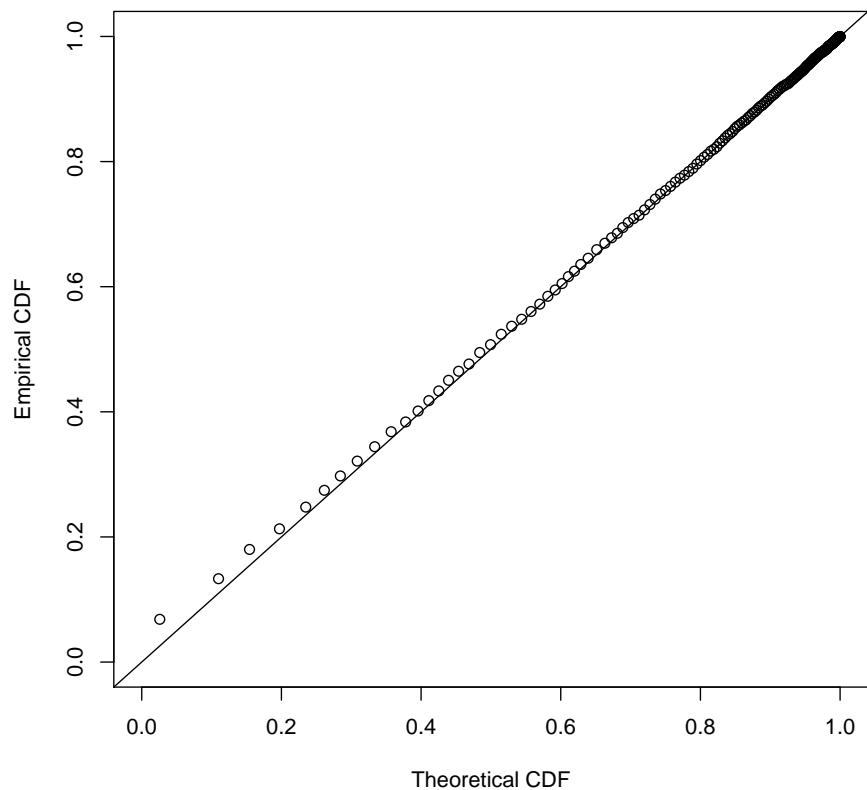
```
plot(x=1:10, y=loglikes, xlab="Number of mixture components",
      ylab="Log-likelihood on testing data")
```

FIGURE 21.5: *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)
invisible(sapply(1:9,plot.normal.components,mixture=snoq.k9))
```

FIGURE 21.6: As in Figure 21.3, 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 21.7: Calibration plot for the nine-component Gaussian mixture.

21.4.5 Interpreting the Mixture Components, or Not

The components of the mixture are far from arbitrary. It appears from Figure 21.6 that as the mean increases, so does the variance. This impression is confirmed from Figure 21.8. 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 21.9). It requires 26 parameters¹¹, 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.

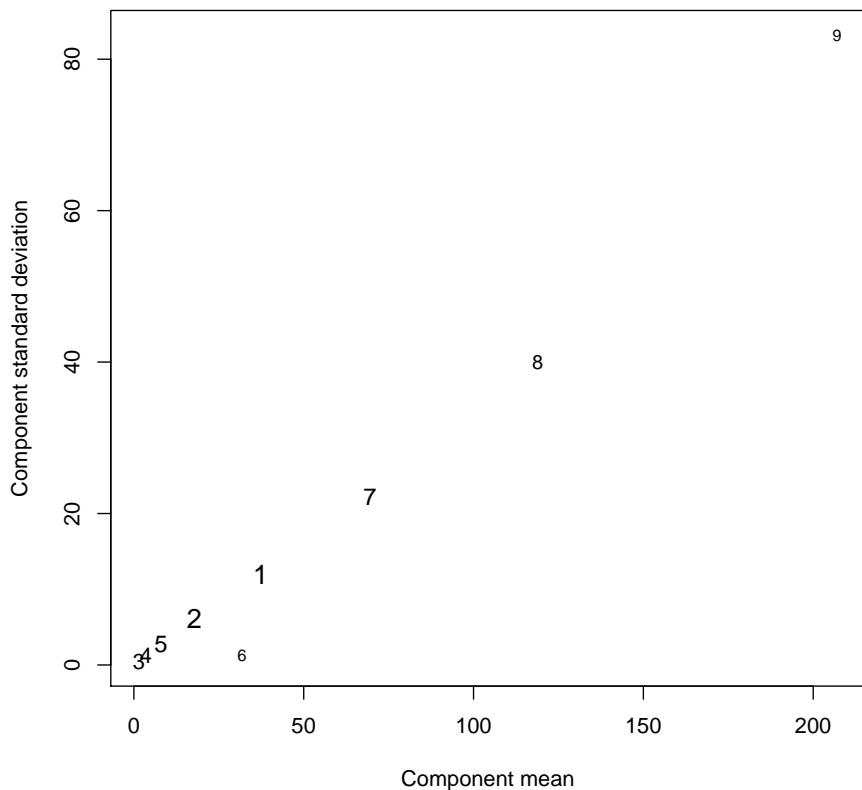
On the substantive side, there are various things we could do to check the idea that wet days really do divide into nine types. These are going to be informed by our background knowledge about the weather. One of the things we know, for example, is that weather patterns more or less repeat in an annual 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)
```

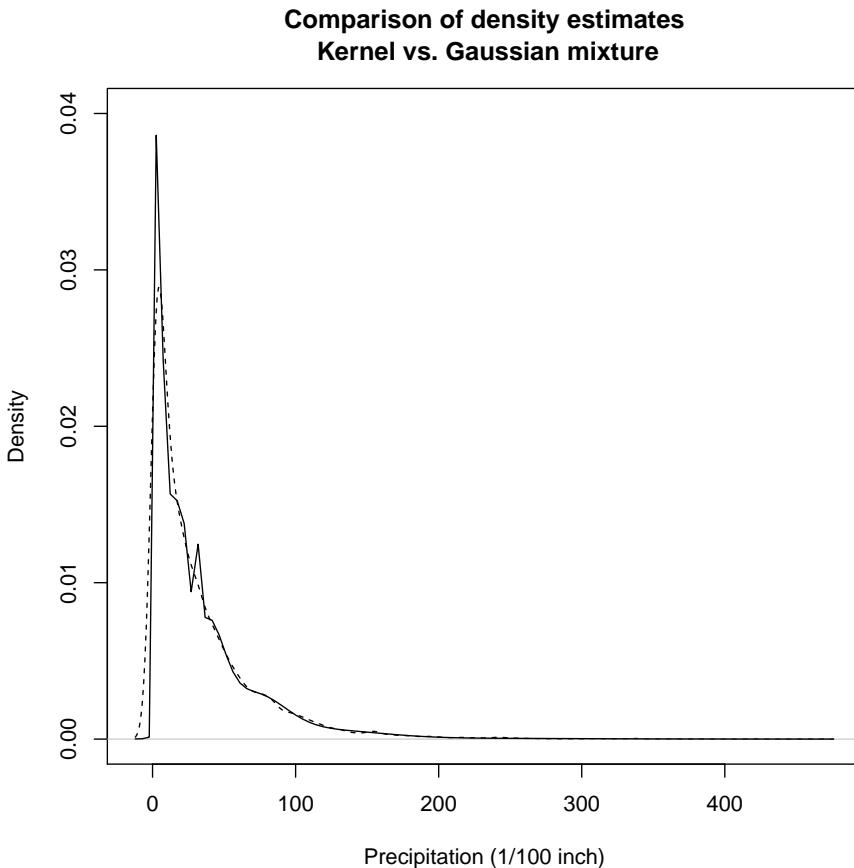
We can’t just plot this and hope to see any useful patterns, because we want to see stuff recurring every year, and we’ve stripped out the dry days, the division into years, the padding to handle leap-days, etc. Thus, we need to do a bit of R magic.

¹¹A 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 21.8: 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 21.9: Dashed line: kernel density estimate. Solid line: the nine-Gaussian mixture. Notice that the mixture, unlike the KDE, gives negligible probability to negative precipitation.

Remember we started with a giant vector `snoqualmie` which had all days, wet or dry; let's copy that into a data frame, to which we'll add the classes and the days of the year.

```
snoqualmie.classes <- data.frame(precip=snoqualmie, class=0)
years <- 1948:1983
snoqualmie.classes$day <- rep(c(1:366,1:365,1:365,1:365),times=length(years)/4)
wet.days <- (snoqualmie > 0)
snoqualmie.classes$class[wet.days] <- day.classes
```

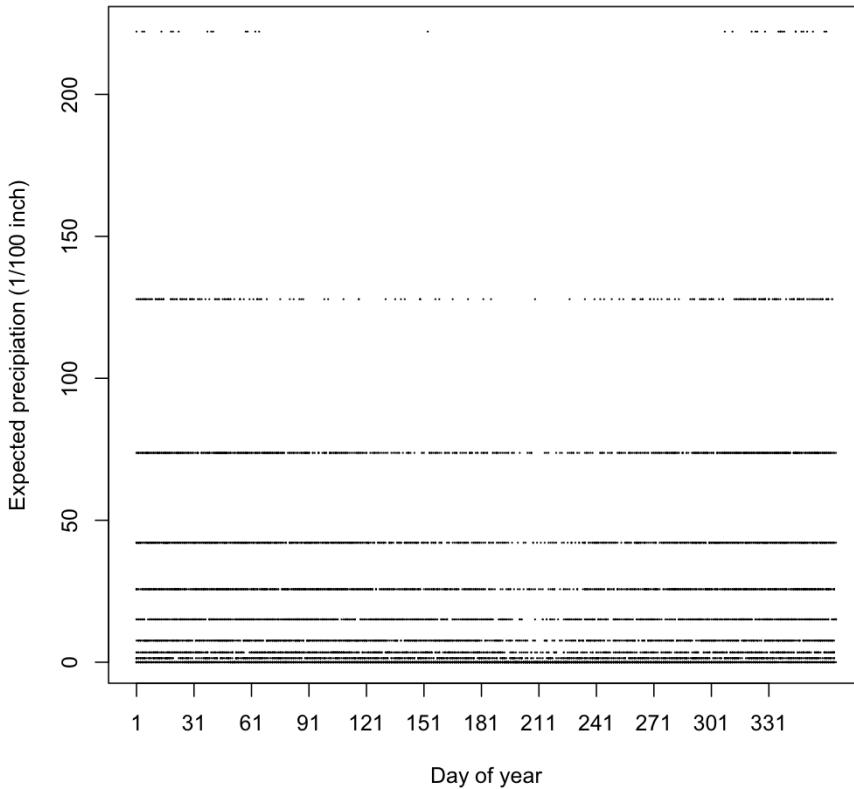
Now, it's somewhat inconvenient that the index numbers of the components do not really tell us about the mean amount of precipitation. Let's try replacing the numerical labels in `snoqualmie.classes` by those means.

```
snoqualmie.classes$class[wet.days] <- snoq.k9$mu[day.classes]
```

This leaves alone dry days (still zero) and NA days (still NA). Now we can plot (Figure 21.10).

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.¹²

¹²A 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(x=snoqualmie.classes$day, y=snoqualmie.classes$class,
      xlim=c(1,366), ylim=range(snoq.k9$mu), xaxt="n",
      xlab="Day of year", ylab="Expected precipitation (1/100 inch)",
      pch=16, cex=0.2)
axis(1,at=1+(0:11)*30)
```

FIGURE 21.10: 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.

21.4.6 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,

$$2(\log L_{k+1} - \log L_k) \sim \chi^2_{\dim(k+1) - \dim(k)} \quad (21.29)$$

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 G.) 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.

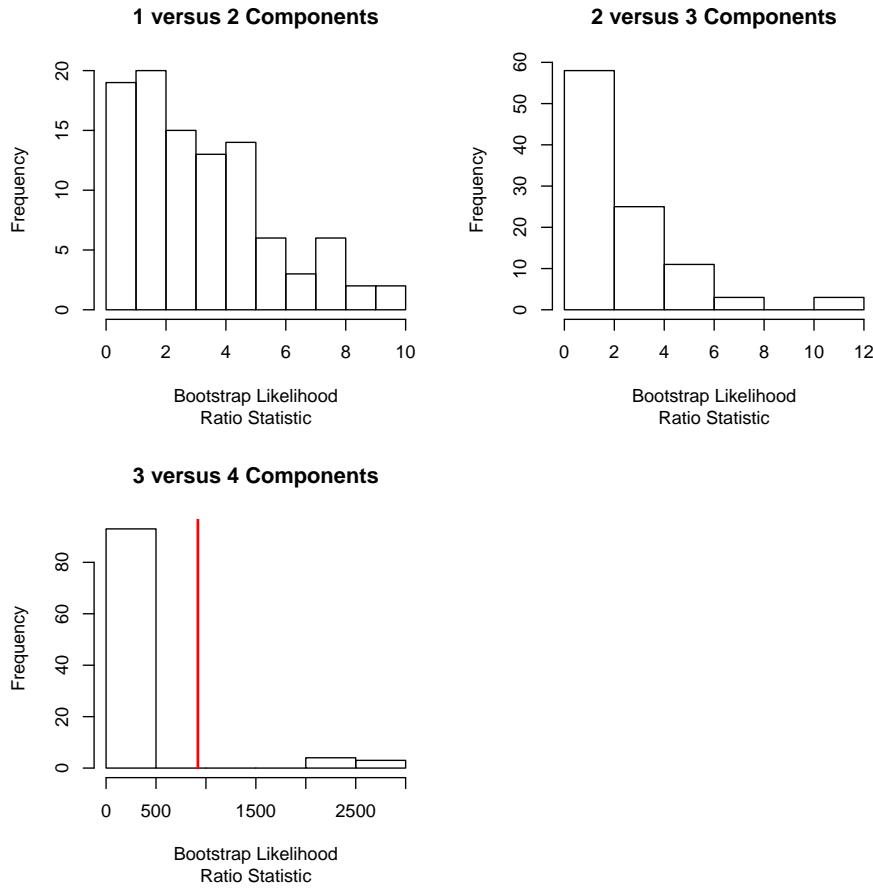
While it is not too hard to program this by hand (Exercise 4), the `mixtools` package contains a function to do this for us, called `boot.comp`, for “bootstrap comparison”. Let’s try it out (Figure 21.11).

The command in the figure 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 three 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 21.11). 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`.

```
str(snoq.boot)
## List of 3
## $ p.values : num [1:3] 0 0 0.07
## $ log.lik   :List of 3
##   ..$ : num [1:100] 7.926 0.48 0.624 1.479 3.335 ...
##   ..$ : num [1:100] 2.75 2.64 3.62 2.09 5.06 ...
##   ..$ : num [1:100] 2.46e+03 1.27 5.57e-02 1.04 4.55 ...
## $ obs.log.lik: num [1:3] 5096 2354 920
```



```
snoq.boot <- boot.comp(snoq,max.comp=10,mix.type="normalmix",
                         maxit=400,epsilon=1e-2)
```

FIGURE 21.11: *Histograms produced by `boot.comp()`. The vertical red lines mark the observed difference in log-likelihoods.*

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 21.5, 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 21.10 is no more encouraging for the reality of the latent classes.

21.5 Exercises

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 (Chapter 17) to check calibration, along the lines of Figure 21.4.

Chapter 22

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.

22.1 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:

$$p(X_1, X_2, \dots, X_p | F_1, F_2, \dots, F_q) = \prod_{i=1}^p p(X_i | F_1, \dots, F_q) \quad (22.1)$$

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

$$p(X_1, X_2, \dots, X_p | F_1, F_2, \dots, F_q) = \prod_{i=1}^p p(X_i | F_{\text{loads}(i)}) \quad (22.2)$$

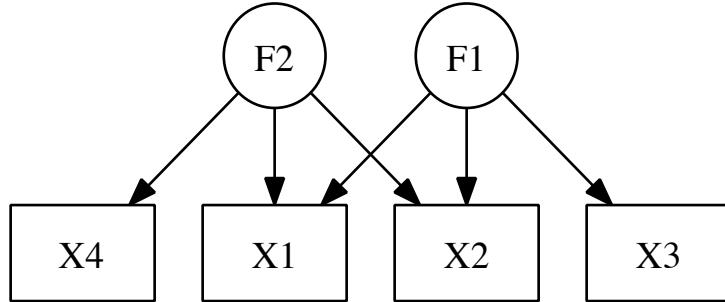


FIGURE 22.1: Illustration of a typical model with two latent factors (F_1 and F_2 , in circles) and four observables (X_1 through X_4).

Consider Figure 22.1. The conditional distribution of observables given factors is

$$p(X_1, X_2, X_3, X_4 | F_1, F_2) = p(X_1 | F_1, F_2)p(X_2 | F_1, F_2)p(X_3 | F_1)p(X_4 | F_2) \quad (22.3)$$

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.

22.2 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”:

$$p(X_1, X_2, \dots, X_p) = \prod_{i=1}^p p(X_i | X_{\text{parents}(i)}) \quad (22.4)$$

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. 22.4, 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. 22.4. (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.

22.2.1 Conditional Independence and the Markov Property

The probabilistic implication of Eq. 22.4 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 I for a brief review of the ideas and jargon of graph theory.

²See §22.4 for remarks on undirected graphical models, and graphs with cycles.

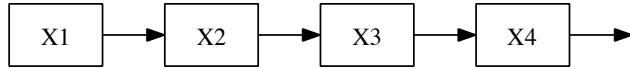


FIGURE 22.2: DAG for a discrete-time Markov process. At each time t , X_t is the child of X_{t-1} and the parent of X_{t+1} .

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,

$$X_{t+1} \perp\!\!\!\perp (X_{t-1}, X_{t-2}, \dots, X_1) | X_t \quad (22.5)$$

from which it follows that

$$(X_{t+1}, X_{t+2}, X_{t+3}, \dots) \perp\!\!\!\perp (X_{t-1}, X_{t-2}, \dots, X_1) | X_t \quad (22.6)$$

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:

$$X_I \perp\!\!\!\perp X_{\text{non-descendants}(I)} | X_{\text{parents}(I)} \quad (22.7)$$

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 22.2⁴.

On the other hand, if we condition on one of X_i ’s children, X_i will generally be dependent on any other parent of that child. If we condition on multiple children of X_i , we’ll generally find X_i is dependent on all its co-parents. It should be plausible, and is in fact true, that X_i is independent of everything else in the graph if we condition on its parents, its children, and its children’s other parents. This set of nodes is called X_i ’s **Markov blanket**.

22.3 Examples of DAG Models and Their Uses

Factor models are examples of DAG models (as we’ve seen). So are mixture models (Figure 22.3) and Markov chains (see above). DAG models are considerably more

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.

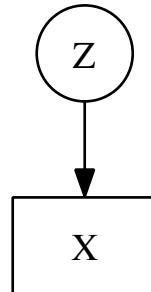


FIGURE 22.3: *DAG for a mixture model. The latent class Z is exogenous, and the parent of the observable random vector X . (If the components of X are conditionally independent given Z , they could be represented as separate boxes on the lower level.)*

flexible, however, and can combine observed and unobserved variables in many ways.

Consider, for instance, Figure 22.4. 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) 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 can-

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.

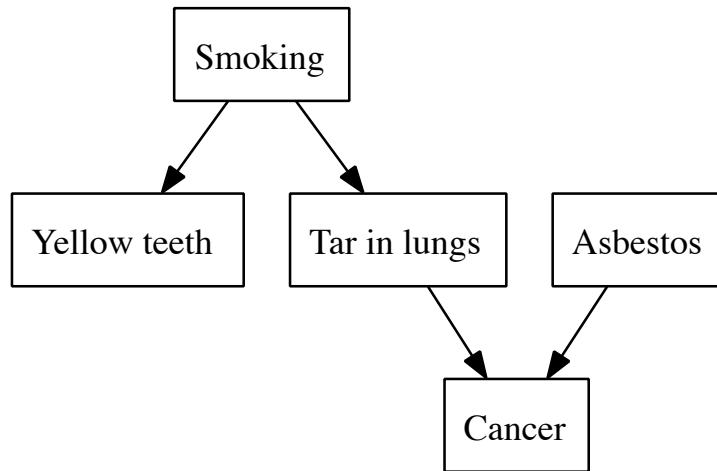


FIGURE 22.4: DAG model indicating (hypothetical) relationships between smoking, asbestos, cancer, and covariates.

cer. 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 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’s really 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 22.4, 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

22.3.1 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:

$$p(X_i|X_1, X_2, X_{i-1}, X_{i+1}, X_p) = \frac{p(X_1, X_2, X_{i-1}, X_i, X_{i+1}, X_p)}{\int p(X_1, X_2, X_{i-1}, x_i, X_{i+1}, X_p) dx_i} \quad (22.8)$$

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 from §22.2.1 that X_i is independent of all the other variables given its Markov blanket, i.e., its parents, its children, and the other parents of its children. We can therefore drop from the conditioning everything which isn’t in the Markov blanket. 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 a relic of the days of ignorance, when our ancestors groped towards truths they could not 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 §23.2.2.

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

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

22.4 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.

22.4.1 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:

$$X_I \perp\!\!\!\perp X_{\text{non-neighbors}(I)} | X_{\text{neighbors}(I)} \quad (22.9)$$

This corresponds to a factorization of the joint distribution, but a more complex one than that of Eq. 22.4, 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

$$p(X_1, X_2, \dots, X_p) \propto \prod_{c \in \text{cliques}(G)} \psi_c(X_{i \in c}) \quad (22.10)$$

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

from the definition of conditional probability, there is nothing distinctively Bayesian here — no subjective probability, or assigning probabilities to hypotheses.

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.

one has

$$p(X_1, X_2, \dots, X_p) \propto e^{-\sum_{c \in \text{cliques}(G)} U_i(X_{i \in c})} \quad (22.11)$$

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.

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. 22.11 is probably extremely suggestive, and you should read Mandelbrot (1962).

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.

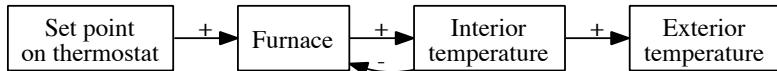


FIGURE 22.5: *Directed but cyclic graphical model of a feedback loop. Signs (+, - on arrows are “guides to the mind”. Cf. Figure 22.6.*

22.4.2 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 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 22.5).

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 22.6). Sometimes¹⁴, it is clear that when people draw a diagram like Figure 22.5, 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 22.6.

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.

22.5 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 in Part III. 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

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 (Fisher, 1983, 2010).

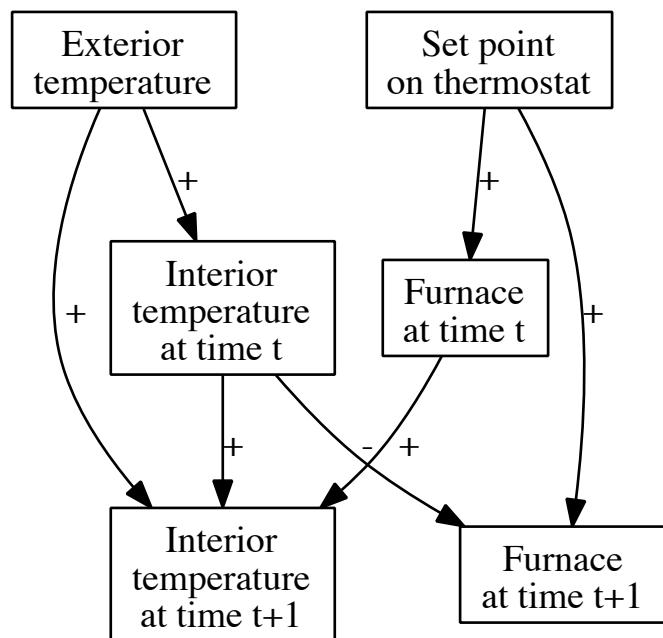


FIGURE 22.6: Directed, acyclic graph for the situation in Figure 22.5, taking into account the fact that it takes time to traverse a feedback loop. One should imagine this repeating to times $t + 2$, $t + 3$, etc., and extending backwards to times $t - 1$, $t - 2$, etc., as well. Notice that there are no longer any cycles.

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 who know less math anyone taking this course. Li (1975), while older, is very enthusiastic and has many interesting applications in biology.

Part III

Causal Inference

Chapter 23

Graphical Causal Models

23.1 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 23.1). 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?

[[TODO: discuss latent variables and measurement either here or in the previous chapter]]

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.

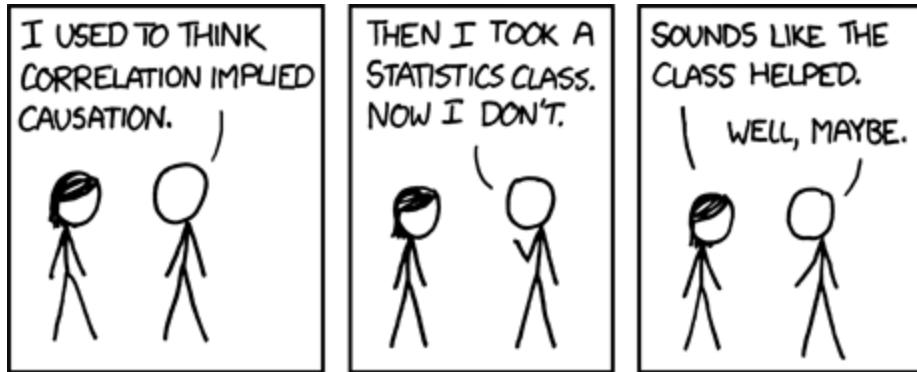


FIGURE 23.1: “Correlation doesn’t imply causation, but it does waggle its eyebrows suggestively and gesture furtively while mouthing ‘look over there’” (Image and text copyright by Randall Munroe, used here under a Creative Commons attribution-noncommercial license; see <http://xkcd.com/552/>. [[TODO: Excise from the commercial version]])

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.

23.2 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.

23.2.1 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,

$$\Pr(X_E|X_c = x_c) \tag{23.1}$$

is probabilistic conditioning, or selecting a sub-ensemble from the old mechanisms; but

$$\Pr(X_E|do(X_c = x_c)) \tag{23.2}$$

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 24.

23.2.2 Back to Teeth

Let us return to the example of Figure 22.4, 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

$$\begin{aligned} & p(\text{Yellow teeth, Smoking, Asbestos, Tar in lungs, Cancer}) \\ &= p(\text{Smoking})p(\text{Asbestos}) \\ &\quad \times p(\text{Tar in lungs}|\text{Smoking}) \\ &\quad \times p(\text{Yellow teeth}|\text{Smoking}) \\ &\quad \times p(\text{Cancer}|\text{Asbestos, Tar in lungs}) \end{aligned} \tag{23.3}$$

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 23.2⁸ 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/>).

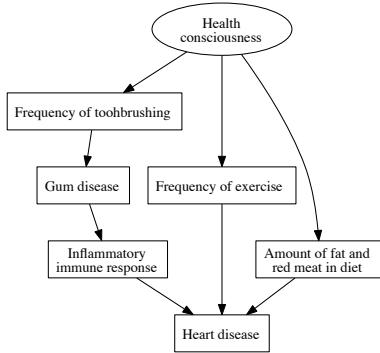


FIGURE 23.2: *Graphical model illustrating hypothetical pathways linking brushing your teeth to not getting heart disease.*

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 23.2 to Figure 23.3, and we would have

$$p(\text{Heart disease} | \text{Brushing} = b) \neq p(\text{Heart disease} | do(\text{Brushing} = b)) \quad (23.4)$$

⁹Hopefully, by ensuring that everyone brushes, rather than keeping people from brushing.

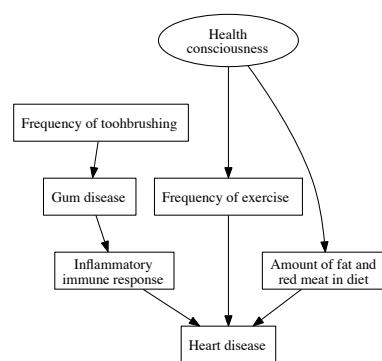


FIGURE 23.3: The previous graphical model, “surgically” altered to reflect a manipulation (do) of brushing.

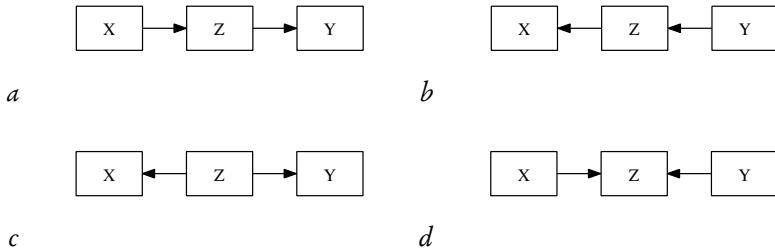


FIGURE 23.4: *Four DAGs for three linked variables. The first two (a and b) are called chains; c is a fork; d is a collider. If these were the whole of the graph, we would have $X \not\perp\!\!\!\perp Y$ and $X \perp\!\!\!\perp Y|Z$. For the collider, however, we would have $X \perp\!\!\!\perp Y$ while $X \not\perp\!\!\!\perp Y|Z$.*

23.3 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 23.4. 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 .

All of this flips around when we condition on the intermediate variable (Z in Figure 23.4). The chains (Figures 23.4a and b), conditioning on the intermediate

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.

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 23.4c) — 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.

23.3.1 D-Separation Illustrated

The discussion of d-separation has been rather abstract, and perhaps confusing for that reason. Figure 23.5 shows a DAG which might make this clearer and more concrete¹⁵.

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).

¹⁵I am grateful to Donald Schoolmaster, Jr., for pointing out errors in an earlier version of this example.

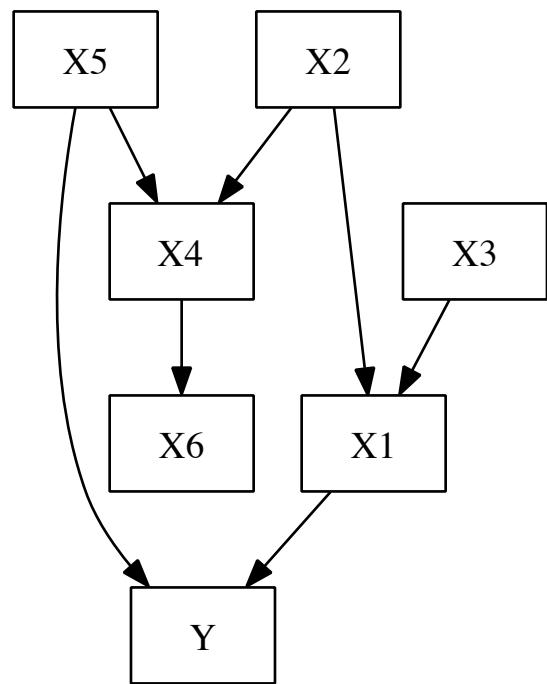


FIGURE 23.5: Example DAG used to illustrate d -separation.

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 $X_3 \rightarrow X_1 \rightarrow Y \leftarrow X_5$. Conditioning on X_2 (and nothing else) 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, again, 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 X_1 opens the path there. 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.

$$Y \not\perp\!\!\!\perp X_3 \quad (23.5)$$

$$Y \perp\!\!\!\perp X_3 | X_1 \quad (23.6)$$

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:

$$Y \not\perp\!\!\!\perp X_3 | X_1, X_4 \quad (23.7)$$

$$Y \not\perp\!\!\!\perp X_3 | X_1, X_6 \quad (23.8)$$

$$Y \not\perp\!\!\!\perp X_3 | X_1, X_4, X_6 \quad (23.9)$$

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

$$Y \perp\!\!\!\perp X_3 | X_1, X_2 \quad (23.10)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_5 \quad (23.11)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_2, X_5 \quad (23.12)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_2, X_4 \quad (23.13)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_2, X_6 \quad (23.14)$$

$$Y \perp\!\!\!\perp X_3 | X_1, X_2, X_5, X_6 \quad (23.15)$$

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.

23.3.2 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:

$$\mathbf{E}[X_i | X_{\text{parents}(i)}] = \sum_{j \in \text{parents}(i)} w_{ji} X_j \quad (23.16)$$

¹⁶Some people use the phrase “structural equation models” for such models exclusively.

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 .
- 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.

23.3.3 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.

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 $\frac{d\mathbb{E}[Y|X=x]}{dx} \geq 0$

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.

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.

23.4 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:

$$X \perp\!\!\!\perp Y \Leftrightarrow p(x, y) = p(x)p(y) \quad (23.17)$$

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:

$$\log \frac{p(x, y)}{p(x)p(y)} \quad (23.18)$$

This will always be exactly 0 when $X \perp\!\!\!\perp Y$. We use its average value as our measure of dependence:

$$I[X; Y] \equiv \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \quad (23.19)$$

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.

(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 **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,

$$-\frac{1}{n} \log \beta_n \rightarrow I[X; Y] \quad (23.20)$$

(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,

$$I[X; Y|Z = z] \equiv \sum_{x,y} p(x, y|z) \log \frac{p(x, y|z)}{p(x|z)p(y|z)} \quad (23.21)$$

And we can average over z ,

$$I[X; Y|Z] \equiv \sum_z p(z) I[X; Y|Z = z] \quad (23.22)$$

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.

23.5 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;

²²Using the same type of convexity argument (“Jensen’s inequality”) we used in Lecture 19 for understanding the details of the EM algorithm.

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 (§23.4), 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.

23.6 Exercises

1. Show, for the graphical model in Figure 22.4, 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 23.5, 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 23.5.

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 24

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.

24.1 Causal Effects, Interventions and Experiments

As a reminder, when I talk about the causal effect of X on Y , which I write

$$\Pr(Y|do(X = x)) \tag{24.1}$$

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

$$\Pr(Y|X = x) \tag{24.2}$$

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 24.1). As we saw in Chapter 22, 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 23, 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 24.1: *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.

24.1.1 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.

24.2 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 at the over-simplification, 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.

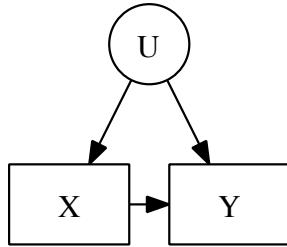


FIGURE 24.1: *The distribution of Y given X , $\Pr(Y|X)$, confounds the actual causal effect of X on Y , $\Pr(Y|do(X=x))$, with the indirect dependence between X and Y created by their unobserved common cause U . (You may imagine that U is really more than one variable, with some internal sub-graph.)*

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 24.1.

In Figure 24.1, 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 24.1 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.

24.3 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

$$\Pr(Y|do(X = x)) = \sum_t \Pr(Y|X = x, \text{Pa}(X) = t) \Pr(\text{Pa}(X) = t) \quad (24.3)$$

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 24.2 illustrates, and as the next homework will drive home, *adding* an ill-chosen covariate to a regression can create confounding.

[[TODO: Fix "the next homework" bit above]]

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.

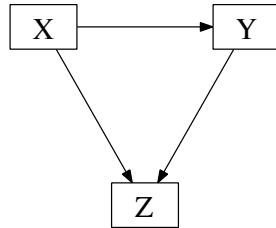


FIGURE 24.2: “Controlling for” additional variables can introduce bias into estimates of causal effects. Here the effect of X on Y is directly identifiable, $\Pr(Y|do(X=x)) = \Pr(Y|X=x)$. If we also condition on Z however, because it is a common effect of X and Y , we’d get $\Pr(Y|X=x, Z=z) \neq \Pr(Y|X=x)$. 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.

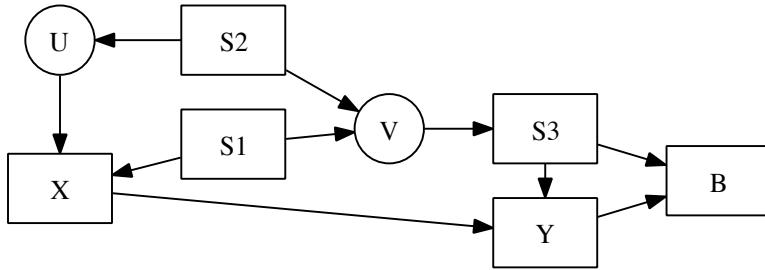


FIGURE 24.3: Illustration of the back-door criterion for identifying the causal effect of X on Y . Setting $S = \{S_1, S_2\}$ satisfies the criterion, but neither S_1 nor S_2 on their own would. Setting $S = \{S_3\}$, or $S = \{S_1, S_2, S_3\}$ also works. Adding B to any of the good sets makes them fail the criterion.

24.3.1 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 24.3.) When S meets the back-door criterion,

$$\Pr(Y|do(X=x)) = \sum_s \Pr(Y|X=x, S=s) \Pr(S=s) \quad (24.4)$$

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. 24.3 that

$$\Pr(Y|do(X=x)) = \sum_t \Pr(\text{Pa}(X)=t) \Pr(Y|X=x, \text{Pa}(X)=t) \quad (24.5)$$

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. 24.4.

Now suppose we can always introduce another set of conditioned variables, if we sum out over them:

$$\Pr(Y|do(X=x)) = \sum_t \Pr(\text{Pa}(X)=t) \sum_s \Pr(Y, S=s | X=x, \text{Pa}(X)=t) \quad (24.6)$$

We can do this for *any* set of variables S , it's just probability. It's also just probability that

$$\begin{aligned} \Pr(Y, S | X=x, \text{Pa}(X)=t) &= \\ \Pr(Y | X=x, \text{Pa}(X)=t, S=s) \Pr(S=s | X=x, \text{Pa}(X)=t) \end{aligned} \quad (24.7)$$

so

$$\begin{aligned} \Pr(Y|do(X=x)) &= \\ \sum_t \Pr(\text{Pa}(X)=t) \sum_s \Pr(Y | X=x, \text{Pa}(X)=t, S=s) \Pr(S=s | X=x, \text{Pa}(X)=t) \end{aligned} \quad (24.8)$$

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

$$\begin{aligned} \Pr(Y|do(X=x)) &= \\ \sum_t \Pr(\text{Pa}(X)=t) \sum_s \Pr(Y | X=x, S=s) \Pr(S=s | X=x, \text{Pa}(X)=t) \end{aligned} \quad (24.9)$$

Point (ii) of the criterion, not containing descendants of X , means (by the Markov property) that $X \perp\!\!\!\perp S | \text{Pa}(X)$. Therefore

$$\begin{aligned} \Pr(Y|do(X=x)) &= \\ \sum_t \Pr(\text{Pa}(X)=t) \sum_s \Pr(Y | X=x, S=s) \Pr(S=s | \text{Pa}(X)=t) \end{aligned} \quad (24.10)$$

Since $\sum_t \Pr(\text{Pa}(X)=t) \Pr(S=s | \text{Pa}(X)=t) = \Pr(S=s)$, we have, at last,

$$\Pr(Y|do(X=x)) = \sum_s \Pr(Y | X=x, S=s) \Pr(S=s) \quad (24.11)$$

as promised. \square

24.3.1.1 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.)

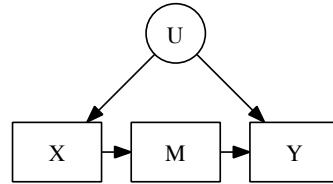


FIGURE 24.4: Illustration of the front-door criterion, after Pearl (2009b, Figure 3.5). X , Y and M are all observed, but U is an unobserved common cause of both X and Y . $X \leftarrow U \rightarrow Y$ is a back-door path confounding the effect of X on Y with their common cause. However, all of the 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 $\Pr(Y|do(X = x))$.

24.3.2 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

$$\begin{aligned} \Pr(Y|do(X = x)) &= \\ &\sum_m \Pr(M = m|X = x) \sum_{x'} \Pr(Y|X = x', M = m) \Pr(X = x') \end{aligned} \tag{24.12}$$

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 :

$$\Pr(Y|do(X = x)) = \sum_m \Pr(Y|do(M = m)) \Pr(M = m|do(X = x)) \tag{24.13}$$

Clause (ii) says that we can get the effect of X on M directly,

$$\Pr(M = m|do(X = x)) = \Pr(M = m|X = x). \tag{24.14}$$

Clause (iii) say that X satisfies the back-door criterion for identifying the effect of M on Y , and the inner sum in Eq. 24.12 is just the back-door computation (Eq. 24.4) of $\Pr(Y|do(M = m))$. So really we *are* using the back door criterion, twice. (See Figure 24.4.)

For example, in the “does tooth-brushing prevent heart-disease?” example of §23.2.2, 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 23.2.

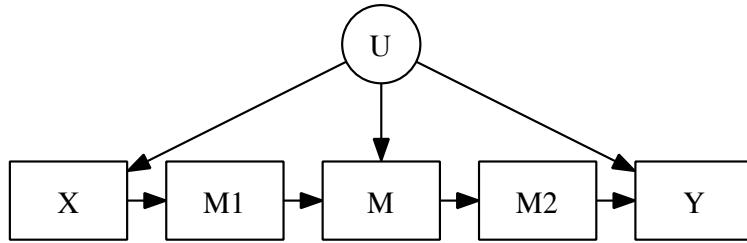


FIGURE 24.5: *The path $X \rightarrow M \rightarrow Y$ contains all the mechanisms by which X influences Y , but is not isolated from the rest of the system ($U \rightarrow M$). The sub-mechanisms $X \rightarrow M_1 \rightarrow M$ and $M \rightarrow M_2 \rightarrow Y$ are isolated, and the original causal effect can be identified by composing them.*

24.3.2.1 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 24.12.

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 24.5. 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. 24.13, 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.

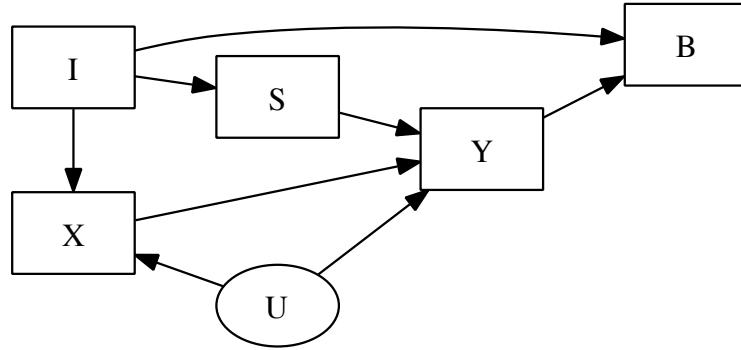


FIGURE 24.6: A valid instrumental variable, I , is related to the cause of interest, X , and influences Y only through its influence on X , at least once control variables block other paths. Here, to use I as an instrument, we should condition on S , but should not condition on B . (If we could condition on U , we would not need to use an instrument.)

24.3.3 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 24.6.)

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

$$\Pr(Y|do(I = i)) = \sum_x \Pr(Y|do(X = x)) \Pr(X = x|do(I = i)) \quad (24.15)$$

as in Eq. 24.3. We can thus identify the causal effect of X on Y whenever Eq. 24.15 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 integral equations¹¹, which is beyond the scope of this class; the upshot is that, in gen-

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).

¹¹If X is continuous, then the analog of Eq. 24.15 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 $\Pr(Y|do(I = i))$.

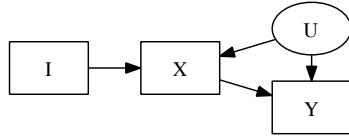


FIGURE 24.7: *I* acts as an instrument for estimating the effect of *X* on *Y*, despite the presence of the confounding, unobserved variable *U*.

eral, 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 24.7, 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

$$X = \alpha_0 + \alpha I + \delta U + \epsilon_X \quad (24.16)$$

and

$$Y = \beta_0 + \beta X + \gamma U + \epsilon_Y \quad (24.17)$$

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,

$$Y = \beta_0 + \beta \alpha_0 + \beta \alpha I + (\beta \delta + \gamma) U + \beta \epsilon_X + \epsilon_Y \quad (24.18)$$

Since *U*, ϵ_X and ϵ_Y are all unobserved, we can re-write this as

$$Y = \gamma_0 + \beta \alpha I + \eta \quad (24.19)$$

where $\eta = (\beta \delta + \gamma) U + \beta \epsilon_X + \epsilon_Y$ has mean zero.

Now take the covariances:

$$\text{Cov}[I, X] = \alpha \text{Var}[I] + \text{Cov}[\epsilon_X, I] \quad (24.20)$$

$$\text{Cov}[I, Y] = \beta \alpha \text{Var}[I] + \text{Cov}[\eta, I] \quad (24.21)$$

$$\begin{aligned} &= \beta \alpha \text{Var}[I] + (\beta \delta + \gamma) \text{Cov}[U, I] \\ &\quad + \beta \text{Cov}[\epsilon_X, I] + \text{Cov}[\epsilon_Y, I] \end{aligned} \quad (24.22)$$

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,

$$\beta = \frac{\text{Cov}[I, Y]}{\text{Cov}[I, X]} \quad (24.23)$$

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

$$\frac{\text{Cov}[X, Y]}{\text{Var}[X]} = \frac{\beta \text{Var}[X] + \gamma \text{Cov}[U, X]}{\text{Var}[X]} \quad (24.24)$$

$$= \beta + \gamma \frac{\text{Cov}[U, X]}{\text{Var}[X]} \quad (24.25)$$

$$= \beta + \gamma \frac{\delta \text{Var}[U]}{\alpha^2 \text{Var}[I] + \delta^2 \text{Var}[U] + \text{Var}[\epsilon_X]} \quad (24.26)$$

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 §25.2 we’ll look at the more complicated estimation methods which can handle this, again assuming linearity.

24.3.3.1 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 24.8). 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 24.9).

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”.

24.3.3.2 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

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 $E[Y|X = x] = r_0 + r_1 x$, then Eq. 24.26 is *exactly* the coefficient we would want to use. OLS is doing its job.

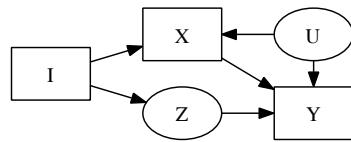


FIGURE 24.8: *I* is not a valid instrument for identifying the effect of *X* on *Y*, because *I* can influence *Y* through a path not going through *X*. If we could control for *Z*, however, *I* would become valid.

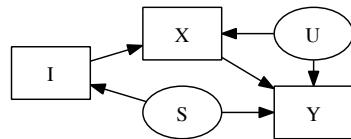


FIGURE 24.9: *I* is not a valid instrument for identifying the effect of *X* on *Y*, because there is an unblocked back-door path connecting *I* and *Y*. If we could control for *S*, however, *I* would become valid.

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 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

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>.

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. 24.23 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”.

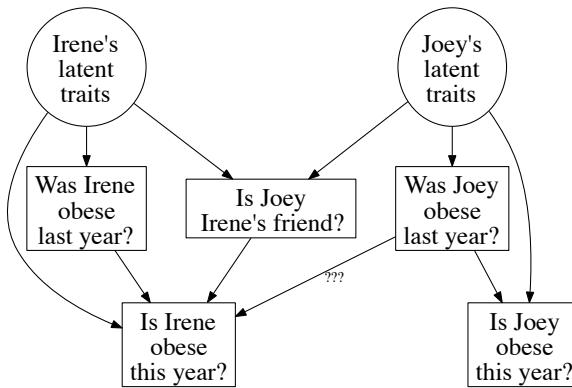


FIGURE 24.10: *Social influence is confounded with selecting friends with similar traits, unobserved in the data.*

24.3.4 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 24.4. 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 24.10. 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 24.10). 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.

24.4 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.

24.4.1 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).

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

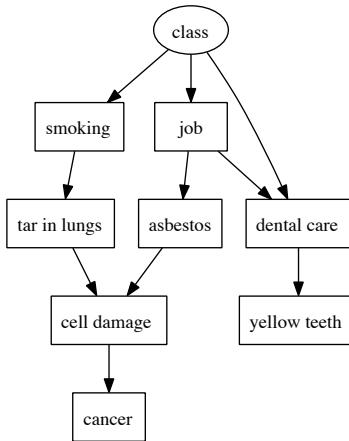


FIGURE 24.11: DAG for Exercise 3.

24.5 Exercises

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. 24.3 from the causal Markov property to the appropriate surgically-altered graph.
3. Refer to Figure 24.11. 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 25

Estimating Causal Effects from Observations

Chapter 24 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.

25.1 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,

$$\Pr(Y = y|do(X = x)) = \sum_s \Pr(S = s) \Pr(Y = y|X = x, S = s) \quad (25.1)$$

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 book 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 16), 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

$$\sum_s \hat{\Pr}(S = s) \hat{\Pr}(Y = y|X = x, S = s) \quad (25.2)$$

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.

25.1.1 Estimating Average Causal Effects

Because $\Pr(Y|do(X = x))$ is a probability distribution, we can ask about $E[Y|do(X = x)]$, when it makes sense for Y to have an expectation value; it's just

$$E[Y|do(X = x)] = \sum_y y \Pr(Y = y|do(X = x)) \quad (25.3)$$

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

$$E[Y|do(X = x)] = \sum_y y \Pr(Y = y|do(X = x)) \quad (25.4)$$

$$= \sum_y y \sum_s \Pr(Y = y|X = x, S = s) \Pr(S = s) \quad (25.5)$$

$$= \sum_s \Pr(S = s) \sum_y y \Pr(Y = y|X = x, S = s) \quad (25.6)$$

$$= \sum_s \Pr(S = s) E[Y|X = x, S = s] \quad (25.7)$$

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.

25.1.2 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

$$\Pr(Y = y|do(X = x)) = \sum_s \Pr(Y = y|X = x, S = s) \Pr(S = s) \quad (25.8)$$

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 ,

$$\frac{1}{n} \sum_{i=1}^n f(s_i) \rightarrow \sum_s f(s) \Pr(S = s) \quad (25.9)$$

Therefore, with a large sample,

$$\Pr(Y = y|do(X = x)) \approx \frac{1}{n} \sum_{i=1}^n \Pr(Y = y|X = x, S = s_i) \quad (25.10)$$

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 28 (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.

25.1.3 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 use 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

$$X \perp\!\!\!\perp S|R \quad (25.11)$$

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

$$\sum_r \Pr(Y = y | X = x, R = r) \Pr(R = r) \quad (25.12)$$

$$= \sum_{r,s} \Pr(Y = y, S = s | X = x, R = r) \Pr(R = r)$$

$$= \sum_{r,s} \Pr(Y = y | X = x, R = r, S = s) \Pr(S = s | X = x, R = r) \Pr(R = r) \quad (25.13)$$

$$= \sum_{r,s} \Pr(Y = y | X = x, S = s) \Pr(S = s | X = x, R = r) \Pr(R = r) \quad (25.14)$$

$$= \sum_{r,s} \Pr(Y = y | X = x, S = s) \Pr(S = s | R = r) \Pr(R = r) \quad (25.15)$$

$$= \sum_s \Pr(Y = y | X = x, S = s) \sum_r \Pr(S = s, R = r) \quad (25.16)$$

$$= \sum_s \Pr(Y = y | X = x, S = s) \Pr(S = s) \quad (25.17)$$

$$= \Pr(Y = y | do(X = x)) \quad (25.18)$$

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 about X as the whole of S does. Consider for instance, the set-up where

$$X \leftarrow \sum_{j=1}^p V_j + \epsilon_X \quad (25.19)$$

$$Y \leftarrow f(X, V_1, V_2, \dots, V_p) + \epsilon_Y \quad (25.20)$$

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. 25.13 to Eq. 25.14 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. 25.14 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 outcome. 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.

25.1.4 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?

$$E[Y|do(X = 1)] - E[Y|do(X = 0)] \quad (25.21)$$

$$\begin{aligned} &= \sum_s \Pr(S = s) E[Y|X = 1, S = s] - \sum_s \Pr(S = s) E[Y|X = 0, S = s] \\ &= \sum_s \Pr(S = s) (E[Y|X = 1, S = s] - E[Y|X = 0, S = s]) \end{aligned} \quad (25.22)$$

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 compare the mean Y for those who are treated ($X = 1$) to the mean Y for those

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 those techniques 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"). It appears to be merely coincidence that (iv) many of the statisticians who make such pronouncements work or have worked for the Educational Testing Service, an organization with an interest in asserting that, strictly speaking, sex and race cannot have any *causal* role in the score anyone gets on the SAT. (Points (i)–(iii) follow Glymour (1986).)

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

$$\begin{aligned} & \sum_s \Pr(S = s)(E[Y|X = 1, S = s] - E[Y|X = 0, S = s]) \\ &= \sum_r \Pr(R = r)(E[Y|X = 1, R = r] - E[Y|X = 0, R = r]) \end{aligned} \quad (25.23)$$

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. 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 ideas 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.

25.2 Instrumental-Variables Estimates

§24.3.3 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 24.7, we saw that when everything is linear, we can find the causal coefficient of Y on X as

$$\beta = \frac{\text{Cov}[I, Y]}{\text{Cov}[I, X]} \quad (25.24)$$

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. 25.24 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 §25.2, 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. 24.15,

$$\Pr(Y|do(I = i)) = \sum_x \Pr(Y|do(X = x)) \Pr(X = x|do(I = i))$$

and likewise finding $E[Y|do(X = x)]$ means solving

$$E[Y|do(I = i)] = \sum_x E[Y|do(X = x)] \Pr(X = x|do(I = i)) \quad (25.25)$$

When, as is generally the case, x is continuous, we have rather an integral equation,

$$E[Y|do(I = i)] = \int E[Y|do(X = x)] p(x|do(I = i)) dx \quad (25.26)$$

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).

25.3 Uncertainty and Inference

The point of the identification strategies from Chapter 24 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.

25.4 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.

25.5 Further Reading

The material in §25.1 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 and 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 24.9. — For a rather more convincing effort to apply ideas about causal inference to understanding the changing prevalence of autism, see Liu *et al.* (2010).

25.6 Exercises

1. Prove Eq. 25.23.
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 24.7, prove that the two-stage least-squares estimate of β is the same as the Wald estimate.

Chapter 26

Discovering Causal Structure from Observations

[[TODO: Style: some redundancy with respect to “how many ways of writing an open triple are there”, and “you can use any CI test you like, honest” — fragments of an incomplete 2012 re-write. Smooth out]]

[[TODO: Mention algorithms for hidden variables (FCI, RFCI)]]

[[TODO: Further examples]]

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

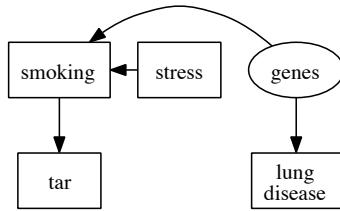


FIGURE 26.1: A hypothetical causal model in which smoking is associated with lung disease, but does not cause it. Rather, both smoking and lung disease are caused by common genetic variants. (This idea was due to R. A. Fisher.) Smoking is also caused, in this model, by stress.

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.

26.1 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 26.1, $\text{lungdisease} \perp\!\!\!\perp \text{tar} | \text{smoking}$. 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.

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

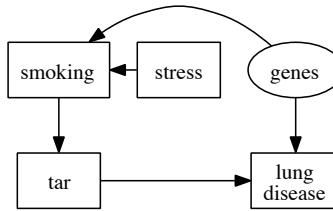


FIGURE 26.2: As in Figure 26.1, but now tar in the lungs does cause lung disease.

checking predictions which *differ* between them. So in both Figure 26.1 and in Figure 26.2, $\text{stress} \perp\!\!\!\perp \text{tar} | \text{smoking}$. 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 26.1, 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.

26.2 Testing Conditional Independence

Recall from §23.4 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 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 $\Pr(Y|X,Z) =$

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.

$\Pr(Y|Z)$. 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 16: 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).

26.3 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 §23.2, 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:

$$X \rightarrow Y \Rightarrow X \not\perp\!\!\!\perp Y \quad (26.1)$$

$$X \leftarrow Y \Rightarrow X \not\perp\!\!\!\perp Y \quad (26.2)$$

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:

$$X \rightarrow Y \rightarrow Z \quad (26.3)$$

$$X \leftarrow Y \leftarrow Z \quad (26.4)$$

$$X \leftarrow Y \rightarrow Z \quad (26.5)$$

$$X \rightarrow Y \leftarrow Z \quad (26.6)$$

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.

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.

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.

26.3.1 Partial Identification of Effects

Chapters 24–25 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.

26.4 Causal Discovery with Known Variables

Section 26.1 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.

Causal discovery is silly with just one variable, and too hard for us with just two.⁵

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 .

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.

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

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." 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.

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 §26.9.

Call the result of the SGS algorithm \hat{G} . If all of the assumptions above hold, and the algorithm is correct in its guesses about when variables are conditionally independent, then $\hat{G} = G$. In practice, of course, conditional independence guesses are really statistical tests based on finite data, so we should write the output as \hat{G}_n , to indicate that it is based on only n samples. If the conditional independence test is consistent, then

$$\lim_{n \rightarrow \infty} \Pr(\hat{G}_n \neq G) = 0 \quad (26.7)$$

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.

26.4.1 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.
- ...
- For each X and Y which are still connected, see if $X \perp\!\!\!\perp Y|$ all the $p - 2$ other variables; 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.

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 §26.9). 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.

26.4.2 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.

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.

26.4.3 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 they are continuous, Gaussian, and linearly related (so that one can test for vanishing partial correlations), though the `pcalg` package does allow users to provide their

¹⁰Peter-Clark

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.

[[TODO: Add references to the recent papers on things like kernel conditional independent tests and kernel distance covariance]]

26.5 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 package 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.

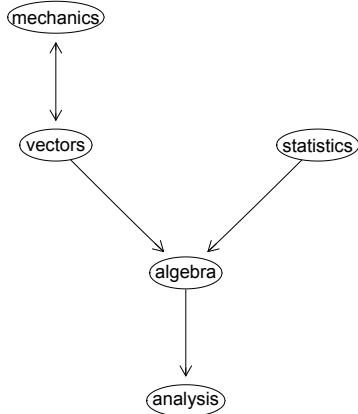
```
library(pca)
library(SMPRACTICALS)
data(mathmarks)
suffStat <- list(C=cor(mathmarks), n=nrow(mathmarks))
pc.fit <- pc(suffStat, indepTest=gaussCItest, p=ncol(mathmarks), alpha=0.005)
```

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 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%).

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).

Inferred DAG for mathmarks



```
library(Rgraphviz)
plot(pc.fit,labels=colnames(mathmarks),main="Inferred DAG for mathmarks")
```

FIGURE 26.3: 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.)

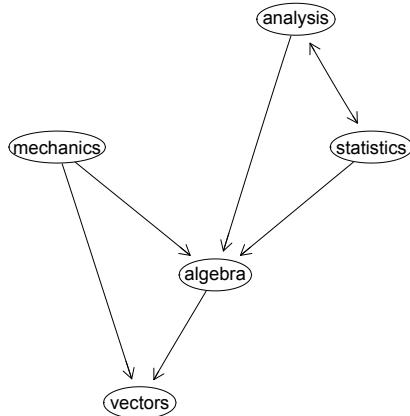
Before going on, I encourage you to run `pc` as above, but with `verbose=TRUE`, and to study the output.

Figure 26.3 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,
  NAdelete = NAdelete, m.max = m.max)

Nmb. edgetests during skeleton estimation:
```



```
plot(pc(suffStat, indepTest=gaussCITest, p=ncol(mathmarks), alpha=0.05),
     labels=colnames(mathmarks), main="")
```

FIGURE 26.4: *Inferred DAG when the size of the test is 0.05.*

```
=====
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 26.4).

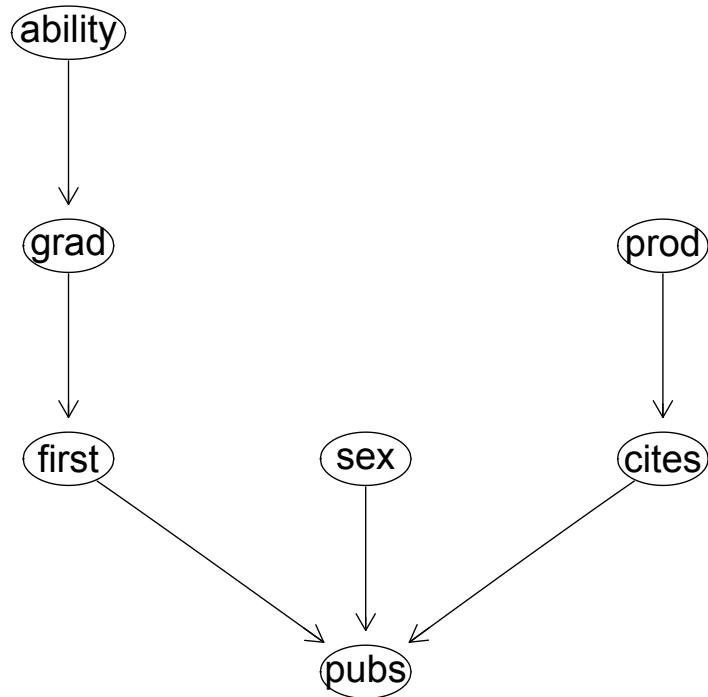
For a second example¹³, let's use some data on academic productivity among psychologists. The two variables of ultimate interest were the publication (pubs) and citation (cites) rates, with possible measured causes including ability (basically, standardized test scores), graduate program quality grad (basically, the program's national rank), the quality of the psychologist's first job, first, a measure of productiv-

¹³Following Spirtes *et al.* (2001, §5.8.1, pp. 98–102).

ity 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 `pcalg` is fairly reasonable (Figure 26.5). 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 26.5: *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.*

26.6 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

$$\lim_{n \rightarrow \infty} \Pr(\hat{G}_n \neq G) = 0 \quad (26.8)$$

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. 26.8 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.

26.7 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 is less well-developed. Important papers on this topic include Richardson (1996) and Lacerda *et al.* (2008).

26.8 Exercises

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`.

26.9 Pseudo-code for the SGS and PC Algorithms

[[Consider whether this is really needed]]

This section may be omitted on first (and maybe even second) reading.

When you see a loop, assume that it gets entered at least once. “Replace” in the sub-functions always refers to the input graph.

26.9.1 The SGS Algorithm

```

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)
}

```

{

26.9.2 The PC Algorithm

[[To come]]

Chapter 27

Experimental Design

[[Chapter to come: basic ideas of experimental design; randomization; blocking; factorial designs; partial factorial designs; optimal designs; within-subject designs including cross-over]]

Part IV

Dependent Data

Chapter 28

Time Series

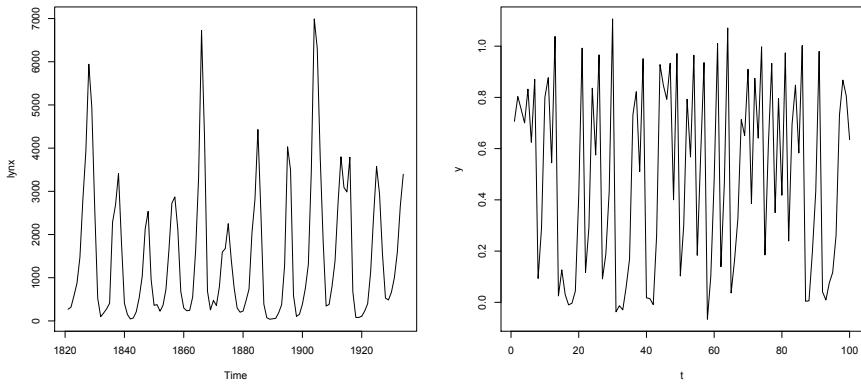
So far, we have assumed that all data points are pretty much independent of each other. In the chapters on regression, we assumed that each Y_i was independent of every other, given its X_i , and we often assumed that the X_i were themselves independent. In the chapters on multivariate distributions 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.

28.1 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 28.1 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.

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. Continuously-



```
data(lynx); plot(lynx)
```

FIGURE 28.1: *Left: annual number of trapped lynxes in the Mackenzie River region of Canada. Right: a toy dynamical model. (See code online for the toy.)*

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.

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.

28.2 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 $\mathbf{E}[X_1] = \mathbf{E}[X_t]$, 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.

28.2.1 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**,

$$\gamma(h) = \text{Cov}[X_t, X_{t+h}] \quad (28.1)$$

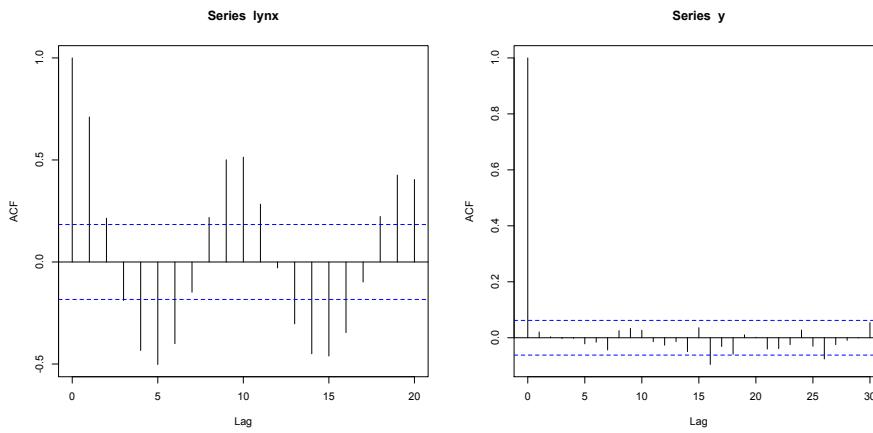
which is well-defined just when the process is weakly stationary. We could equally well use the **autocorrelation**,

$$\rho(h) = \frac{\text{Cov}[X_t, X_{t+h}]}{\text{Var}[X_t]} = \frac{\gamma(h)}{\gamma(0)} \quad (28.2)$$

again using stationarity to simplify the denominator.

As I said, for most time series $\gamma(h) \rightarrow 0$ as h grows. Of course, $\gamma(h)$ could be exactly zero while X_t and X_{t+h} are strongly dependent. Figure 28.2 shows the autocorrelation 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 28.3). More general measures of dependence would include looking at the Spearman rank-correlation of X_t and X_{t+h} , 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 28.2: 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.

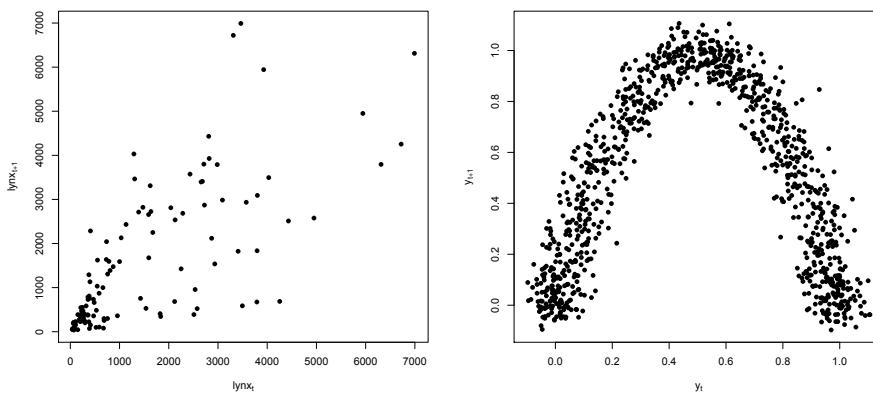


FIGURE 28.3: Plots of X_{t+1} versus X_t , for the lynx (left) and the simulation (right). (See code online.) Note that even though the correlation between successive iterates is next to zero for the simulation, there is clearly a lot of dependence.

28.2.2 The Ergodic Theorem

With IID data, the ultimate basis of all our statistical inference is the law of large numbers, which told us that

$$\frac{1}{n} \sum_{i=1}^n X_i \rightarrow E[X_1] \quad (28.3)$$

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.

28.2.2.1 The World's Simplest Ergodic Theorem

Suppose X_t is weakly stationary, and that

$$\sum_{b=0}^{\infty} |\gamma(b)| = \gamma(0)\tau < \infty \quad (28.4)$$

(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,

$$\bar{X}_n = \frac{1}{n} \sum_{t=1}^n X_t \quad (28.5)$$

This **time average** is a random variable. Its expectation value is

$$E[\bar{X}_n] = \frac{1}{n} \sum_{t=1}^n E[X_t] = E[X_1] \quad (28.6)$$

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?

$$\text{Var} [\bar{X}_n] = \text{Var} \left[\frac{1}{n} \sum_{t=1}^n X_t \right] \quad (28.7)$$

$$= \frac{1}{n^2} \left[\sum_{t=1}^n \text{Var} [X_t] + 2 \sum_{t=1}^n \sum_{s=t+1}^n \text{Cov} [X_t, X_s] \right] \quad (28.8)$$

$$= \frac{1}{n^2} \left[n \text{Var} [X_1] + 2 \sum_{t=1}^n \sum_{s=t+1}^n \gamma(s-t) \right] \quad (28.9)$$

$$\leq \frac{1}{n^2} \left[n\gamma(0) + 2 \sum_{t=1}^n \sum_{s=t+1}^n |\gamma(s-t)| \right] \quad (28.10)$$

$$\leq \frac{1}{n^2} \left[n\gamma(0) + 2 \sum_{t=1}^n \sum_{b=1}^n |\gamma(b)| \right] \quad (28.11)$$

$$\leq \frac{1}{n^2} \left[n\gamma(0) + 2 \sum_{t=1}^n \sum_{b=1}^{\infty} |\gamma(b)| \right] \quad (28.12)$$

$$= \frac{n\gamma(0)(1+2\tau)}{n^2} \quad (28.13)$$

$$= \frac{\gamma(0)(1+2\tau)}{n} \quad (28.14)$$

Eq. 28.9 uses stationarity again, and then Eq. 28.13 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 ,

$$\Pr(|Z - E[Z]| > \epsilon) \leq \frac{\text{Var}[Z]}{\epsilon^2} \quad (28.15)$$

so

$$\Pr(|\bar{X}_n - E[X_1]| > \epsilon) \leq \frac{\gamma(0)(1+2\tau)}{n\epsilon^2} \quad (28.16)$$

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.

28.2.2.2 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.

28.2.2.3 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

$$\frac{1}{n} \sum_{t=1}^n f(X_t^{t+k-1}) \rightarrow E[f(X_1^k)] \quad (28.17)$$

for any reasonable function f . This is (again very roughly) equivalent to

$$\frac{1}{n} \sum_{t=1}^n \Pr(X_1^k \in A, X_t^{t+k-1} \in B) \rightarrow \Pr(X_1^k \in A) \Pr(X_1^l \in B) \quad (28.18)$$

which is a kind of asymptotic independence-on-average³

If our data source is ergodic, then what Eq. 28.17 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 a 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 $n^{-1} \sum_{t=0}^{n-1} f(T^t Y) \rightarrow E[f(Y)]$, with probability 1. Since a function of the trajectory can be a function of a block of length k , we get Eq. 28.17.

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 21. (This is called the "ergodic

28.3 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

$$\frac{1}{n} \sum_{t=1}^n \mathbf{1}_A(X_t) \rightarrow \Pr(X_1 \in A) \quad (28.19)$$

Since this also applies to functions of blocks,

$$\frac{1}{n} \sum_{t=1}^n \mathbf{1}_{A,B}(X_t, X_{t+1}) \rightarrow \Pr(X_1 \in A, X_2 \in B) \quad (28.20)$$

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 16.

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:

$$X_{t+1} \perp\!\!\!\perp X_1^{t-1} | X_t \quad (28.21)$$

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. 28.17. 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,

$$X_{t+1} \perp\!\!\!\perp X_1^{t-k} | X_{t-k+1}^t \quad (28.22)$$

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.

28.3.1 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.

[[TODO: Maximum likelihood for Markov models]]

[[TODO: Variable length Markov chains]]

t	x	lag0	lag1	lag2	lag3
1821	269				
1822	321	871	585	321	269
1823	585	1475	871	585	321
1824	871	2821	1475	871	585
1825	1475	\Rightarrow	3928	2821	1475
1826	2821	5943	3928	2821	1475
1827	3928	4950	5943	3928	2821
1828	5943	...			
1829	4950				
...					

FIGURE 28.4: 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 40: Example code for turning a time series into a design matrix, suitable for regression.

28.4 Autoregressive Models

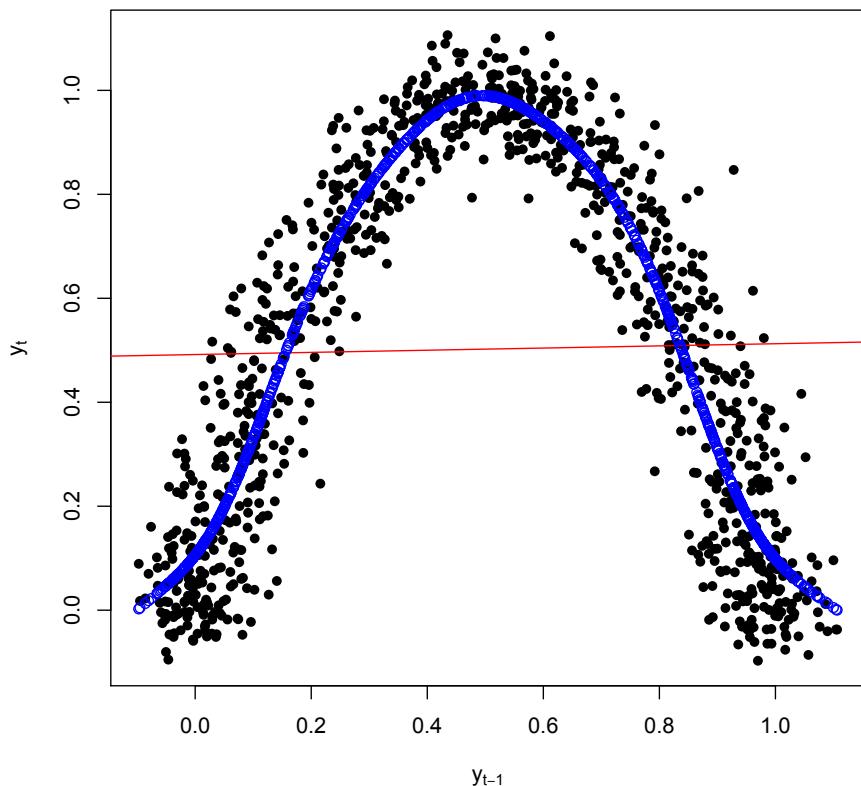
Instead of trying to estimate the whole conditional distribution of X_t , we can just look at its conditional expectation. This is a regression problem, but since we are regressing X_t on earlier values of the series, it's called an **autoregression**:

$$\mathbb{E} \left[X_t | X_{t-p}^{t-1} = x_1^p \right] = r(x_1^p) \quad (28.23)$$

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 28.4); see Code Example 40.

Suppose $p = 1$. Then we essentially want to draw regression curves through plots like those in Figure 28.3. Figure 28.5 shows an example for the artificial series.



```

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 28.5: 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).

28.4.1 Autoregressions with Covariates

Nothing keeps us from adding a variable other than the past of X_t to the regression:

$$\mathbb{E}[X_{t+1}|X_{t-k+1}^t, Z] \quad (28.24)$$

or even another time series:

$$\mathbb{E}[X_{t+1}|X_{t-k+1}^t, Z_{t-l+1}^t] \quad (28.25)$$

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.

28.4.2 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:

$$\mathbb{E}[X_t|X_{t-p}^{t-1}] \approx \alpha_0 + \sum_{j=1}^p g_j(X_{t-j}) \quad (28.26)$$

Example: The lynx Let's try fitting an additive model for the lynx. Code Example 41 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)
```

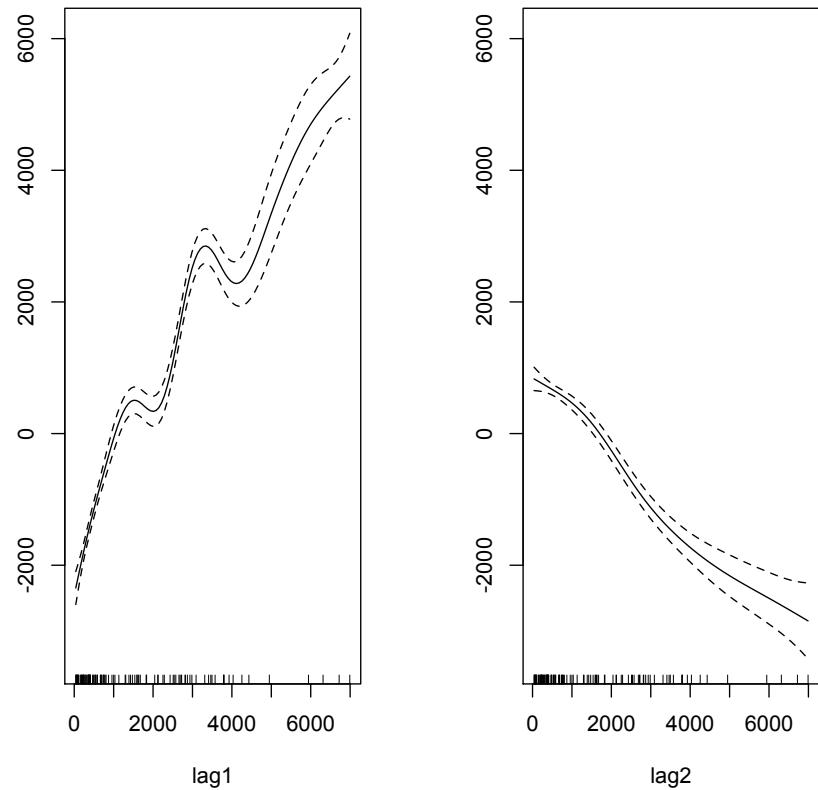
This inherits everything we can do with a GAM, so we can do things like plot the partial response functions (Figure 28.6), plot the fitted values against the actual (Figure 28.7), etc. To get a sense of how well it can actually extrapolate, Figure 28.8 re-fits the model to just the first 80 data points, and then predicts the remaining 34.

28.4.3 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.

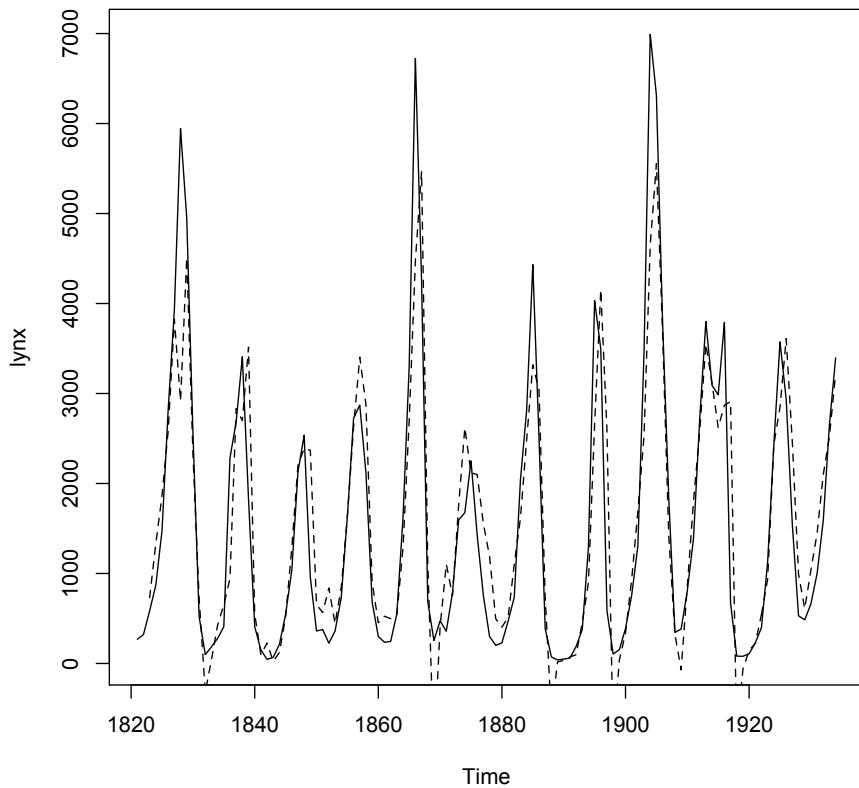
The analysis we did in Chapter 2 of how to find the optimal linear predictor carries over with no change whatsoever. If we want to predict X_t as a linear combination of the last k observations, $X_{t-1}, X_{t-2}, \dots, X_{t-p}$, then the ideal coefficients β are

$$\beta = (\text{Var}[X_{t-p}^{t-1}])^{-1} \text{Cov}[X_{t-p}^{t-1}, X_t] \quad (28.27)$$



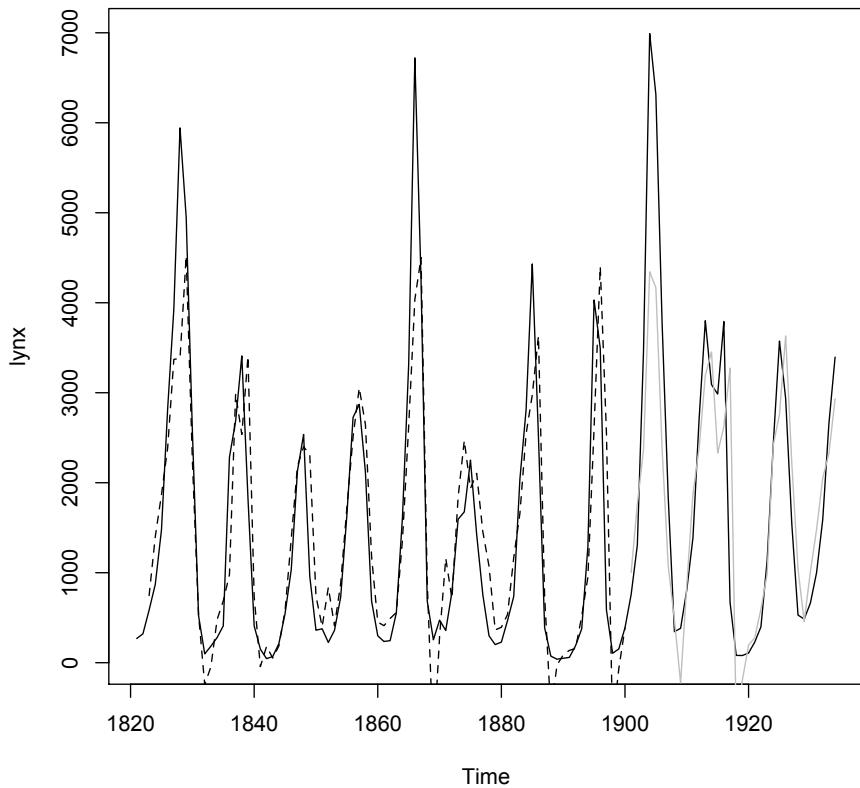
```
plot(lynx.aar2, pages=1)
```

FIGURE 28.6: 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 28.7: 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 28.8: *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 41: *Fitting an additive autoregression of arbitrary order to a time series. See online for comments.*

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 28.9 is pretty typical.

28.4.3.1 “Unit Roots” and Stationary Solutions

Suppose we really believed a first-order linear autoregression,

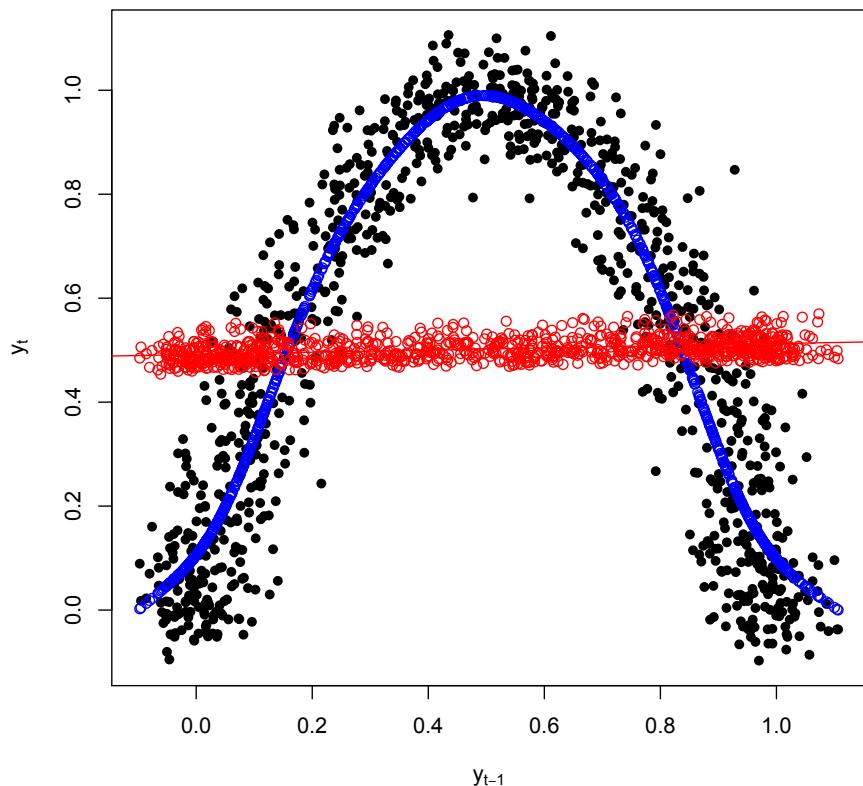
$$X_{t+1} = \alpha + \beta X_t + \epsilon_t \quad (28.28)$$

with ϵ_t some IID noise sequence. Let’s suppose that the mean is zero for simplicity, so $\alpha = 0$. Then

$$X_{t+2} = \beta^2 X_t + \beta \epsilon_t + \epsilon_{t+1} \quad (28.29)$$

$$X_{t+3} = \beta^3 X_t + \beta^2 \epsilon_t + \beta \epsilon_{t+1} + \epsilon_{t+2}, \quad (28.30)$$

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 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 $|\beta| < 1$.



```
library(tseries)
yar8 <- arma(y,order=c(8,0))
points(y[-length(y)],fitted(yar8)[-1],col="red")
```

FIGURE 28.9: Adding the predictions of an eighth-order linear AR model (red dots) to Figure 28.5. 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.

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

$$\sum_{j=1}^p \beta_j z^j - 1 \quad (28.31)$$

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.

28.4.4 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.

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 28.10 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`.

28.4.5 Regression with Correlated Noise; Generalized Least Squares

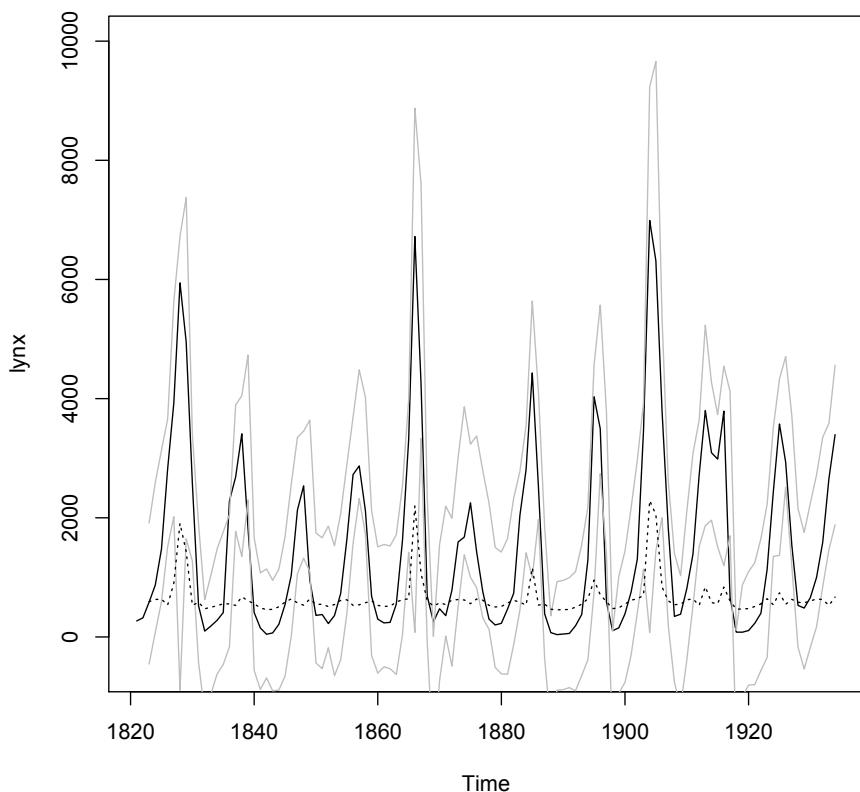
Suppose we have an old-fashioned regression problem

$$Y_t = r(X_t) + \epsilon_t \quad (28.32)$$

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.)

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 28.10: *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.*

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 $\gamma(h)$ 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 H) tells us that the most efficient estimate is the solution to the **generalized least squares** problem,

$$\hat{m}_{GLS} = \underset{m}{\operatorname{argmin}} \frac{1}{n} (\mathbf{y} - \mathbf{m}(\mathbf{x}))^T \Gamma^{-1} (\mathbf{y} - \mathbf{m}(\mathbf{x})) \quad (28.33)$$

as opposed to just minimizing the mean-squared error,

$$\hat{m}_{OLS} = \underset{m}{\operatorname{argmin}} \frac{1}{n} (\mathbf{y} - \mathbf{m}(\mathbf{x}))^T (\mathbf{y} - \mathbf{m}(\mathbf{x})) \quad (28.34)$$

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 $\gamma(h)$ 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.

28.5 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.

28.5.1 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.

28.5.2 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	269
1822	321	321	321
1823	585	585	585
1824	871	871	871
	\Rightarrow	585	871
1825	1475	871	1475
1826	2821	1475	2821
1827	3928	2821	3928
1828	5943	3928	5943
		1828	5943
\Rightarrow			
lag2	lag1	lag0	
269	321	585	
871	1475	2821	
585	871	1475	
			\Rightarrow
1821	269	1824	871
1822	321	1825	1475
1823	585	1826	2821
1827	585	1827	585
1828	871	1828	871

FIGURE 28.11: 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.

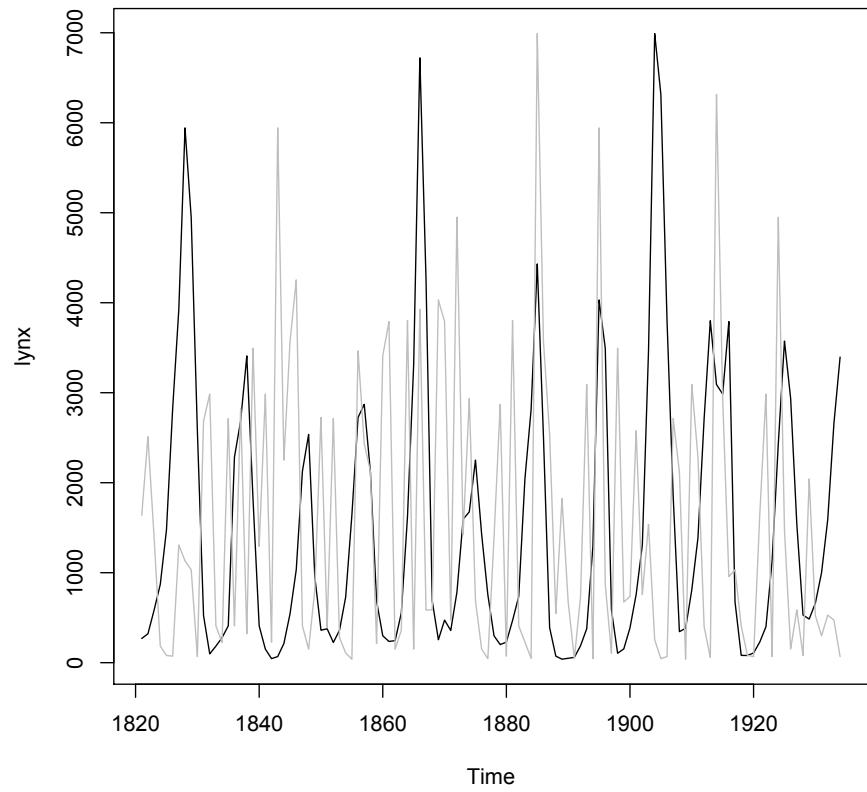
sub-blocks within each block with replacement. This gives a somewhat better idea of longer-range dependence, though we have to pick two block-lengths.

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.

28.5.3 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. We then (slowly) let them get more complicated as we get more data¹⁰. One popular

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.



```
plot(lynx)
lines(1821:1934, rblockboot(lynx,4), col="grey")
```

FIGURE 28.12: *The lynx time series, and one run of resampling it with a block bootstrap, block length = 4. (See online for the code to rblockboot.)*

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.

28.6 Trends and De-Trending

The sad fact is that a lot of important real time series are not even approximately stationary. For instance, Figure 28.13 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,

$$\begin{aligned} Y_t &= X_t + Z_t \\ \text{series} &= \text{fluctuations + trend} \end{aligned} \quad (28.35)$$

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:

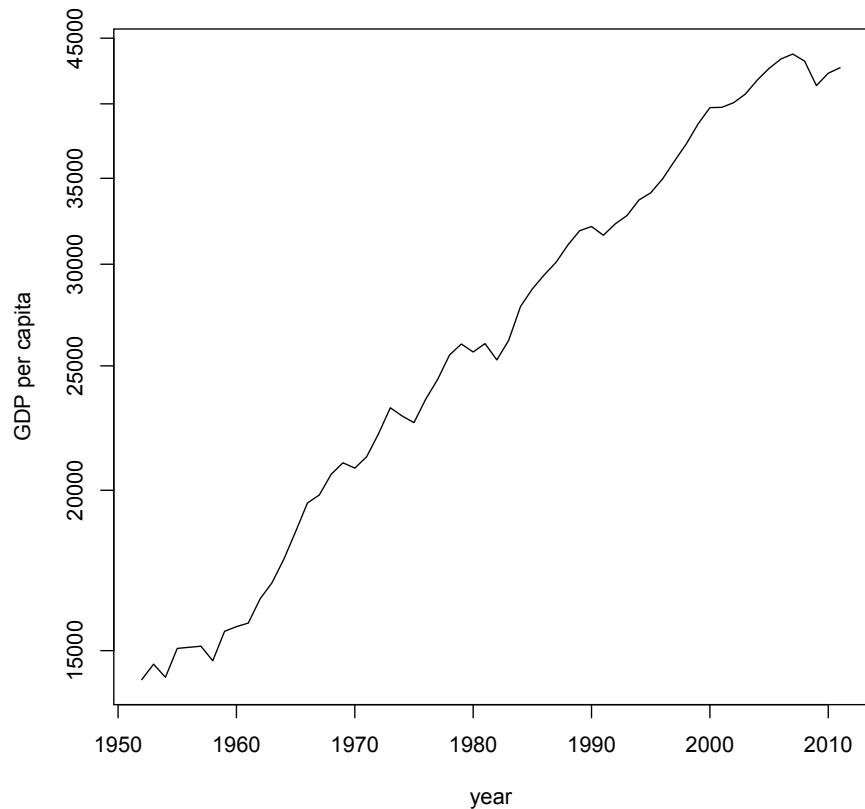
$$Z_t = E[Y_{i,t}] \approx \sum_{i=1}^m Y_{i,t} \quad (28.36)$$

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, some (simple) models of economic growth predict that series like the one in Figure 28.13 should, on average, grow at a steady exponential rate¹². We could then estimate

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.

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.



```
gdppc <- read.csv("gdp-pc.csv")
gdppc$y <- gdppc$y*1e6
plot(gdppc,log="y",type="l",ylab="GDP per capita")
```

FIGURE 28.13: *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.)*

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.

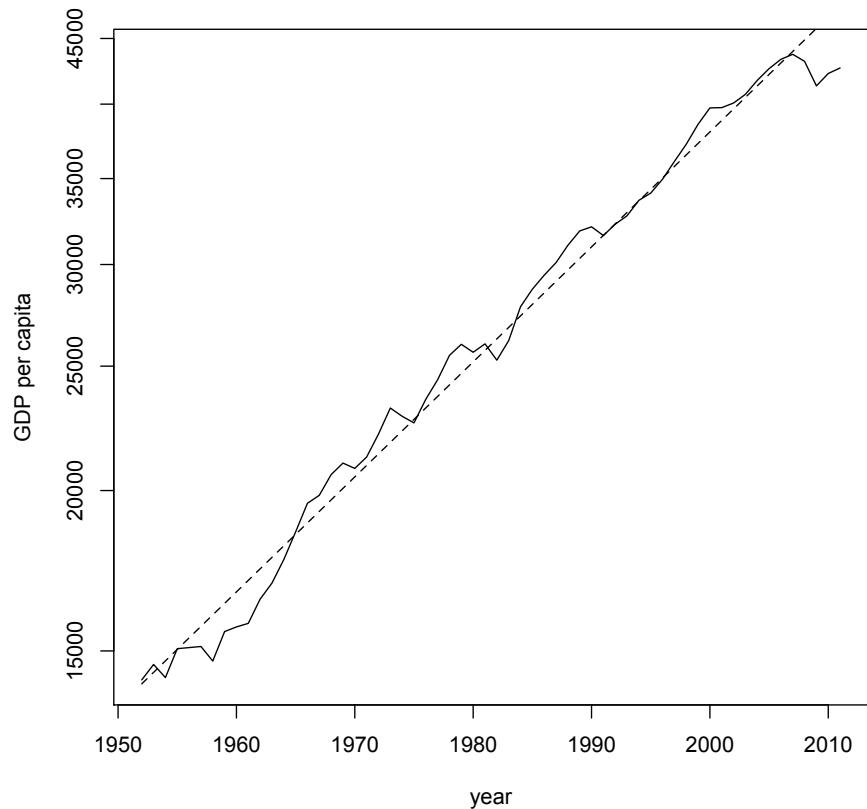
28.6.1 Forecasting Trends

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 28.16. 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. 171). 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

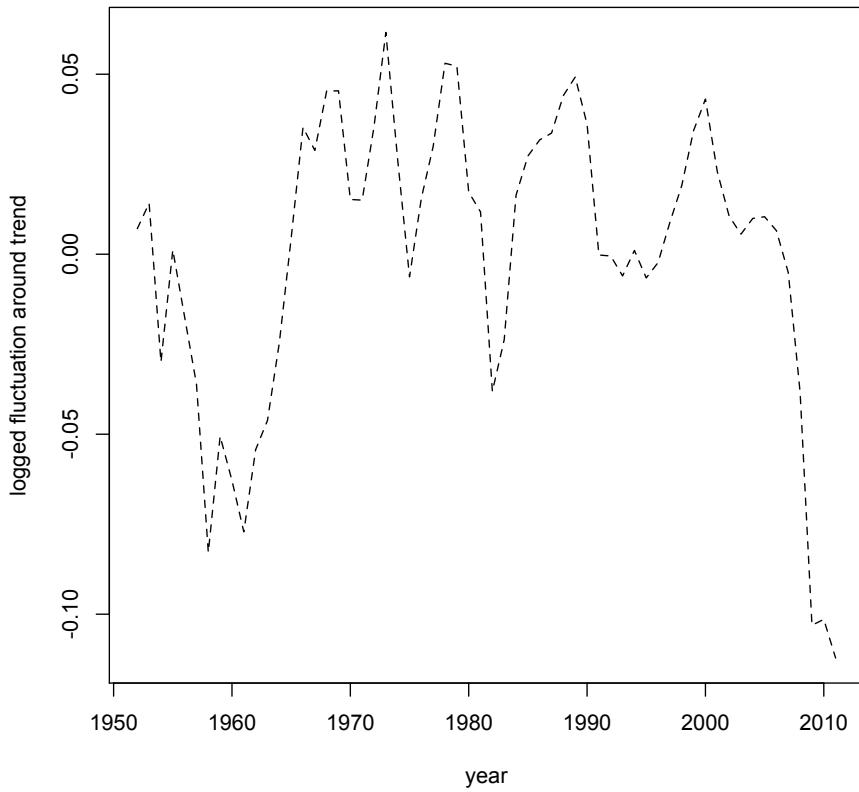
¹³I am being deliberately vague about what "the characteristic time scale of Z_t " means. Intuitively, it's the amount of time required for Z_t to change substantially. You might think of it as something like $n^{-1} \sum_{t=1}^{n-1} 1/|Z_{t+1} - Z_t|$, if you promise not to treat that too seriously. 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 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.



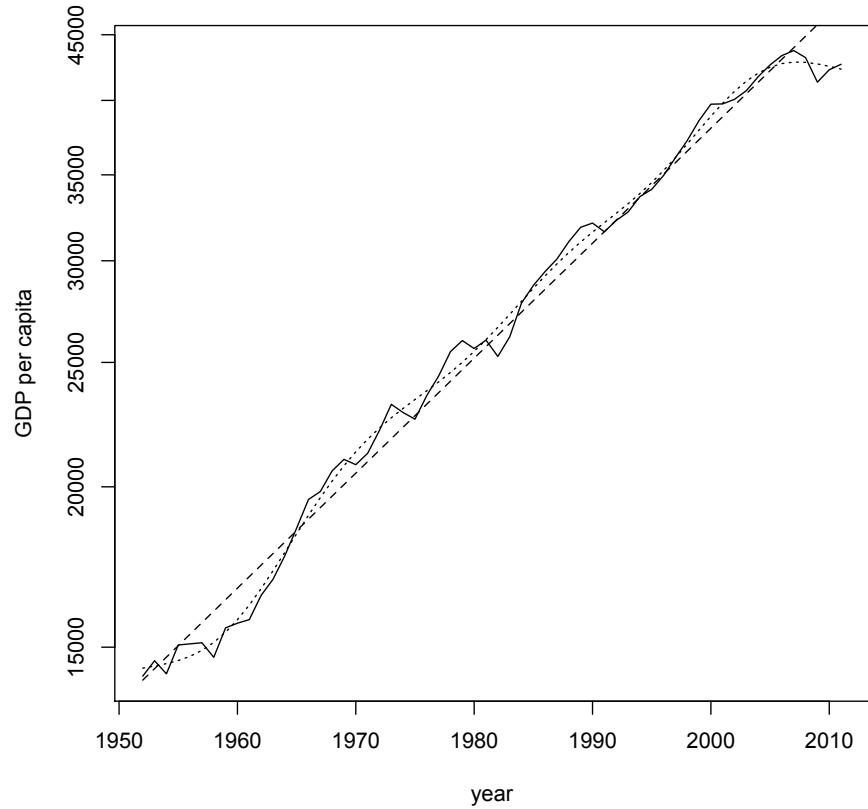
```
gdppc.exp <- lm(log(y) ~ year,data=gdppc)
beta0 <- exp(coefficients(gdppc.exp)[1])
beta <- coefficients(gdppc.exp)[2]
curve(beta0*exp(beta*x),lty="dashed",add=TRUE)
```

FIGURE 28.14: As in Figure 28.13, but with an exponential trend fitted.



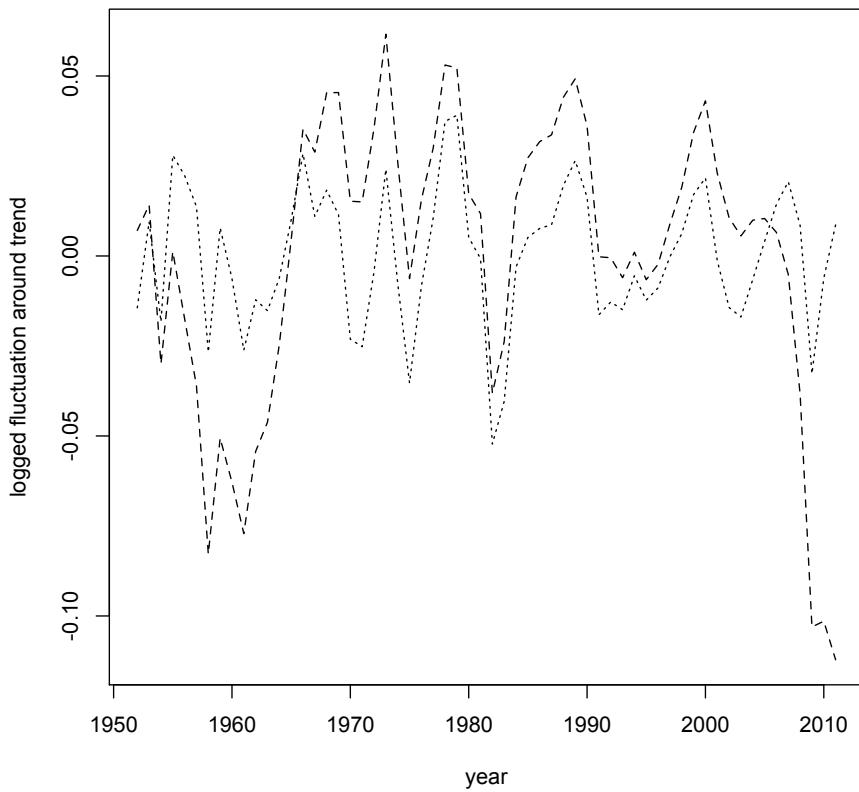
```
plot(gdppc$year,residuals(gdppc.exp),xlab="year",
     ylab="logged fluctuation around trend",type="l",lty="dashed")
```

FIGURE 28.15: *The hopefully-stationary fluctuations around the exponential growth trend in Figure 28.14. Note that these are $\log \frac{Y_t}{\beta_0 e^{\beta t}}$, and so unitless.*



```
gdp.spline <- fitted(gam(y~s(year), data=gdppc))
lines(gdppc$year, gdp.spline, lty="dotted")
```

FIGURE 28.16: Figure 28.14, 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 28.17: Adding the logged deviations from the spline trend (dotted) to Figure 28.15.

28.6.2 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

$$Y_t = X_t + Z_t + S_t \quad (28.37)$$

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

$$S_t \approx \frac{1}{m} \sum_{j=0}^{m-1} Y_{t+jT} \quad (28.38)$$

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.

28.6.3 Detrending by Differencing

Suppose that Y_t has a linear time trend:

$$Y_t = \beta_0 + \beta t + X_t \quad (28.39)$$

with X_t stationary. Then if we take the difference between successive values of Y_t , the trend goes away:

$$Y_t - Y_{t-1} = \beta + X_t - X_{t-1} \quad (28.40)$$

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

$$Z_t = Z_{t-1} + \epsilon_t \quad (28.41)$$

where the “innovations” or “shocks” ϵ_t are IID, and that

$$Y_t = Z_t + X_t \quad (28.42)$$

¹⁵Exercise: come up with an example of a time series where the periodicity *should* be 355 days.

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

$$Y_t - Y_{t-1} = \epsilon_t + X_t - X_{t-1} \quad (28.43)$$

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 28.18.) 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.

28.6.4 Bootstrapping with Trends

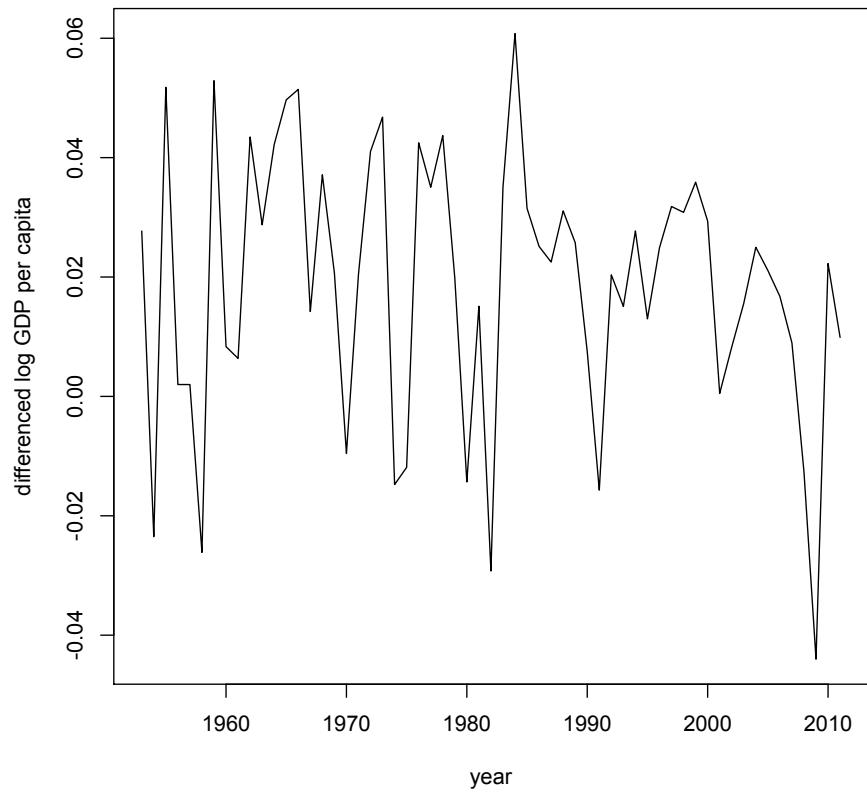
All the bootstraps discussed in §28.5 work primarily for stationary time series. (Parametric bootstraps are an exception, since we *could* include trends in the model.) If we have done extensive de-trending, the reasonable thing to do is to use a bootstrap to generate a series of fluctuations, add it to the estimated trend, and then repeat the *whole* analysis on the new, non-stationary surrogate series, including the de-trending. This works on the same sort of principle as resampling residuals in regressions (§6.4, especially 6.4.3).

28.7 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



```
plot(gdppc$year[-1],diff(log(gdppc$y)),type="l",xlab="year",
     ylab="differenced log GDP per capita")
```

FIGURE 28.18: *First differences of log GDP per capita, i.e., the year-to-year growth rate of GDP per capita.*

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 Guttorm (1995).

The basic ergodic theorem in §28.2.2.1 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), 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*.

28.8 Exercises

1. In Eq. 28.33, assume that $m(x)$ has to be a linear function, $m(x) = \beta \cdot x$. Solve for the optimal β in terms of y , x , and Γ . This “generalized least squares” (GLS) solution should reduce to ordinary least squares when $\Gamma = \sigma^2 I$.
2. If $Z_t = Z_{t-1} + \epsilon_t$, with ϵ_t IID, prove that Z_t is not stationary. *Hint:* consider $\text{Var}[Z_t]$.

Chapter 29

Time Series with Latent Variables

[[To come]]

Chapter 30

Simulation-Based Inference

30.1 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**.

[[TODO: Intro/transition]]
 [[ATTN: Is this chapter excessive? Online supplement?]]

[[TODO: Check cross-refs as text yanked from Chapter 5.]]

30.1.1 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,

$$m = g(\theta) \tag{30.1}$$

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

$$\theta = g^{-1}(m) \tag{30.2}$$

The method of moments *estimator* takes the observed, sample moments \hat{m} , and plugs them into Eq. 30.2:

$$\widehat{\theta}_{MM} = g^{-1}(\hat{m}) \tag{30.3}$$

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:

$$\widehat{\theta}_{MM} = \operatorname{argmin}_{\theta} \|g(\theta) - \hat{m}\|^2 \tag{30.4}$$

For the minimization version, we just have to calculate moments from parameters $g(\theta)$, not vice versa. To see that Eqs. 30.3 and 30.4 do the same thing, notice that

(i) the squared¹ 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**.

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

$$(g(\theta) - \hat{m}) \cdot \mathbf{w} (g(\theta) - \hat{m}) \quad (30.5)$$

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.

30.1.2 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. 30.4, the estimator would be

$$\widehat{\theta}_{SMM} = \operatorname{argmin}_{\theta} \left\| \tilde{g}_{s,T}(\theta) - \hat{m} \right\|^2 \quad (30.6)$$

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 .²

30.1.3 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.

¹Why squared? Basically because it makes the function we're minimizing smoother, and the optimization nicer.

²A 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.

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))
```

[[TODO: Update data set, include dividend-payment adjustments]]

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:

$$X_t = Z_t + \theta Z_{t-1} \quad (30.7)$$

$$Z_t \sim \mathcal{N}(0, \sigma^2) \text{ i.i.d.} \quad (30.8)$$

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 information³. 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 reasons⁴, 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 $\text{Cov}[X_t, X_{t-1}]$.

$$\text{Var}[X_t] = \text{Var}[Z_t] + \theta^2 \text{Var}[Z_{t-1}] \quad (30.9)$$

$$= \sigma^2 + \theta^2 \sigma^2 \quad (30.10)$$

$$= \sigma^2(1 + \theta^2) \equiv v(\theta, \sigma) \quad (30.11)$$

³Nobody will ever say "What? It's snowing in Pittsburgh in February? Call my broker!"

⁴Think 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).

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

$$\text{Cov}[X_t, X_{t-1}] = \mathbb{E}[(Z_t + \theta Z_{t-1})(Z_{t-1} + \theta Z_{t-2})] \quad (30.12)$$

$$= \theta \mathbb{E}[Z_{t-1}^2] \quad (30.13)$$

$$= \theta \sigma^2 \equiv c(\theta, \sigma) \quad (30.14)$$

We can solve the system of equations for the parameters, starting with eliminating σ^2 :

$$\frac{c(\theta, \sigma)}{v(\theta, \sigma)} = \frac{\sigma^2 \theta}{\sigma^2(1 + \theta^2)} \quad (30.15)$$

$$= \frac{\theta}{1 + \theta^2} \quad (30.16)$$

$$0 = \theta^2 \frac{c}{v} - \theta + \frac{c}{v} \quad (30.17)$$

This is a quadratic in θ ,

$$\theta = \frac{1 \pm \sqrt{1 - 4 \frac{c^2}{v^2}}}{2c/v} \quad (30.18)$$

and it's easy to confirm⁵ that this has only one solution in the meaningful range, $-1 \leq \theta \leq 1$. Having found θ , we solve for σ^2 ,

$$\sigma^2 = c/\theta \quad (30.19)$$

The method of moments estimator takes the sample values of these moments, \hat{v} and \hat{c} , and plugs them in to Eqs. 30.18 and 30.19. 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.⁶

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

$$\begin{bmatrix} \hat{\theta}_{MM} \\ \hat{\sigma^2}_{MM} \end{bmatrix} = \underset{\theta, \sigma^2}{\operatorname{argmin}} \left\| \begin{bmatrix} \sigma^2 \theta - \hat{c} \\ \sigma^2(1 + \theta^2) - \hat{v} \end{bmatrix} \right\|^2 \quad (30.20)$$

Computationally, it would go something like Code Example 42.

⁵For example, plot c/v as a function of θ , and observe that any horizontal line cuts the graph at only one point.

⁶It 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 42: *Code for implementing method of moments estimation of a first-order moving average model, as in Eq. 30.20. See Appendix 30.3 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)
}

```

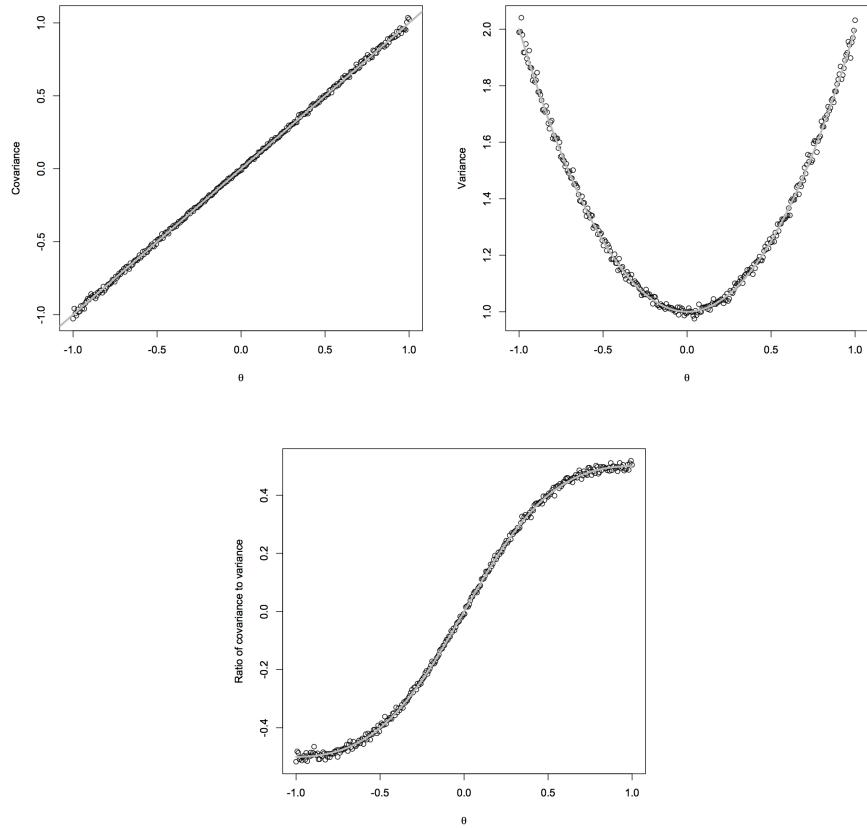
CODE EXAMPLE 43: *Function which simulates s independent runs of a first-order moving average model, each of length n, with given noise variance sigma2 and after-effect theta. See online for the version with comments on the code details.*

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 43 shows the solution.

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 44).

Figure 30.1 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



```

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="Varianc")
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 varianc")
curve(x/(1+x^2),col="grey",lwd=3,add=TRUE)

```

FIGURE 30.1: 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.

```

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 44: Functions for calculating the variance and covariance for specified parameter values from simulations.

ones.⁷ The agreement is quite good, though of course not quite perfect.⁸

Conceptually, we could estimate θ by just taking the observed value \hat{c}/\hat{v} , running a horizontal line across Figure 30.1c, 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 45. The code is practically identical to that in Code Example 42, 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.

30.2 Exercises

Section 30.1 explained the method of simulated moments, where we try to match expectations of various functions of the data. Expectations of functions are summary

[[TODO: Create proper section on II]]

⁷I 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.

⁸If 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.

```

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

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

$$\hat{\theta}_{II} = \underset{\theta}{\operatorname{argmin}} \|\tilde{\beta}(\theta) - \hat{\beta}\|^2 \quad (30.21)$$

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

$$\hat{\beta} \rightarrow \beta \quad (30.22)$$

$$\tilde{\beta}(\theta) \rightarrow b(\theta) \quad (30.23)$$

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

$$\hat{\theta}_{II} \rightarrow \theta \quad (30.24)$$

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

$$X_t = \beta_0 + \beta_1 X_{t-1} + \epsilon_t \quad (30.25)$$

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 30.1.)
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?

30.3 Appendix: Some Design Notes on the Method of Moments Code

Go back to Section 30.1.3 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.⁹

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” fashion¹⁰. 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

⁹Here `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

¹⁰What qualifies as “substantial coding” depends on how much experience you have

30.3. APPENDIX: SOME DESIGN NOTES ON THE METHOD OF MOMENTS CODE

which controlled whether to calculate the predicted moments by simulation or by a formula. You may try your hand at writing this.

Chapter 31

Longitudinal, Spatial and Network Data

[[To come]]

[[Data arranged in space: spatial autocorrelation, spatial smoothing ("kriging"; also Kafadar 1996). Data arranged on a network; Laplacian smoothing smoothing. Multiple time series: the distraction of "Granger causality" and the possibility of real causality.]]

Part V

Data-Analysis Problem Sets

All of the following problem sets have been used in class at least once. They are arranged in an order approximately matching the order of the chapters, but many of them draw on multiple chapters. Each one is scored out of 100 points; in a typical semester, students would do one problem set a week, 12–14 in all. A few provide much less “scaffolding” to guide students through the analysis; these were assigned as take-home exams.

[[TODO: Create a special numbering scheme for problem sets, rather than making them chapters]]

[[TODO: Add references to the source papers]]

Chapter 32

What's That Got to Do with the Price of Condos in California?

AGENDA: As a warm-up and refresher in using linear regression to explore relationships between variables, we will look at a large data set on real estate prices.

The Census Bureau divides the country up into geographic regions, smaller than counties, called “tracts” of a few thousand people each, and reports much of its data at the level of tracts. This data set, drawn from the 2011 American Community Survey, contains information on the housing stock and economic circumstances of every tract in California and Pennsylvania. For each tract, the data file records a large number of variables (not all of which will be used in this assignment):

- A geographic ID code, a code for the state, a code for the county, and a code for the tract
- The population, latitude and longitude of the tract
- Its name
- The median value of the housing units in the tract
- The total number of units and the number of vacant units
- The median number of rooms per unit
- The mean number of people per household which owns its home, the mean number of people per renting household
- The median and mean income of households (in dollars, from all sources)

- The percentage of housing units built in 2005 or later; built in 2000–2004; built in the 1990s; in the 1980s; in the 1970s; in the 1960s; in the 1950s; in the 1940s; and in 1939 or earlier
- The percentage of housing units with 0 bedrooms; with 1 bedroom; with 2; with 3; with 4; with 5 or more bedrooms
- The percentage of households which own their home, and the percentage which rent

Remember that these are not values for individual houses or families, but summaries of all of the houses and families in the tract.

The basic question here has to do with how the quality of the housing stock, the income of the people, and the geography of the tract relate to house values in the tract. We will look at several different linear models, and see if they have reasonable interpretations, and/or make systematic errors.

1. (3 pts) Not all variables are available for all tracts. Remove the rows containing NA values. All subsequent problems will be done on this cleaned data set.
Hint: Recipe 5.27.

- (a) (1) How many tracts are eliminated?
- (b) (1) How many people live in those tracts?
- (c) (1) What happens to the summary statistics for median house value and median income?

2. (7) House value and income

- (a) (1) Linearly regress median house value on median household income. Report the intercept and the coefficient (to reasonable precision), and explain what they mean.
- (b) (2) Regress median house value on mean household income. Report the intercept and the coefficient (to reasonable precision), and explain what they mean. Why are the coefficients for two different measure of household income different?
- (c) (4) Regress median house value on both mean and median household income. Report the estimates, and interpret the coefficients, as before. Does this interpretation seem reasonable? Explain.

3. (10) Regress median house value on median income, mean income, population, number of housing units, number of vacant units, percentage of owners, median number of rooms, mean household size of homeowners, and mean household size of renters. Report all the estimated coefficients and their standard errors to reasonable precision, and explain what they mean. Why are the coefficients on income different from in the previous models?

4. (5) Which three variables are most important, in this model, for predicting house values? Explain your reasoning for deciding on this. *Hint:* make sure your answers wouldn't change if we changed the units of measurement for the predictor variables.
5. (20) *Checking residuals* for the model from problem 3.
 - (a) (5) Make a Q – Q plot of the regression residuals.
 - (b) (5) Make scatter-plots of the regression residuals against each of the predictor variables, and add kernel smoother curves (as in Chapter 1). Describe any patterns you see. (A *very* rough rule of thumb is that the bandwidth should be about $\sigma n^{-1/5}$, where σ is the standard deviation of the predictor variable and n is the sample size.)
 - (c) (5) Make scatter-plots of the squared residuals against each of the predictor variables, and add kernel smoother curves. Describe any patterns you see.
 - (d) (5) Explain, using these plots, whether the residuals appear Gaussian and independent of the predictors.
6. (12) Fit the model from 3 to data from California alone, and again to data from Pennsylvania alone.
 - (a) (5) Report the two sets of coefficients and standard errors. Explain whether or not it is plausible that the true coefficients are really equally.
 - (b) (2) What are the square root of the mean squared error (RMSEs) of the Pennsylvania and California coefficients, on their own data?
 - (c) (5) Use the California coefficients to predict the Pennsylvania data. What is the RMSE? What is the correlation between the California coefficients' predictions for Pennsylvania, and the Pennsylvania coefficients' predictions? *Hint:* Recipe 11.18.
7. (10) Make a map of median house values. The vertical coordinate should be latitude, the horizontal coordinate should be longitude, and the house value should be indicated either by the color of the points (*Hint:* recipe 10.23), or by using a third dimension in a perspective plot. Describe the patterns that you see.
8. (10) Make a map of the regression residuals for the model from problem 3. Are they randomly scattered over space, or are there regions where the model systematically over- or under-predicts? Are there regions where the errors are unusually large in both directions? (You might also want to make a map of the absolute value of the residuals.) — If you cannot make a map, you can still get partial credit for scatter-plots of residuals against latitude and longitude.
9. (8) Fit a linear regression with all the variables from problem 3, as well as latitude and longitude. Report the new coefficients and their standard errors. What do the coefficients on latitude and longitude mean? How important are latitude and longitude in this new model?

10. (5) Make a map of the regression residuals for the new model from problem 9. Compare and contrast it with the map of the residuals from the previous model. Are the new residuals spatially uniform, or are there patterns?
11. (10) Regress the *log* of median house value on the same variables as in problem 9. Which model more accurately predicts housing prices? How can you tell?

Chapter 33

The Advantages of Backwardness

Many theories of economic growth say that it's easier for poor countries to grow faster than rich countries — “catching up”, or the “advantages of backwardness”. One argument for this is that poor countries can grow by copying existing, successful technologies and ways of doing business from rich ones. But rich countries are already using those technologies, so they can only grow by finding new ones, and copying is faster than innovation. So, all else being equal, poor countries should grow faster than rich ones. One way to check this is to look at how growth rates are related to other economic variables.

Our data for examining this will be taken from the “Penn World Table” (http://pwt.econ.upenn.edu/php_site/pwt_index.php), for selected countries and years. The data file is `penn-select.csv` on the class website. Each row of this table gives, for a given country and a five-year period, the starting year, the initial population of the country, the initial gross domestic product (GDP)¹ per capita (adjusted for inflation and local purchasing power), the average annual growth rate of GDP over that period, the average population growth rate, the average percentage of GDP devoted to investment, and the average percentage ratio of trade (imports plus exports) to GDP².

We will use the `np` package on CRAN to do kernel regression.³ Install it, and load the data file `penn-select.csv` (link on the class website).

1. (5 points) Fit a linear model of `gdp.growth` on `log(gdp)`. What is the coefficient? What does it suggest about catching-up?

¹Annual gross domestic product is the total value of all goods and services produced in the country in a given year. It has some pathologies — an earthquake which breaks everyone's windows could *increase* GDP by the value of the repairs — but it's a standard measure of economic output.

²The Penn tables call this variable “openness”. It can be bigger than 100, if, for instance, a country re-exports lots of its imports.

³In addition to the examples in Chapter 4 of the notes, the package has good help files, and a tutorial at <http://www.jstatsoft.org/v27/i05>.

2. (5 points) Fit a linear model of `gdp.growth` on `log(gdp)`, `pop.growth`, `invest` and `trade`. What is the coefficient on `log(gdp)`? What does it suggest about catching-up?
3. (5 points) It is sometimes suggested that the catching-up effect only works for countries which are open to trade with, and learning from, more-developed economies. Add an interaction between `log(gdp)` and `trade` to the model from Problem 2. What are the relevant coefficients? What do they suggest about catching-up?
4. (15 points) Use data-set splitting, as in Chapter 3 of the notes, to decide which of these three linear models predicts best. (You can adapt the code from that chapter or write your own.) Which one is the winner?
5. (15 points) The `npreg` function in the `np` package does kernel regression. By default, it uses a combination of cross-validation and sophisticated but very slow optimization to pick the best bandwidth. In this problem, we will force it to use fixed bandwidths, and do the cross-validation ourselves.

```
penn.0.1 <- npreg(gdp.growth~log(gdp), bws=0.1, data=penn)
```

does a kernel regression of `growth` on `log(gdp)`, using the default kernel (which is Gaussian) and bandwidth 0.1. (You don't have to call the data `penn`.) You can run `fitted`, `predict`, etc., on the output of `npreg` just as you can on the output of `lm`. (There are more examples of using `npreg` in Chapter 4.)

The code at the end of this assignment (also online) uses five-fold cross-validation to estimate the mean-squared error for the six bandwidths 0.05, 0.1, 0.2, 0.3, 0.4, 0.5. Use it to create a plot of cross-validated MSE versus bandwidth. Add to the same plot the in-sample MSEs of those six bandwidths on the *whole* data. What bandwidth predicts best?

6. (10 points) Make a scatterplot of `log(gdp)` versus `growth`. Add the line for the linear model from problem 1. Add the fitted values for the kernel curve with the best bandwidth (according to the previous problem). What does this suggest about catching up?

(There are at least two ways to get the fitted values for the kernel regression, using `fitted` or `predict`.)

7. (5 points) `npreg` will also do kernel regressions with multiple input variables. This time, use the built-in bandwidth selector:

```
penn.npr <- npreg(gdp.growth ~ log(gdp) + pop.growth + invest
+ trade, data=penn, tol=0.1, ftol=0.1)
```

(The last two arguments tell the bandwidth selector not to try very hard to optimize; it may still take several minutes.) What are the selected bandwidths? (Use `summary`.)

8. (5 points) Explain why we cannot add an interaction between $\log(gdp)$ and $trade$ to the nonparametric regression from the previous problem.

9. (15 points) Sub-divide the data into points where the initial GDP per capita is $\leq \$700$ and those where it is above. For each data point, use the kernel regression from problem 7 to predict the *change* in growth-rate from a 10% decrease in initial GDP (not a 10% decrease in log-GDP). Report the averages over the initially-poorer and the initially-richer data points. Describe what this suggests about catching up.

Hints: use `predict()` with partially-modified data; *do not* estimate another regression with artificially-lowered initial GDPs; make sure you are changing initial GDP by 10%, and not changing the log of GDP by 10%.

10. (10 points) To choose between the best linear model (as picked by you in problem 4) and the kernel regression from problem 7, use cross-validation again. Modify the code provided to use five-fold cross-validation to get CV MSEs for both the linear regression and for the nonparametric regression (with automatic bandwidth selection). Which predicts better?
11. (10 points) Based on your analysis, does the data support the idea of catching up, undermine it, support its happening under certain conditions, or provide no evidence either way? (As always, explain your answers.)

```

# Compare predictive ability of different bandwidths using k-fold CV
# Inputs: number of folds, vector of bandwidths, dataframe
# Presumes: data frame contains variables called "gdp.growth" and "gdp"
# Output: vector of cross-validated MSEs for the different bandwidths
# The default bandwidths here are NOT good ones for other problems
cv.growth.folds <- function(nfolds=5, bandwidths=c(0.05,(1:5)/10), df=penn) {
  require(np)
  case.folds <- rep(1:nfolds,length.out=nrow(df))
  # divide the cases as evenly as possible
  case.folds <- sample(case.folds) # randomly permute the order
  fold.mses <- matrix(0,nrow=nfolds,ncol=length(bandwidths))
  colnames(fold.mses) = as.character(bandwidths)
  # By naming the columns, we'll won't have to keep track of which bandwidth
  # is in which position
  for (fold in 1:nfolds) {
    # What are the training cases and what are the test cases?
    train <- df[case.folds!=fold,]
    test <- df[case.folds==fold,]
    for (bw in bandwidths) {
      # Fit to the training set
      # First create a "bandwidth object" with the fixed bandwidth
      current.npr.bw <- npregbw(gdp.growth ~ log(gdp), data=train, bws=bw,
        bandwidth.compute=FALSE)
      # Now actually use it to create the kernel regression
      current.npr <- npreg(bws=current.npr.bw)
      # Predict on the test set
      predictions <- predict(current.npr, newdata=test)
      # What's the mean-squared error?
      fold.mses[fold,paste(bw)] <- mean((test$gdp.growth - predictions)^2)
      # Using paste() here lets us access the column with the right name...
    }
  }
  # Average the MSEs
  bandwidths.cv.mses <- colMeans(fold.mses)
  return(bandwidths.cv.mses)
}

```

Chapter 34

The Size of a Cat's Heart

The goal of this homework is to practice using bootstrapping to quantify the uncertainty in regression models.

The data set `cats` in the library `MASS` contains measurements of the total body weight for 47 female and 97 male adult cats, as well as the weights of their hearts. The medical rationale for this experiment was that the dosage of many heart medicines needs to be calibrated to the mass of the heart, and so one wants to know how to predict that from the total body weight.

Load the data, and check the loading, as follows:

```
> library(MASS)
> data(cats)
> summary(cats)
Sex           Bwt                  Hwt
F:47   Min.    :2.000   Min.    : 6.30
M:97   1st Qu.:2.300   1st Qu.: 8.95
          Median :2.700   Median :10.10
          Mean   :2.724   Mean   :10.63
          3rd Qu.:3.025   3rd Qu.:12.12
          Max.   :3.900   Max.   :20.50
```

Body weights (`Bwt`) are in kilograms, and heart weights (`Hwt`) are in grams — a cat's heart is not actually bigger than its entire body.

The goal here is to provide an accurate estimate of the weight of a cat's heart from its body weight, including some measure of uncertainty, and to assess whether the prediction should take account of the cat's sex.

[[Modified from "Practical" 1 in Davison and Hinkley (1997, §6.8, pp. 321–322). TODO: See if different enough to use]]

[[TODO: Fix references to text]]

1. (5 points) Use `lm` to linearly regress heart weight on body weight, without using sex as a predictor variable, and ensuring that the regression line goes through the origin. (That is, forcing the intercept to be zero.) Report the estimated coefficient, the standard errors given by R, and the corresponding 95% confidence interval (using a *t*-distribution).

Extra Credit: (5 points) Why is it reasonable to force the intercept to be zero in this case?

2. (5 points) Plot the distribution of residuals from the model you fit in Problem 1. Does it look Gaussian? (Explain.)

Extra credit (5 points): Suggest a formal test of the hypothesis that the residuals are Gaussian. Is the departure from a Gaussian distribution significant at the 5% level? At the 1% level?

3. (5 points) Using `lm`, fit a linear regression model for heart weight, in which body weight *interacts* with Sex. Again, ensure that the regression lines for both females and males go through the origin. How many coefficients should there be? Report the estimated coefficients, the standard errors calculated by `lm`, and the corresponding 95% confidence intervals.

4. *Testing the significance of a model expansion*

- (a) (5 points) Describe, briefly and in your own words, a method for formally testing the hypothesis that adding `Sex` as a predictor in the model, as in Problem 3, does not significantly improve over the model you fit in Problem 1. Clearly state the assumptions underlying the test. (Here a “formal test” means one where you calculate both a test statistic and a *p*-value for the test statistic under the null distribution, e.g., the “Partial *F*-test” you learned in 36-401.)

- (b) (5 points) Apply this test to the `cats` data and the models from Problems 1 and 3. What is the value of your test statistic? Is the difference significant at the 5% level?

5. We now compare the measures of uncertainty from problem 1, which are calculated assuming Gaussian and homoskedastic noise, with the measures obtained by bootstrapping the data points. *Hint:* Look at section 4.1 of the notes for [[lecture 8]].

- (a) (5 points) Write a function `resample.cats`, to resample the data points in `cats`. It should take no arguments, but produce a new data frame with the same column names as `cats`. Check that it is working properly by running `summary(resample.cats())` and confirming that the result is close to that of `summary(cats)`.

- (b) (5 points) Write a function `fit.cats.1` to re-estimate the coefficient of the model from Problem 1 on a new data frame. It should take a data frame as an argument, estimate the same model as in Problem 1, and return the value of the regression coefficient (a single number, not the whole regression object). Check that it works by confirming that `fit.cats.1(cats)` gives the same number as the regression coefficient you got in Problem 1.

- (c) (5 points) Using your functions `resample.cats` and `fit.cats.1` from Problems 3a and 3b, write a function `cats.1.se` to find the bootstrap standard error for the coefficient in this linear model. The function should

take the number of bootstrap replicates, and return the estimated standard error. What standard error do you find with 100 replicates? With 1000?

- (d) (5 points) Using your functions `resample.cats` and `fit.cats.1` from Problems 3a and 3b, write a function `cats.1.cis` to find confidence intervals for the coefficient in this model. The function should take as arguments the number of bootstrap replicates and one minus the confidence level. It should return the upper and lower confidence limits. What 95% confidence interval do you get with 100 replicates? With 1000 replicates?
- (e) (5 points) How do your results in (5c) and (5d) compare to those from (1)? Based on your findings in (2), which set of error estimates seems more trustworthy?

6. Cross-validation on a model expansion

- (a) (5 points) Explain, briefly and in your own words, how to use cross-validation to tell whether including `Sex` in the model improves its ability to generalize.
- (b) (5 points) Check, using five-fold cross-validation, whether adding `Sex` to the model improves it.
- (c) (5 points) Can you say whether the cross-validation comparison is significant at the 5% level?

7. Bootstrap testing of a model expansion.

- (a) (5 points) Write a function to simulate new data sets from the model you fit in Problem 1 by re-sampling the residuals. The function should take no arguments, but return a data frame with columns `Sex`, `Bwt` and `Mwt`. *(Hint:* Look at section 4.3 of the notes for [[lecture 8]]).
- (b) (10 points) Write a function to calculate the test statistic from your hypothesis test in Problem 4. The input should be a data frame, which you can assume has the columns `Sex`, `Bwt` and `Hwt`, and the output should be the value of the statistic. Check your function by seeing that it gives the right value of the test statistic when applied to the original `cats` data frame, i.e., the one you calculated in Problem 4b. *Hint:* Look at Code Example 8 in the notes for lecture 8.
- (c) (10 points) Using the simulator from (7a) and the test statistic calculator from (7b), find a bootstrap *p*-value for the significance of adding `Sex` as a predictor. Use at least 500 replicates. Is it significant at the 5% level? *Hint:* Look at [[Code Example 8]] in the notes for [[lecture 8]].
- 8. (10 points) A veterinarian wants to know whether they should adjust for a cat's sex when calibrating how much heart medicine to administer. Based on your findings in problems 4, 6 and 7, what would you recommend, and why?

Chapter 35

It's Not the Heat that Gets You, It's the Sustained Conjunction of Heat with Elevated Levels of Atmospheric Pollutants

The data set `chicago`, in the package `gamair`, contains data on the relationship between air pollution and the death rate in Chicago from 1 January 1987 to 31 December 2000. The seven variables are: the total number of (non-accidental) deaths each day (`death`); the median density over the city of large pollutant particles (`pm10median`); the median density of smaller pollutant particles (`pm25median`); the median concentration of ozone (O_3) in the air (`o3median`); the median concentration of sulfur dioxide (SO_2) in the air (`so2median`); the time in days (`time`); and the mean daily temperature (`tmpd`).

We will model how the death rate changes with pollution and temperature. Epidemiologists tell us that risk factors usually multiply together rather than adding, so we will fit additive models to the logarithm of the number of deaths. These problems *can* all be done using either the `gam` or the `ngcv` packages for fitting additive models, but you will probably find it easier to use `mgcv`.

Warning: The bootstrapping in Problem 7g might be time-consuming; don't wait to the last minute.

1. (5) Load the data set and run `summary` on it.
 - (a) (1) Is temperature given in degrees Fahrenheit or degrees Celsius?
 - (b) (2) The pollution variables are negative at least half the time. What might this mean?

- (c) (2) We will ignore the `pm25median` variable in the rest of this problem set.
Why is this reasonable?
2. (10) Fit a spline smoothing of `log(death)` on time. (You can use either `smooth.spline` or `gam`.)
- (3) Plot the smoothing spline along with the actual values.
 - (4) There should be four large outliers, right next to each other in time.
When are they? For full credit, give calendar dates, not day numbers.
(Hint: day 0 was 31 December 1993.)
 - (3) How many degrees of freedom did your smoothing spline have? Add curves to the plot which would result from using 10, 50, 100 and 2000 degrees of freedom. (Make sure these differ in color and/or line-style.)
What happens to the spline curves as you change the degrees of freedom?
3. (15) Use `gam` to fit an additive model for `log(death)` on `pm10median`, `o3median`, `so2median`, `tmpd` and `time`. Use spline smoothing for each of these predictor variables.
- (7) Plot the partial response functions, with partial residuals. Describe the partial response functions in words.
 - (4) Plot the fitted values as a function of time, along with the actual values of `log(death)`.
 - (4) Are the outliers still there? Are they any better?
4. (15) It is medically implausible to suppose that deaths on day t are only due to heat or pollution on that day, and not on earlier ones.
- (8) Suppose that on any given day, we want to know the average value of some variable over today and the previous k days. Explain how the following code computes that.
- ```
lag.mean <- function(x, window) {
 n <- length(x)
 y <- rep(0, n-window)
 for (t in 0:window) {
 y <- y + x[(t+1):(n-window+t)]
 }
 return(y/(window+1))
}
```
- In particular, how is  $k$  related to the arguments?
- (b) (7) Create a new data frame with the same column names as `chicago`, but where, on each day, the value of the pollution concentrations and temperature is the average of that day's value with the previous three days. How many rows should this data frame have? Make sure that the `time` and `death` columns are properly aligned with the new, time-average predictor variables. How can you check that this is working properly?

5. (10) Fit an additive model, as in problem 3, with the time-averaged pollution and temperature variables. (Do not average `time` or `death`.)
  - (a) (5) Plot the partial response functions and their partial residuals.
  - (b) (5) Plot the fitted values as a function of time, and the actual values. What has happened to the outliers?
6. (15) *Variable examination*
  - (a) (4) Find the rows in the data frame (with the time-averaged values) corresponding to the large-death outliers. Look at all variables for them, and for three days on either side. Now compare this to the same stretch of time a year earlier. Which two variables, aside from `death`, are unusually high or low around the outliers?
  - (b) (7) Re-fit the model from problem 5, with an interaction between the two variables you just picked out. Plot the partial response functions.
  - (c) (4) Plot the fitted values versus time. What has happened to the outliers?
7. (25) Using the last model you fit, we will consider the predicted impact of a 2°C Celsius increase in temperature on `log(death)`, taking the last full year of the data as a baseline.<sup>1</sup>.
  - (a) (1) Prepare a data frame containing only the last full year of the data. What is the average predicted value of `log(deaths)`?
  - (b) (1) Modify this data frame to increase all temperatures by 2°C.
  - (c) (3) Find the new average *change* in the predicted values of `log(deaths)` associated with a 2°C warming.
  - (d) (5) Find a standard error for this average predicted change, using the standard errors for the prediction on each day, and assuming no correlation among them. Also give the corresponding Gaussian 95% confidence interval.
  - (e) (5) Find the predicted change in the number of deaths (not change in `log(death)`) from a 2°C warming over the course of a whole year. *Hint:* remember that  $\bar{e^x} \neq e^{\bar{x}}$ .
  - (f) (5) Explain how you could use bootstrapping to give a 95% confidence interval for the average increase in `log(death)` over the year. More credit will be given for more precise, complete and clear explanations.
  - (g) (5) Implement your bootstrapping scheme and give the confidence interval.

<sup>1</sup>2°C is in the middle range of current projections for the global average effect of climate change by the end of this century ([http://www.ipcc.ch/publications\\_and\\_data/ar4/wg1/en/contents.html](http://www.ipcc.ch/publications_and_data/ar4/wg1/en/contents.html)). Of course it's unrealistic to suppose that would be an even shift throughout the year, or for that matter that Chicago would necessarily warm by the average amount. In fact, some of the models ([http://www.ipcc.ch/publications\\_and\\_data/ar4/wg1/en/ch11s11-5-3.html](http://www.ipcc.ch/publications_and_data/ar4/wg1/en/ch11s11-5-3.html), Figure 11.11) have 4°C of warming in the middle of their prediction intervals for central North America.

8. (5) Explain at least one reason that this estimate of what would happen if Chicago warmed by 2°C might be systematically flawed. (Do not repeat the problems mentioned in the footnote. Doubts that such warming will happen do not count.) For full credit, suggest ways of improving the estimates.

## Chapter 36

# Nice Demo City, but Will It Scale?

[[TODO: Integrate the two versions of this problem set, perhaps by just picking one]]

This version was used as a take-home exam, hence less scaffolding; flag as such in the guide to the problems

[[TODO: Yank references from *Preface to Urban Economics*]]

### 36.1 Version 1

#### 36.1.1 Background

It has been known for a long time that larger cities tend to be more economically productive than smaller ones. That is, the economic output per person of a city or other settlement ( $Y$ ) tends to increase with the population ( $N$ ). Recently, there has been some controversy over the exact form of the relationship, and over its explanation.

In particular, it has been claimed<sup>1</sup> that urban incomes show “power-law scaling”, meaning that

$$Y \approx y_0 N^\alpha$$

for some constant  $y_0 > 0$  (the same across cities) and some *scaling exponent*  $\alpha > 0$  (the same across cities). Equivalently<sup>2</sup>,

$$\log Y \approx c + \alpha \log N$$

The scientists who first postulated power law scaling for urban economies thought that the tendency for bigger cities to be more productive was largely due to what are called “increasing returns to scale”<sup>3</sup>, which would be stronger in larger cities. Additionally, having more people around, and more different sorts of people around, could lead to exchanges of ideas and so to new and better ways of doing business. According to this view, the primary determinant of a city’s economy is simply its size, and larger cities are just “scaled up” versions of smaller ones.

<sup>1</sup>By Geoffrey West and collaborators; there’s a good video online of Prof. West giving a talk about the work at a TED conference, if you’re interested.

<sup>2</sup>Why is it equivalent, and how is  $c$  related to  $y_0$ ?

<sup>3</sup>This is when the cost of producing the same item, with the same factory, employees, etc., is lower when the volume being produced is high, perhaps because the system runs more efficiently, or each sale has to recover a smaller share of the fixed cost of setting up the factory. A constant sale price minus lower costs equals higher profits.

An alternative explanation is that different industries have different levels of income per worker, and that some industries tend to be concentrated in larger cities and others in smaller towns. Large cities tend especially to be the places where one finds highly skilled providers of very specialized services, though their services are used, often indirectly, more or less everywhere<sup>4</sup>. In this view, the association between the population of cities and their economic productivity is due to the kind of industries that go with being big cities, not some effect of size as such. There is no reason, according to this “urban hierarchy” view, why the relationship between per-capita income  $Y$  and urban population  $N$  should be a power law. In fact, the urban-hierarchy model doesn’t even specify a particular functional relationship between how much of a city’s economy comes from high-value industries and the city’s income, just that the relationship is increasing.

Note that neither the power-law nor the urban-hierarchy model predicts Gaussian distributions.

In this exam, you will assess the evidence for power law scaling, and whether the “urban hierarchy” idea can explain the relationship between income and population.

### 36.1.2 Data

For data-collection purposes, urban regions of the United States are divided into several hundred “Metropolitan Statistical Areas” based on patterns of residence and commuting; these cut across the boundaries of legal cities and even states. In the last decade, the U.S. Bureau of Economic Analysis has begun to estimate “gross metropolitan products” for these areas — the equivalent of gross national product, but for each metropolitan area. (See Homework 2 for the definition of “gross national product”.) Our data set contains the following variables, derived from the BEA:

- the name of each metropolitan area;
- its per-capita gross metropolitan product, in dollars ( $Y$ );
- its population ( $N$ );
- the share of its economy derived from finance (as a fraction between 0 and 1);
- the share of “professional and technical services”;
- the share of “information, communication and technology” (ICT);
- and the share of “management of firms and enterprises”.

Note that the last four columns have some missing values (NAs), since the BEA does not release those figures when doing so would disclose sensitive information about individual companies.

---

<sup>4</sup>There are probably few, if any, electrochemical engineers who design liquid crystal displays working in Altoona, PA, but everyone there who buys a cellphone indirectly pays for the time and training of such engineers who live elsewhere.

### 36.1.3 Tasks and Questions

You are to write a report assessing the (1) whether the power-law scaling model accurately represents the relationship between urban population and urban per-capita income; (2) whether, as the “urban hierarchy” idea implies, the relationship can be explained away by controlling for which industries are found in which cities; and (3) whether the power-law scaling or the urban-hierarchy idea provides a better model of urban economies.

Your report should have the following sections: an introduction, laying out the questions being investigated and the approach taken; a description of the data; detailed analyses; and conclusions. Your report should deal with *at least* the following specific points:

- The estimation of the scaling exponent  $\alpha$  from the data, including its uncertainty<sup>5</sup>;
- An estimate of the out-of-sample error of the power-law-scaling model;
- An examination of that model’s residuals;
- A comparison of that model to non-parametric models of the size-income relationship (including, but not limited to, out-of-sample errors);
- Whether larger cities tend to have higher shares of the four high-value industries measured in the data set, and if so, what the size-industry relationship is;
- Whether cities with higher shares for those industries have higher incomes, and if so, what the industry-income relationship is;
- Whether, and in what sense, the income-industry relationships can explain the size-income relationship;
- How missing values were handled, and why;
- Appropriate quantifications of uncertainty for all estimates and hypothesis tests.

Adequately dealing with these points may, of course, lead to others.

## 36.2 Version 2

### 36.2.1

[[This version was a pair of homework assignments, so the points add up to 200]]

For data-collection purposes, urban areas of the United States are divided into several hundred “Metropolitan Statistical Areas” based on patterns of residence and commuting; these cut across the boundaries of legal cities and even states. In the last decade, the U.S. Bureau of Economic Analysis has begun to estimate “gross metropolitan

---

<sup>5</sup>Hint: You should get a value in the range (0,0.5).

products” for these areas — the equivalent of gross national product, but for each metropolitan area. (See Homework 2 for the definition of “gross national product”.) Even more recently, it has been claimed that these gross metropolitan products show a simple quantitative regularity, called “supra-linear power-law scaling”. If  $Y$  is the gross metropolitan product in dollars, and  $N$  is the number of people in the city, then, the claim goes,

$$Y \approx cN^b \quad (36.1)$$

where the exponent  $b > 1$  and the scale factor  $c > 0$ . This homework will use the tools built so far to test this hypothesis.

1. (15 points) A metropolitan area’s gross per capita product is  $y = Y/N$ . Show that if Eq. 36.1 holds, then

$$\log y \approx \beta_0 + \beta_1 \log N$$

How are  $\beta_0$  and  $\beta_1$  related to  $c$  and  $b$ ?

2. (15 points) The data files `gmp_2006.csv` and `pcgmp_2006.csv` on the class website contain the total gross metropolitan product ( $Y$ ) in millions of dollars, and the per capita gross metropolitan product ( $y$ ) in dollars, for all metropolitan areas in the US in 2006. Read them in and use them to calculate the metropolitan populations ( $N$ ). If it’s done correctly, then running `summary` on the population figures should give

| Min.  | 1st Qu. | Median | Mean   | 3rd Qu. | Max.     |
|-------|---------|--------|--------|---------|----------|
| 54980 | 135600  | 231500 | 680900 | 530900  | 18850000 |

(Your exact results may differ very slightly because of rounding and display settings.) What is the variance of  $\log y$ ?

3. (20 points) *Estimating the power-law scaling model.* Use `lm` to linearly regress  $\log$  per capita product,  $\log y$ , on  $\log$  population,  $\log N$ . How does estimating this statistical model relate to Equation 36.1? What are the estimated coefficients? Are they compatible with the idea of supra-linear scaling? What is the mean squared error for  $\log y$ ?
4. (15 points) Plot per capita product  $y$  against  $N$ , along with the fitted power-law relationship from problem 3. (Be careful about  $\log$ s!)
5. (15 points) Fit a non-parametric smoother to  $\log y$  and  $\log N$ . (You can use kernel regression, a spline, or any other non-parametric smoother.) What is the mean squared error for  $\log y$ ? Describe, in words, how this curve compares to the power-law model from problem 3.
6. (20 points) Using the method from [[lecture 10, section 1]], test whether the power-law relationship is correctly specified. What is the  $p$ -value? What do you conclude about the validity of the power-law model, based not just on this problem but the previous ones as well?

### 36.2.2

We continue to investigate the relationship between how big cities are, and how economically productive they are. The scientists who first postulated power laws for urban economies thought that the tendency for bigger cities to be more productive was largely due to what are called “increasing returns to scale”<sup>6</sup>, which would be bigger in larger cities. Additionally, having more people around, and more different sorts of people around, could lead to exchanges of ideas and so to new and better ways of doing business.

An alternative explanation is that different industries have different levels of income per worker, and that some industries tend to be concentrated in larger cities and others in smaller towns. Large cities tend especially to be the places where one finds highly skilled providers of very specialized services, though their services are used, often indirectly, more or less everywhere<sup>7</sup>. In this view, the association between the population of cities and their economic productivity is due to the kind of industries that go with being big cities, not some effect of size as such.

In this exam, you will do a fairly simple test of these two explanations.

#### 36.2.2.1 Data

A data file has been e-mailed to you at your Andrew account. It is a comma-separated text file (CSV), containing the following columns, in order, for each metropolitan area:

- the name of the metropolitan area;
- its per-capita gross metropolitan product (in dollars)
- its population;
- the share of its economy derived from finance (as a fraction between 0 and 1);
- the share of “professional and technical services”;
- the share of “information, communication and technology” (ICT);
- and the share of “management of firms and enterprises”.

The first three columns you saw in the last homework. The last four columns came from the same source. However, those columns have some missing values (NAs), since the Bureau of Economic Analysis does not release the data when doing so would disclose sensitive information about individual companies.

---

<sup>6</sup>This is when the cost of producing the same item, with the same factory, employees, etc., is lower when the volume being produced is high, perhaps because the system runs more efficiently, or each sale has to recover a smaller share of the fixed cost of setting up the factory. A constant sale price minus lower costs equals higher profits.

<sup>7</sup>There are probably very few electrochemical engineers who design liquid crystal displays in Altoona, but everyone there who buys a cellphone indirectly pays for the time and training of such engineers who live elsewhere.

### 36.2.3 Problems

1. *More specialist service industries in bigger cities?*

- (a) (2 points) For each of the four industries, create a scatter-plot of the share of that industry in the economy as a function of population. If a city is missing a value for an industry, omit it from that plot.
- (b) (5 points) Add a nonparametric smoothing curve to each plot. Use kernel regression, local linear regression, a smoothing spline, etc., as you wish, but make sure that you use cross-validation to adapt the amount of smoothing to the roughness of the data.
- (c) (3 points) Describe the patterns made by these plots. In particular, do larger cities have more of these industries?

2. *Higher productivity from specialist service industries?*

- (a) (2 points) For each of the four industries, create a scatter-plot of per-capita GMP as a function of the share of that industry in the city's economy. If a city is missing a value for an industry, omit it from the plot.
- (b) (5 points) Add a nonparametric smoothing curve to each plot. (Use the same smoothing method you did for problem 1.)
- (c) (3 points) Describe the patterns made by these plots. In particular, do cities which are more dependent on these industries have higher productivity?

3. *Are bigger cities more productive, controlling for industry shares?* Using the `gam` function from the `mgcv` package, fit the semi-parametric log-additive model

$$\ln y = \alpha_0 + b \ln N + \sum_{j=1}^4 f_j(x_j) + \epsilon$$

where  $y$  is per-capita GMP,  $N$  is population, and  $x_1$  through  $x_4$  are the shares of the four industries.

- (a) (5 points) Explain how this model is related to, but different than, the power-law scaling model from the last homework. Which terms in the model are parametric, and which are non-parametric?
- (b) (2 points) What R command did you use to fit this?
- (c) (2 points) Report your estimated values for  $\alpha_0$ ,  $b$ , and the residual standard error.
- (d) (6 points) Provide plots of each of the four partial response functions  $f_j$ . Compare them to the plots from question 2 — do they suggest the same relationships between industry shares and the level of productivity, and if not, how do they differ? Hint: `help(plot.gam, package="mgcv")`
- (e) (5 points) Do the residuals seem to have a Gaussian distribution? (Justify your answer.)

- (f) (5 points) Running summary on your fitted model will produce output which includes approximate standard errors and  $p$ -values for the parametric terms, assuming homoskedastic Gaussian noise. What standard error and  $p$ -value does it report for  $b$ ? Is that term significant? Do you think you can trust those calculations in this case?

4. *Predictive comparisons*

- (a) (5 points) Take the fitted power-law scaling model from the last homework. (If you were unable to complete that homework, follow the solutions.) For each city, compute the predicted change in  $\ln y$  from increasing that city's population by 10%. Report the average change over all cities.
- (b) (5 points) Repeat this calculation, for the cities where complete data is available, for the model you fit in Problem 3, assuming that *only* population changes.
- (c) (5 points) Do the two models seem to lead to different conclusions about the effect of population on productivity? Explain

5. *Model comparisons*

- (a) (3 points) What is the in-sample mean squared error, for  $\ln y$ , of the additive model you fit in Problem 3? How much smaller is it than the linear (power law) model from the last homework? Explain why the additive model should always have a smaller in-sample error than the linear model.
- (b) (11 points) Describe, concisely and in your own words, a technique for determining whether the additive model from Problem 3 is better able to generalize than the pure power law model. Explain why this technique should be reliable here. (You are free to use a method from 36-401, if you can explain why it is applicable.)
- (c) (11 points) Implement this comparison and report your results. Which model is favored?

6. *Evaluation*

- (a) (10 points) Based on what you have done so far, does it seem that city size directly effects productivity? Specifically, if an American city wanted to increase its per-capita economic output, should it try to increase population, or change its industries?
- (b) (5 points) Suggest additional data, models or comparisons which could improve your analysis.

# Chapter 37

# Diabetes

Diabetes is a family of diseases where the body does not metabolize sugar properly. Ordinarily, metabolism of blood sugar is regulated by a protein called insulin. In type I, the body stops producing enough insulin; in type II diabetes, the body stops responding to insulin<sup>1</sup>. While they are related, the treatments are very different: type I diabetics need regular injections of insulin, and benefit from treatments which increase the body's production of insulin; these are of no use for type II diabetes.<sup>2</sup>

C-peptide is a protein which is produced along with insulin, but is easier to measure, and provides an accurate proxy for their blood insulin levels. Our data set contains information about 43 patients with type I diabetes: their age when they were first diagnosed with diabetes, the logarithm of their C-peptide concentration (in picomoles per milliliter), and `base.deficit`, a measure of how acid their blood is compared to normal controls<sup>3</sup>. The questions of immediate interest are about predicting c-peptide levels from age and base-deficit. The questions of ultimate interest are about what can be done to increase insulin levels for patients suffering from type I diabetes.

1. (10 total) Fit an additive model,

$$\hat{y}_{AM}(\text{age}, \text{base.deficit}) = \alpha + f_1(\text{age}) + f_2(\text{base.deficit})$$

with c-peptide level as the response.

- (a) (5) Plot and describe your estimates of the partial response functions  $f_1$  and  $f_2$ .

---

<sup>1</sup>Type I and type II diabetes used to be called “juvenile” and “adult-onset”, respectively. While it is true that most people who become diabetic as children have type I, and that type II tends to develop later in life, there are plenty exceptions in both directions. (I know someone whose type I diabetes first manifested in his mid-30s, on his honeymoon.)

<sup>2</sup>If you need to know more, there are much better sources than Wikipedia, such as the National Diabetes Information Clearninghouse, run by the National Institutes of Health.

<sup>3</sup>There are several reasons why this was of interest. One is that diabetes can lead to a metabolic condition called *ketoacidosis*, when the body, starved for sugars, starts breaking down proteins, with waste-products that make the blood acid (and give the breath a sweet, fruity smell); this can also occur with starvation, or extreme alcoholism.

- (b) (5) Plot the predicted surface  $\hat{y}_{AM}(\text{age}, \text{base.deficit})$ . (Contour, heatmap and wireframe plots are all acceptable. Make sure the axes are labeled with variable names as well as numerical units.)

2. (15 total) *Conditional predictions*

- (a) (10) Plot the predicted c-peptide level for a five-year-old patient against `base.deficit` level, letting `base.deficit` run over the whole observed range, i.e., plot

$$\hat{y}_{AM}(5, \text{base.deficit})$$

as a function of `base.deficit`.

- (b) (2) Add to this plot two more lines, showing the predicted c-peptide level as a function of base-deficit for patients aged 10 and 12 years.

- (c) (3) Are the three lines for the three ages parallel to each other? Should they be?

3. (10 total) *Kernel model*

- (a) (5) Fit a kernel regression jointly to `age` and `base.deficit`. Plot the predicted surface  $\hat{y}_{KM}(\text{age}, \text{base.deficit})$ . Describe the surface and compare it to the surface for the additive model.

- (b) (5) Plot predicted c-peptide levels as functions of base-deficit for patients aged 5, 10 and 12 years, as in Problem 2. Are the three lines for the three different ages parallel to each other? Should they be?

4. (25 total) *Model comparison* Should we use a strictly additive model here, or should we allow for an interaction between `age` and `base.deficit`?

- (a) (8) Describe a procedure which could be used to decide between these options.

- (b) (4) Explain in what sense the model picked by this procedure ought to be better than the one it doesn't pick.

- (c) (5) Explain why this procedure *reliably* picks the better model.

- (d) (8) Apply the procedure to the data and report the result.

5. (25 total) *Predicting response to changes*

- (a) (5) On average, by how much would increasing the age of each patient by 1 year change their c-peptide levels? Answer in terms of the model you picked in Problem 4, and assume patients' `base.deficit` levels remain unchanged.

- (b) (5) On average, by how much would increasing `base.deficit` by one tenth of a standard deviation (i.e., moving it towards zero) change the c-peptide level? Again, answer in terms of the model you picked in Problem 4, and assume that patients' ages remain unchanged.

- (c) (5) Calculate standard errors for both of these average predicted changes. Be sure to explain both the method you used to find the standard errors, and why that method is appropriate to this problem.
  - (d) (5) Calculate a standard error for the *difference* in these average predicted changes.
  - (e) (5) If these two changes are equally easy to bring about, which one would be better? (Assume that higher c-peptide levels are better, generally, than lower ones.) How sure should you be of this answer? What would it mean to increase the patients' ages?
6. (15) Summarize your analysis in one page of prose. Refer to your work on earlier problems for details, but try, as much as you can, to be clear to someone who had not taken 402, or even 401.

Based on the data set presented on p. 304 of Hastie and Tibshirani's *Generalized Additive Models*, but re-typed by me since it's not online. Also based on the analysis for interactions in the last chapter of Hart's *Nonparametric Smoothing and Lack-of-Fit Tests*

## Chapter 38

# Fair's Affairs

In 1969, the magazine *Psychology Today* did a survey of its readers that included questions about (among other things) how often the respondents had had extra-marital sex in the previous twelve months. In 1978 the economist Ray C. Fair used this data to develop a “theory of extramarital affairs” (Fair, 1978)<sup>1</sup>, with the idea that people optimize a trade-off between working, spending time with their spouse, and spending time with a “paramour”. The model and data have become very well known (there are at least a hundred later papers and books which reference it), and is available as *Affairs* in the package *AER* on CRAN.

The variable *affairs* records the answer to “How often did you engage in extra-marital sexual intercourse during the past year”, with values of “once a month”, or more frequently, all coded as 12. Other variables are sex, age, how many years the respondent had been married<sup>2</sup>, whether they had children, how religious they were (on a scale of 1–5), their level of education, how much prestige their occupation had (on a scale of 1–7), and how happy they were with their marriage (on a scale of 1–5).

1. (30 points) *Two specifications*
  - (a) (15 points) Using logistic regression, fit a model for the *number* of times respondents said they had extramarital sex during the previous year. Describe, in words, the predictions of the model. Which variables are significant predictors?
  - (b) (15 points) Repeat (1a), but use logistic regression to fit a model for *whether* respondents said they had extramarital sex at all during the previous year.
2. (10 points) Are the same variables significant in both models in problem 1? Do they have the same signs in both models? Should the models match in this way? Explain.
3. (20 points) *Comparing predictions*

---

<sup>1</sup>This paper also used a similar survey of readers of *Redbook* in 1974, not part of this data set.

<sup>2</sup>Prof. Fair removed respondents who had never married, or had married more than once.

- (a) (5 points) For each person in the data set, calculate the predicted probability, under both models, that they did *not* have an affair.
- (b) (10 points) Plot these against each other. Describe the plot in words.
- (c) (5 points) Do the models agree with each other in their predictions? Should they? Explain.

4. (20 points) *Calibration*

- (a) (2 points) Consider all the people for whom the predicted probability of an affair, according to the model from problem (1a), is less than 10%. What fraction of them report having affairs?
  - (b) (3 points) Repeat this calculation for predicted probabilities between 10% and 20%, 20% and 30%, etc. Plot the actual frequencies against the predicted probabilities.
  - (c) (5 points) Make a similar plot for the other model. (You can combine the plots, if the result is clear.)
  - (d) (10 points) For which model do the predictions seem to match the data best? Explain with reference to your plots.
5. (10 points) Download Fair's paper and read Table I (p. 53). Does it make sense to use a linear response for all of the variables (as in problem 1 above), or would it be better to treat some variables as categorical? Explain.

6. (10 points) *Evaluation*

- (a) (5 points) Do either of these models seem to provide an adequate description of the data? (Explain.) If not, what else could one try?
- (b) (5 points) Is it reasonable to use this data to develop theories about contemporary behavior? Explain.

## Chapter 39

# How the North American Paleofauna Got a Crook in Its Regression Line

Our problem set this week concerns an important question for evolutionary biology and paleontology. It has been argued that larger organisms tend to have selective advantage over smaller ones of the same species, but larger bodies demand more specialized internal structure, more “division of labor”, than small ones, indirectly driving the evolution of increased biological complexity (Bonner, 1988). To evaluate this, it is important to know whether species tend to get larger over evolutionary time, and, if so, to characterize this accurately.

Our data set this week is taken from the North American Mammalian Paleofaunal Database, which contains information on the typical body mass of about 2000 living and extinct species of mammals native to North America. (You can find it on the website, <http://www.stat.cmu.edu/~cshalizi/uADA/13/hw/04/nampd.csv>.) Specifically, the columns of the data give: the scientific name of the species; the natural logarithm of its typical body mass (measured in grams); the natural logarithm of the mass of its ancestor (in grams); how long ago it first appeared in the fossil record (in millions of years); and how recently it last appeared (in millions of years; an NA in this column indicates the species is still alive). We will model how the *change* body mass is related to the body mass of the ancestral species. In particular, paleontologists have suggested that the correct model relating change in log mass to ancestral log mass should be piece-wise linear: a downward-sloping line for small ancestral log masses, and flat for larger ancestral masses. In this problem set, you will fit that model, and examine its predictions.

### 1. (10) *Basics*

- (a) (5) Load the data. Create a vector which gives each species’ change in log body mass from its ancestor, and add it to the data frame as a new

column. Explain, in your own words, what it would mean for a species to have a value of +0.7 in this column. Check that this column has NA values in the correct places. Explain how you know that those are the correct places. Remove all the rows with NA values for the change in log mass, and use this cleaned version of the data for all subsequent parts of the assignment.

- (b) (5) Plot the change in log body mass versus ancestral log body mass. Describe the plot briefly.

**2. (10) Linear model**

- (a) (2) Linearly regress the change in log body mass on the ancestral log body mass. Report the coefficients to reasonable precision.
- (b) (3) Create a new figure which is the scatter-plot from problem 1b, plus your fitted regression line.
- (c) (5) Based on the estimates 2a and the plot from 2b, does this model support or undermine the idea that new species tend to be larger than their ancestors? Explain.

**3. (15) Piecewise linear model**

- (a) (5) The piece-wise linear model predicts the following mean response as a function of the input  $x$ :

$$\hat{y}(x) = \begin{cases} a + bx & \text{if } x \leq d \\ c & \text{if } x \geq d \end{cases}$$

Assuming that this is continuous at  $d$ , solve for  $a$  in terms of  $b$ ,  $c$  and  $d$ . Explain why, in this application, it is reasonable to assume continuity.

- (b) (10) Write a function in R, called<sup>1</sup> `deac`, that takes in a vector of numbers  $x$ , and three parameters  $b$ ,  $c$ , and  $d$ , and returns the prediction of the model at each value of  $x$ .

Check that your `deac` function is working properly by seeing that when  $b = -1$ ,  $c = 0.05$  and  $d = 2$ , giving  $x=c(1, 1.5, 3)$  outputs

```
[1] 1.05 0.55 0.05
```

Plot `deac`, with those parameters, as  $x$  goes over the range  $(0, 4)$ . Does it look right?

*Hints:* `ifelse` for writing `deac`, `curve` for plotting.

4. (15) Because `deac` varies nonlinearly with parameter  $d$ , we cannot estimate it by linear regression. However, we can still estimate the parameters by least squares. To do this, we need to write a function, make a starting guess about the parameters, and use the built-in optimization function `optim` (see recipe

---

<sup>1</sup>From the initials of the scientists who proposed this model; they didn't give it a name.

13.2 in *The R Cookbook*).<sup>2</sup> The following function fits the model to a data set by numerically minimizing the sum of squared errors:

```
my.start <- c(b=-1,c=0.2,d=10)
fit.a.deac <- function(data,start=my.start) {
 sse <- function(par) {
 preds <- deac(data$ln_old_mass,par[1],par[2],par[3])
 sum((data$delta_ln_mass - preds)^2)
 }
 fit <- optim(par=start,fn=sse,method="Nelder-Mead")
 coefficients <- fit$par
 fitted <- deac(data$ln_old_mass,coefficients[1],coefficients[2],
 coefficients[3])
 residuals <- data$delta_ln_mass - fitted
 mse <- mean(residuals^2)
 return(list(coefficients=coefficients,fitted=fitted,residuals=residuals,
 mse=mse,data=data))
}
```

(See online for the commented version; you'll want to source that, rather than typing this in and adding original errors.)

- (a) (7) Explain what the inner function, `sse`, does.
  - (b) (8) What sort of output does `fit.a.deac` give — a vector, a list, an array, what? What do the various components of the output represent, in terms of the statistical problem?
5. (15) *Starting positions* The code given above looks for a vector of initial parameters called `my.start`, if no other starting point is supplied. The line before the function makes up some values for `my.start`; they are bad ones. We will see in a later problem set that a reasonable guess for  $d$  is about 5.
- (a) (5) Use this more-reasonable value of  $d$  to get a rough guess for  $c$  by taking the average change in log mass over all animals whose ancestral log mass exceeds  $d$ . Explain why this is a reasonable way to guess at  $c$ .
  - (b) (5) Get a rough guess for  $b$  by linearly regressing the change in log mass on ancestral log mass for animals where the ancestral log mass is less than  $d$ . Explain why this is a reasonable way to guess at  $b$ .
  - (c) (5) Re-define `my.start` to contain your improved guesses for  $b$ ,  $c$  and  $d$ . Run `fit.a.deac` to get a fitted model, which you should call `nampd.deac`. Plot the fitted values as a function of log ancestral mass on a scatter-plot of change in log mass versus log ancestral mass.

---

<sup>2</sup>R has a built-in function, `nls`, for such “nonlinear least-squares” estimation, working more like `lm`. Unfortunately, `nls` can be flaky when the model doesn’t have continuous derivatives, which is the case here. Besides, writing your own code builds character.

6. (20) *Bootstrapping will continue until morale improves.* Use resampling of residuals, not cases, in both parts. Note: You can use the same resampled data-frames for both parts of this problem, but it needs more clever programming. 1000 bootstrap replicates takes 1–2 minutes on my computer.
  - (a) (10) Find bootstrap standard errors, and 95% confidence intervals, for the parameters  $b$ ,  $c$  and  $d$ . Report all these quantities.
  - (b) (10) Find 95% bootstrap confidence bands for the fitted curve, and add them to your plot from problem 5c.
7. (15) *Linear vs. Piecewise Linear* One way to compare two models is to see which one can predict the other's parameter values. We will compare the simple linear model from problem 2a with the piecewise linear model `deac` model from problem 5c.
  - (a) (5) Simulate the fitted `deac` model, using resampling of residuals, and estimate the linear model on the simulation. What coefficients do you estimate? Are they compatible with the ones you estimated from the data?
  - (b) (5) Simulate the fitted linear model, using resampling of residuals, and estimate the `deac` model on the simulation. What coefficients do you get? Are they compatible with the ones you estimated from the data?
  - (c) (5) Use five-fold cross-validation to compare the linear model from problem to the piecewise-linear `deac` model. Which one predicts mass changes better?

## Chapter 40

# How the *Hyracotherium* Got Its Mass

AGENDA: Using nonparametric smoothing to check parametric models; more practice with simple simulations and function-writing.

We continue to work with the fossil data set from §39. As mentioned there, some paleontologists have suggested that the right curve relating change in log mass to ancestral log mass should be piece-wise linear and homoskedastic: a downward-sloping line for small ancestral log masses, flat for larger ancestral masses, and constant conditional variance:

$$\begin{aligned} Y &= \begin{cases} a + bx + \epsilon & \text{if } x \leq d \\ c + \epsilon & \text{if } x \geq d \end{cases} \\ E[\epsilon|x] &= 0 \\ \text{Var}[\epsilon|x] &= \sigma^2 \end{aligned}$$

In the last problem set, you fit that model; in this one, you will see whether the data support non-linear corrections.

You will first need to load the data from the other problem set, and add the column of change in log mass to the data frame.

The `mgcv` package is recommended for the additive model in Problem 5. Earlier problems call for spline smoothing, and can be done with either the `smooth.spline` function or with the `gam` function.

1. (10) *Plotting the Parametric Model*
  - (a) (5) Make a scatter-plot showing the change in log mass as a function of the log ancestral mass.
  - (b) (5) Add the estimated piecewise linear model from homework 4. You may refer to the solutions for code and parameter estimates, but must explain, in your own words, any code you borrow from there.

2. (25) *Residual inspections*

- (a) (5) Calculate the residuals of the estimated piecewise linear model and plot them against the log ancestral mass. Describe any patterns to the plot in words; you should address whether the model systematically over- or under-predicts in certain ranges of ancestral mass, but there may be other important features.
- (b) (5) The column `first_appear_Mya` lists how many millions of years ago each species first appeared. Plot the residuals against this variable; describe any patterns.
- (c) (5) Plot the squared residuals against the log ancestral mass. Add a smoothing spline. Explain whether the scatter-plot and the spline show evidence of heteroskedasticity.
- (d) (5) Plot the squared residuals against date of first appearance and add a smoothing spline. Explain whether the scatter-plot and the spline show evidence of heteroskedasticity.
- (e) (5) Plot the histogram of the residuals (not the squared residuals). Are they Gaussian? Should they be, under the model?

3. (10) *A nonparametric alternative*

- (a) (7) Fit a spline regression of the change in log mass against log ancestral mass. Plot this spline on the same graph as the data and the estimated piece-wise linear model. Compare, in words, the shape of the spline to that of the parametric model.
- (b) (3) Find the in-sample root-mean-square error of both the parametric model and the smoothing spline. Which fits better?

4. (20) *Testing parametric forms*

- (a) (3) Write a function to fit the smoothing spline to a data set. Check that it works by making sure it gives the right answer on the original data.
- (b) (2) Write a function to calculate the MSE of a fitted smoothing spline. Check that it works by making sure it gives the right answer on the original data.
- (c) (5) Write a function to take in a data set and return the difference in MSEs between the parametric model and the smoothing spline. Check that it works by making sure it gives the right answer on the original data.
- (d) (5) Write a function to simulate from the estimated piecewise-linear model by resampling the residuals. You can borrow from the solutions to homework 4, but must explain, in your own words, how that code works. How can you check that the simulation works?
- (e) (5) Combine your functions to draw 1000 samples from the distribution of this test statistic, under the null hypothesis that the parametric model is right. What is the  $p$ -value of this test of the null hypothesis?

5. (25) *Additional Variables* The piecewise linear model implicitly assumes that the relationship between ancestral mass and change in mass is the same at all times. An alternative is that this relationship has itself evolved.
  - (a) (5) Estimate an additive model which regresses the change in log mass against the log ancestral mass and the date of first appearance. Plot the two partial response functions, and describe, in words, the shape of the curves. Compare the shape of the partial response function for log ancestral mass to the spline curve from Problem 3a.
  - (b) (4) Does the estimated additive model support or undermine the idea that the relationship between ancestral mass and descendant mass is invariant over time? Explain.
  - (c) (1) What is the in-sample root-mean-square error of the additive model?
  - (d) (10) Explain what you would have to change from your code in Problem 4 to test the piecewise-linear model against the additive model, and what pieces of code could stay the same.
  - (e) (5) Write the new code called for by Problem 5d and run the test. What is the  $p$ -value?
6. (10) Is the piecewise-linear, homoskedastic parametric model an acceptable representation of the data? Justify your answer by referring to your work above.

## Chapter 41

# How the Recent Mammals Got Their Size Distribution

Problem sets 39 and 40 used regression to study how the typical mass of (mammalian) species changes over evolution: on average new species are heavier than their ancestors, especially if the ancestor was very small, but with a wide variation. If we combine this with the facts that new species branch off from old ones, and that sometimes species go extinct without leaving descendants, we get a model for how the distribution of body masses changes over time. It's not feasible to say much about this model mathematically, but we can simulate it, and check the simulated distribution against the real distribution of body masses today.

The objects in this model are species, each described by its typical mass. (We assume that this does not change over the lifespan of the species.) Each species can produce new species, whose mass is related to that of its ancestor according to our previously-learned regression model, or go extinct. As time goes on, the distribution of body masses will fluctuate randomly, but should do so around a steady, characteristic distribution.

More specifically, each species  $i$  has a mass  $X_i$ , which is required to be between  $x_{\min}$ , the smallest possible mass for a mammal, and  $x_{\max}$ , the largest possible mass. At each point in time, one current species  $A$  is uniformly selected to evolve into exactly two new species. Each descendant has a mass  $X_D$  which depends on the mass of its ancestor,  $X_A$ , according to the regression model, plus independent noise:

$$\log X_D = \log X_A + Z + \begin{cases} a + b \log X_A & \text{if } \log X_A \leq d \\ c & \text{if } \log X_A \geq d \end{cases} \quad (41.1)$$

where  $Z \sim \mathcal{N}(0, \sigma^2)$ . Continuity means that  $a = c - bd$ ; we also need to impose the constraints that  $x_{\min} \leq X_D \leq x_{\max}$ .

Species become extinct with a probability that depends on their body mass,

$$p_e(x) = \beta x^\rho \quad (41.2)$$

All of this is shameless ripped off from <http://arxiv.org/abs/0901.0251> but Aaron said it was OK

Unless otherwise specified, you should use  $\sigma^2 = 0.63$ ;  $x_{\min} = 1.8$  grams and  $x_{\max} = 10^{15}$  grams;  $\rho = 0.025$ ;  $\beta = 1/5000$ ; and the values of  $b$ ,  $c$  and  $d$  from the solutions to Homework 4.

1. (10) Write a function, `rdeac.1`, which takes as inputs a single ancestral mass  $X_A$  (not  $\log X_A$ ), the parameters  $b$ ,  $c$ ,  $d$  and  $\sigma^2$ , and the limits  $x_{\min}$  and  $x_{\max}$ . It should generate a candidate value for  $X_D$  (not  $\log X_D$ ) from Eq. 41.1 and return it if it is between the limits, otherwise it should discard the candidate value and try again.
  - (a) (2) Set  $X_A$  to 40 grams and check, by simulating many times, that the output is always between  $x_{\min}$  and  $x_{\max}$ , even when those values are brought close to 40 grams.
  - (b) (8) Simulate a single  $X_D$  value for 100 values of  $X_A$  evenly spaced between 1 and 100 grams. Treat this as real data and re-estimate the parameters  $b$ ,  $c$  and  $d$  according to the methods of Homework 4; are they reasonably close to those in the simulation?
2. (10) Write a function, `rdeac`, which takes the same inputs as `rdeac.1` *plus* an integer  $n$ , and returns a vector containing  $n$  independent draws from this distribution. We will test this with  $n = 2$ , but your code must be more general for full credit.
  - (a) (4) Check, by simulating, that the first component of the returned vector has the same marginal distribution as the output of `rdeac.1`.
  - (b) (4) Check that the second component of the returned vector has the same marginal distribution as the first component.
  - (c) (2) Check that the two components are uncorrelated.
3. (10) Write a function, `speciate`, which takes the same arguments as `rdeac.1`, except that  $X_A$  is replaced by a vector of ancestral masses. The function should select one entry from the vector to be  $X_A$ , and generate two independent values of  $X_D$  from it. One of these should replace the entry for  $X_A$ , and the other should be added to the end of the vector.
  - (a) (2) Check, by simulating, the output always has one more entry than the input vector of masses, no matter how long the input is.
  - (b) (8) If the input has length  $n$ , check that  $n - 1$  of the entries in the output match the input.
4. (15) Write a function, `extinct.probs`, which takes as inputs a vector of species masses, an exponent  $\rho$ , and a baseline-rate  $\beta$ , and returns the extinction probability for each species, according to Eq. 41.2.
  - (a) (1) Check that if the input masses are 2 grams and 2500 grams, with the default parameters the output probabilities  $\approx 2.0 \times 10^{-4}$  and  $2.4 \times 10^{-4}$  respectively.

- (b) (2) Check that if  $\rho = 0$ , then the output probabilities are always  $\beta$ , no matter what the masses are.
- (c) (2) Check that if there input masses are all equal, then the output probabilities are all the same, no matter what  $\rho$  and  $\beta$  are.
- (d) (10) Write a function, `extinction`, which takes a vector of species masses,  $\rho$  and  $\beta$ , and returns a possibly-shorter vector which removes the masses of species which have been selected for extinction. *Hint:* What does `rbinom(n, size=1, prob=p)` do when  $p$  is a vector of length  $n$ ?
5. (15) *Evolve!*
- (a) (5) Write a function, `evolve.1`, which takes as inputs a vector of species masses,  $b$ ,  $c$ ,  $d$ ,  $\sigma^2$ ,  $x_{\min}$ ,  $x_{\max}$ ,  $\rho$  and  $\beta$ , and first does one speciation step, then one round of extinction, and returns the resulting vector of species masses.
- (b) (5) Write a function, `evolve`, which takes the same inputs as `evolve.1`, plus an integer  $t$ , and iterates `evolve.1`  $t$  times.
- (c) (5) How do you know that your functions are working properly?
6. (15) *Re-running history*
- (a) (5) Run `evolve` starting from a single species with a mass of 40 grams for  $t = 2 \times 10^5$  steps. Save the output vector of species masses as  $y_1$ . Plot the density of  $y_1$ .
- (b) (5) Repeat the last step to get a different vector  $y_2$ . Does it have the same distribution as  $y_1$ ? How can you tell?
- (c) (5) Change the initial mass to 1000 grams and get a vector of final masses  $y_3$ . How does its distribution differ from that of  $y_1$ ?
7. (25) The data file `MOM_data_full.txt` gives the masses of a large (and representative) sample of currently-living species of mammals. The column `mass` gives the mass in grams; the columns `species`, `genus`, `family`, `order` and `code` are identifiers for the particular species, which do not matter to us. Finally, the column `land` is 1 for species which live on land and 0 for those which live in the water.
- (a) (5) Load the data and plot the density of masses for land species.
- (b) (10) Describe, in words, how the distribution of current species masses compares to that produced by the simulation model in  $y_1$ .
- (c) (10) Use the relative distribution method from Chapter 17 to compare the actual distribution to the distribution of  $y_1$ . Describe the results and what they say about how the data differ from the model.

## Chapter 42

# Red Brain, Blue Brain

The data set n90\_pol.csv contains information on 90 university students who participated in a psychological experiment designed to look for relationships between the size of different regions of the brain and political views. The variables `amygdala` and `acc` indicate the volume of two particular brain regions known to be involved in emotions and decision-making, the amygdala and the anterior cingulate cortex; more exactly, these are residuals from the predicted volume, after adjusting for height, sex, and similar body-type variables. The variable `orientation` gives the subjects' locations on a five-point scale from 1 (very conservative) to 5 (very liberal). `orientation` is an ordinal but not a metric variable, so scores of 1 and 2 are not necessarily as far

[[TODO: Fix point totals apart as scores of 2 and 3.  
as off after hybridizing versions]]

1. (35) *Predicting brain sizes from political views*
  - (a) (5) Ignoring the fact that `orientation` is an ordinal variable, what is the correlation between it and the volume of the amygdala? Between `orientation` and the volume of the ACC?
  - (b) (5) Using case resampling, give 95% bootstrap confidence intervals for these correlations.
  - (c) (5) The function `rank`, applied to a data vector, returns the vector of ranks, where 1 indicates the smallest value, 2 the next-smallest, etc. What are the correlations between the ranks of `orientation` and the ranks of `amygdala`? Between `orientation` and `acc`? *Hint:* What does `cor(x,y,method="spearman")` do?
  - (d) (5) Using case resampling, give 95% bootstrap confidence intervals for the rank correlations.
  - (e) (15) Using `npcdens`, plot the conditional density of the volume of the amygdala as a function of political orientation. Do the same for the volume of the ACC. Make sure that in both cases you are treating `orientation` as an ordinal variable. You will be graded on how easy your plots are to read.

2. (10) *Creating a binary response variable*

- (a) (2) Create a vector, `conservative`, which is 1 when the subject has `orientation`  $\leq 2$ , and 0 otherwise.
- (b) (3) Explain why the cut-off was put at an orientation score of 2 (as opposed to some other cut-off).
- (c) (4) Check that your `conservative` vector has the proper values, *without* manually examining all 90 entries.
- (d) (1) Add `conservative` to your data frame. (Creating a new data frame with a new name will only get you partial credit.)

3. (10) *Logistic regression*

- (a) (5) Fit a logistic regression of `conservative` (not `orientation`) on `amygdala` and `acc`. Report the coefficients to no more than three significant digits. Explain what the coefficients mean.
- (b) (5) Using case resampling, give bootstrap standard errors and 95% confidence intervals for the coefficients. Was the restriction to three significant digits reasonable?

4. (10) *Generalized additive model*. Fit a generalized additive model for `conservative` on `amygdala` and `acc`. (Be sure to smooth both the input variables.) Make sure you are using a logistic link function. Report the intercept with reasonable precision. Plot the partial response functions, and explain what they mean (be careful!).

5. (15) *Kernel conditional probability estimation*

- (a) (5) Using `npcdens`, find the conditional probability of `conservative` given `amygdala` and `acc`. Make sure `npcdens` treats `conservative` as a categorical variable and not a continuous one. Report the bandwidths.
- (b) (5) Plot the estimated conditional probability that `conservative` is 1, with `acc` set to its median value and `amygdala` running over the range  $[-0.07, 0.09]$ . (The plotting range for `amygdala` exceeds the range of values found in the data.) *Hint:* your code will need to provide values for `acc`, for `amygdala` and for `conservative` (why?).
- (c) (5) Plot the estimated conditional probability that `conservative` is 1, with `amygdala` set to its median value and `acc` running over the range  $[-0.04, 0.06]$ . (This plotting range also requires extrapolating outside the data.)

6. *Probability surfaces* (15) For each of the three models, create a plot showing the estimated probability that `conservative` is 1, given `amygdala` or `acc`. The range for `amygdala` should be  $[-0.07, 0.09]$ , and the range for `acc` should be  $[-0.04, 0.06]$ . Compare and contrast the three plots.

Contour, wireframe and heatmap/level-plot plots are all acceptable, but all access must be clearly labeled with numerical scales, and, if you use color, there must be a color key.

*Hint:* use `predict`; be careful that you are predicting the right thing.

7. *Calibration* (15) Make calibration plots for each of the three models, as in [[chapter 13]] of the notes. Which models (if any) seem reasonably calibrated? Explain with reference to your plots.
8. *Classification* (10) The models from problems 3–5 predict probabilities for *conservative*. If we have to make a point prediction of whether someone is conservative or not, we should predict 1 if the probability is  $\geq 0.5$  and 0 otherwise. Find such predictions for each subject, under each of the three models. What fraction of subjects are mis-classified? What fraction would be mis-classified by “predicting” that none of them are conservative?
9. (10) *Summing up* Explain what you can conclude from this data about the relationship between brain anatomy and political orientation. Refer to your answers to earlier problems.

## Chapter 43

# Patterns of Exchange

There are many variables which influence the rate at which one currency is exchanged against another: changes in interest rates, changes in inflation rates, changes in country's trade balances, the actions of speculators and central banks, etc.<sup>1</sup> Often these variables act in almost the same way at the same time in countries which occupy similar positions in the global economy, so we might expect substantial correlation in movements of exchange rates. This suggests that it might be useful to try to reduce the large number of exchange rates to a smaller number of components.

The data set for this week, `fx.csv` on the class website, contains the sequence of exchange rates, against the Swiss franc<sup>2</sup>, of some commercially important and widely traded currencies: the US dollar, the Japanese yen, the Euro, and nineteen others. The data set runs from early 1995 to early 2010, with dates given as row names<sup>3</sup>. We will use principal components to analyze these exchange rate histories.

1. *Exploration and basic understanding* (10 points) Load the data file. It should have 3779 rows (trading days) and 22 columns (currencies).
  - (a) (3 points) The column names are abbreviated three-letter codes standing for currencies. What are the names of the countries and of the currencies? (*Hint:* use a search engine and/or the library.)
  - (b) (2 points) The exchange rate of dollars to yen on 28 February 2003 was approximately 118.22 yen to the dollar. Explain how to calculate this from the data. (*Hint:* The data give exchange rates for both in terms of Swiss francs.)
  - (c) (5 points) Produce basic numerical or graphical summaries for each currency. Do they seem to have the same range and distribution? Should they?

---

<sup>1</sup>See Barry Eichengreen's *Globalizing Capital: A History of the International Monetary System* (Princeton University Press, 1996) for an excellent introduction and history.

<sup>2</sup>We could give prices for all the currencies in terms of any one of them, and it won't matter which one is the "numeraire". This week's data is a cleaned-up part of `FXRatesCHF` from the `fxregime` package, whose authors are Swiss.

<sup>3</sup>For the first few years of the data, the "Euro" is really the old German deutsche mark.

2. *Data transformation* (10 points) Economic theory for exchange rates is mostly about changes to rates, and in fact mostly about the proportional changes, as measured by  $\log \frac{r_t}{r_{t-1}}$ .
  - (a) (5 points) Create a new data frame where each column gives the logarithmic change in the exchange rate for the corresponding currency in the original data. *Hint:* use `diff` and `apply`.
  - (b) (5 points) Produce the same sort of summaries for currencies as in problem 1c. Are they now more similar to one another, or less?
3. *Covariance matrix* (20 points)
  - (a) (8 points) Calculate the matrix of covariances between changes in exchange rates. This should be a  $22 \times 22$  matrix. Provide it as a table, with at most three decimal places for each entry.
  - (b) (8 points) Make a visual display of the matrix, using grey-scale, contour plots, color, or a perspective plot. Make sure the result is clearly legible when printed.
  - (c) (4 points) Are there any groups of currencies which seem particularly strongly correlated?
4. *PCA* (25 points)
  - (a) (4 points) Perform PCA on the *transpose* of the rate-of-change data frame, so that countries are rows and dates are columns. (*Hint:* use the `t()` function.) Include your code.
  - (b) (8 points) Make a plot of how much variance is accounted for by the first  $k$  components, for  $k$  from 1 to 22. How much variance do the first two components, taken together, retain? How many components are needed to capture 50% of the variance? To capture 90%?
  - (c) (9 points) Make a figure where the three-letter symbol for each currency is plotted according to the currency's score on the first two principal components. *Hint:* see examples in the notes for the lecture on PCA.
  - (d) (4 points) Describe any clusters or patterns you see in the plot from the previous part. Do these match the strongly correlated currencies from the previous question? Should they?
5. *Examining the components* (15 points)
  - (a) (5 points) Plot the sequence of weights (or “rotation”) for the first principal component. Using the row names of the data set, and the `axis` command, add regularly spaced dates to the horizontal axis.
  - (b) (5 points) Can you relate this time series to major economic events since 1995?
  - (c) (5 points) Repeat parts (a) and (b) for the second principal component.

6. *Reconstruction error* (10 points)

- (a) (2 points) Plot the logarithmic changes for the US dollar as a function of time (with dates as in part (a) of the previous problem).
- (b) (3 points) Plot the approximation to the dollar's history based on the first three principal components. Describe the similarities and differences.
- (c) (5 points) Calculate the mean squared error between the actual history of the dollar and the approximation based on the first  $q$  principal components, for  $q$  from 1 to 22.

EXTRA CREDIT (20 points): Fit a factor analysis model. Use hypothesis testing to determine the number of factors. Produce a figure like that of problem 5a for the leading factor. Comment on how it resembles or differs from the figure in 5a.

## Chapter 44

# Is This Assignment Really Necessary?

The dataframe `mathmarks` in the library `SMPRACTICALS`<sup>1</sup> contains scores for 88 university students in five mathematical subjects: vectors, algebra, analysis, statistics, and mechanics. All subjects were scored out of 100 points. We will use multivariate methods to explore how this data, and the relationships between the variables.

When asked to report numbers, give only two significant figures (not decimal places!) unless told otherwise.

1. (5) What is the sample correlation matrix among the grades?
2. (5) Find the least-squares coefficients for predicting the grade in statistics as a linear combination of the other four grades. What is the (in-sample) root-mean-squared error of this regression?
3. (10) Fit a factor model with one common factor, using `factanal`. Report the factor loadings and the uniquenesses, and explain what the numbers mean.
4. (10) Calculate the correlation matrix among the grades implied by the factor model. Why is it not the same as the sample correlation matrix? How does it differ?
5. (5) Calculate the covariance matrix implied by the factor model. How does it compare to the sample covariance matrix?
6. (10) Calculate the coefficients for predicting the grade in statistics as a linear combination of the other four grades, under the factor model. What is the (in-sample) root-mean-squared error? *Hint:* §14.2.2.
7. (5) The `factanal` function reports a test which compares the factor model to an unrestricted multivariate Gaussian. What is the  $p$ -value of this test, and what does it tell you about how well the factor model fits?

---

<sup>1</sup>You can also get the data file from <http://statwww.epfl.ch/davison/SM/>.

8. (10) Under the factor model, each variable should have a Gaussian distribution. Check this for each of the five grades, using `ddst.norm.test`, and describe your results.
9. (10) Using `mvnrmalmixEM` from the `mixtools` package, fit a multivariate Gaussian mixture model with two components. Report the mean vector of each Gaussian, its covariance matrix, and the two mixing weights.
10. *Mixture of Gaussians vs. factor model* (30 total)
  - (a) (10) Write a function to draw  $n$  vectors from a mixture of two multivariate Gaussians. *Hint:* select a random Gaussian (how?) and then call `rmvnorm`.
  - (b) (5) Write a function which takes a data frame, fits a one-factor model to it, and returns the  $p$ -value. Check that it works by making sure it gives the same answer as what you got in Problem 7
  - (c) (10) Repeatedly simulate data sets of size 88 from the two-Gaussian mixture you fit in problem 9. What fraction of them fit a one-factor model at least as well as the data?
  - (d) (5) What can you conclude about whether to prefer a one-factor or two-mixture model for this data?

## Chapter 45

# Mystery Multivariate Analysis

[[TODO: Put the data set on the website]]

1. *Initial exploration* (15 points)
  - (a) (4 points) Check whether the marginal distribution for each variable is Gaussian; both graphical and quantitative tests are acceptable.
  - (b) (5 points) Explain what the test you used in problem 1a does, and why it works.
  - (c) (3 points) Explain a *different* procedure that you could have used.
  - (d) (3 points) Explain why making a scatter-plot of variable values against row numbers would *not* let you check whether a distribution is Gaussian.
2. *A joint distribution* (15 points)
  - (a) (4 points) Using `npudens` from the `np` package, or any similar function, make a kernel density estimate of the joint distribution of `X.1` and `X.10`. Plot it; contour, color or perspective plots are all acceptable.
  - (b) (5 points) Fit a two-dimensional Gaussian to the same data, and plot it in the same way.
  - (c) (1 point) Plot the difference between the non-parametric and parametric density estimates. Comment.
  - (d) (5 points) Could you use the same procedure as in Problem 1 to check that the joint distribution is Gaussian? If so, explain how to modify it and why it still works. If not, explain why it cannot be adjusted, and describe a different procedure which you could use. EXTRA CREDIT (10 points): implement your test and report your results.
3. *Principal components* (15 points)
  - (a) (10 points) Find the first five principal components. Make a plot of these components. The horizontal axis should run over the integers from 1 to 10, the vertical axis should run from  $-1$  to  $+1$ , and the points should

indicate the projection of the components on to the corresponding observable variables. Put all five components in the same plot, using color or line style to distinguish them. (Alternately, use a three-dimensional plot.) Make sure the results come through clearly when printed. Comment on any patterns you see in the components. Grading *will* reflect how much visual clarity you can give this plot.

- (b) (5 points) Plot the amount of variance retained by the first  $q$  components vs.  $q$ , for up to 10 components.

4. *Factor analysis* (15 points)

- (a) (5 points) Fit a factor model with one factor. Plot the factor loadings of the ten observable variables, as in the previous problem. Does this match the first principal component? Should it?
- (b) (5 points) Fit a two-factor model, and plot the loadings of both factors. Has the first factor changed? If so, what does this mean? If not, is this a coincidence, or should the first factor never change when other factors are added?
- (c) (5 points) Plot the amount of variance retained by the first  $q$  factors vs.  $q$ , for up to 5 factors. Does this match the variance plot for PCA (up to five components)? Should it?

5. *Factor model selection* (20 points)

- (a) (7 points) Describe how to use the log likelihood ratio to select the number of factors.
- (b) (3 points) Of the models fit in the previous question, which one is favored by the log likelihood ratio test?
- (c) (7 points) Describe a way to test whether the discrepancy between your selected factor model and the data is significant.
- (d) (3 points) Is the discrepancy significant?

6. *Mystery R* (20 points)

```
myfunction <- function(x, t=100, eps=1e-2/sqrt(nrow(x))) {
 stopifnot(require(mixtools), is.array(x))
 n <- nrow(x)
 w <- rep(c(0,1), length.out=n)
 w <- sample(w)
 del <- Inf
 i <- 0
 while ((del > eps) && (i < t)) {
 i <- i+1
 l1 <- mean(abs(w))
 l2 <- 1 - l1
 m1 <- apply(x, 2, weighted.mean, w=w)
```

```

m2 <- apply(x,2,weighted.mean,w=(1-w))
s1 <- cov.wt(x,wt=w)$cov
s2 <- cov.wt(x,wt=(1-w))$cov
p1 <- dmvnorm(x,m1,s1)
p2 <- dmvnorm(x,m2,s2)
wn <- l1*p1/(l1*p1+l2*p2)
del <- max(abs(wn-w))
w <- wn
}
return(list(m1=m1,m2=m2,s1=s1,s2=s2,l1=l1,l2=l2,w=w,t=i,del=del))
}

```

- (a) (6 points) Explain what this function does. What are the inputs? What are the outputs?
- (b) (6 points) Explain what each line does.
- (c) (4 points) Is the `abs` necessary in the second line of the `while` loop? Is it necessary in the next-to-last line of the `while` loop?
- (d) (4 points) Run this function on your data. Describe the results.
7. *Mixtures* (EXTRA CREDIT, 50 points) Install the `mixtools` package from CRAN, and read sections 1, 2, and 6.1 of the paper describing it by Benaglia *et al.* (<http://www.jstatsoft.org/v32/i06>).

- (10 points) Using `mvnrmalmixEM()`, fit Gaussian mixture models to your data, varying the number of clusters or mixture components from 2 to 6. Plot the likelihood as a function of the number of clusters.

*Hint:* The default settings for `maxit` and `epsilon` will take forever; more reasonable ones here would be `maxit=100` and `epsilon=1e-1`. Explain, in your write-up, what these settings mean. Fitting the model with seven clusters might then still take up to an hour on a slow machine.

- (10 points) Describe how the `boot.comp` function works.
- (10 points) Use `boot.comp` to select the number of components to use.  
*Hint:* You will want to use fewer than the default number of bootstrap replicates, and also pass along the fitting arguments. Even so, this may take a very long time.
- (10 points) Describe a way to decide whether to use this selected mixture model, or the factor model you selected earlier, and explain why this comparison should be reliable.
- (10 points) Implement your comparison. Which model is favored?

## Chapter 46

# Separated at Birth

In many species of plants and animals, the genetic causes of bodily traits or behavior can be studied by breeding, or experimentally creating, organisms with specific genetic characteristics, and comparing different “strains” or “lines” which differ only in their genes, in known ways. This is not allowed for people<sup>1</sup>. To try to get around this, researchers in psychology and medicine have used what are called “twin designs”, which compare identical twins raised together to identical twins raised apart, hoping to indirectly isolate genetic influences on aspects of the body (e.g., height, weight, cholesterol) or behavior (e.g., extroversion, school grades, voting, belief in astrology)<sup>2</sup>. This problem set will use graphical causal models to examine the basis of such studies.

Unless specifically noted otherwise, you can assume that all variables are continuous, with mean zero, and each variable is a sum of its causal parents plus noise. You can assume that the noise term is independent and mean-zero for each variable, but you cannot assume that they all have the same variance; write the noise variance for  $X$  as  $\sigma_X^2$ .

[[TODO: Include a data set, obviously — KR might have one?]]

Figures 46.1 shows the models which are usually assumed in such studies, for identical twins raised together and identical twins raised apart. The letters on the arrows are the names of the path coefficients.

### 1. Paths and equations (10)

- (a) (4) For the models in Figure 46.1, write the equation for the trait of twin A in terms of its causal parents, the path coefficients, and the noise.
- (b) (1) How must these equations be changed for twin B?
- (c) (5) Explain why we can assume the path coefficients  $a$  and  $b$  are both always 1.

---

<sup>1</sup>Even if it was ethical, it wouldn't be very practical, because it takes so long to raise a new generation.

<sup>2</sup>All of these are traits which have actually been studied with twin designs. [[TODO: References]]

2. *Genetic component of variance* (25) The **heritability** of a trait is the fraction of its total variance which is genetically caused<sup>3</sup>. We will focus on the genetically-caused component of variance, i.e., the numerator of the heritability.
  - (a) (4) Find the variances of the trait of twin A in both models, in terms of the path coefficients and the variances of its causal parents.
  - (b) (1) Find the variances of the trait of twin B in both models.
  - (c) (5) Find the covariances between twin A and twin B in both models.
  - (d) (5) Find the variance of the genes in terms of the observable variances and covariances. (You may have to compare twins raised together to twins raised apart.)
  - (e) (5) Find an expression for the family-environment variance, in terms of the variances and covariances of the observable variables.
  - (f) (5) Can you find the heritability from the previous results? If so, what is it? If not, what else would you need to know?
3. *Not separated that far* (20) Twins “raised apart” are placed in families through the same process at the same time; in many documented cases they end up living with relatives of their parents, and sometimes grow up in the same town, attending the same schools. This is referred to as a **community effect**: see Figure 46.2.
  - (a) (5) Find the variances of the twins’ traits in the new models, in terms of the variances of the traits causal parents and the path coefficients. Explain why the path coefficients can all be taken to be 1.
  - (b) (5) Find the covariance between twins raised together and the covariance between twins raised apart in the new models.
  - (c) (5) Is your expression from Problem 2d still equal to the variance of the genes? If not, how would using the sample covariance between twins raised apart lead to a biased estimate of genetic variance?
  - (d) (5) Is your expression from Problem 2e still equal to the family-environment variance? If not, how would using the sample version of your expression lead to a biased estimate?
4. *Family proxies* (20) Suppose we can imperfectly measure family environment.
  - (a) (5) Modify the models in Figure 46.2 to include an observable variable which measures each family environment. Call its path coefficient  $m$ .
  - (b) (5) Does conditioning on the shared genes and the measurements of family environments d-separate the twins when they are raised apart? Does it d-separate them when they are raised together? (Show your work.)

---

<sup>3</sup>There is also a “narrow” heritability, which has to do with genetic causation which adds up across genes without interactions; we won’t try to deal with that.

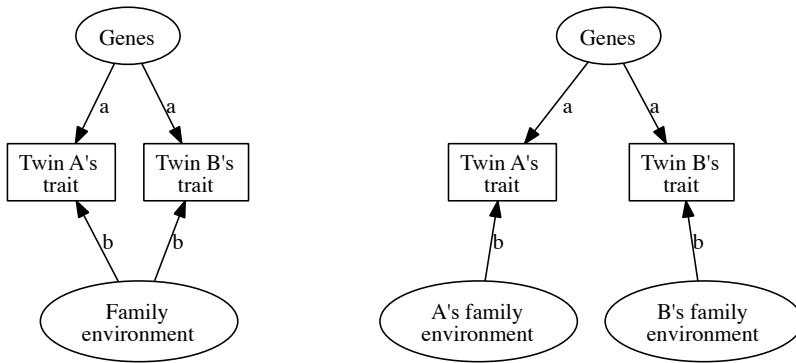


FIGURE 46.1: Standard models for twins raised together (left) and twins raised apart (right). Variables in boxes are observable, variables in ellipses are latent. Letters on edges indicate path coefficients.

- (c) (10) Conditional on the measurements of family environments, find the variances and covariances of twins raised together and twins raised apart. For this problem only, you may assume all variables are Gaussian, if that helps. *Hint:* Chapter 14.
- 5. *Not separated before birth* (25) Twins share the same environment for nine months (or so) before birth. This is called the **maternal environment** or **maternal effect**.
  - (a) (5) Draw graphical models including the maternal effect for both twins raised together and twins raised apart.
  - (b) (5) Find the variances of the twins' traits in the new models, in terms of the variances of the traits causal parents and the path coefficients. Explain why the path coefficients can all be taken to be 1.
  - (c) (5) Find the covariance between twins raised together and the covariance between twins raised apart in the new models.
  - (d) (5) Is your expression from Problem 2d still equal to the variance of the genes? If not, how would using the sample covariance between twins raised apart lead to a biased estimate of genetic variance?
  - (e) (5) Is your expression from Problem 2e still equal to the variance of the family environment? If not, how would using the sample version of your expression lead to a biased estimate?

[[TODO: More scaffolding here, this is where many students got lost]]

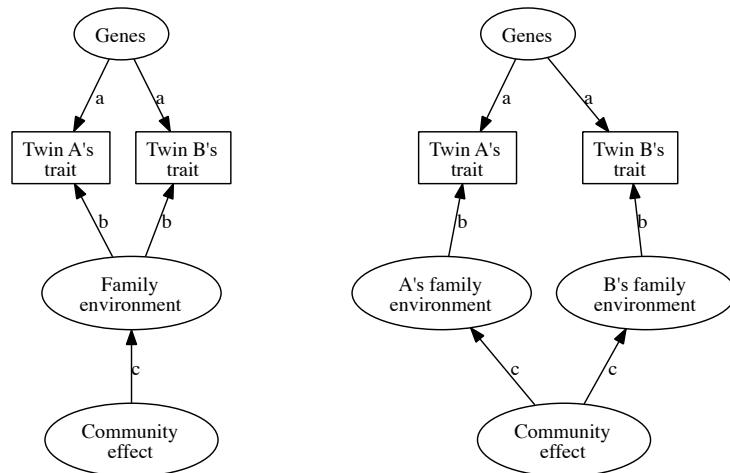


FIGURE 46.2: Models for twins raised together and apart, incorporating community effects.

## Chapter 47

# Brought to You by the Letters D, A and G

The file `sesame.csv` contains data on an experiment which sought to learn whether regularly watching *Sesame Street* caused an increase in cognitive skills, at least on average. The experiment consisted of randomly selecting some children, the treated, and *encouraging* them to watch the show, while others received no such encouragement. The children were tested before and after the experimental period on a range of cognitive skills. (Table 47.1 lists the variables.)

For questions that ask you to write code or manipulate data, include the relevant commands in the body of your answer.

1. *Data manipulation* (5) For each of the skills variables, find the difference between pre-test and post-test scores, and add the corresponding column to the data frame. Name these columns `deltabody`, `deltalet`, etc. Describe and run a check that the values in these columns are at least approximately right (without examining them all).
2. *Naive comparison* (5 total)
  - (a) (2) Find the mean `deltalet` scores for children who were regular watchers, and for children who were not regular watchers.
  - (b) (3) What must be assumed for the difference between these means to be a sound estimate of the average causal effect of switching from not watching to regularly watching *Sesame Street*? Is that plausible? Suggest a way the assumption could be tested.
3. “*Holding all else constant*” (20 total)
  - (a) (5) Linearly regress the change in reading scores on regular watching, and all other variables except `id`, `viewcat`, and the post-tests. (Be careful of which variables are categorical.) Report the coefficients to *reasonable*

precision. You will lose points for unjustified precision. *Hint:* R's default is definitely to report to unjustified precision.

- (b) (5) What would someone who had only taken 401 report as the average effect of making a child become a regular watcher of *Sesame Street*?
  - (c) (5) Explain why `id`, `viewcat`, and the `post` variables had to be left out of the regression. (The reasons need not all be the same.)
  - (d) (5) What would we have to assume for this to be a sound estimate of the average causal effect? Is that plausible?
4. (20 total) Consider the graphical model in Figure 47.1.
- (a) (10) Find a set of variables which satisfies the back-door criterion for estimating the effect of regular watching on `deltalet`.
  - (b) (5) Linearly regress `deltalet` on `regular` and the variables you selected in 4a. What is the corresponding estimate of the average effect of causing a child to become a regular watcher?
  - (c) (5) Do a kernel regression for the same variables. (Be careful about which variables are categorical.) Find the corresponding estimate of the average effect of causing a child to become a regular watcher.
5. (25 total) Consider the graphical model in Figure 47.2.
- (a) (5) There is at least one set of variables which meets the back-door criterion in Figure 47.2 which did not meet it in Figure 47.1. Find such a set, and explain why it meets the criterion in the new graph, but did not meet it in the old one.
  - (b) (5) Explain whether or not the set of control variables you found in 4a still works in the new graph.
  - (c) (5) Linearly regress `deltalet` on `regular` and the variables you selected in 5a. What is the corresponding estimate of the average causal effect of causing a child to become a regular watcher?
  - (d) (5) Do a kernel regression for the same variables. (Be careful about which variables are categorical.) Find the corresponding estimate of the average effect of causing a child to become a regular watcher.
  - (e) (5) Find a pair of variables which are conditionally (or marginally) independent in Figure 47.1 but are not in Figure 47.2, and vice versa. Explain why. *Note:* Both the conditioned and conditioning variables must be observed; the point is to find something which could be checked with the data.
  - (f) (Extra credit: 5) Test whether either of the two conditional independence relations from 5e hold in the data.
6. *Instrumental encouragement* (25 total) Some children were randomly selected for encouragement to watch *Sesame Street*. This is encoded in the variable `encour`.

- (a) (5) Explain why `encour` is a valid instrument in Figure 47.1. (You may need to also control for some other variables.)
- (b) (5) Explain why `encour` is a valid instrument in Figure 47.2. (You may need to also control for some other variables.)
- (c) (5) Describe a DAG in which `encour` would not be a valid instrument.
- (d) (5) Use the two-stage least-squares method to estimate the average effect of causing a child to become a regular watcher.

|                            |                                                                                                                                                                                                                                                                |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>id</b>                  | subject ID number                                                                                                                                                                                                                                              |
| <b>site</b>                | categorical; social background<br>1: Disadvantaged inner-city children, 3–5 yr old<br>2: Advantaged suburban children, 4 yr old<br>3: Advantaged rural children, various ages<br>4: Disadvantaged rural children<br>5: Disadvantaged Spanish-speaking children |
| <b>sex</b>                 | male=1, female=2                                                                                                                                                                                                                                               |
| <b>age</b>                 | in months                                                                                                                                                                                                                                                      |
| <b>setting</b>             | categorical; whether show was watched at home (1) or school (2)                                                                                                                                                                                                |
| <b>viewcat</b>             | categorical; frequency of viewing <i>Sesame Street</i><br>1: watched < 1/wk<br>2: watched 1 – –2/wk<br>3: watched 3 – –5/wk<br>4: watched > 5/wk                                                                                                               |
| <b>regular</b>             | 0: watched < 1/wk, 1: watched $\geq$ 1/wk                                                                                                                                                                                                                      |
| <b>encour</b>              | encouraged to watch = 1, not encouraged=0                                                                                                                                                                                                                      |
| <b>peabody</b>             | mental age, according to the Peabody Picture Vocabulary test<br>(to measure vocabulary knowledge)                                                                                                                                                              |
| <b>prelet, postlet</b>     | pre-experiment and post-experiment scores on knowledge of letters                                                                                                                                                                                              |
| <b>prebody, postbody</b>   | pre-test and post-test on body parts                                                                                                                                                                                                                           |
| <b>preform, postform</b>   | pre-test and post-test on geometric forms                                                                                                                                                                                                                      |
| <b>prenumb, postnumb</b>   | tests on numbers                                                                                                                                                                                                                                               |
| <b>prerelat, postrelat</b> | tests on relational terms                                                                                                                                                                                                                                      |
| <b>preclasf, postclasf</b> | pre-test and post-test on classification skills<br>("one of these things is not like the others")<br>("one of these things just doesn't belong")                                                                                                               |

TABLE 47.1: *Variables in the sesame data file. The pre- and post- experiment test scores are integers, but can be treated as continuous.*

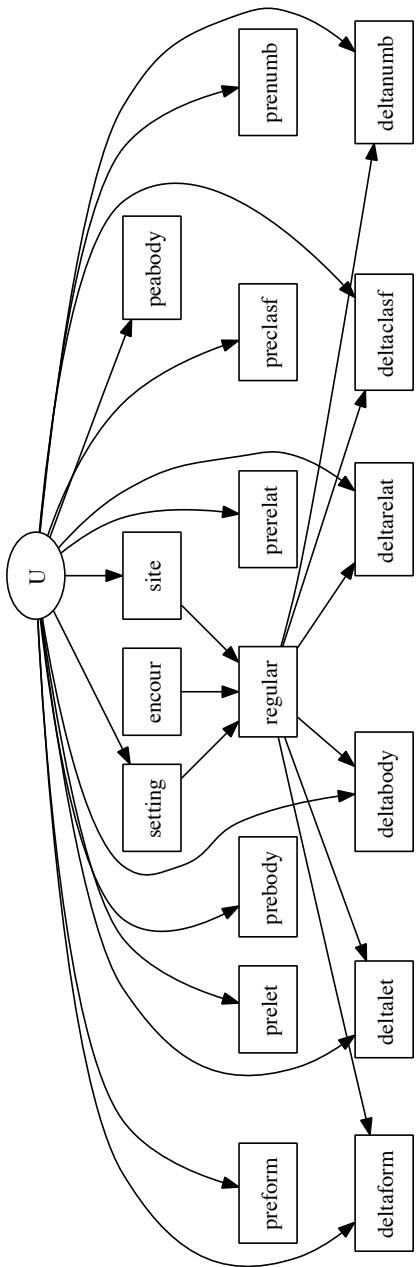


FIGURE 47.1: First DAG.

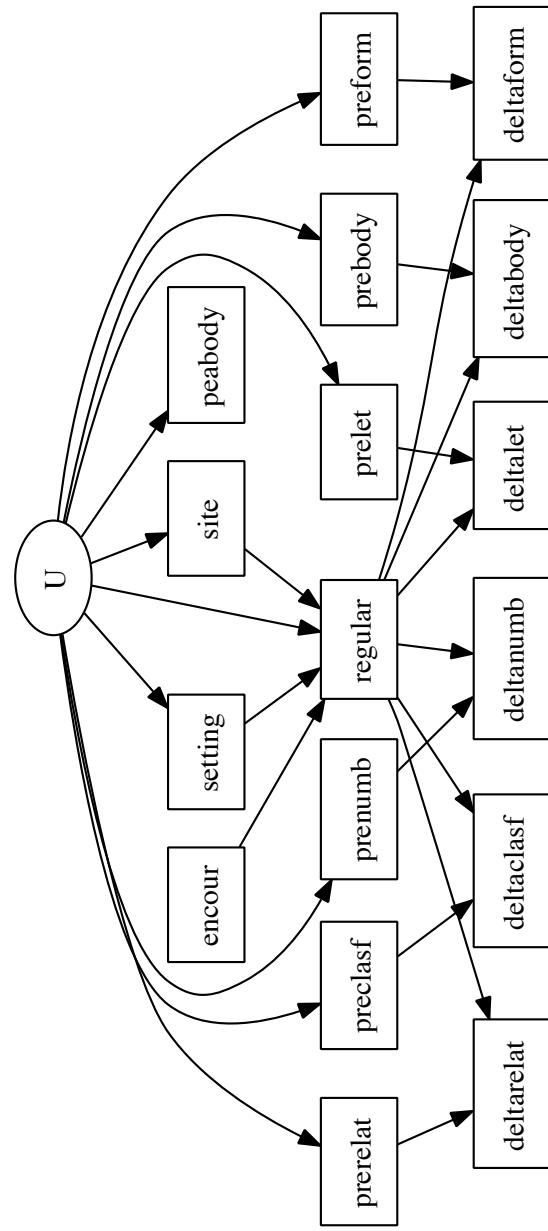


FIGURE 47.2: Second DAG.

## Chapter 48

# Estimating with DAGs

This homework will illustrate some of the advantages of using a known DAG structure. You will need to read the lectures on graphical models carefully in order to do it.

Figure 48.1 is an elaboration of the graph used in lectures. All problems refer to it, unless otherwise specified.

The file `fake-smoke.csv` contains some (synthetic) data, for use in problem 5.

1. *Parents and children* (10 points)
  - (a) (5 points) For each variable in the model, list its parents; or, if it has no parents, say so.
  - (b) (5 points) For each variable in the model, list its children. (Some variables have no children.)
2. *Joint distributions and factorization* (10 points) Using the graph, list the smallest collection of marginal and conditional distributions which must be estimated in order to get the joint distribution of all variables.
3. *Associations* (20 points) Should there be a positive association, a negative association, or no association between the following variables? Explain with reference to the graph. (2 points each)
  - (a) Yellowing of teeth and cancer?
  - (b) Yellowing of teeth and cancer, controlling for smoking?
  - (c) Yellowing of teeth and cancer, controlling for occupational prestige?
  - (d) Yellowing of teeth and cancer, controlling for smoking and exposure to asbestos?
  - (e) Smoking and cancer, controlling for the amount of tar in the lungs?
  - (f) Asbestos and cancer, controlling for cellular damage?
  - (g) Smoking and cancer, controlling for asbestos?

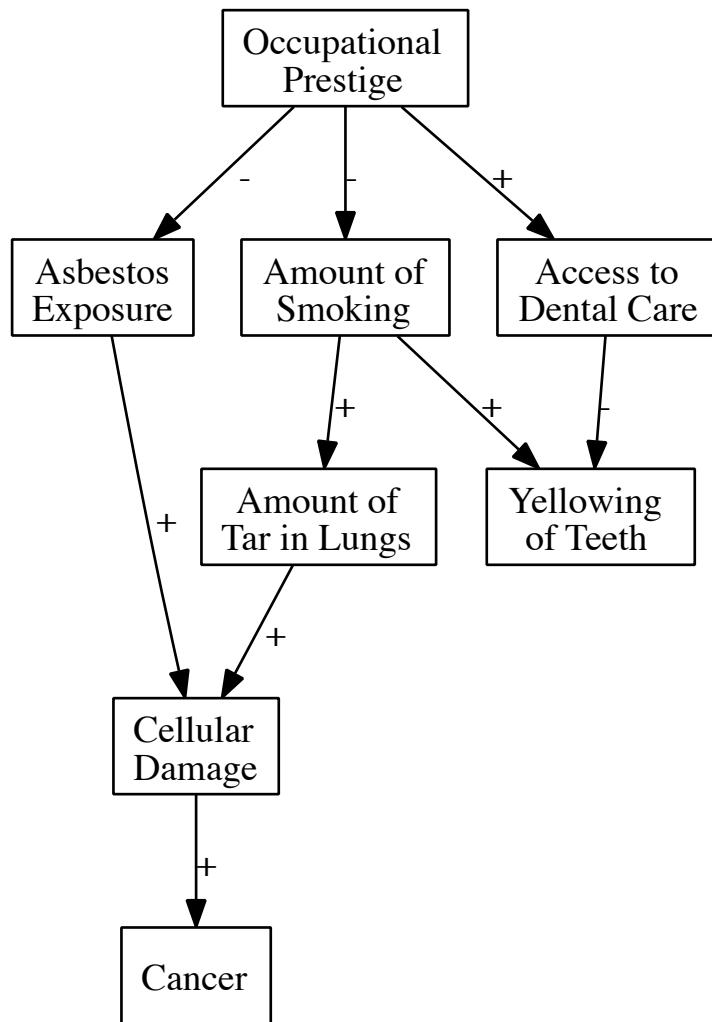


FIGURE 48.1: Graphical model for use in all problems, except part of the last. Signs on arrows indicate the sign of the associations (not necessarily linear) between parents and children.

- (h) Smoking and asbestos, controlling for cellular damage?
- (i) Tar in lungs and cancer, controlling for asbestos, smoking, and yellowing of teeth?
- (j) Smoking and cancer, controlling for asbestos and occupational prestige?

4. *Using conditional independence to specify regressions* (40 points)

- (a) (10 points) We wish to know the conditional risk of cancer given smoking. What other variables should be controlled for? Which other variables do not need to be controlled for?
- (b) (10 points) Using the `fake-smoke.csv` data from the class website, fit a logistic regression model for the risk of cancer given the level of smoking, controlling for any appropriate covariates.
- (c) (10 points) Using the same data set, fit another logistic regression for the risk of cancer using *all* the covariates. What does this say about the relationship between smoking and cancer? Why is this different than what is implied by the model in 4b?
- (d) (5 points) A medical insurance company needs to predict the risk of cancer among customers in order to set rates. Should it use the model from 4b or the one from 4c? Why? (Assume, for the sake of the problem, that the training data and the insurance customers are both representative samples of the general population.)
- (e) (5 points) A doctor wants to advise their patients about what actions to take to reduce their risk of cancer. Should they use the model from 4b or 4c? Why?

5. (20 points) Consider the alternative graph in Figure 48.2.

- (a) (10 points) Repeat problem 3 with the new graph. Clearly indicate in your response which associations differ for the two DAGs.
- (b) (10 points) Suggest an experiment, or an observational analysis, which could let us check which structure was right; explain, in terms of the graphs.

6. (10 points) EXTRA CREDIT: Which DAG did the example data come from? How can you tell?

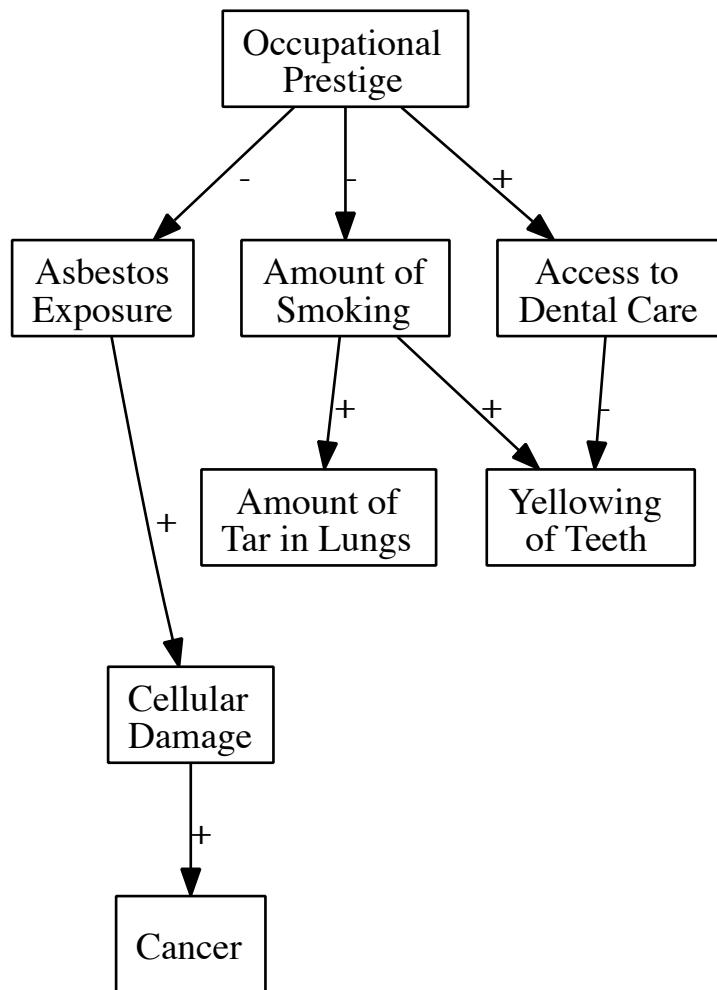


FIGURE 48.2: An alternative DAG for the same variables.

## Chapter 49

# Use and Abuse of Conditioning

1. (30 points) Refer to figure [[1]] in Problem Set 48.
  - (a) (5 points) Using the back door criterion, describe a way to estimate the causal effect of smoking on cancer.
  - (b) (5 points) Using the front door criterion, describe a *different* way to estimate the causal effect of smoking on cancer.
  - (c) (5 points) Is there a way to use instrumental variables to estimate the causal effect of smoking on cancer in this model? Explain.
  - (d) (5 points) Using your back-door identification strategy and the data file from last time, estimate  $\Pr(\text{cancer} = 1 | \text{do(smoking} = 1.5))$ .
  - (e) (5 points) Repeat this using your front-door identification strategy.
  - (f) (5 points) Do your two estimates of the causal effect match? Explain.
2. (25 points) Take the model in Figure 49.1. Suppose that  $X \sim \mathcal{N}(0, 1)$ ,  $Y = \alpha X + \epsilon$  and  $Z = \beta_1 X + \beta_2 Y + \eta$ , where  $\epsilon$  and  $\eta$  are mean-zero Gaussian noise with common variance  $\sigma^2$ . Set this up in R and regress  $Y$  twice, once on  $X$  alone and once on  $X$  and  $Z$ . Can you find any values of the parameters where the coefficient of  $X$  in the second regression is even approximately equal to  $\alpha$ ? (It's possible to solve this problem exactly through linear algebra instead.)
3. (25 points) Take the model in Figure 49.2 and parameterize it as follows:  $U \sim \mathcal{N}(0, 1)$ ,  $X = \alpha_1 U + \epsilon$ ,  $Z = \beta X + \eta$ ,  $Y = \gamma Z + \alpha_2 U + \xi$ , where  $\epsilon, \eta, \xi$  are independent Gaussian noises with mean zero and common variance  $\sigma^2$ . If you regress  $Y$  on  $Z$ , what coefficient do you get, on average? If you regress  $Y$  on  $Z$  and  $X$ ? If you do a back-door adjustment for  $X$ ? (Approach this either analytically or through simulation, as you like.)
4. (20 points) Continuing in the set-up of the previous problem, what coefficient do you get for  $X$  when you regress  $Y$  on  $Z$  and  $X$ ? Now compare this to the front-door adjustment for the effect of  $X$  on  $Y$ .

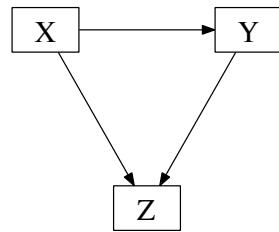


FIGURE 49.1: DAG for problem 2.

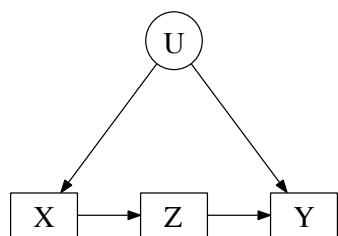


FIGURE 49.2: DAG for problems 3 and 4.

## Chapter 50

# What Makes the Union Strong?

Finding the factors which control the frequency and severity of strikes by organized workers is an important problem in economics, sociology and political science<sup>1</sup>. Our data set, <http://www.stat.cmu.edu/~cshalizi/uADA/12/hw/06/strikes.csv>, kindly provided by a distinguished specialist in the field, contains information about the incidence of strikes, and several variables which are plausibly related to that, for 18 developed (OECD) countries during 1951–1985:

- Country name
- Year
- Strike volume, defined as “days [of work] lost due to industrial disputes per 1000 wage salary earners”
- Unemployment rate (percentage)
- Inflation rate (consumer prices, percentage)
- “parliamentary representation of social democratic and labor parties”. (For the United States, this is the fraction of Congressional seats held by the Democratic Party.)
- A measure of the centralization of the leadership in that country’s union movement, on a scale of 0 to 1<sup>2</sup>.
- Union density, the fraction of salary earners belonging to a union (only available from 1960).

Note that some variables are missing (NA) for some cases.

[[TODO: Fix point assignments]]

---

<sup>1</sup>Or it used to be, anyway.

<sup>2</sup>This measure really should be a constant for each country over the period, but having a variable with only 8 levels is trouble for the spline smoother used in Problem 3, so a very small amount of artificial noise ( $\pm 0.005$  at most) has been added to each value.

1. Estimate a linear model to predict strike volume in terms of all of the other variables, *except* country and year.
  - (a) Report the coefficients, with 90% (not 95%) confidence intervals calculated according to
    - i. (2) The standard formulas
    - ii. (9) Resampling of the residuals
    - iii. (9) Resampling of the cases

Do not use more digits than you can justify.
  - (b) (10) Describe the meaning of the coefficients *qualitatively*. (I.e., do not write “A one unit change in foo produces a change of bar units in strike volume” over and over.)
  - (c) (5) Rank the predictor variables from most to least important, with “importance” measured by the magnitude of the predicted change to strike volume in response to a 1% relative change of the predictor away from its mean value.
  - (d) (5) Rank the predictor variables from most to least important in terms of predicted response to a 1 standard deviation change in the variable.
  - (e) (5) Do the two rankings agree? Should they? Which one seems more reasonable for this problem?
2. Some theories suggest that English-speaking countries have legal and political institutions which make strikes operate differently than in other industrialized countries. Figure out which countries in the data set are primarily English-speaking, create an indicator (dummy) variable for whether a case belongs to one of those countries, and add it to the data set.
  - (a) (5) Fit a linear model in which the predictors from Problem 1 interact with the English-using variable. Report the new coefficients (to *reasonable* precision)
  - (b) (5) Explain how (if at all) this model differs qualitatively from the model in Problem 1.
  - (c) (5) Use five-fold cross-validation to compare this model to the model in Problem 1. Which one does better?
3. Fit an additive model for strike volume as a smooth function of all the variables except country and year.
  - (a) (5) Plot all the partial response functions. Do they agree qualitatively with the conclusions you drew from the model in Problem 1?
  - (b) (5) Consider increasing each of the predictor variables by 1% from its mean, leaving the other variables alone. Rank the predictors according to the magnitude of this model’s predicted change in strike volume. Would the ranking be the same for a 1% decrease? *Hint:* use `predict` and a data frame with artificial data.

- (c) (5) Consider increasing each of the predictor variables by one standard deviation from its mean, leaving the other variables alone. Rank the predictors according to the magnitude of this model's predicted change in strike volume.
- (d) (5) Discuss the contrast (if any) between these rankings, and the corresponding ones for the linear model.
4. (10) Use the methods of Chapter 10 to test whether the linear model from Problem 1 is well-specified against an additive alternative.
5. *Continuing past the training data*
- (a) (2) What were the values of unemployment, inflation, union density, and `left.parliament` for the United States in 2009? *Hint:* You can get most of these from the last *The Statistical Abstract of the United States*.
- (b) (4) Assuming the union centralization variable for the US in 2009 was 0, what strike volume was predicted by (i) the model from problem 1, (ii) the English-is-different model from problem 2, and (iii) the additive model from problem 3?
- (c) (4) The actual strike volume for the United States in 2009 was 0.8. Is this plausible under any of the models? *Hint:* How much do you expect actual values to differ from predicted values?
6. (a) (5) Use `pc()` from `pca1g` to obtain a graph, assuming all relations between variables are linear. Report the causal parents (if any) and children (if any) of every variable. If the algorithm is unable to orient one or more of the edges, report this, and in later parts of this problem, consider all the graphs which result from different possible orientations.
- Note:* See <http://bactra.org/weblog/914.html> for help with installing `pca1g`. The most troublesome component is the `Rgraphviz` package. If you are unable to get `Rgraphviz` to work, you can still extract the information from the fitted model returned by `pc`: if that's `pc.fit`, then `pc.fit@graph@edgeL` is the "edge list" of the graph, listing, for each node, the nodes it has arrows to. With this information, you can make your own picture of the DAG.
- (b) (10) Linearly model each variable as a function of its parents. Report the coefficients (to reasonable precision), the standard deviation of the regression noise (ditto), and 95% confidence intervals for all of these, as determined by bootstrapping the residuals.
- (c) (10 total) You should find that strike volume and union density are not connected, but that there is at least one directed path linking them — either density is an ancestor of strike volume, or the other way around.
- i. (5) Find the expected change in the descendant from a one-standard-deviation increase in the ancestor above its mean value.

- ii. (5) Linearly regress the descendant on all the other variables, including the ancestor. According to this regression, what is the expected change in the descendant, when the ancestor increases one SD above its mean value and all other variables are at their mean values?
- (d) (15 total) Check the linearity assumption for each variable which has a parent. (Putting in interactions and/or quadratic terms is inadequate and will result in only partial credit at best.)
  - i. (5) Describe your method, and why it should work.
  - ii. (5) Report the  $p$ -value for each case, to reasonable precision.
  - iii. (5) What is your over-all judgment about whether it is reasonable to model each endogenous variable as linearly related to its parents? If you need more information than just  $p$ -values to reach a decision, describe it.
- (e) (10) Discuss the over-all adequacy of the model, on both statistical grounds (goodness-of-fit, appropriateness of modeling assumptions, etc.) and substantive, scientific ones (whether it makes sense, given what is known about the processes involved).

## Chapter 51

# An Insufficiently Random Walk Down Wall Street

In this assignment, you will work with a data set of historical values for the S&P 500 stock index, which also features in the notes. You will need to download `SPhistory.short.csv` from the class website. This data set records the actual *prices* of the index, say  $P_t$  on day  $t$ , but in finance we actually care about the returns,  $\frac{P_t}{P_{t-1}}$ , or about the logarithmic returns,

$$R_t = \log \frac{P_t}{P_{t-1}}$$

since we care more about whether we're making 1% on our investment than \$1 per share. In this assignment, "returns" always means "logarithmic returns".

Problems 2 and 3 are about estimating the first percentile of the return distribution,  $Q(0.01)$ , under various assumptions. The returns will be larger than this 99% of the time, so  $Q(0.01)$  gives an idea of how bad the bad performance will be, which is useful for planning. Note that a calendar year contains about 250 trading days, and so should average two or three days when returns are even worse than  $Q(0.01)$ . Problems 4 and 5 are about predicting future returns from historical returns, and the uncertainty in this. Doing all the bootstrapping for problem 5 may be time-consuming, and should not be left to the last minute.

1. (5) Load the data file, take the last column (containing the daily closing price), and calculate the logarithmic returns. Note that the file is in reverse chronological order (newest first). When you are done, if everything worked right, running `summary` on the returns series should give

| Min.      | 1st Qu.   | Median   | Mean      | 3rd Qu.  | Max.     |
|-----------|-----------|----------|-----------|----------|----------|
| -0.094700 | -0.006440 | 0.000467 | -0.000064 | 0.006310 | 0.110000 |

*Hint:* `help(rev)` and Recipe 14.8 in *The R Cookbook*.

[[TODO: Brad DeLong (!) points out by e-mail that one should really define returns here as  $\log((P_{t+1} + D_{t+1})/P_t)$ , where  $D$  is the dividend series — prices aren't the only thing that matters with the S&P! Obtain a historical dividend series, or a dividend-adjusted price series.]]

2. In finance, it is common to model daily returns as independent Gaussian variables.
  - (a) (5) Find the mean and standard deviation of the returns. What is  $Q(0.01)$  of the corresponding Gaussian distribution? *Hint:* `qnorm`.
  - (b) (5) Write an expression which will generate a series of independent Gaussian values of the same length as the returns, with the mean and standard deviation you found in 2a. Check that the mean and standard deviation of the output is *approximately* right, and that their histogram looks like a bell-curve.
  - (c) (10) Write a function which takes in a data vector, calculates its mean and standard deviation, and returns  $Q(0.01)$  according to the corresponding Gaussian distribution. Check that it works by seeing that it matches what the answer you got in 2a when run on the actual returns.
  - (d) (10) Using the code you wrote in 2b and 2c, find a 95% confidence interval for  $Q(0.01)$  from 2a. *Hint:* Look at the examples in the notes of parametric bootstrapping.
  - (e) (5 points) What is the first percentile of the data? Is it within the confidence interval you found in 2d? *Hint:* `quantile`.
3. (a) (5) Use `hist` to plot the histogram of returns. Also plot, on the same graph, the probability density function of the Gaussian distribution you fit in problem 2a. Comment on their differences.
- (b) (5) Write a function to resample the returns; it should generate a different random vector of the sample length as the data every time it is run. Check that running `summary` on these vectors produces results close to those on the data. *Hint:* Look at the examples in the notes of resampling.
- (c) (5) Write a function to calculate  $Q(0.01)$  from an arbitrary vector, *without* assuming a Gaussian distribution. Check that it works by seeing that its answer, when run on the real data, matches what you found in 2e.
- (d) (10) Using the code you wrote in 3b and 3c, find a 95% confidence interval for  $Q(0.01)$ . Compare this to your answer in 2d. Which is more believable, and why? *Hint:* Look at the examples in the notes of non-parametric bootstrapping.
4. (10) Using `npreg`, fit a kernel regression of  $R_{t+1}$ , tomorrow's returns, on  $R_t$ , today's returns. (Use the automatic bandwidth selector.) Report the selected bandwidth and the in-sample mean-squared error. Make a scatter-plot with  $R_t$  on the horizontal axis and  $R_{t+1}$  on the vertical axis, and add the estimated kernel regression function. Comment on the shape of the curve. *Hints:* Make a data frame with  $R_t$  as one column and  $R_{t+1}$  as another column. Also, see examples in the notes of plotting fitted models from `npreg`.
5. (25) *Uncertainty in the kernel regression*

- (a) (5) Write a function which resamples  $(R_t, R_{t+1})$  pairs from the returns series, and produces a new data frame of the same size as the original. Check that it works by running `summary` on it, and seeing that both columns *approximately* match the summaries of the data. *Hint:* look at the examples of resampling cases for regression in the notes.
- (b) (10) Write a function which takes a data frame with appropriately-named columns, and runs a kernel regression of  $R_{t+1}$  on  $R_t$ . It should return fitted values at 30 evenly-spaced values of  $R_t$  which span its observed range.
- (c) (10) Using your code from 5a and 5b, add 95% confidence bands for the kernel regression to your plot from problem 4. *Hint:* See the examples of plotting bootstrapped nonparametric regressions in the notes.

1. (5 points) Load the data file, take the last column (containing the daily closing price), and calculate the logarithmic returns. Note that the file is in reverse chronological order (newest first). When you are done, if everything worked right, running `summary` on the returns series should give

| Min.      | 1st Qu.   | Median   | Mean      | 3rd Qu.  | Max.     |
|-----------|-----------|----------|-----------|----------|----------|
| -0.094700 | -0.006440 | 0.000467 | -0.000064 | 0.006310 | 0.110000 |

*Hint:* Read the notes for 1 February.

2. In many applications in finance, it is common to model daily returns as independent Gaussian variables.
- (a) (5 points) Find the mean and standard deviation of the best-fitting Gaussian, and the  $Q(0.01)$  it implies.
- (b) (5 points) Write a function which simulates a data set of the same size as the real data, using the independent Gaussian model you fit in (2a), and returns a list, with components named `mean` and `sd`, containing the parameter values estimated from the simulation output.
- (c) (5 points) Write a function which takes as arguments a list with components named `mean` and `sd`, and returns the first percentile of the corresponding Gaussian distribution. Check that it works by verifying that when run with `mean` 5 and `sd` 2, it returns 0.347. *Hint:* Look at the examples in the notes of parametric bootstrapping.
- (d) (10 points) Using the code you wrote in (2b) and (2c), find a 95% confidence interval for  $Q(0.01)$  from (2a). *Hint:* Look at the examples in the notes of parametric bootstrapping.
- (e) (5 points) What is the first percentile of the data? Is it within the confidence interval you found in (2d)?

[[TODO: Integrate this version with the one above]]

[[TODO: Clarify "best-fitting" means maximum likelihood. Separate into two sentences. Clarify using `qnorm`.]]

[[TODO: Insisting on a list here and in the next part was a mistake, in the future just say 2 arguments]]

[[TODO: Clarify using quantile here.]]

3. (a) (5 points) Use `density()`, or any other suitable non-parametric density estimator, to plot the distribution of returns. Also plot, on the same graph, the Gaussian distribution you fit in problem 2. Comment on their differences.
- (b) (10 points) Write a function to re-sample the returns, and calculate  $Q(0.01)$  on each surrogate data set. Use this to find a 95% confidence interval for  $Q(0.01)$ . *Hint:* Look at the examples in the notes of non-parametric bootstrapping.
4. (15 points) In an **autoregressive** model, the measurement at time  $t$  is regressed on the measurement at time  $t - 1$ ,  $X_t = \phi_0 + \phi_1 X_{t-1} + \epsilon_t$ . Use `lm` to fit an autoregressive model to the returns. Give the estimates of  $\phi_0$ ,  $\phi_1$  and  $\text{Var}[\epsilon]$ , and try to interpret what they mean. Also give the reported standard error for  $\widehat{\phi}_1$ .
5. *Hint:* Look at the examples in the notes of re-sampling regression residuals.
  - (a) (5 points) Write a function which re-samples the residuals of the autoregressive model from (4). Check that the mean and standard deviation of its output are close to those of the residuals.
  - (b) (15 points) Write a function which simulates the autoregressive model you fit in (4), with noise provided by the function you wrote for (5a).
  - (c) (5 points) Write a function which takes a time series, fits an autoregressive model, and returns the estimate of  $\phi_1$ . Check that it works by seeing that when it's given the data, the output matches what you found in (4).
  - (d) (10 points) Using the function you wrote in (5c), and the simulator you wrote in (5b), find the bootstrap standard error for  $\widehat{\phi}_1$ . Does it match what `lm` reported in (4)?

*Note:* If you cannot solve (5b), you can get full credit for (5d) using the built-in function `arima.sim` instead, but make sure that the distribution of innovations or noise comes from the function you wrote in (5a). If you cannot solve (5a), you can get full credit for (5b) and (5d) by providing suitable Gaussian noise.

## Chapter 52

# Macroeconomic Forecasting

The data set <http://www.stat.cmu.edu/~cshalizi/uADA/13/exams/3/macro.csv> on the class website contains five standard macroeconomic time series for the United States, from the beginning of 1948 to the beginning of 2010: total national income or GDP; value of goods consumed; investment spending; hours worked; and output per hour worked for all non-financial firms. (Some of these series are in inflation-adjusted dollars, some of them are in hours, and some of them are indexes where a particular date has been set as 100 and others are expressed relative to that.) All variables are measured “quarterly”, i.e., four times a year.

Most macroeconomic forecasting models do not concern themselves directly with these values, but only with the logged fluctuations around their long-run trends.

For full credit on the modeling questions, you must use models which go beyond those available in 401, or you must use appropriate methods to show that linear model are justified here.

It is first necessary to remove trends; macroeconomists traditionally do this with the following function.

```
hpfilter <- function(y, w=1600){
 eye = diag(length(y))
 d = diff(eye,d=2)
 ybar = solve(eye + w*crossprod(d), y)
 yhat = log(y) - log(ybar)
 return(list(fluctuation=yhat,trend=ybar))
}
```

1. (10) Create five plots, showing each of the variables and its trend (as returned by `hpfilter`) as functions of time. Use a logged scale for the vertical axis. Report  $R^2$ , with and without logging, for each of the five trends.
2. (10) Plot the logged fluctuations around trend (as returned by `hpfilter`) for each of the five variables. Does it make sense to compare these fluctuations across variables? Do the fluctuations look stationary? — After this problem,

references to the variables always mean their logged fluctuations around their trends.

3. (10) Are the variables Gaussian? (You can do better than looking at a histogram.)
4. (20) For the first four variables (GDP, consumption, investment, hours worked), fit an additive regression of each variable on the values of all four at the previous time-step. Use only data up to, but not including, 2005 (“the training period”). Report the mean squared error on the training data (to reasonable precision), and include plots of the partial response functions. Describe, in words, what the partial response functions say about the relations between these variables.
5. (20 total) Using the circular block bootstrap, with blocks of length 24, generate new time series which are as long as the training data.
  - (a) (4) Write a function to calculate the mean squared errors of the fitted models from Problem 4, on a time series. (Each of the four variables should have its own MSE.) Check that it works by making sure that it gives the right answer for the training data.
  - (b) (6) Report the mean MSEs, and the standard error of these means, from enough bootstrap replicates that the standard errors are no more than 10% of the means.
  - (c) (10) What do you need to assume for the numbers from 5b to be good estimates of the generalization error of this model?
6. (20 total) “Real” (as opposed to “monetary”) business cycle theories hold that fluctuations in macroeconomic variables are ultimately caused by exogenous “real shocks”, especially changes to productivity. The productivity variable in `macro.csv` is a measurement of this variable, which, according to these theories, should be exogenous. The other variables, in such theories, are endogenous.
  - (a) (10) Fit a model for each of the four endogenous variables, as an additive function of the endogenous variables in the previous quarter, and productivity for the previous four quarters. Report the MSEs and include plots of the partial response functions. Compare the plots to those in Problem 4.
  - (b) (4) Describe a method which could be used to decide whether including productivity in this way really improves predictive performance. Discuss the assumptions of the method, and why you think they apply here.
  - (c) (6) Implement your method. For which variables does including productivity actually help? How confident are you of this conclusion?
7. (10 total) Now consider the period 2005–2010. What are the mean squared errors, on this data, of
  - (a) (4) Predicting according to the additive model from Problem 4?

- (b) (4) Predicting according to the additive model from Problem 6?
  - (c) (2) Predicting the mean of each variable, as estimated from the training period?
8. (5, extra credit) Explain how what `hpfilter` does is related to spline smoothing.

## Chapter 53

# Debt Needs Time for What It Kills to Grow In

An important and controversial question in macroeconomics and political economy is whether high levels of government debt causes the economy to grow more slowly or even shrink. There are several plausible-sounding reasons why it might<sup>1</sup>; some economists claim that there is a threshold level of debt, perhaps around 90% of GDP, above which growth rates plummet.

Against this, there are other reasons why high levels of debt might *not* cause growth to slow, at least not always<sup>2</sup>. In particular, since “high levels of government debt” are defined relative to the size of the economy, as a high ratio of debt to GDP, slow growth itself might cause higher levels of government debt.

This week’s data set contains information on GDP and government debt for a selection of countries since World War II. For each country and year, we should have the GDP (nominal, i.e., not adjusted for inflation or differences in exchange rates) and the size of government debt (also nominal). Unfortunately, one or both values may be missing for some countries in some years.

1. (10) The data set contains a variable, *growth*, which is the annual growth rate in real (inflation-adjusted) GDP for each country and year. It also contains a variable, *ratio* which is the ratio of government debt to GDP. Make a scatter-plot with *growth* on the vertical axis and *ratio* on the horizontal. Describe the patterns you see, if any.

---

<sup>1</sup>High levels of government borrowing might “crowd out” investing in the private sector, by using up available savings and/or raising the interest rates at which businesses can borrow; capitalists might anticipate that the debt will either be paid off through high taxes or discharged through inflation, and prefer to spend their money on luxuries now, rather than invest and see the investment go away later; high levels of debt might lead to lower confidence that the government generally knows what it’s doing, making investment seem too risky; etc.

<sup>2</sup>A depressed economy has unused resources, so government employment needn’t lead to crowding out; the things government spends money on (roads, schools, hospitals, basic research, honest markets) increase the value of private investments; governments which can borrow large sums are receiving a market endorsement of their willingness and ability to pay their debts; etc.

2. (15) Run a nonparametric regression of `growth` on `ratio`, and plot the resulting curve. Describe and interpret the curve. Does it suggest an abrupt slowing of growth above some threshold level of debt?
3. (10) Since changes in government debt levels might take some time to affect economic growth, we would like to compare growth in year  $t + 1$  to `ratio` in year  $t$ . Create a new variable, `growth.lead1`, which records for each country/year the *next* year's GDP growth, with NAs in the right places when it is not available. Describe, in words, how your code works. Add `growth.lead1` to the data frame.

*Hints:* Make sure that you do not confuse growth rates from different countries (so that, e.g., the last year for Austria gets a growth rate from Belgium). You may find Recipes 14.7 (and 6.6) from *The R Cookbook* helpful.

4. (10) Plot `growth.lead1` against `ratio`, and do a nonparametric regression of the former on the latter. Describe the results, and compare them to those of Problem 2.
5. (15) Economic growth rates tend to be rather persistent over time within countries. Estimate an additive model where `growth.lead1` is predicted from `growth` and `ratio`. Is the partial response to the previous year's growth nearly linear? Should it be? Compare the partial response function for debt to the curves from problems 2 and 4.
6. (10) Create a new variable, `growth.lag1`, which represents the *previous* year's growth rate (with NAs in appropriate places), and add it to the data set. Plot it against `ratio` and fit a nonparametric regression. Does `ratio` do a better job of predicting `growth` or `growth.lag1`?
7. (15) Estimate an additive model in which the current year's `ratio` is predicted by last year's `ratio`, last year's `growth`, and the current year's `growth`. (You may have to create a new column.) Describe the partial response functions, and whether any predictor variables could be dropped.
8. (15) Explain what we would have to assume for the model in Problem 5 to give us an unconfounded estimate of the causal effect of government debt on future economic growth; be as specific as possible. (You may want to draw some DAGs, and include them in your write-up.) Comment on how plausible those assumptions are, and on what might go wrong if the assumptions fail.

# Appendices

## Appendix A

# Reminders from Linear Algebra

[[TODO: Actually write; some chunks of this are in various chapters]]

This appendix is in no way a replacement for actually learning linear algebra; it's only intended for actual reminders and quick reference.

### A.1 Eigenvalues and Eigenvectors of Matrices

A non-zero vector  $\vec{v}$  is an **eigenvector** of a square matrix  $\mathbf{a}$ , with **eigenvalue**  $\lambda$ , when

$$\mathbf{a}\vec{v} = \lambda\vec{v} \tag{A.1}$$

If  $\vec{v}$  is an eigenvector, then  $b\vec{v}$  is also an eigenvector with eigenvalue  $\lambda$ . It's usual to **normalize** the eigenvector to have length 1, or perhaps to have its entries sum to 1.

An  $p \times p$  matrix  $\mathbf{a}$  has at most  $p$  distinct eigenvalues. If one puts the eigenvalues in order, the largest one is the **leading** eigenvalue, and the corresponding eigenvector is also the leading eigenvector. A matrix's **spectrum** is its set of eigenvalues.

Eigenvectors with distinct eigenvalues are orthogonal.

If a matrix has fewer than  $p$  distinct eigenvalues, the “repeated” ones are **degenerate**. If  $\lambda$  is degenerate, then there are at least two vectors  $\vec{v}_1$  and  $\vec{v}_2$  such that  $\mathbf{a}\vec{v}_1 = \lambda\vec{v}_1$  and  $\mathbf{a}\vec{v}_2 = \lambda\vec{v}_2$ . Therefore for any  $b, c$ ,  $b\vec{v}_1 + c\vec{v}_2$  is also an eigenvector with eigenvalue  $\lambda$ . If  $\lambda$  is  $k$ -fold degenerate, then there is  $(k - 1)$ -dimensional space of eigenvectors with eigenvalue  $\lambda$ .

[[trace, determinant, characteristic polynomial; possibility of complex eigenvalues]]

#### A.1.1 Singular Value Decomposition

[[write]]

### A.1.2 Square Root of a Matrix

Any matrix  $\mathbf{b}$  such that  $\mathbf{b}^T \mathbf{b} = \mathbf{a}$  is a **square root** of  $\mathbf{a}$ . There are infinitely many of them, but a fairly straightforward one can be defined through the singular value decomposition:

$$\mathbf{a}^{1/2} = \lambda^{1/2} \mathbf{v}$$

where  $\lambda^{1/2}$  takes the square root of each element of  $\lambda$ .

## A.2 Special Kinds of Matrix

A square matrix is **symmetric** when it is equal to its own transpose,  $\mathbf{a}^T = \mathbf{a}$ .

A square matrix is **positive semi-definite** when, for all non-zero vectors  $\vec{v}$ ,  $\vec{v} \cdot \mathbf{a}\vec{v} \geq 0$ . If  $\vec{v} \cdot \mathbf{a}\vec{v} > 0$ , then the matrix is **positive definite**. All the eigenvectors of a positive semi-definite matrix are  $\geq 0$ ; all the eigenvectors of a positive definite matrix are  $> 0$ . [[Squiggly signs for PSD and PD.]]

A matrix is **orthogonal** when it is inverse is the same as its transpose,  $\mathbf{o}^T = \mathbf{o}^{-1}$ . All eigenvalues of an orthogonal matrix have magnitude 1. Since  $\mathbf{o}^T \mathbf{o} = \mathbf{I}$ , if  $\mathbf{b}^T \mathbf{b} = \mathbf{a}$ , then  $(\mathbf{ob})^T (\mathbf{ob})$  is also equal to  $\mathbf{a}$ .

## A.3 Orthonormal Bases

A collection of vectors  $v_1, \dots, v_p$  are a **basis** for a vector space  $\mathcal{V}$  when, given any  $v \in \mathcal{V}$ ,

$$v = \sum_{j=1}^p a_j v_j$$

The basis is **orthogonal** when  $\vec{v}_i \cdot \vec{v}_j = 0$  unless  $i = j$ , and **normal** when each  $\vec{v}_i$  has length 1. If it is both orthogonal and normal, the basis is **orthonormal**. The unit vectors along the different coordinate axes form an orthonormal basis, so no orthonormal basis has to have more basis vectors than  $\mathcal{V}$  has dimensions. (A non-orthogonal basis might need more.)

If a matrix is non-degenerate, then its eigenvectors form an orthonormal basis.

If a basis is orthonormal, then the coefficients  $a_j$  in the expansion are just given by inner products:

$$v = \sum_{j=1}^p (v \cdot v_j) v_j$$

## A.4 Orthogonal Projections

[[Vector = its projection on to your favorite space + the part orthogonal to the space]]

## A.5 Function Spaces

Spaces of functions can often be treated as linear vector spaces. There are several possible definitions of inner product, but the usual one is

$$\langle f, g \rangle = \int f(x)g(x)dx$$

This leads to a norm for functions,

$$\|f\|_2^2 = \int f^2(x)dx$$

Functions where  $\|f\|_2 < \infty$  are **square-integrable**, and the space of square-integrable functions is called  $L_2$  (or sometimes  $L^2$ ). (Exercise: Suggest a definition of  $L_p$ , without looking it up.) Sometimes it is convenient to modify the definition of inner product to

$$\langle f, g \rangle = \int f(x)g(x)\mu(x)dx$$

and

$$\|f\|_2^2 = \int f^2(x)\mu(x)dx$$

and then the space of function is called  $L_2(\mu)$ . You can check (EXERCISE) that the inner product is linear in both  $f$  and  $g$ , with or without the factor of  $\mu$ .

Similarly, one can define inner products and square-integrability for functions on a restricted domain, e.g.,  $\int_0^1 f^2(x)dx$ , or combine a domain restriction with a non-uniform base distribution.

### A.5.1 Bases

Because  $\langle f, g \rangle$  acts like an ordinary inner product, lots of what's familiar from vector spaces carries over to  $L_2$ . In particular, it makes sense to speak of a sequence of functions  $\psi_1, \psi_2, \dots$  being a basis, and even of its being an orthonormal basis, in which case

$$f = \sum_{j=1}^{\infty} \langle f, \psi_j \rangle \psi_j$$

Notice that the mononomials  $1, x, x^2, \dots$  are a basis for  $L_2$  on  $[0, 1]$ , but they are neither orthogonal nor normalized. [[On the other hand, Fourier basis.]]

### A.5.2 Eigenvalues and Eigenfunctions of Operators

An **operator**  $\mathcal{O}$  is a function which maps functions to other functions. (You know two from calculus. Taking a derivative transforms one function,  $f$ , into another,  $df/dx$ . Likewise integration transforms  $f$  into  $\int_{x=-\infty}^a f(x)dx$ , a function of  $a$ .)

Operators can have eigenvalues and eigenfunctions, just as matrices have eigenvalues and eigenvectors:

$$\mathcal{O}f = \lambda f$$

You know examples of this, too, from calculus:  $e^{\alpha x}$  is an eigenfunction of both differentiation and integration. (What are the eigenvalues?)

As the last example suggests, in general operators on function spaces have infinitely many eigenvalues and eigenvectors.

## A.6 Further Reading

There are many, many decent references on linear algebra, often as part of a more general reference on mathematical methods, e.g., Boas (1983). Axler (1996) is notable for presenting the whole subject “from the ground up”, but at the same time from the abstract perspective characteristic of modern mathematics (as opposed to sheer calculational procedures). I’m not sure how easy it would be to learn it from there, but it’s an interesting perspective.

## 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  $c g(n) = O(f(n))$  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

# Taylor Expansions

[[TODO: write appendix!]]

[[What a Taylor series is]]

A function whose Taylor series converges everywhere is **analytic**.

[[Error of truncating a Taylor series at a given order]]

[[Taylor series in more than one dimension]]

[[TODO: Replace with a  
copyright-free picture, or get  
permission]]

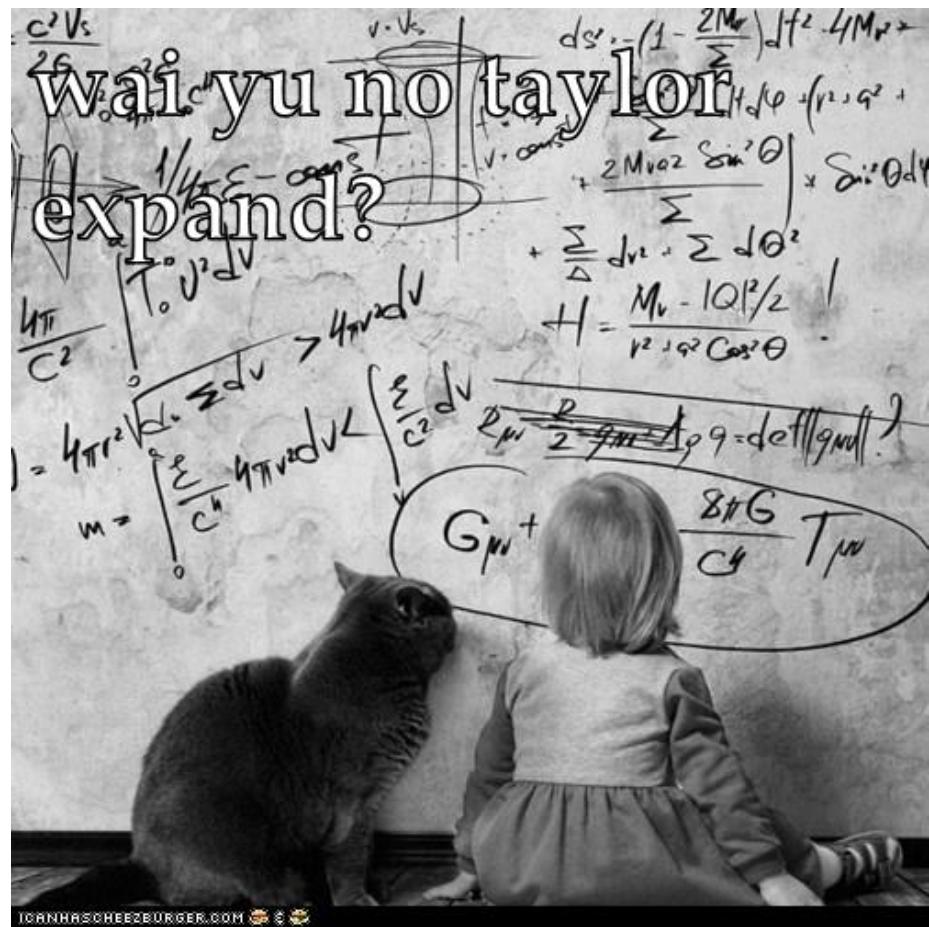


FIGURE C.1: Sound advice when stuck on almost any problem in statistical theory.

## Appendix D

# Propagation of Error, and Standard Errors for Derived Quantities

A reminder about how we get approximate standard errors for functions of quantities which are themselves estimated with error.

Suppose we are trying to estimate some quantity  $\theta$ . We compute an estimate  $\hat{\theta}$ , based on our data. Since our data is more or less random, so is  $\hat{\theta}$ . One convenient way of measuring the purely statistical noise or uncertainty in  $\hat{\theta}$  is its standard deviation. This is the **standard error** of our estimate of  $\theta$ .<sup>1</sup> Standard errors are not the only way of summarizing this noise, nor a completely sufficient way, but they are often useful.

Suppose that our estimate  $\hat{\theta}$  is a function of some intermediate quantities  $\hat{\psi}_1, \hat{\psi}_2, \dots, \hat{\psi}_p$ , which are also estimated:

$$\hat{\theta} = f(\hat{\psi}_1, \hat{\psi}_2, \dots, \hat{\psi}_p) \quad (\text{D.1})$$

For instance,  $\theta$  might be the difference in expected values between two groups, with  $\psi_1$  and  $\psi_2$  the expected values in the two groups, and  $f(\psi_1, \psi_2) = \psi_1 - \psi_2$ . If we have a standard error for each of the original quantities  $\hat{\psi}_i$ , it would seem like we should be able to get a standard error for the **derived quantity**  $\hat{\theta}$ . There is in fact a simple if approximate way of doing so, which is called **propagation of error**<sup>2</sup>.

We start with (what else?) a Taylor expansion (App. C). We'll write  $\psi_i^*$  for the

---

<sup>1</sup>It is not, of course, to be confused with the standard deviation of the data. It is not even to be confused with the standard error of the mean, unless  $\theta$  is the expected value of the data and  $\hat{\theta}$  is the sample mean.

<sup>2</sup>Or, sometimes, the **delta method**.

true (ensemble or population) value which is estimated by  $\widehat{\psi}_i$ .

$$f(\psi_1^*, \psi_2^*, \dots, \psi_p^*) \approx f(\widehat{\psi}_1, \widehat{\psi}_2, \dots, \widehat{\psi}_p) + \sum_{i=1}^p (\psi_i^* - \widehat{\psi}_i) \frac{\partial f}{\partial \psi_i} \Big|_{\psi=\widehat{\psi}} \quad (\text{D.2})$$

$$f(\widehat{\psi}_1, \widehat{\psi}_2, \dots, \widehat{\psi}_p) \approx f(\psi_1^*, \psi_2^*, \dots, \psi_p^*) + \sum_{i=1}^p (\widehat{\psi}_i - \psi_i^*) \frac{\partial f}{\partial \psi_i} \Big|_{\psi=\widehat{\psi}} \quad (\text{D.3})$$

$$\widehat{\theta} \approx \theta^* + \sum_{i=1}^p (\widehat{\psi}_i - \psi_i^*) f'_i(\widehat{\psi}) \quad (\text{D.4})$$

introducing  $f'_i$  as an abbreviation for  $\frac{\partial f}{\partial \psi_i}$ . The left-hand side is now the quantity whose standard error we want. I have done this manipulation because now  $\widehat{\theta}$  is a linear function (approximately!) of some random quantities whose variances we know, and some derivatives which we can calculate.

Remember the rules for arithmetic with variances: if  $X$  and  $Y$  are random variables, and  $a$ ,  $b$  and  $c$  are constants,

$$\text{Var}[a] = 0 \quad (\text{D.5})$$

$$\text{Var}[a + bX] = b^2 \text{Var}[X] \quad (\text{D.6})$$

$$\text{Var}[a + bX + cY] = b^2 \text{Var}[X] + c^2 \text{Var}[Y] + 2bc \text{Cov}[X, Y] \quad (\text{D.7})$$

While we don't know  $f(\psi_1^*, \psi_2^*, \dots, \psi_p^*)$ , it's constant, so it has variance 0. Similarly,  $\text{Var}[\widehat{\psi}_i - \psi_i^*] = \text{Var}[\widehat{\psi}_i]$ . Repeatedly applying these rules to Eq. D.4,

$$\text{Var}[\widehat{\theta}] \approx \sum_{i=1}^p (f'_i(\widehat{\psi}))^2 \text{Var}[\widehat{\psi}_i] + 2 \sum_{i=1}^{p-1} \sum_{j=i+1}^p f'_i(\widehat{\psi}) f'_j(\widehat{\psi}) \text{Cov}[\widehat{\psi}_i, \widehat{\psi}_j] \quad (\text{D.8})$$

The standard error for  $\widehat{\theta}$  would then be the square root of this.

If we follow this rule for the simple case of group differences,  $f(\psi_1, \psi_2) = \psi_1 - \psi_2$ , we find that

$$\text{Var}[\widehat{\theta}] = \text{Var}[\widehat{\psi}_1] + \text{Var}[\widehat{\psi}_2] - 2 \text{Cov}[\widehat{\psi}_1, \widehat{\psi}_2] \quad (\text{D.9})$$

just as we would find from the basic rules for arithmetic with variances. The approximation in Eq. D.8 comes from the nonlinearities in  $f$ .

If the estimates of the initial quantities are uncorrelated, Eq. D.8 simplifies to

$$\text{Var}[\widehat{\theta}] \approx \sum_{i=1}^p (f'_i(\widehat{\psi}))^2 \text{Var}[\widehat{\psi}_i] \quad (\text{D.10})$$

and, again, the standard error of  $\widehat{\theta}$  would be the square root of this. The special case of Eq. D.10 is sometimes called *the* propagation of error formula, but I think it's better to use that name for the more general Eq. D.8.

# Appendix E

# Optimization Theory

[[TODO: Multiple sections either need to be written from the beginning, or seriously re-written]]

Many statistical problems are conveniently cast as optimization problems. This is particularly true of finding point estimates. This appendix therefore reviews some basic ideas of optimization (§E.1), the asymptotics of optimizing noisy objective functions (§E.2) and its implications for maximum likelihood (§E.2.1), and the theory of constrained and penalized optimization (§§E.3.1–E.3.3), including constrained linear regression (ridge regression and the lasso) as an application (§E.3.4). Appendix F covers numerical algorithms for approximately solving optimization problems.

## E.1 Basic Concepts of Optimization

[[Agenda: Optima, local and global, relation to 2nd derivatives, transformations, importance of convexity]]

Following is yanked from slides for statistical computing, hence fragmentary

- Examples of Optimization Problems
  - Minimize mean-squared error of regression surface (Gauss, c. 1800) [[TODO: Re-read Stigler for the actual history]]
  - Maximize likelihood of distribution (Fisher, c. 1918)
  - Maximize output of plywood from given supplies and machines (Kantorovich, 1939)
  - Maximize output of tanks from given supplies and factories; minimize number of bombing runs to destroy factory (any number of Allied mathematicians, c. 1939–1945)
  - Maximize return of portfolio for given volatility (Markowitz, 1950s)
  - Minimize cost of airline flight schedule (Kantorovich vs. the Americans, 1950s)

- Maximize reproductive fitness of an organism (Maynard Smith)

- Mathematically: Given an **objective function**  $f : \mathcal{D} \mapsto R$ , find

$$\theta^* = \operatorname{argmin}_{\theta} f(\theta)$$

- Basics: maximizing  $f$  is minimizing  $-f$ :

$$\operatorname{argmin}_{\theta} -f(\theta) = \operatorname{argmax}_{\theta} f(\theta)$$

- If  $h$  is strictly increasing (e.g., log), then

$$\operatorname{argmin}_{\theta} f(\theta) = \operatorname{argmin}_{\theta} h(f(\theta))$$

- Considerations

- Approximation: How close can we get to  $\theta^*$ , and/or  $f(\theta^*)$ ?
- Time complexity: How many computer steps does that take?

Varies with precision of approximation, niceness of  $f$ , size of  $\mathcal{D}$ , size of data, method...

Often optimization algorithms use **successive approximation**, so distinguish number of iterations from cost of each iteration

- As you remember from calculus...

- Suppose  $x$  is one dimensional and  $f$  is smooth

If  $x^*$  is an **interior minimum / maximum / extremum point**

$$\left. \frac{df}{dx} \right|_{x=x^*} = 0$$

If  $x^*$  a minimum,

$$\left. \frac{d^2f}{dx^2} \right|_{x=x^*} > 0$$

(Note: not only if, consider  $f(x) = x^4$  and the minimum at  $x^* = 0$ )

- This all carries over to multiple dimensions:

At an **interior extremum**,

$$\nabla f(\theta^*) = 0$$

At an **interior minimum**,

$$\nabla^2 f(\theta^*) \geq 0$$

meaning for any vector  $v$ ,

$$v^T \nabla^2 f(\theta^*) v \geq 0$$

$\nabla^2 f$  = the **Hessian**,  $H$

$\theta$  might just be a **local minimum**

[[TODO: Cross-ref. to linear algebra appendix on matrix positivity]]

## E.2 Small-Noise Asymptotics for Optimization

The basic ideas underlying asymptotic estimation theory are very simple; most presentations rather cloud the basics, because they include lots of detailed conditions needed to show *rigorously* that everything works.

[[TODO: Very rough draft, re-work for notational consistency, tone, etc.]]

We have a statistical model, which tells us, for each sample size  $n$ , the probability that the observations  $X_1, X_2, \dots, X_n \equiv X_{1:n}$  will take on any particular value  $x_{1:n}$ , or the probability density if the observables are continuous. This model contains some unknown parameters, bundled up into a single object  $\theta$ , which we need to calculate those probabilities. That is, the model's probabilities are  $m(x_{1:n}; \theta)$ , not just  $m(x_{1:n})$ . Because this is just baby stats., we'll say that there are only a finite number of unknown parameters, which don't change with  $n$ , so  $\theta \in \mathbb{R}^d$ . Finally, we have a loss function, which tells us how badly the model's predictions fail to align with the data:

$$\lambda_n(x_{1:n}, m(\cdot; \theta)) \quad (\text{E.1})$$

For instance, each  $X_i$  might really be a  $(U_i, V_i)$  pair, and we try to predict  $V_i$  from  $U_i$ , with loss being mean-weighted-squared error:

$$\lambda_n = \frac{1}{n} \sum_{i=1}^n \frac{(v_i - \mathbb{E}_\theta[V_i | U_i = u_i])^2}{\text{Var}_\theta[V_i | U_i = u_i]} \quad (\text{E.2})$$

(If I don't subscript expectations  $\mathbb{E}[\cdot]$  and variances  $\text{Var}[\cdot]$  with  $\theta$ , I mean that they should be taken under the true, data-generating distribution, whatever that might be. With the subscript, calculate assuming that the  $m(\cdot; \theta)$  distribution is right.)

Or we might look at the negative mean log likelihood,

$$\lambda_n = -\frac{1}{n} \sum_{i=1}^n \log m(x_i | x_{1:i-1}; \theta) \quad (\text{E.3})$$

Being simple folk, we try to estimate  $\theta$  by minimizing the loss:

$$\hat{\theta}_n = \underset{\theta}{\operatorname{argmin}} \lambda_n \quad (\text{E.4})$$

We would like to know what happens to this estimate as  $n$  grows. To do this, we will make two assumptions, which put us at the mercy of two sets of gods.

The first assumption is about what happens to the loss functions.  $\lambda_n$  depends both on the parameter we plug in and on the data we happen to see. The latter is random, so the loss at any one  $\theta$  is really a random quantity,  $\Lambda_n(\theta) = \lambda_n(X_{1:n}, \theta)$ . Our first assumption is that these random losses tend towards non-random limits: for each  $\theta$ ,

$$\Lambda_n(\theta) \rightarrow \ell(\theta) \quad (\text{E.5})$$

where  $\ell$  is a deterministic function of  $\theta$  (and nothing else). It doesn't particularly matter to the argument *why* this is happening, though we might have our suspicions<sup>1</sup>, just that it is. This is an appeal to the gods of stochastic convergence.

Our second assumption is that we always have a unique interior minimum with a positive-definite Hessian: with probability 1,

$$\nabla \Lambda_n(\hat{\theta}_n) = 0 \quad (\text{E.6})$$

$$\nabla \nabla \Lambda_n(\hat{\theta}_n) > 0 \quad (\text{E.7})$$

(All gradients and other derivatives will be with respect to  $\theta$ ; the dimensionality of  $x$  is irrelevant.)

Moreover, we assume that the limiting loss function  $\ell$  also has a nice, unique interior minimum at some point  $\theta^*$ , the minimizer of the limiting, noise-free loss:

$$\theta^* = \operatorname{argmin}_{\theta} \ell \quad (\text{E.8})$$

$$\nabla \ell(\theta^*) = 0 \quad (\text{E.9})$$

$$\nabla \nabla \ell(\theta^*) > 0 \quad (\text{E.10})$$

Since the Hessians will be important, I will abbreviate  $\nabla \nabla \Lambda_n(\hat{\theta}_n)$  by  $\mathbf{H}_n$  (notice that it's a random variable), and  $\nabla \nabla \ell(\theta^*)$  by  $\mathbf{j}$  (notice that it's not random).

These assumptions about the minima, and the derivatives at the minima, place us at the mercy of the gods of optimization.

To see that these assumptions are not *empty*, here's an example. Suppose that our models are Pareto distributions for  $x \geq 1$ ,  $m(x; \theta) = (\theta - 1)x^{-\theta}$ . Then  $\lambda_n(\theta) = \overline{\log x_n} - \log(\theta - 1)$ , where  $\overline{\log x_n} = n^{-1} \sum_{i=1}^n \log x_i$ , the sample mean of the log values. From the law of large numbers,  $\ell(\theta) = \theta \mathbb{E}[\log X] - \log(\theta - 1)$ . To show the convergence, Figure E.1 plots  $\lambda_{10}$ ,  $\lambda_{1000}$  and  $\lambda_{10^5}$  for a particular random sample, and the corresponding  $\ell$ . I chose this example in part because the Pareto distribution is heavy tailed, and I actually generated data from a parameter value where the variance of  $X$  is infinite (or undefined, for purists). The objective functions, however, converge just fine.

With these assumptions made, we use what is about the only mathematical device employed in statistical theory at this level, which is a Taylor expansion. Specifically, we expand the gradient  $\nabla \Lambda_n$  around  $\theta^*$ :

$$\nabla \Lambda_n(\hat{\theta}_n) = 0 \quad (\text{E.11})$$

$$\approx \nabla \Lambda_n(\theta^*) + \mathbf{H}_n(\hat{\theta}_n - \theta^*) \quad (\text{E.12})$$

$$\hat{\theta}_n = \theta^* - \mathbf{H}_n^{-1} \nabla \Lambda_n(\theta^*) \quad (\text{E.13})$$

The first term on the right hand side,  $\theta^*$ , is the population/ensemble/true minimizer of the loss. If we had  $\ell$  rather than  $\Lambda_n$ , we'd get that for the location of the

---

<sup>1</sup>"In fact, all epistemologic value of the theory of the probability is based on this: that large-scale random phenomena in their collective action create strict, nonrandom regularity" — Gnedenko and Kolmogorov (1954, p. 1).

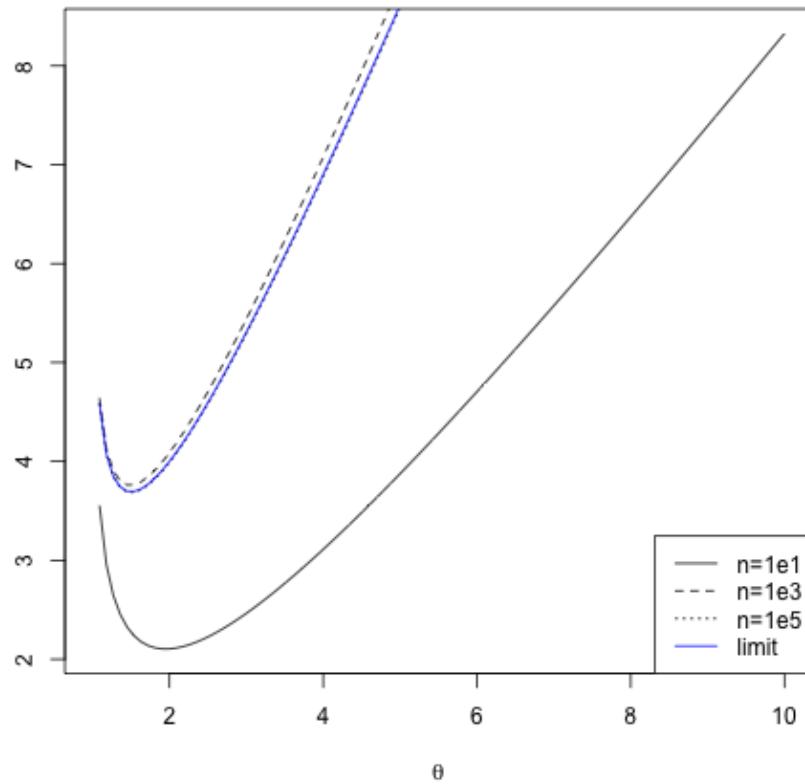


FIGURE E.1: Convergence of objective functions, here, negative average log-likelihoods. Note that the limiting,  $n = \infty$  objective function (solid blue line) is extremely close to what we see at  $n = 10^5$  already. See Code Example 46 for code.

```

source("pareto.R")
for pareto.R, which includes rpareto() and pareto.loglike(), see
http://www.santafe.edu/~aaronc/powerlaws/pli-R-v0.0.3-2007-07-25.tgz
Generate 10^6 samples from a Pareto distribution with exponent 1.5
Note that Var(X) = infinity because exponent < 2
x <- rpareto(1e6,threshold=1,exponent=1.5)
Define the negative mean log-likelihood function on the first n samples
lambda.once <- function(n,theta) {
 -(1/n)*pareto.loglike(x[1:n],threshold=1,exponent=theta)
}
Vectorize over theta for plotting
lambda <- Vectorize(lambda.once)
Start with curve based on first ten samples
curve(lambda(n=10,theta=x),from=1,to=10,xlab=expression(theta),ylab="")
Add curved based on first 10^3
curve(lambda(n=1000,theta=x),add=TRUE,lty="dashed")
Add curved based on first 10^5
curve(lambda(n=1e5,theta=x),add=TRUE,lty="dotted")
Add curve for the infinite-sample limit
Uses E[log(X)]=2 from knowledge of the Pareto
curve(x*2 - log(x-1),add=TRUE,col="blue")
Decorate with a legend
legend("bottomright",legend=c("n=1e1","n=1e3","n=1e5","limit"),
lty=c("solid","dashed","dotted","solid"),
col=c("black","black","black","blue"))

```

CODE EXAMPLE 46: *Code for Figure E.1.*

minimum. But since we see  $\ell$  corrupted by noise, we need to include the extra term  $-\mathbf{H}_n^{-1}\nabla\Lambda_n(\theta^*)$ . The Hessian  $\mathbf{H}_n$  tells us how sharply curved  $\Lambda_n$  is near its minimum; the bigger this is, the easier, all else being equal, to find the location of the minimum. The other factor,  $\nabla\Lambda_n(\theta^*)$ , indicates how much noise there is — not so much in the function being minimized, as in its gradient, since after all  $\nabla\ell(\theta^*)=0$ . We would like  $\hat{\theta}_n \rightarrow \theta^*$ , so we have to convince ourselves that the rest is asymptotically negligible, that  $\mathbf{H}_n^{-1}\nabla\Lambda_n(\theta^*)=o(1)$ .

Start with the Hessian.  $\mathbf{H}_n$  is the matrix of second derivatives of a random function:

$$\mathbf{H}_n(\hat{\theta}_n) = \nabla\nabla\Lambda_n(\hat{\theta}_n) \quad (\text{E.14})$$

Since  $\Lambda_n \rightarrow \ell$ , it would be reasonable to hope that

$$\mathbf{H}_n(\hat{\theta}_n) \rightarrow \nabla\nabla\ell(\hat{\theta}_n) = \mathbf{j}(\hat{\theta}_n) \quad (\text{E.15})$$

We'll assume that everything is nice ("regular") enough to let us "exchange differentiation and limits" in this way. Since  $\mathbf{H}_n(\hat{\theta}_n)$  is tending to  $\mathbf{j}(\hat{\theta}_n)$ , it follows that  $\mathbf{H}_n = O(1)$ , and consequently  $\mathbf{H}_n^{-1} = O(1)$  by continuity. With more words: since  $\Lambda_n$  is tending towards  $\ell$ , the curvature of the former is tending towards the curvature of the latter. But this means that the *inverse* curvature is also stabilizing.

Our hope then has to be the noise-in-the-gradient factor,  $\nabla\Lambda_n(\theta^*)$ . Remember again that

$$\nabla\ell(\theta^*)=0 \quad (\text{E.16})$$

and that  $\Lambda_n \rightarrow \ell$ . So if, again, we can exchange taking derivatives and taking limits, we do indeed have

$$\nabla\Lambda_n(\theta^*) \rightarrow 0 \quad (\text{E.17})$$

and we're done. More precisely, we've established **consistency**:

$$\hat{\theta}_n \rightarrow \theta^* \quad (\text{E.18})$$

Of course it's not enough just to know that an estimate will converge, one also wants to know something about the uncertainty in the estimate. Here things are mostly driven by the fluctuations in the noise-in-the-gradient term. We've said that  $\nabla\Lambda_n(\theta^*)=o(1)$ ; to say anything more about the uncertainty in  $\hat{\theta}_n$ , we need to posit more. It is very common to be able to establish that  $\nabla\Lambda_n(\theta^*) = O(1/\sqrt{n})$ , often because  $\Lambda_n$  is some sort of sample- or time- average, as in my examples above, and so an ergodic theorem applies. In that case, because  $\mathbf{H}_n^{-1} = O(1)$ , we have

$$\hat{\theta}_n - \theta^* = O(1/\sqrt{n}) \quad (\text{E.19})$$

If we can go further, and find

$$\text{Var}[\nabla\Lambda_n(\theta^*)] = \mathbf{k}_n \quad (\text{E.20})$$

then we can get a variance for  $\hat{\theta}_n$ , using propagation of error:

$$\text{Var} [\hat{\theta}_n] = \text{Var} [\hat{\theta}_n - \theta^*] \quad (\text{E.21})$$

$$= \text{Var} [-\mathbf{H}_n^{-1} \nabla \Lambda_n(\theta^*)] \quad (\text{E.22})$$

$$\approx \mathbf{j}^{-1}(\theta^*) \text{Var} [\nabla \Lambda_n(\theta^*)] \mathbf{j}^{-1}(\theta^*) \quad (\text{E.23})$$

$$= \mathbf{j}^{-1} \mathbf{k}_n \mathbf{j}^{-1} \quad (\text{E.24})$$

the infamous <strong>sandwich covariance matrix</strong>. If  $\nabla \Lambda_n(\theta^*) = O(1/\sqrt{n})$ , then we should have  $n \mathbf{k}_n \rightarrow \mathbf{k}$ , for a limiting variance  $\mathbf{k}$ . Then  $n \text{Var} [\hat{\theta}_n] \rightarrow \mathbf{j}^{-1} \mathbf{k} \mathbf{j}^{-1}$ .

Of course we don't know  $\mathbf{j}(\theta^*)$ , because that involves the parameter we're trying to find, but we can estimate it by  $\mathbf{j}(\hat{\theta}_n)$ , or even by  $\mathbf{H}_n^{-1}(\hat{\theta}_n)$ . That still leaves getting an estimate of  $\mathbf{k}_n$ . If  $\Lambda_n$  is an average over the  $x_i$ , as in my initial examples, then we can often use the variance of the gradients at each data point as an estimate of  $\mathbf{k}_n$ . In other circumstances, we might actually have to think.

Finally, if  $\nabla \Lambda_n(\theta^*)$  is asymptotically Gaussian, because it's governed by a central limit theorem, then so is  $\hat{\theta}_n$ , and we can use Gaussians for hypothesis testing, confidence regions, etc.

A case where we can short-circuit a lot of thinking is when the model is correctly specified, so that the data-generating distribution is  $m(\cdot; \theta^*)$ , and the loss function is the negative mean log-likelihood. (That is, we are maximizing the likelihood.) Then the negative of the limiting Hessian  $\mathbf{j}$  is the **Fisher information**. More importantly, under reasonable conditions, one can show that  $\mathbf{j} = \mathbf{k}$ , that the variance of the gradient is exactly the limiting negative Hessian. Then the variance of the estimate simplifies to just  $\mathbf{j}^{-1}$ . This turns out to actually be the best variance you can hope for, at least with unbiased estimators (the **Cramér-Rao inequality**). But the bulk of the analysis doesn't depend on the fact that we're estimating by maximum likelihood; it goes the same way for minimizing any well-behaved objective function.

Now, there are a *lot* of steps here where I am being very loose. (To begin with: In what sense is the random function  $\Lambda_n$  converging on  $\ell$ , and does it have to converge everywhere, or just in a neighborhood of  $\theta^*$ ?) That is, I am arguing like a physicist. Turning this sketch into a rigorous argument is the burden of good books on asymptotic statistics. But this is the core. It does not require the use of likelihood, or correctly specified models, or independent data, just that the loss function we minimize be converging, in a well-behaved way, to a function which is nicely behaved around its minimum.

[[Further reading: Barndorff-Nielsen and Cox (1995); Geyer (2013); Huber (1967); van der Vaart (1998)]]

### E.2.1 Application to Maximum Likelihood

### E.2.2 Aside: The Akaike Information Criterion

[[appendix to come, from other notes]]

## E.3 Constrained and Penalized Optimization

### E.3.1 Constrained Optimization

Suppose we want to minimize<sup>2</sup> 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

$$\frac{\partial L}{\partial u} = 0 \quad (\text{E.25})$$

$$\frac{\partial L}{\partial v} = 0 \quad (\text{E.26})$$

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:

$$\frac{dL}{dv} = \frac{\partial L}{\partial v} + \frac{\partial L}{\partial u} \frac{\partial h}{\partial v} \quad (\text{E.27})$$

which we then set to zero and solve for  $v$ . Except in quite rare cases, this is messy.

### E.3.2 Lagrange Multipliers

A superior alternative is the method of **Lagrange multipliers**. We introduce a new variable  $\lambda$ , the Lagrange multiplier, and a new objective function, the **Lagrangian**,

$$\mathcal{L}(u, v, \lambda) = L(u, v) + \lambda(g(u, v) - c) \quad (\text{E.28})$$

which we minimize with respect to  $\lambda$ ,  $u$  and  $v$  and  $\lambda$ . That is, we solve

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \quad (\text{E.29})$$

$$\frac{\partial \mathcal{L}}{\partial u} = 0 \quad (\text{E.30})$$

$$\frac{\partial \mathcal{L}}{\partial v} = 0 \quad (\text{E.31})$$

Notice that minimize  $\mathcal{L}$  with respect to  $\lambda$  always gives us back the constraint equation, because  $\frac{\partial \mathcal{L}}{\partial \lambda} = g(u, v) - c$ . Moreover, when the constraint is satisfied,  $\mathcal{L}(u, v, \lambda) =$

---

<sup>2</sup>Maximizing  $L$  is of course just minimizing  $-L$ .

$L(u, v)$ . 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  $\lambda$  yields, as I said, the constraint equation. The other derivatives are however yields

$$\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial L}{\partial u} + \lambda \frac{\partial g}{\partial u} \quad (\text{E.32})$$

$$\frac{\partial \mathcal{L}}{\partial v} = \frac{\partial L}{\partial v} + \lambda \frac{\partial g}{\partial v} \quad (\text{E.33})$$

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  $c$ ,

$$\frac{\partial L}{\partial c} = -\lambda \quad (\text{E.34})$$

That is, at the constrained minimum, increasing the constraint level from  $c$  to  $c + \delta$ , with  $\delta$  very small, would change the value of the objective function by  $-\lambda\delta$ . (Note that  $\lambda$  might be negative.) This makes  $\lambda$  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**<sup>3</sup>.

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 constraint<sup>4</sup>.

---

<sup>3</sup>In 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.

<sup>4</sup>A full and precise statement of this idea is the Karush-Kuhn-Tucker theorem of optimization, which you can look up.

### E.3.3 Penalized Optimization

So much for constrained optimization; how does this relate to penalties? Well, once we fix  $\lambda$ , the  $(u, v)$  which minimizes the full Lagrangian

$$L(u, v) + \lambda g(u, v) + \lambda c \quad (\text{E.35})$$

has to be the same as the one which minimizes

$$L(u, v) + \lambda g(u, v) \quad (\text{E.36})$$

This is a *penalized* optimization problem. Changing the magnitude of the penalty  $\lambda$  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.3.4 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

$$Y = \beta X + \epsilon \quad (\text{E.37})$$

where  $\epsilon$  is noise, and  $X$  and  $Y$  are both scalars. We want to estimate the optimal value of the slope  $\beta$ , 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.,

$$L(\beta) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta x_i)^2 \quad (\text{E.38})$$

Call the unconstrained optimum  $\hat{\beta}$ :

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} L(\beta) \quad (\text{E.39})$$

As was said above in §E.3.3, 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

$$\mathcal{L}(\beta, \lambda) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta x_i)^2 + \lambda(\beta^2 - c) \quad (\text{E.40})$$

and we optimize:

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \quad (\text{E.41})$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = 0 \quad (\text{E.42})$$

$$(E.43)$$

The first of these just gives us the constraint back again,

$$\tilde{\beta}^2 = c \quad (\text{E.44})$$

writing  $\tilde{\beta}$  for the constrained optimum. The second equation is

$$\frac{1}{n} \sum_{i=1}^n 2(y_i - \tilde{\beta}x_i)(-x_i) + 2\lambda\tilde{\beta} = 0 \quad (\text{E.45})$$

(If it weren't for the  $\lambda$  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  $\lambda$ , and two equations. Let's solve for  $\lambda$ . The equation  $\tilde{\beta}^2 = c$  can also be written  $\tilde{\beta} = \sqrt{c} \operatorname{sgn} \tilde{\beta}$ , so, plugging in to Eq. E.45,

$$0 = \frac{2}{n} \sum_{i=1}^n x_i y_i - \sqrt{c} \operatorname{sgn} \tilde{\beta} \frac{2}{n} \sum_{i=1}^n x_i^2 + \lambda \sqrt{c} \operatorname{sgn} \tilde{\beta} \quad (\text{E.46})$$

$$\lambda = \frac{\frac{2}{n} \sum_{i=1}^n x_i y_i - c \operatorname{sgn} \tilde{\beta} \frac{2}{n} \sum_{i=1}^n x_i^2}{\sqrt{c} \operatorname{sgn} \tilde{\beta}} \quad (\text{E.47})$$

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.37, with the true  $\beta = 4$ ,  $X$  uniformly distributed on  $[-1, 1]$ , and  $\epsilon$  having a  $t$  distribution with 2 degrees of freedom (Figure E.2). Figure E.3 shows the MSE as a function of  $\beta$ , i.e., the  $L(\beta)$  of Eq. E.38. If  $\sqrt{c}$  is smaller than  $\hat{\beta} \approx 3.95$ , then the constraint is active and  $\lambda$  is non-zero. Figure E.4 plots  $\lambda$  against  $c$  from Eq. E.47. Notice how, as the constraint comes closer and closer to including the unconstrained optimum, the Lagrange multiplier  $\lambda$  becomes closer and closer to 0, finally crossing when  $c = \hat{\beta}^2 \approx 15.6$ .

Turned around, we could fix  $\lambda$  and try to solve the penalized optimization problem

$$\tilde{\beta} = \operatorname{argmin}_{\beta} \mathcal{L}(\beta, \lambda) \quad (\text{E.48})$$

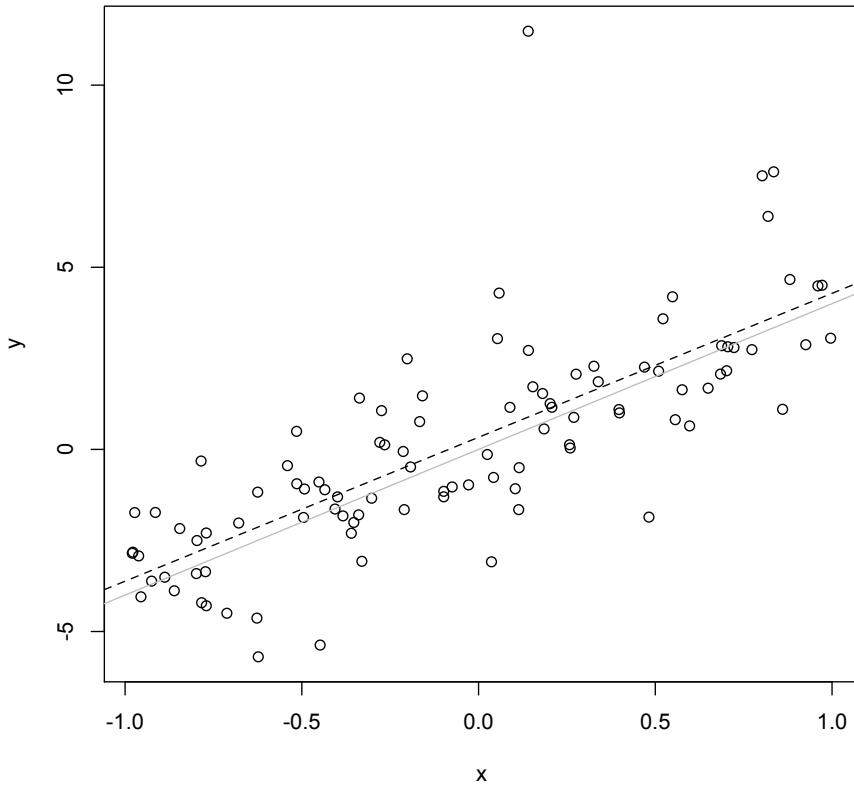
$$= \operatorname{argmin}_{\beta} \frac{1}{n} \sum_{i=1}^n (y_i - \beta x_i)^2 + \lambda \beta^2 \quad (\text{E.49})$$

Taking the derivative with respect to  $\beta$ ,

$$0 = \frac{\partial \mathcal{L}}{\partial \beta} \quad (\text{E.50})$$

$$0 = \frac{1}{n} \sum_{i=1}^n 2(y_i - \tilde{\beta}x_i)(-x_i) + 2\lambda\tilde{\beta} \quad (\text{E.51})$$

$$\tilde{\beta} = \frac{\frac{1}{n} \sum_{i=1}^n x_i y_i}{\lambda + \frac{1}{n} \sum_{i=1}^n x_i^2} \quad (\text{E.52})$$

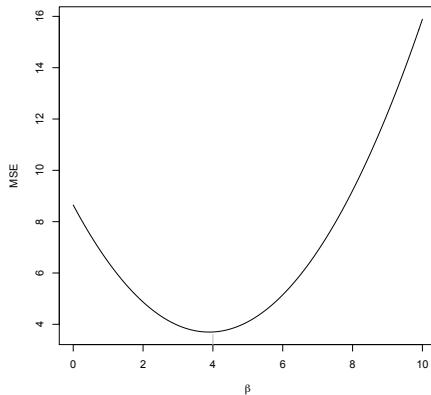


```

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)

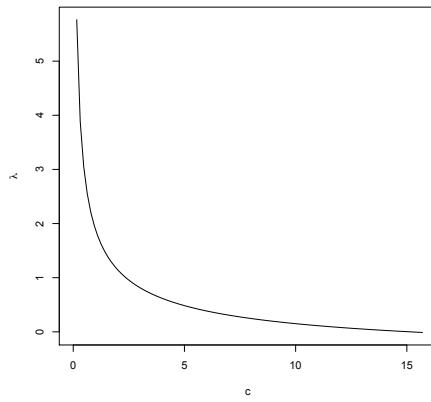
```

FIGURE E.2: 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.3: *Mean squared error as a function of  $\beta$ . The grey tick marks the true  $\beta = 4$ ; the minimum of the curve is at  $\hat{\beta} = 3.95$ .*



```
lambda.from.c <- function(c) { (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.4:  $\lambda$  as a function of the constraint level  $c$ , according to Eq. E.47 and the data in Figure E.2.

which is of course just Eq. E.45 again. Figure E.5 shows how  $\tilde{\beta}$  and  $\tilde{\beta}^2$  change with  $\lambda$ . The fact that the latter plot shows the same curve as Figure E.4 only turned on its side reflects the general correspondence between penalized and constrained optimization.

### E.3.5 Statistical Remark: “Ridge Regression” and “The Lasso”

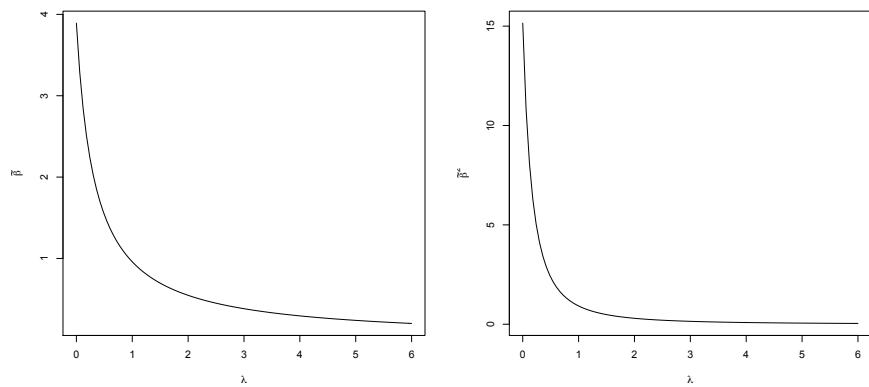
The idea of penalizing or constraining the coefficients of a linear regression model can be extended to having more than one coefficient. The general case, with  $p$  covariates, is that one penalizes the sum of the squared coefficients,  $\beta_1^2 + \dots + \beta_p^2$ , which of course is just the squared length of the coefficient vector,  $\|\beta\|^2$ . This is called **ridge regression** (Hoerl and Kennard, 1970), and yields the estimates

$$\tilde{\beta} = (\mathbf{x}^T \mathbf{x} + \lambda \mathbf{I})^{-1} \mathbf{x}^T \mathbf{y} \quad (\text{E.53})$$

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.53, but it can be computed efficiently.

Examining Eq. E.53 should convince you that  $\tilde{\beta}$  is generally smaller than the unpenalized estimate  $\hat{\beta}$ . (This may be easier to see from Eq. E.52.) 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(1) { return(mean(x*y)/(1+mean(x^2))) }
curve(beta.from.lambda(x),from=0,to=6,
 xlab=expression(lambda),ylab=expression(tilde(beta)))
curve(beta.from.lambda(x)^2,from=0,to=6,
 xlab=expression(lambda),ylab=expression(tilde(beta)^2))

```

FIGURE E.5: Left: The penalized estimation of the regression slope, as a function of the strength of the penalty  $\lambda$ . Right: Square of the penalized regression slope.

# Appendix F

# Optimization Methods

[[Approximations; time-error trade-offs]]

## F.1 Optimization with First- and Second- Derivatives

### F.1.1 Gradient Descent

Gradients and changes to  $f$ :

$$\begin{aligned} f'(x_0) &= \left. \frac{df}{dx} \right|_{x=x_0} = \lim_{x \rightarrow x_0} \frac{f(x) - f(x_0)}{x - x_0} \\ f(x) &\approx f(x_0) + (x - x_0)f'(x_0) \end{aligned}$$

i.e., locally, the function looks linear

To minimize a linear function, move down the slope!

Multivariate version:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot \nabla f(\theta_0)$$

$\nabla f(\theta_0)$  points in the direction of fastest ascent at  $\theta_0$

The gradient descent algorithm:

1. Start with initial guess for  $\theta$ , step-size  $\eta$
2. While ((not too tired) and (making adequate progress))
  - (a) Find gradient  $\nabla f(\theta)$
  - (b) Set  $\theta \leftarrow \theta - \eta \nabla f(\theta)$
3. Return final  $\theta$  as approximate  $\theta^*$

Variations: adaptively adjust  $\eta$  to make sure of improvement; or search along the gradient direction for minimum

Our starting point:

$$f(x) \approx f(x_0) + \frac{1}{2}(x - x_0)^T H(x_0)(x - x_0)$$

and

$$\nabla f(x) \approx H(x_0)(x - x_0)$$

So

$$f(x - \lambda \nabla f) \approx f(x_0) + \frac{1}{2}(x - \lambda \nabla f - x_0)^T H(x_0)(x - \lambda \nabla f - x_0) \quad (\text{F.1})$$

$$= f(x_0) - (\lambda \nabla f)^T H(x_0)(x - x_0) + \frac{1}{2}\lambda^2 \nabla f^T H(x_0) \nabla f \quad (\text{F.2})$$

$$= f(x_0) - \lambda(x - x_0)^T H^T(x_0)H(x_0)(x - x_0) + \frac{1}{2}\lambda^2(x - x_0)^T H^T(x_0)H(x_0)(x - x_0) \quad (\text{F.3})$$

The coefficient of  $\lambda$  is  $> 0$  as is that of  $\lambda^2$ , but the terms are  $O(\lambda)$  and  $O(\lambda^2)$ , respectively.

If  $\lambda$  is small enough, we improve by taking one gradient step:

$$\begin{aligned} \|x - \lambda \nabla f - x_0\|^2 &= \langle x - \lambda H(x - x_0) - x_0, x - \lambda H(x - x_0) - x_0 \rangle \\ &= \langle x - x_0, x - x_0 \rangle - 2\lambda \langle H(x - x_0), x - x_0 \rangle + \lambda^2 \langle H(x - x_0), H(x - x_0) \rangle \end{aligned} \quad (\text{F.4})$$

which is an over-all negative change if  $\lambda$  is small enough. But then we might need to take *many* steps.

How Much Work is Gradient Descent?

Pro:

- For nice  $f$ ,  $f(\theta) \leq f(\theta^*) + \epsilon$  in  $O(\epsilon^{-2})$  iterations [[cite]]
- For *very* nice  $f$ , only  $O(\log \epsilon^{-1})$  iterations [[cite]]
- To get  $\nabla f(\theta)$ , take  $p$  derivatives,  $\therefore$  each iteration costs  $O(p)$

Con:

- Taking derivatives can slow down as data grows — each iteration might really be  $O(np)$

### F.1.2 Newton's Method

[[TODO: Smooth out writing for current location, as originally taken from Chapter 12]]

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,  $\beta^*$ . We suppose that  $f$  is smooth, and that  $\beta^*$  is a regular interior minimum, meaning that the derivative at  $\beta^*$  is zero and the second derivative is positive. Near the minimum we could make a Taylor expansion:

$$f(\beta) \approx f(\beta^*) + \frac{1}{2}(\beta - \beta^*)^2 \left. \frac{d^2 f}{d\beta^2} \right|_{\beta=\beta^*} \quad (\text{F.6})$$

(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:

$$f(\beta) \approx f(\beta^{(0)}) + (\beta - \beta^{(0)}) \left. \frac{df}{d\beta} \right|_{\beta=\beta^{(0)}} + \frac{1}{2}(\beta - \beta^{(0)})^2 \left. \frac{d^2 f}{d\beta^2} \right|_{\beta=\beta^{(0)}} \quad (\text{F.7})$$

Now it's easy to minimize the right-hand side of equation F.7. 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  $\beta$ , and set it equal to zero at a point we'll call  $\beta^{(1)}$ :

$$0 = f'(\beta^{(0)}) + \frac{1}{2}f''(\beta^{(0)})2(\beta^{(1)} - \beta^{(0)}) \quad (\text{F.8})$$

$$\beta^{(1)} = \beta^{(0)} - \frac{f'(\beta^{(0)})}{f''(\beta^{(0)})} \quad (\text{F.9})$$

The value  $\beta^{(1)}$  should be a better guess at the minimum  $\beta^*$  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:

$$\beta^{(n+1)} = \beta^{(n)} - \frac{f'(\beta^{(n)})}{f''(\beta^{(n)})} \quad (\text{F.10})$$

Notice that the true minimum  $\beta^*$  is a **fixed point** of equation F.10: 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  $\beta^*$ , 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.<sup>1</sup>

---

<sup>1</sup>`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.

### F.1.2.1 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

$$\beta^{(n+1)} = \beta^{(n)} - \mathbf{H}^{-1}(\beta^{(n)}) \nabla f(\beta^{(n)}) \quad (\text{F.11})$$

where  $\nabla f$  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  $\nabla f$  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.

---

[[ Yet another set of Newton's method notes for meger ]]

Newton's method: "reverse the polarity flow!" We know  $\nabla f(x)$  and  $\nabla^2 f(x)$ , though we don't know  $H(x_0)$ .

$$f(x_0) \approx f(x) + \langle (x_0 - x), \nabla f(x) \rangle + \frac{1}{2} \langle (x_0 - x), H(x)(x_0 - x) \rangle$$

Since  $\nabla f(x_0) = 0$ ,  $\nabla f(x) + H(x)(x_0 - x) = 0$ . Thus

$$\nabla f(x) = -H(x)(x_0 - x) \quad (\text{F.12})$$

$$-H^{-1}(x)\nabla f(x) = x_0 - x \quad (\text{F.13})$$

$$x - H^{-1}(x)\nabla f(x) = 0 \quad (\text{F.14})$$

Newton's method: try it, and keep repeating if it doesn't work. It's exact for quadratic functions, and for others,

$$\|x - x_0 - H^{-1}\nabla f(x)\|^2 = \|x - x_0\|^2 - 2\langle x - x_0, H^{-1}(x)\nabla f(x) \rangle + \|H^{-1}\nabla f(x)\|^2$$

The negative term is  $O(\|\nabla f(x)\|)$ , while the positive one is  $O(\|\nabla f(x)\|^2)$ , so the former is guaranteed to dominate if we start close enough to the optimum.

---

What if we do a quadratic approximation to  $f$ ?

[[ TODO: Smooth out, yet another set of notes ]]

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

Simplifies if  $x_0$  is a minimum since then  $f'(x_0) = 0$ :

$$f(x) \approx f(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

That is, near a minimum, smooth functions look like parabolas. This carries over to the multivariate case:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot \nabla f(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta_0)(\theta - \theta_0)$$

Now, minimizing a quadratic is easy. If we know

$$f(x) = ax^2 + bx + c$$

we can minimize exactly:

$$\begin{aligned} 2ax^* + b &= 0 \\ x^* &= \frac{-b}{2a} \end{aligned}$$

If

$$f(x) = \frac{1}{2}a(x - x_0)^2 + b(x - x_0) + c$$

then

$$x^* = x_0 - a^{-1}b$$

To get from here to Newton's method, we Taylor-expand for the value *at the minimum*  $\theta^*$

$$f(\theta^*) \approx f(\theta) + (\theta^* - \theta)\nabla f(\theta) + \frac{1}{2}(\theta^* - \theta)^T \mathbf{H}(\theta)(\theta^* - \theta)$$

Take gradient, set to zero, solve for  $\theta^*$ :

$$\begin{aligned} 0 &= \nabla f(\theta) + \mathbf{H}(\theta)(\theta^* - \theta) \\ \theta^* &= \theta - (\mathbf{H}(\theta))^{-1}\nabla f(\theta) \end{aligned}$$

This works *exactly* if  $f$  is quadratic (and  $\mathbf{H}^{-1}$  exists, etc.)

1. Start with guess for  $\theta$
2. While ((not too tired) and (making adequate progress))
  - (a) Find gradient  $\nabla f(\theta)$  and Hessian  $\mathbf{H}(\theta)$
  - (b) Set  $\theta \leftarrow \theta - \mathbf{H}(\theta)^{-1}\nabla f(\theta)$
3. Return final  $\theta$  as approximation to  $\theta^*$

Like gradient descent, but with inverse Hessian giving the step-size (“This is about how far you can go with that gradient”)<sup>2</sup>.

Advantages and Disadvantages of Newton's Method

Pro:

- Step-sizes chosen adaptively through 2nd derivatives, much harder to get zig-zagging, over-shooting, etc.
- Like gradient descent, guaranteed to need  $O(\epsilon^{-2})$  steps to get within  $\epsilon$  of optimum [[cite]]

---

<sup>2</sup>Of course the inverse Hessian could also be applying a rotation, so that's a bit of a lie.

- Only  $O(\log \log \epsilon^{-1})$  for very nice functions [[cite]]
- Typically many fewer iterations than gradient descent

Cons:

- Hopeless if  $\mathbf{H}$  doesn't exist or isn't invertible
- Need to take  $O(p^2)$  second derivatives *plus*  $p$  first derivatives
- Need to solve  $\mathbf{H}\theta_{\text{new}} = \mathbf{H}\theta_{\text{old}} - \nabla f(\theta_{\text{old}})$  for  $\theta_{\text{new}}$  (inverting  $\mathbf{H}$  is  $O(p^3)$ , but cleverness gives  $O(p^2)$  for solving)

Getting around finding the Hessian: we want to use the Hessian to improve convergence, but we don't want to have to keep computing the Hessian at each step. How might we cheat?

- Use knowledge of the system to get some approximation to the Hessian, use that instead of taking derivatives ("Fisher scoring") [[cite]]
- Use only diagonal entries ( $p$  unmixed 2nd derivatives)
- Use  $\mathbf{H}(\theta)$  at initial guess, hope  $\mathbf{H}$  changes *very* slowly with  $\theta$
- Re-compute  $\mathbf{H}(\theta)$  every  $k$  steps,  $k > 1$
- Fast, approximate updates to the Hessian at each step (BFGS) [[cite]]

### F.1.3 Stochastic Approximation

Typical statistical objective function, mean-squared error:

$$f(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - m(x_i, \theta))^2$$

Getting a value of  $f$  is  $O(n)$ ,  $\nabla f$  is  $O(np)$ ,  $\mathbf{H}$  is  $O(np^2)$  — worse still if  $m$  slows down with  $n$ . Perhaps this isn't so bad bad when  $n = 100$  or even  $n = 10^4$ , but if  $n = 10^9$  or  $n = 10^{12}$  we don't even know which way to move.

Solution via sampling: Pick *one* data point  $I$  at random (uniform on  $1:n$ )

Loss there,  $(y_I - m(x_I, \theta))^2$ , is random, but

$$\mathbb{E}[(y_I - m(x_I, \theta))^2] = f(\theta)$$

Generally, if  $f(\theta) = n^{-1} \sum_{i=1}^n f_i(\theta)$  and  $f_i$  are well-behaved,

$$\begin{aligned} \mathbb{E}[f_I(\theta)] &= f(\theta) \\ \mathbb{E}[\nabla f_I(\theta)] &= \nabla f(\theta) \\ \mathbb{E}[\nabla^2 f_I(\theta)] &= \mathbf{H}(\theta) \end{aligned}$$

**Stochastic Gradient Descent:** Draw lots of one-point samples, let their noise cancel out:

[[TODO: Decide how much of this is needed past Newton's method]]

[[TODO: Replace slide-speak with proper text]]

1. Start with initial guess  $\theta$ , learning rate  $\eta$
2. While ((not too tired) and (making adequate progress))
  - (a) At  $t^{\text{th}}$  iteration, pick random  $I$  uniformly
  - (b) Set  $\theta \leftarrow \theta - t^{-1}\eta \nabla f_I(\theta)$
3. Return final  $\theta$

Shrinking step-size by  $1/t$  ensures noise in each gradient dies down

(Variants: put points in some random order and step through them, only check progress after going over each point once; adjust  $1/t$  rate; average a couple of random data points rather than just one, etc.)

[[TODO: Add comments]]

```

stoch.grad.descent <- function(f,theta,df,max.iter=1e6,rate=1e-6) {
 stopifnot(require(numDeriv))
 for (t in 1:max.iter) {
 g <- stoch.grad(f,theta,df)
 theta <- theta - (rate/t)*g
 }
 return(x)
}

stoch.grad <- function(f,theta,df) {
 i <- sample(1:nrow(df),size=1)
 noisy.f <- function(theta) { return(f(theta, data=df[i,])) }
 stoch.grad <- grad(noisy.f,theta)
 return(stoch.grad)
}

```

### F.1.3.1 Stochastic Newton's Method

a.k.a. 2nd order stochastic gradient descent

1. Start with initial guess  $\theta$
  2. While ((not too tired) and (making adequate progress))
    - (a) At  $t^{\text{th}}$  iteration, pick uniformly-random  $I$
    - (b)  $\theta \leftarrow \theta - t^{-1}\mathbf{H}_I^{-1}(\theta)\nabla f_I(\theta)$
  3. Return final  $\theta$
- + all the Newton-ish tricks to avoid having to recompute the Hessian

### F.1.4 Pros and Cons of Stochastic Gradient Methods

Pros:

- Each iteration is fast (and constant in  $n$ )
- Never need to hold all data in memory
- Does converge eventually

Cons:

- Noise *does* reduce precision — more iterations to get within  $\epsilon$  of optimum than non-stochastic GD or Newton

Often low computational cost to get within *statistical* error of the optimum (cf. App. E.2).

## F.2 Derivative-Free Optimization Techniques

[[Appendix to come: simplex method; simulated annealing; evolutionary searches]]

### F.2.1 Nelder-Mead, a.k.a. the Simplex Method

Try to cage  $\theta^*$  with a **simplex** of  $p + 1$  points

Order the trial points,  $f(\theta_1) \leq f(\theta_2) \dots \leq f(\theta_{p+1})$

$\theta_{p+1}$  is the worst guess — try to improve it

$\theta_0 = \frac{1}{n} \sum_{i=1}^n \theta_i$  = center of the not-worst

[[TODO: Expand this into proper text rather than slide-speak]]

- **Reflection:** Try  $x_0 - (x_{p+1} - x_0)$ , across the center from  $x_{p+1}$ 
  - if it's better than  $x_p$  but not than  $x_1$ , replace the old  $x_{p+1}$  with it
  - **Expansion:** if the reflected point is the new best, try  $x_0 - 2(x_{p+1} - x_0)$ ; replace the old  $x_{p+1}$  with the better of the reflected and the expanded point
- **Contraction:** If the reflected point is worse than  $x_p$ , try  $x_0 + \frac{x_{p+1} - x_0}{2}$ ; if the contracted value is better, replace  $x_{p+1}$  with it
- **Reduction:** If all else fails,  $x_i \leftarrow \frac{x_1 + x_i}{2}$

Why these moves?

- Reflection: try the opposite of the worst point
- Expansion: if that really helps, try it some more
- Contraction: see if we overshot when trying the opposite

- Reduction: if all else fails, try being more like the best point

Pros:

- Each iteration  $\leq 4$  values of  $f$ , plus sorting (at most  $O(p \log p)$ , usually much better)
- No derivatives used, can even work for dis-continuous  $f$

Con:

- Can need *many* more iterations than gradient methods

### F.2.2 Simulated Annealing

Consider using Metropolis [[TODO: Cross-reference]] to sample from a density  $\propto e^{-f(\theta)/T}$ . The samples will tend to be near small values of  $f$ , and closer the smaller we make  $T$ . Now keep lowering  $T$  as we go along (“cooling”, “annealing”).

1. Set initial  $\theta, T > 0$
2. While ((not too tired) and (making adequate progress))
  - (a) Proposal:  $Z \leftarrow r(\cdot|\theta)$  (e.g., Gaussian noise)
  - (b) Draw  $U \sim \text{Unif}(0, 1)$
  - (c) Acceptance: If  $U < e^{-\frac{f(Z)-f(\theta)}{T}}$  then  $\theta \leftarrow Z$
  - (d) Reduce  $T$  a little
3. Return final  $\theta$

Always moves to lower values of  $f$ , sometimes moves to higher  
No derivatives, works for discrete problems, few guarantees

## F.3 Methods for Constraints

[[“Method of multipliers” a.k.a. augmented Lagrangian, interior point methods]]

## F.4 R Notes

[[optim, constrOptim, nlm, nls, alabama, probably others]]

## Appendix G

# Where the $\chi^2$ Null Distribution for the 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  $\theta$ , lies in the restricted class of models  $\omega$ . So there are  $q$  components to  $\theta$  which matter, and the other  $p - q$  are fixed by the constraints defining  $\omega$ . 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

$$\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_q, 0, \dots, 0) \quad (\text{G.1})$$

The unrestricted MLE does not have to obey the constraints, so it's

$$\hat{\Theta} = (\hat{\Theta}_1, \hat{\Theta}_2, \dots, \hat{\Theta}_q, \hat{\Theta}_{q+1}, \dots, \hat{\Theta}_p) \quad (\text{G.2})$$

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  $\chi^2$  distribution with one degree of freedom. A whole bunch of factors cancel out, leaving us with a sum of  $p - q$  independent  $\chi^2$  variables, which has a  $\chi^2_{p-q}$  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.

$$\begin{aligned} L(\hat{\theta}) &\approx L(\hat{\Theta}) + \sum_{i=1}^p (\hat{\Theta}_i - \hat{\theta}_i) \left( \frac{\partial L}{\partial \theta_i} \Big|_{\hat{\Theta}} \right) + \frac{1}{2} \sum_{i=1}^p \sum_{j=1}^p (\hat{\Theta}_i - \hat{\theta}_i) \left( \frac{\partial^2 L}{\partial \theta_i \partial \theta_j} \Big|_{\hat{\Theta}} \right) (\hat{\Theta}_j - \hat{\theta}_j) \\ &= L(\hat{\Theta}) + \frac{1}{2} \sum_{i=1}^p \sum_{j=1}^p (\hat{\Theta}_i - \hat{\theta}_i) \left( \frac{\partial^2 L}{\partial \theta_i \partial \theta_j} \Big|_{\hat{\Theta}} \right) (\hat{\Theta}_j - \hat{\theta}_j) \end{aligned} \quad (\text{G.3})$$

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

$$L(\hat{\Theta}) - L(\hat{\theta}) \approx -\frac{1}{2} \sum_{i=1}^p (\hat{\Theta}_i - \hat{\theta}_i)^2 L_{,ii} \quad (\text{G.4})$$

$$2 \left[ L(\hat{\Theta}) - L(\hat{\theta}) \right] \approx -\sum_{i=1}^q (\hat{\Theta}_i - \hat{\theta}_i)^2 L_{,ii} - \sum_{i=q+1}^p (\hat{\Theta}_i)^2 L_{,ii} \quad (\text{G.5})$$

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

$$\hat{\Theta}_i \rightsquigarrow \mathcal{N}(\theta_i, -1/nF_{ii}) \quad (\text{G.6})$$

$$\hat{\theta}_i \rightsquigarrow \mathcal{N}(\theta_1, -1/nF_{ii}) \quad 1 \leq i \leq q \quad (\text{G.7})$$

$$\hat{\theta}_i = 0 \quad q+1 \leq i \leq p \quad (\text{G.8})$$

Also,  $(1/n)L_{,ii} \rightarrow -F_{ii}$ .

Putting all this together, we see that each term in the second summation in Eq. G.5 is (to abuse notation a little)

$$\frac{-1}{nF_{ii}} (\mathcal{N}(0, 1))^2 L_{,ii} \rightarrow \chi_1^2 \quad (\text{G.9})$$

so the whole second summation has a  $\chi_{p-q}^2$  distribution<sup>1</sup>. The first summation, meanwhile, goes to zero because  $\hat{\Theta}_i$  and  $\hat{\theta}_i$  are actually strongly correlated, so their

---

<sup>1</sup>Thanks 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:

$$\hat{\Theta} \rightsquigarrow \mathcal{N}(\theta, (1/n)F^{-1}) \quad (\text{G.10})$$

Then, at the cost of more linear algebra, we don't have to assume that the Hessian is diagonal.

## Appendix H

# 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

$$\hat{\beta}_{WLS} = \mathbf{h}(w)\mathbf{y} \quad (\text{H.1})$$

where the matrix  $\mathbf{h}(w)$  is

$$\mathbf{h}(w) = (\mathbf{x}^T \mathbf{w} \mathbf{x})^{-1} \mathbf{x}^T \mathbf{w} \quad (\text{H.2})$$

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

$$(\mathbf{h}_0 + \mathbf{c})\mathbf{x}\beta = \beta \quad (\text{H.3})$$

but using the inverse-variance weights is a particular WLS estimate so

$$\mathbf{h}_0 \mathbf{x} \beta = \beta \quad (\text{H.4})$$

and so we can deduce that

$$\mathbf{c}\mathbf{x} = 0 \quad (\text{H.5})$$

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:

$$\text{Var}[\tilde{\beta}] = \text{Var}[(\mathbf{h}_0 + \mathbf{c})\mathbf{Y}] \quad (\text{H.6})$$

$$= (\mathbf{h}_0 + \mathbf{c})\text{Var}[Y](\mathbf{h}_0 + \mathbf{c})^T \quad (\text{H.7})$$

$$= (\mathbf{h}_0 + \mathbf{c})\mathbf{w}_0^{-1}(\mathbf{h}_0 + \mathbf{c})^T \quad (\text{H.8})$$

$$= \mathbf{h}_0\mathbf{w}_0^{-1}\mathbf{h}_0^T + \mathbf{c}\mathbf{w}_0^{-1}\mathbf{h}_0^T + \mathbf{h}_0\mathbf{w}_0^{-1}\mathbf{c}^T + \mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T \quad (\text{H.9})$$

$$= (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}_0\mathbf{w}_0^{-1}\mathbf{w}_0\mathbf{x}(\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1} \quad (\text{H.10})$$

$$+ \mathbf{c}\mathbf{w}_0^{-1}\mathbf{w}_0\mathbf{x}(\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}$$

$$+ (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}_0\mathbf{w}_0^{-1}\mathbf{c}^T$$

$$+ \mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T$$

$$= (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}_0\mathbf{x}(\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1} \quad (\text{H.11})$$

$$+ \mathbf{c}\mathbf{x}(\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1} + (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1}\mathbf{x}^T\mathbf{c}^T$$

$$+ \mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T$$

$$= (\mathbf{x}^T\mathbf{w}_0\mathbf{x})^{-1} + \mathbf{c}\mathbf{w}_0^{-1}\mathbf{c}^T \quad (\text{H.12})$$

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 I

# 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**<sup>1</sup>, 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 I.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 I.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 I.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.)

---

<sup>1</sup>Or, more rarely, a **Guthrie diagram**.

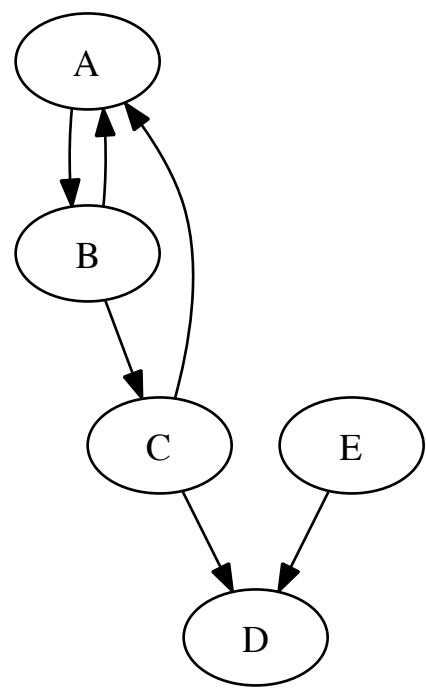


FIGURE I.1: Example for illustrating the concepts of graph theory.

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 J

# Uncorrelated versus Independent

A reminder of about the difference between two variables being uncorrelated and their being independent.

Two random variables  $X$  and  $Y$  are **uncorrelated** when their correlation coefficient is zero:

$$\rho(X, Y) = 0 \quad (\text{J.1})$$

Since

$$\rho(X, Y) = \frac{\text{Cov}[X, Y]}{\sqrt{\text{Var}[X] \text{Var}[Y]}} \quad (\text{J.2})$$

being uncorrelated is the same as having zero covariance. Since

$$\text{Cov}[X, Y] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \quad (\text{J.3})$$

having zero covariance, and so being uncorrelated, is the same as

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] \quad (\text{J.4})$$

One says that “the expectation of the product factors”. If  $\rho(X, Y) \neq 0$ , then  $X$  and  $Y$  are **correlated**.

Two random variables are **independent** when their joint probability distribution is the product of their marginal probability distributions: for all  $x$  and  $y$ ,

$$p_{X,Y}(x,y) = p_X(x)p_Y(y) \quad (\text{J.5})$$

Equivalently<sup>1</sup>, the conditional distribution is the same as the marginal distribution:

$$p_{Y|X}(y|x) = p_Y(y) \quad (\text{J.6})$$

---

<sup>1</sup>Why is this equivalent?

If  $X$  and  $Y$  are not independent, then they are **dependent**. If, in particular,  $Y$  is a function of  $X$ , then they always dependent<sup>2</sup>

If  $X$  and  $Y$  are independent, then they are also uncorrelated. To see this, write the expectation of their product:

$$\mathbb{E}[XY] = \int \int xy p_{X,Y}(x,y) dx dy \quad (\text{J.7})$$

$$= \int \int xy p_X(x) p_Y(y) dx dy \quad (\text{J.8})$$

$$= \int x p_X(x) \left( \int y p_Y(y) dy \right) dx \quad (\text{J.9})$$

$$= \left( \int x p_X(x) dx \right) \left( \int y p_Y(y) dy \right) \quad (\text{J.10})$$

$$= \mathbb{E}[X] \mathbb{E}[Y] \quad (\text{J.11})$$

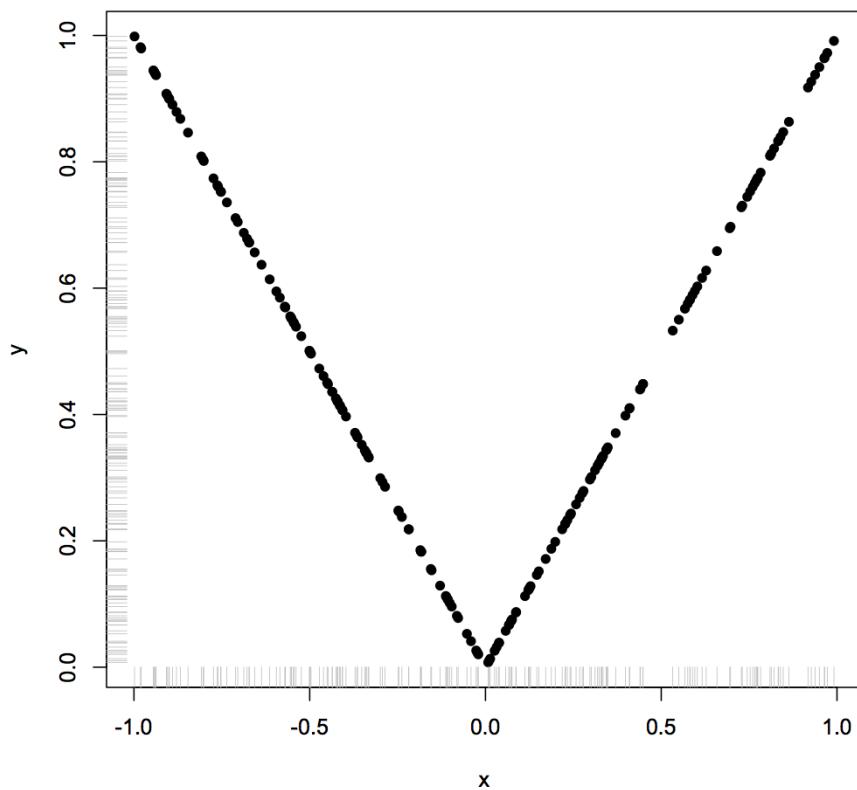
However, if  $X$  and  $Y$  are uncorrelated, then they can *still* be dependent. To see an extreme example of this, let  $X$  be uniformly distributed on the interval  $[-1, 1]$ . If  $X \leq 0$ , then  $Y = -X$ , while if  $X$  is positive, then  $Y = X$ . You can easily check for yourself that:

- $Y$  is uniformly distributed on  $[0, 1]$
- $\mathbb{E}[XY|X \leq 0] = \int_{-1}^0 -x^2 dx = -1/3$
- $\mathbb{E}[XY|X > 0] = \int_0^1 x^2 dx = +1/3$
- $\mathbb{E}[XY] = 0$  (*hint*: law of total expectation).
- The joint distribution of  $X$  and  $Y$  is not uniform on the rectangle  $[-1, 1] \times [0, 1]$ , as it would be if  $X$  and  $Y$  were independent (Figure J.1).

The only general case when lack of correlation implies independence is when the joint distribution of  $X$  and  $Y$  is a multivariate Gaussian.

---

<sup>2</sup>For the sake of mathematical quibblers: a *non-constant* function of  $X$ .



```
x <- runif(200,min=-1,max=1)
y <- ifelse(x>0,x,-x)
plot(x,y,pch=16)
rug(x,side=1,col="grey")
rug(y,side=2,col="grey")
```

FIGURE J.1: An example of two random variables which are uncorrelated but strongly dependent. The grey “rug plots” on the axes show the marginal distributions of the samples from  $X$  and  $Y$ .

## Appendix K

# Information Theory

[[TODO: write appendix, drawing in material about information theory, relative entropy/Kullback-Leibler divergence, and expected log-likelihood currently scattered across chapters in trees, density estimation, and EM]]

## Appendix L

# 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, build their own tools. Someone who just knows how to run canned routines is not a data analyst but a human interface to 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 appendix tries to illustrate what's involved, starting from the very beginning. It is redundant for many students, but included through popular demand.

### L.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.

## L.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

$$f(x; \alpha, x_0) = \begin{cases} \frac{\alpha-1}{x_0} \left(\frac{x}{x_0}\right)^{-\alpha} & x \geq x_0 \\ 0 & x < x_0 \end{cases} \quad (\text{L.1})$$

Consequently, the CDF is

$$F(x; \alpha, x_0) = 1 - \left(\frac{x}{x_0}\right)^{-\alpha+1} \quad (\text{L.2})$$

and the quantile function is

$$Q(p; \alpha, x_0) = x_0(1-p)^{-\frac{1}{\alpha-1}} \quad (\text{L.3})$$

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  $1 \times 10^6$ , 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.

### L.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.

### L.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  $\alpha$  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?)

## L.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.<sup>1</sup>

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)

```

---

<sup>1</sup>I actually made this exact mistake the first time I wrote the function, back in 2004.

```

 50%
1.598253
> qpareto.4(p=0.1,exponent=2.5,threshold=1)
[1] 1.072766
> quantile(r,0.1)
 10%
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 L.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.

### L.4.1 More on Debugging

Everyone who writes their own code spends a lot of time debugging<sup>2</sup>. 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?

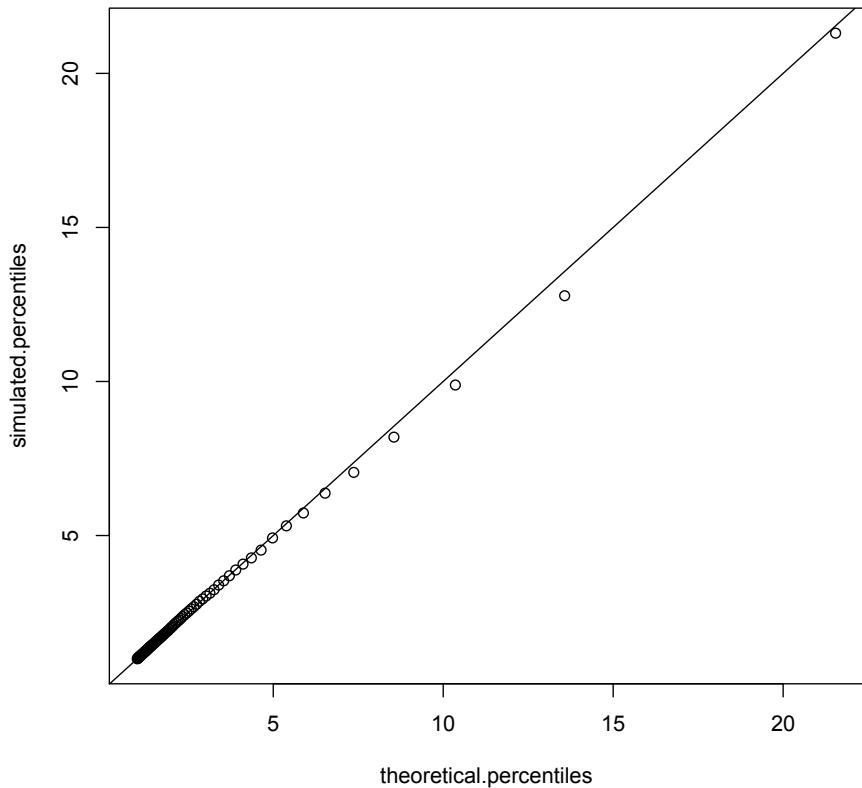
**Localize the Bug** The problem *may* be a diffuse all-pervading wrongness, but often it's a lot more localized, to a few lines or even just one line of code; it helps to know where! We have seen some tools for localizing the bug above: `traceback()` and `stopifnot()`. Another very helpful one is to 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 effect<sup>3</sup>.

**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?

---

<sup>2</sup>Those who don't write their own code but use computers anyway spend a lot of time putting up with other people's bugs.

<sup>3</sup>Real 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 L.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.)

## L.5 Automating Repetition and Passing Arguments

The match between the theoretical quantiles and the simulated ones in Figure L.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 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.)

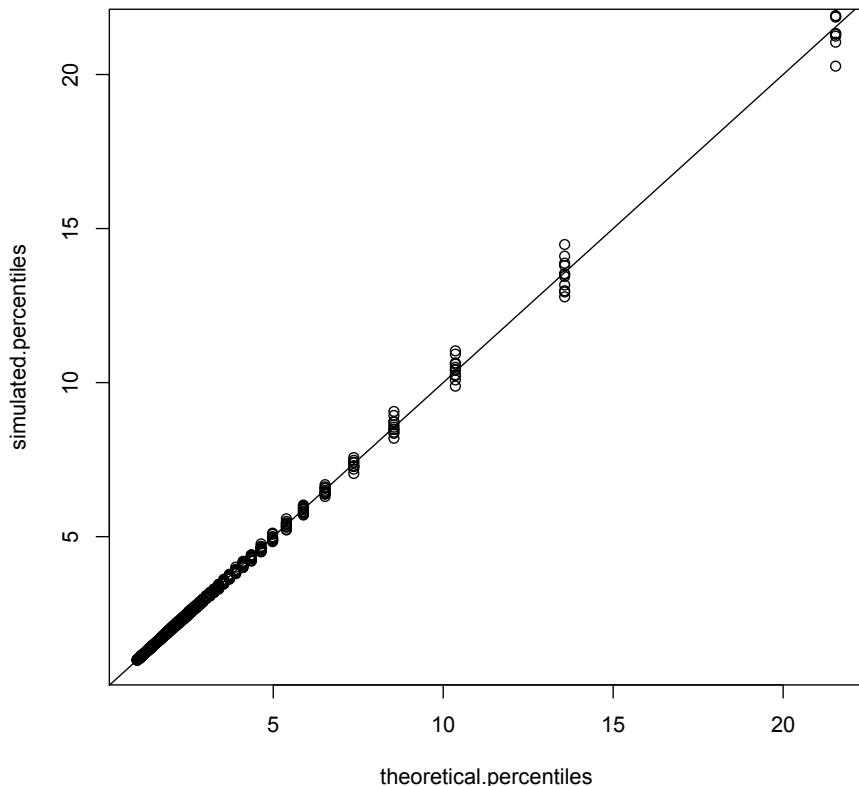
Figure L.2 shows how we can use it to produce multiple simulation runs. We can see that, looking over many simulation runs, the quantiles seem to be too large about as often, and as much, as they are too low, which is reassuring.

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, ...)
}
```

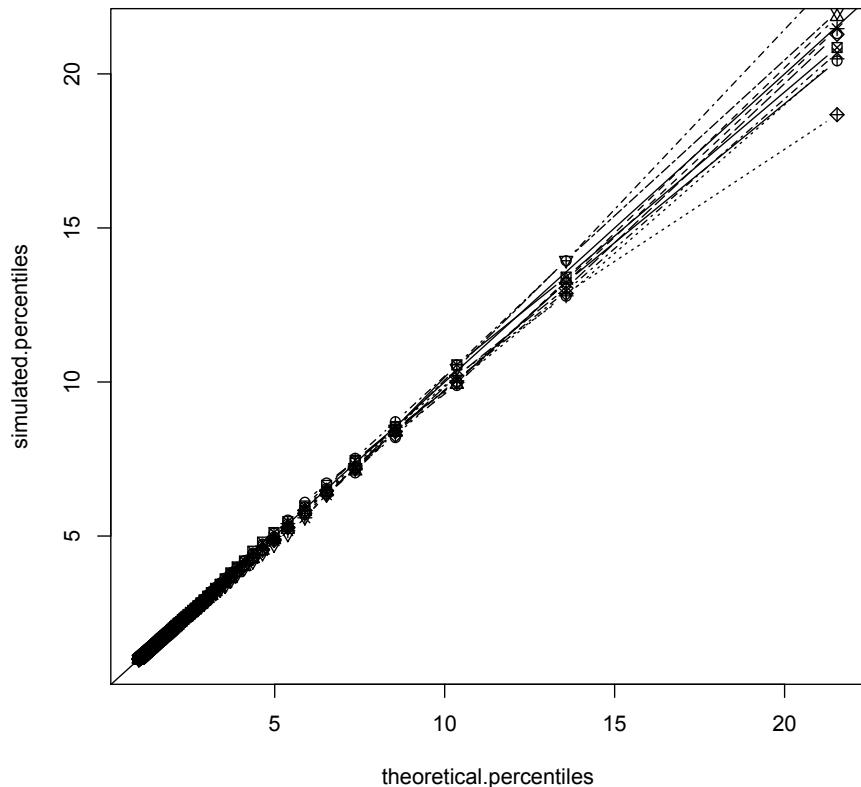
Putting the ellipses (...) in the argument list means that we can give `pareto.sim.vs.theory` an arbitrary collection of arguments, but with the expectation that it will pass them along unchanged to some other function that it will call with ... — here, that's the `points` function. Figure L.3 shows 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:



```
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 L.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 L.3: As Figure L.2, but using the ability to pass along arguments to a subsidiary function to distinguish separate simulation runs.

```

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)),
 # 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`.<sup>4</sup>

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 L.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)

```

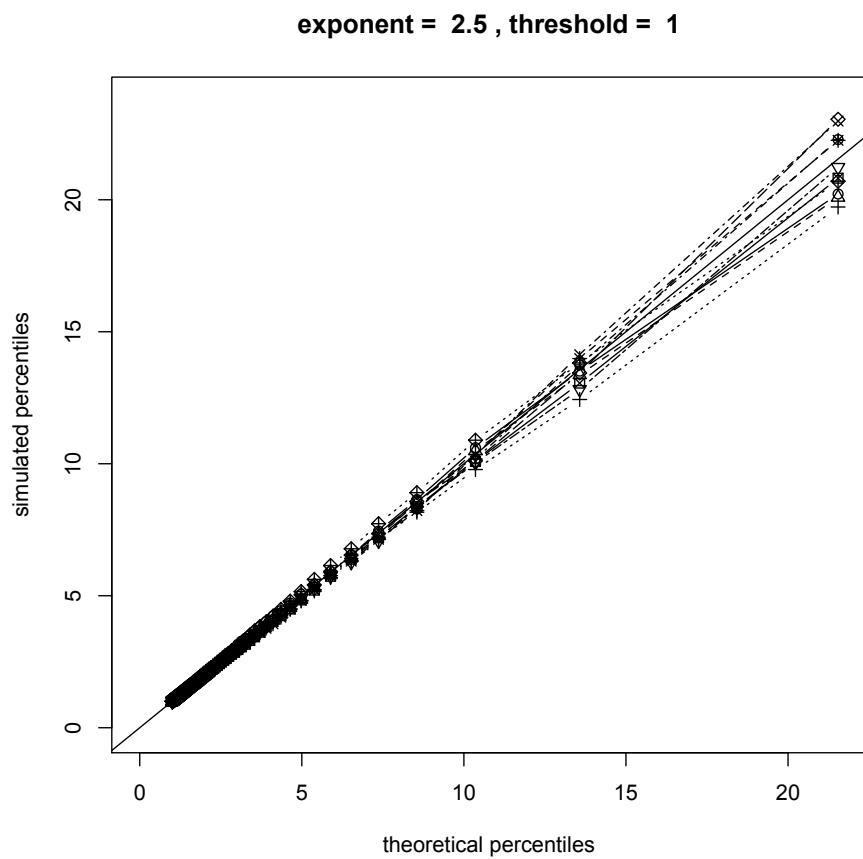
Unfortunately, changing the arguments reveals a bug (Figure L.5). Notice that the vertical coordinates of the points, coming from the simulation, look like they have about the same range as the theoretical quantiles, used to lay out the plotting window. But the horizontal coordinates are all pretty much 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,...)
```

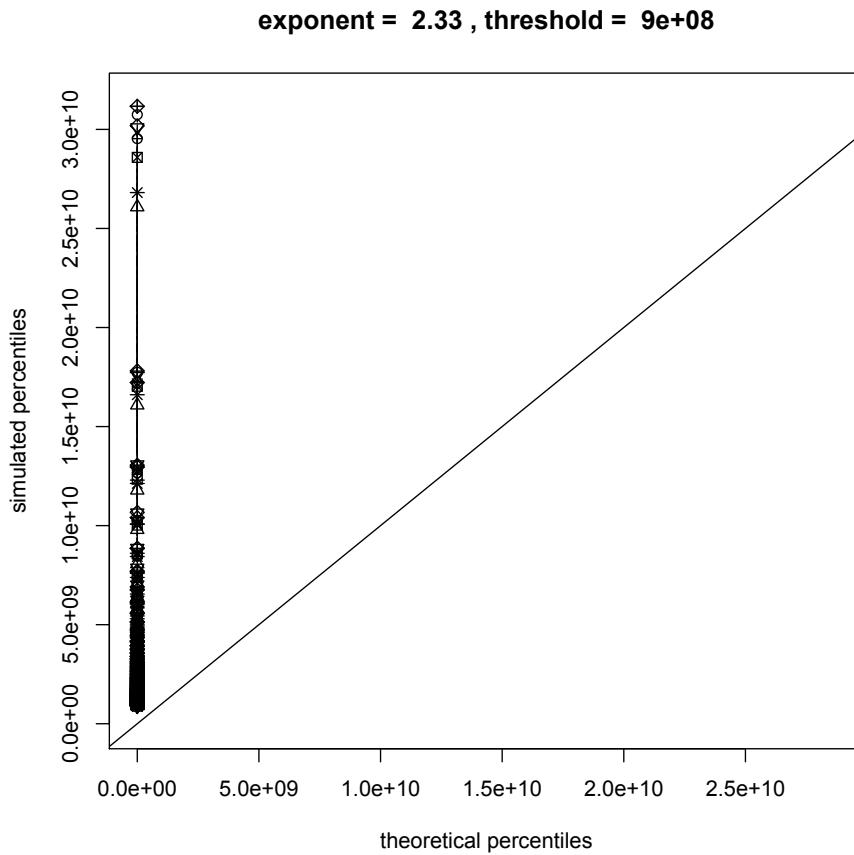
---

<sup>4</sup>Try running `check.rpareto()`, follows by `warnings()`.



```
check.rpareto()
```

FIGURE L.4: Automating the checking of `rpareto`.



```
check.rpareto(n=1e4, exponent=2.33, threshold=9e8)
```

FIGURE L.5: *A bug in check.rpareto.*

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 too small in Figure L.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.<sup>5</sup>

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) {
```

---

<sup>5</sup>There 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.)

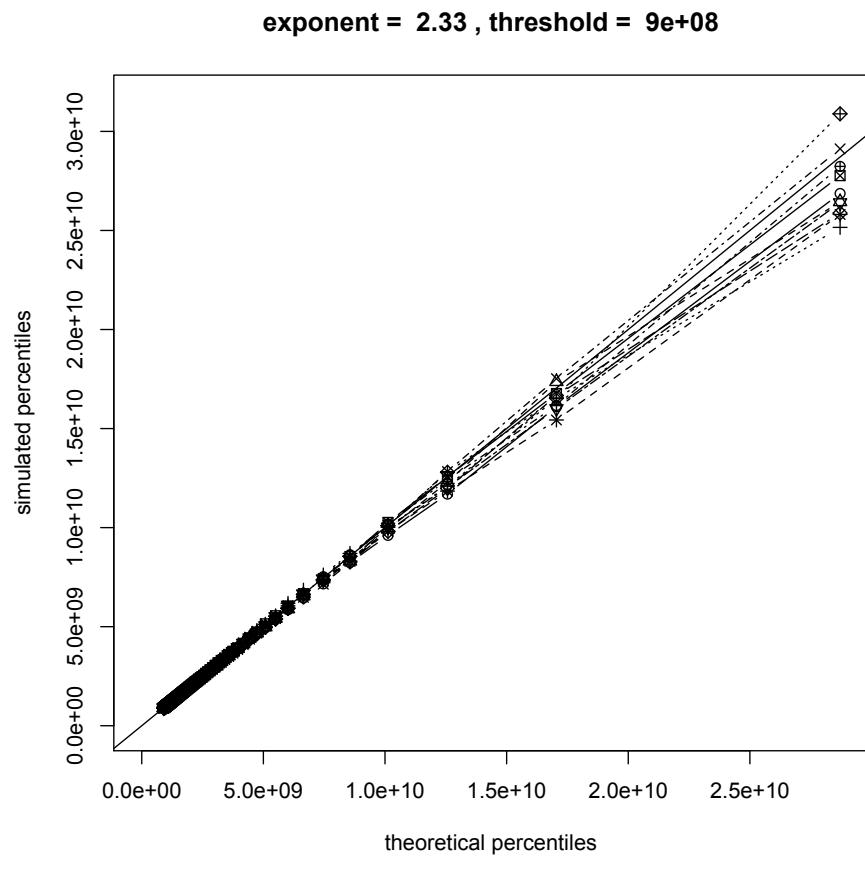
```

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)),
 # 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 L.6 shows that this succeeds.



```
check.rpareto(1e4,2.33,9e8)
```

FIGURE L.6: Using the corrected simulation checker.

## L.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 iteration<sup>6</sup>, 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)
}
```

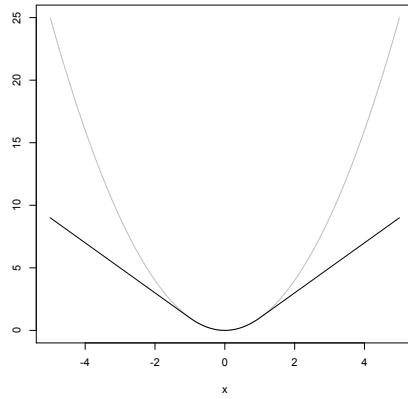
This feeds `qpareto.4` a *vector* of quantiles `p`, of length `n`, which in turn gets passed along to `qpareto.1`, which finally tries to evaluate

```
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.

---

<sup>6</sup>Roughly 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 L.7: The Huber loss function  $\psi$  (black) versus the squared error loss (grey).

### L.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 loss<sup>7</sup>,

$$\psi(x) = \begin{cases} x^2 & \text{if } |x| \leq 1 \\ 2|x| - 1 & \text{if } |x| > 1 \end{cases} \quad (\text{L.4})$$

which isn't so vulnerable to outliers, as in Figure L.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)
}
```

---

<sup>7</sup>One applies this not to the residuals directly, but to residuals divided by some robust measure of dispersion.

This is not very easy to 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 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)
 }
}
```

## L.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
```

```
[1] 21.54435
```

```
[1] 100
```

```
[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 lead to 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 |              |              |

## L.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

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i \quad (\text{L.5})$$

which is just the sample mean, and variance

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2 \quad (\text{L.6})$$

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 factor<sup>8</sup>. 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

---

<sup>8</sup>Clearly, 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 variance<sup>9</sup>. 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
```

## L.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)
```

---

<sup>9</sup>If  $n$  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 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.

## L.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 future<sup>10</sup>. 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.

### L.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.

---

<sup>10</sup>And, in this class, with your graders.

### L.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).

### L.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 functions 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.)

#### L.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.

#### L.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-Sinai category, but there are many much worse ways to start.)

#### L.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.

## L.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 M

# Generating Random Variables

[[TODO: Opening/back-ref to Ch. 5.]]

[[ATTN: Make this a purely online supplement?]]

[[ATTN: Move figure files]]

Note to students: This section may be skipped for 2015

[[TODO: Move to online supplement]]

### M.1 Rejection Method

The quantile method of §5.2.2.3 is a very general approach to drawing from a given distribution, but presumes we can compute the quantile function efficiently. Another general alternative, which avoids needing quantiles, 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.  $\rho$ , 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  $\rho$  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  $\rho$ .
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  $\rho$  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 47.

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 function we are looking for. If  $\rho$  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

```
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 47: 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.

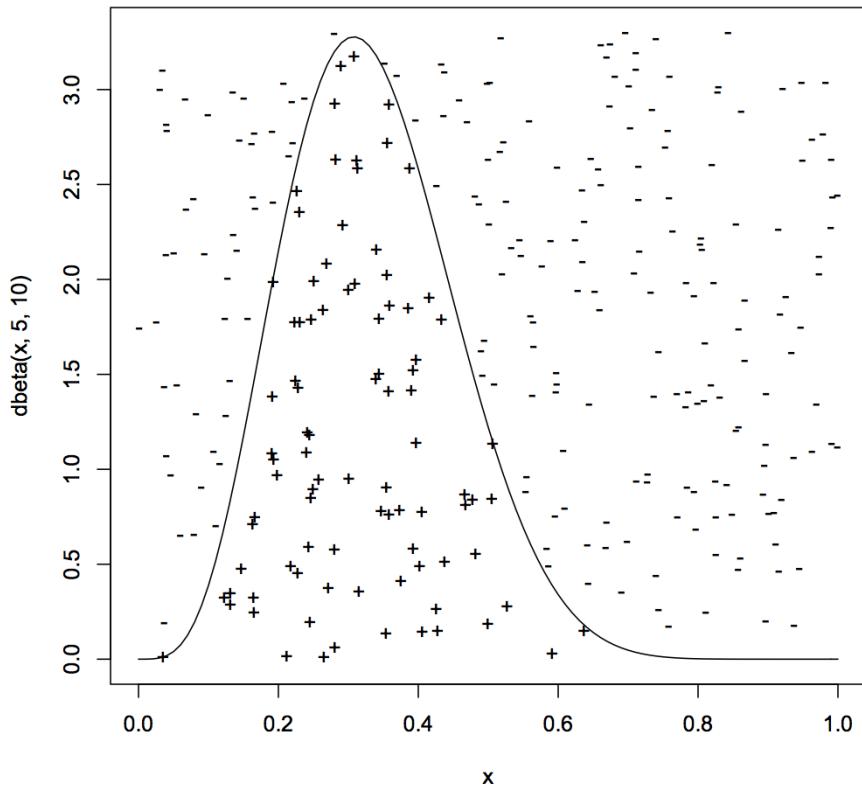
which fall under the curve. When  $\rho$  is not uniform, but we can sample from it nonetheless, then we are uniformly sampling from the area under  $M\rho$ , and keeping only the points which are also below  $f_Z$ .

*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 interval<sup>1</sup>. 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 M.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 1a). Thus produce  $n$  random variables from it takes, on average,  $nM$  cycles (exercise 1b). Clearly, we want  $M$  to be as small, which means that we want the proposal distribution  $\rho$  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.

[[TODO: Rejection method when proposal distribution is non-uniform]]

<sup>1</sup>Here  $\Gamma(a) = \int_0^\infty dx e^{-x} x^{a-1}$ . 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*dunif(r,min=0,max=1) <= dbeta(r,5,10))
points(r[below],M*u[below],pch="+")
points(r[-below],M*u[-below],pch="-")
```

FIGURE M.1: Illustration of the rejection method for generating random numbers from a  $\text{Beta}(5,10)$  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.

## M.2 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.<sup>2</sup>

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

---

<sup>2</sup>And if the dependence between  $Z_t$  and  $Z_{t+1}$  bothers us, we can always randomly permute them, once we have them.

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  $Z_0$  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  $\rho$  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.

### M.3 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.

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 or **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.

[[TODO: Cites on Kolmogorov complexity]]

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

$$\theta_t = \theta_0 + t\alpha \bmod 60 \quad (\text{M.1})$$

If  $\phi$  is **commensurable** with 60, meaning  $\alpha/60 = k/m$  for some integers  $k, m$ , then the positions would just repeat every  $60k$  seconds. If  $\alpha$  is **incommensurable**, because it is an irrational number, then  $\theta_t$  never repeats. In this case, not only does  $\theta_t$  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 2).

You *could* use this as a pseudo-random number generator, with  $\theta_0$  as the seed, but it would not be a very good one, for two reasons. First, exactly representing an irrational number  $\alpha$  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  $\theta_t$  are really too close to each other, and too similar. Even if we only took, say, every 50<sup>th</sup> 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  $\beta$  is incommensurable with both 60 and with  $\alpha$ . We record  $\theta_t$  when  $\phi_t$  is within some small window of 0.<sup>3</sup>

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

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

This is known as “Arnold’s cat map”, after the great Soviet mathematician V. I. Arnold, and Figure M.2. We can think of this as the second-hand  $\theta_t$  advancing not by a fixed amount  $\alpha$  every second, but by a varying amount  $\phi_t$ . The variable  $\phi_t$ , 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  $\theta_t$ , we get a sequence of numbers which, while deterministic, is uniformly distributed, and very hard to predict (Figure M.3).

[[TODO:  
twisters]]

Mersenne

## M.4 Further Reading

There are good discussions of methods of random variable generation in Press *et al.* (1992) [[Monahan]] [[others]]. At a higher level of technicality, [[Devroye]] is authoritative. Press *et al.* (1992) also covers techniques for generating uniform random numbers. Also, [[refs. on discrepancy theory/equidistribution.]] Ruelle (1991) provides a thought-provoking and inspiring guide to the idea that a deterministic system can look random, among many other topics.

[[TODO: Clean up]]

On Monte Carlo: [[Robert and Casella]] is a standard authority on applications and techniques common in statistics. Newman and Barkema (1999) is excellent if you know some physics, especially thermodynamics.

On Markov chain Monte Carlo, the books already named, together with the [[Gilks book]], are worth consulting. Anyone attempting to use the technique should read both Geyer (1992), attacking such practices as “burn-in”, “thinning”, the use of convergence diagnostics, etc., and their defense by [[Gelman and Rubin]]. Geyer has, to my mind, the better of the argument, but people I respect strongly disagree.

---

<sup>3</sup>The 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).

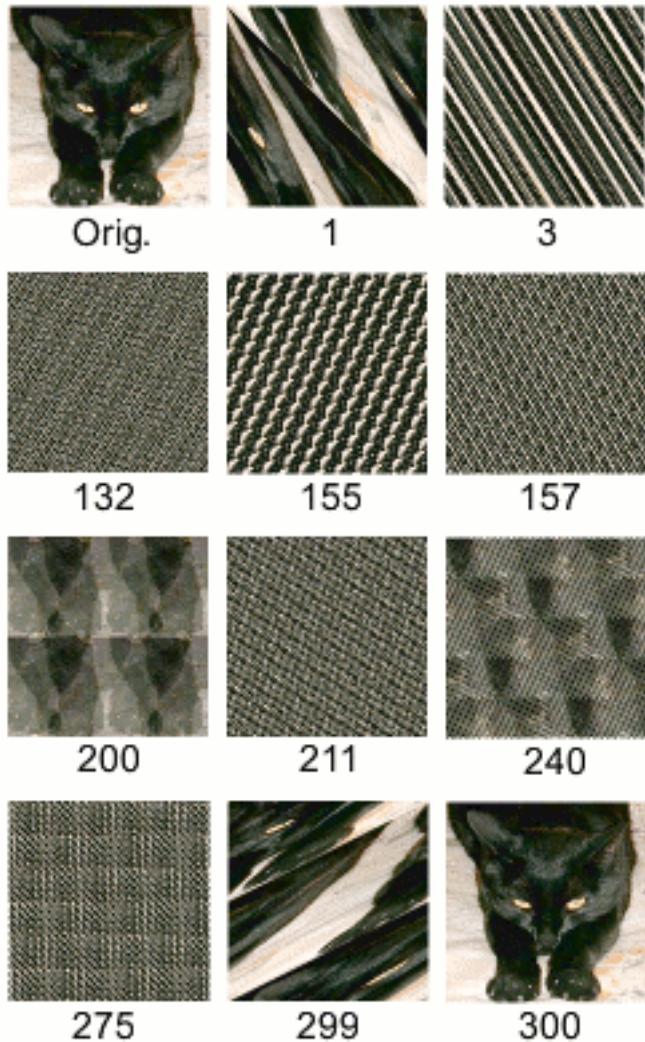


FIGURE M.2: Effect of the Arnold cat map. The original image is  $300 \times 300$ , and mapped into the unit square. The cat map is then applied to the coordinates of each pixel separately, giving a new pixel which inherits the old color. (This can most easily seen in the transition from the original 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>.)

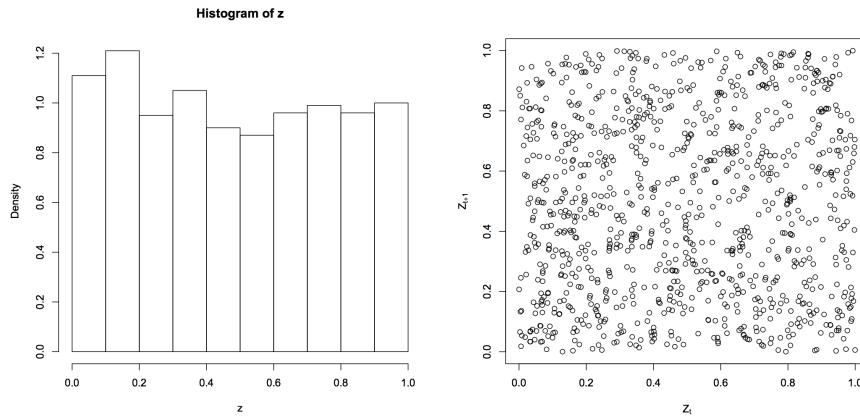
```

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 48: A function implementing the Arnold cat map (Eq. M.3), 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 M.3: Left: histogram from 1000 samples of the  $\theta$  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.

## M.5 Exercises

[[TODO: Currently scattered through the text.]]

1. (a) Prove that a draw in the rejection method of §M.1 is accepted with probability  $1/M$ . That is, the rejection probability of the rejection method is  $1 - 1/M$ .  
(b) Prove that generating  $n$  random values with the rejection method requires on average  $nM$  proposals.
2. Prove that when  $\alpha$  in Eq. M.1 is irrational, then the fraction of times at which  $\theta_t$  falls into any given interval  $[a, b]$  is proportional to its length:

$$\frac{1}{n} \sum_{t=1}^n \mathbf{1}_{[a,b]} \theta_t \xrightarrow{n \rightarrow \infty} \frac{b-a}{60} \quad (\text{M.3})$$

# Acknowledgments

I am grateful to my students in 36-402 for their putting up with early drafts, and for their feedback. Martin Gold and Danny Yee made detailed and valuable comments on early versions. Thanks for specific comments and corrections to Bob Carpenter, Kathy Chen, Brad DeLong, Beatriz Estefania Etchegaray, Njál Foldnes, Manuel Garber, Ben Hansen, Anas Hoque, Rafael Izbicki, Xi Jin, Kent Johnson, Kidong “Justin” Kim, Shubao Liu, Terra Mack, Shaina Mitchell, Sinnott Murphy, Spencer Nelson, Brendan O’Connor, Mark T. Patterson, Dana Peck, Daniel Posen, Calvin Price, David Pugh, Janet E. Rosenbaum, Donald Schoolmaster, Jr., Howard Seltzman, Stephanie Stern, Nicholas Thieme, Ryan Tibshirani, Johan Ugander, and Peter Windridge.

While I was working on this book, my research was supported by grants from the National Science Foundation, the National Institutes of Health, and the Institute for New Economic Thinking. While not intended to pay for writing a textbook, money and time are fungible, so they helped. None of those funders are, of course, in any way responsible for what I have written here.

# 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.
- Amari, Shun-ichi and Hiroshi Nagaoka (1993/2000). *Methods of Information Geometry*. Providence, Rhode Island: American Mathematical Society. Translated by Daishi Harada. As *Joho Kika no Hoho*, Tokyo: Iwanami Shoten Publishers.
- Anthony, Martin and Peter L. Bartlett (1999). *Neural Network Learning: Theoretical Foundations*. Cambridge, England: Cambridge University Press.
- 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.
- Arlot, Sylvain and Alain Celisse (2010). “A survey of cross-validation procedures for model selection.” *Statistics Surveys*, 4: 40–79. URL <http://projecteuclid.org/euclid.ssu/1268143839>.
- Arnold, Barry C. (1983). *Pareto Distributions*. Fairland, Maryland: International Cooperative Publishing House.
- Arnol'd, V. I. (1973). *Ordinary Differential Equations*. Cambridge, Massachusetts: MIT Press. Trans. Richard A. Silverman from *Obyknovennye differentsiyal'nye Uravneniya*.
- Axler, Sheldon (1996). *Linear Algebra Done Right*. Undergraduate Texts in Mathematics. Berlin: Springer-Verlag.

- Bai, Jushan (2003). “Testing Parametric Conditional Distributions of Dynamic Models.” *The Review of Economics and Statistics*, 85: 531–549. doi:10.1162/003465303322369704.
- Barndorff-Nielsen, O. E. (1978). *Information and Exponential Families in Statistical Theory*. New York: John Wiley and Sons.
- Barndorff-Nielsen, O. E. and D. R. Cox (1995). *Inference and Asymptotics*. London: Chapman and Hall.
- 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>.
- Belkin, Mikhail and Partha Niyogi (2003). “Laplacian Eigenmaps for Dimensionality Reduction and Data Representation.” *Neural Computation*, 15: 1373–1396. URL [http://www.cse.ohio-state.edu/~mbelkin/papers/LEM\\_NC\\_03.pdf](http://www.cse.ohio-state.edu/~mbelkin/papers/LEM_NC_03.pdf). doi:10.1162/089976603321780317.
- 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.
- (2008). *Statistical Learning from a Regression Perspective*. Springer Series in Statistics. New York: Springer-Verlag.
- Biecek, Przemyslaw and Teresa Ledwina (2010). *ddst: Data driven smooth test*. URL <http://CRAN.R-project.org/package=ddst>. R package, version 1.02.
- Blackwell, David and M. A. Girshick (1954). *Theory of Games and Statistical Decisions*. New York: Wiley.

- 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>.
- Boas, Mary L. (1983). *Mathematical Methods in the Physical Sciences*. New York: Wiley, 2nd edn.
- Bonner, John Tyler (1988). *The Evolution of Complexity, by Means of Natural Selection*. Princeton, New Jersey: Princeton University Press.
- Borsboom, Denny (2005). *Measuring the Mind: Conceptual Issues in Contemporary Psychometrics*. Cambridge, England: Cambridge University Press.
- (2006). “The Attack of the Psychometricians.” *Psychometrika*, 71: 425–440. URL <https://sites.google.com/site/borsboomdenny/BorsboomPM2006.pdf>. doi:10.1007/s11336-006-1447-6.
- 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](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.
- Breiman, Leo, Jerome Friedman, R. Olshen and C. Stone (1984). *Classification and Regression Trees*. Belmont, California: Wadsworth.
- Breiman, Leo and Jerome H. Friedman (1985). “Estimating Optimal Transformations for Multiple Regression and Correlation.” *Journal of the American Statistical Association*, 80: 580–598. doi:10.1080/01621459.1985.10478157.
- Brown, Lawrence D. (1986). *Fundamentals of Statistical Exponential Families: with Applications in Statistical Decision Theory*. Hayward, California: Institute of Mathematical Statistics. URL <http://projecteuclid.org/euclid.lnms/1215466757>.
- 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 and Brian Ripley (2013). “boot: Bootstrap R (S-Plus) Functions.” R package version 1.3-9. URL <http://cran.r-project.org>.
- 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.
- Casella, George and R. L. Berger (2002). *Statistical Inference*. Belmont, California: Duxbury Press, 2nd edn.
- 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 P. E. Hart (1967). “Nearest Neighbor Pattern Classification.” *IEEE Transactions on Information Theory*, 13: 21–27. URL <http://www.stanford.edu/~cover/papers/transIT/0021cove.pdf>.
- 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.1si.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.
- DuMouchel, William H. and Greg J. Duncan (1983). "Using Sample Survey Weights in Multiple Regression Analyses of Stratified Samples." *Journal of the American Statistical Association*, 78: 535–543. URL <http://www.jstor.org/stable/2288115>.
- 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.), pp. 256–264. URL <http://jmlr.org/proceedings/papers/v31/entner13a.html>.
- 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>.
- Fair, Ray C. (1978). “A Theory of Extramarital Affairs.” *Journal of Political Economy*, **86**: 45–61. URL <http://fairmodel.econ.yale.edu/rayfair/pdf/1978A200.PDF>.
- Fan, Jianqing and Qiwei Yao (2003). *Nonlinear Time Series: Nonparametric and Parametric Methods*. Berlin: Springer-Verlag.
- Faraway, Julian (2004). *Linear Models with R*. Boca Raton, Florida: Chapman and Hall/CRC Press.
- Faraway, Julian J. (2006). *Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models*. Boca Raton, Florida: Chapman and Hall/CRC.
- Fisher, Franklin M. (1983). *Disequilibrium Foundations of Equilibrium Economics*. Cambridge, England: Cambridge University Press.
- (2010). “The Stability of General Equilibrium — What Do We Know and Why Is It Important?” In *General Equilibrium Analysis: A Century after Walras* (Pascal Bridel, ed.), pp. 34–45. London: Routledge. URL <http://economics.mit.edu/files/6988>.
- 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.
- Gershenfeld, Neil (1999). *The Nature of Mathematical Modeling*. Cambridge, England: Cambridge University Press.
- Geyer, Charles J. (1992). "Practical Markov Chain Monte Carlo." *Statistical Science*, **7**: 473–483. doi:10.1214/ss/1177011137.
- (2013). "Asymptotics of Maximum Likelihood without the LLN or CLT or Sample Size Going to Infinity." In *Advances in Modern Statistical Theory and Applications: A Festschrift in honor of Morris L. Eaton* (Galin Jones and Xiaotong Shen, eds.), pp. 1–24. Beachwood, Ohio: Institute of Mathematical Statistics. URL <http://arxiv.org/abs/1206.4762>. doi:10.1214/12-IMSCOLL1001.
- 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>.
- (1998). "What Went Wrong? Reflections on Science by Observation and *The Bell Curve*." *Philosophy of Science*, **65**: 1–32. URL <http://www.hss.cmu.edu/philosophy/glymour/glymour1998.pdf>.
- (2001). *The Mind's Arrows: Bayes Nets and Graphical Causal Models in Psychology*. Cambridge, Massachusetts: MIT Press.
- Gnedenko, B. V. and A. N. Kolmogorov (1954). *Limit Distributions for Sums of Independent Random Variables*. Cambridge, Massachusetts: Addison-Wesley. Translated from the Russian and annotated by K. L. Chung, with an Appendix by J. L. Doob.
- Godfrey, L. G. (1988). *Misspecification Tests in Econometrics; The Lagrange Multiplier Principle and Other Approaches*. Cambridge, England: Cambridge University 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.
- Grünwald, Peter D. (2007). *The Minimum Description Length Principle*. Cambridge, Massachusetts: MIT Press.
- Guckenheimer, John and Philip Holmes (1983). *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields*. New York: Springer-Verlag.
- 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.
- Haldane, J. B. S. (1985). *On Being the Right Size, and Other Essays*. Oxford: Oxford University Press. Edited and introduced by John Maynard Smith.
- 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>.
- Hand, David, Heikki Mannila and Padhraic Smyth (2001). *Principles of Data Mining*. Cambridge, Massachusetts: MIT Press.
- 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](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.
- Hong, Yongmiao and Halbert White (1995). "Consistent Specification Testing Via Nonparametric Series Regression." *Econometrica*, 63: 1133–1159. URL <http://www.jstor.org/stable/2171724>.
- 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 Léon Bottou, eds.), pp. 689–696. Cambridge, Massachusetts: MIT Press. URL [http://books.nips.cc/papers/files/nips21/NIPS2008\\_0266.pdf](http://books.nips.cc/papers/files/nips21/NIPS2008_0266.pdf).
- Huber, Peter J. (1967). "The Behavior of Maximum Likelihood Estimates under Nonstandard Conditions." In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* (Lucien M. Le Cam and Jerzy Neyman, eds.), vol. 1, pp. 221–233. Berkeley: University of California Press. URL <http://projecteuclid.org/euclid.bsmsp/1200512988>.
- 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." arxiv.org. 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. URL <ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-1440.pdf>. doi:10.1162/neco.1994.6.2.181.
- 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>.

- Kass, Robert E. and Paul W. Vos (1997). *Geometrical Foundations of Asymptotic Inference*. New York: Wiley.
- 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/comm1/](http://www.ams.org/online_bks/comm1/).
- 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 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.
- Lambert, Diane and Kathryn Roeder (1995). “Overdispersion Diagnostics for Generalized Linear Models.” *Journal of the American Statistical Association*, 90: 1225–1236. URL <http://www.jstor.org/stable/2291513>.
- 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/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.
- Lawrie, Ian D. (1990). *A Unified Grand Tour of Theoretical Physics*. Bristol, England: Adam Hilger.

- Lee, Ann B. and Larry Wasserman (2010). "Spectral Connectivity Analysis." *Journal of the American Statistical Association*, 105: 1241–1255. URL <http://arxiv.org/abs/0811.0121>. doi:10.1198/jasa.2010.tm09754.
- 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/buhlmann/maathuisetal2010.pdf>. doi:10.1038/nmeth0410-247. See also <http://stat.ethz.ch/Manuscripts/buhlmann/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](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.
- Mitchell, Tom M. (1997). *Machine Learning*. New York: McGraw-Hill.
- 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.
- Nadaraya, E. A. (1964). "On Estimating Regression." *Theory of Probability and Its Applications*, **9**: 141–142. doi:10.1137/1109020.
- 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.
- Newman, Mark E. J. and G. T. Barkema (1999). *Monte Carlo Methods in Statistical Physics*. Oxford: Clarendon Press.
- 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.
- Press, William H., Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge, England: Cambridge University Press, 2nd edn. URL <http://www.nrbook.com/>.
- 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.
- Racine, Jeffrey S. (2008). "Nonparametric Econometrics: A Primer." *Foundations and Trends in Econometrics*, **3**: 1–88. URL <http://socserv.mcmaster.ca/racine/EC00301.pdf>. doi:10.1561/0800000009.
- Raginsky, Maxim (2011). "Directed Information and Pearl's Causal Calculus." In *Proceedings of the 49th Annual Allerton Conference on Communication, Control and Computing* (S. Meyn and B. Hajek, eds.), pp. 958–965. IEEE. URL <http://arxiv.org/abs/1110.0718>. doi:10.1109/Allerton.2011.6120270.

- 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.
- Reid, Constance (1982). *Neyman from Life*. New York: Springer-Verlag.
- Reinsch, Christian H. (1967). "Smoothing by Spline Functions." *Numerische Mathematik*, 10: 177–183.
- 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.
- Ripley, Brian D. (1996). *Pattern Recognition and Neural Networks*. Cambridge, England: Cambridge University Press.
- 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.
- Roweis, Sam T. and Laurence K. Saul (2000). "Nonlinear Dimensionality Reduction by Locally Linear Embedding." *Science*, 290: 2323–2326. doi:10.1126/science.290.5500.2323.
- 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*. 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>.
- Saul, Lawrence K. and Sam T. Roweis (2003). “Think Globally, Fit Locally: Supervised Learning of Low Dimensional Manifolds.” *Journal of Machine Learning Research*, 4: 119–155. URL <http://jmlr.csail.mit.edu/papers/v4/saul03a.html>.
- Schoenberg, I. J. (1964). “Spline Functions and the Problem of Graduation.” *Proceedings of the National Academy of Sciences (USA)*, 52: 947–950.
- Schutz, Bernard F. (1980). *Geometrical Methods of Mathematical Physics*. Cambridge, England: Cambridge University Press.
- Schwarz, Gideon (1978). “Estimating the Dimension of a Model.” *Annals of Statistics*, 6: 461–464. URL <http://projecteuclid.org/euclid-aos/1176344136>.
- Scott, Clayton and Robert Nowak (2005). “A Neyman-Pearson Approach to Statistical Learning.” *IEEE Transactions on Information Theory*, 51: 3806–3819. URL <http://www.ece.wisc.edu/~nowak/np.pdf>. doi:10.1109/TIT.2005.856955.
- 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.
- Shawe-Taylor, John and Nello Cristianini (2004). *Kernel Methods for Pattern Analysis*. Cambridge, England: Cambridge University 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.
- Silverman, B. W. (1984). “Spline Smoothing: The Equivalent Variable Kernel Method.” *Annals of Statistics*, 12: 898–916. doi:10.1214/aos/1176346710.
- (1985). “Some Aspects of the Spline Smoothing Approach to Non-Parametric Regression Curve Fitting.” *Journal of the Royal Statistical Society B*, 47: 1–52. URL <http://www.jstor.org/stable/2345542>.
- 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.
- Spain, Seth M., Kristin L. Sotak, Joey (Chou-Yu) Tsai, P. D. Harms and Sean T. Hannah (2012). “Testing the Form of Theoretical Models by Relaxing Assumptions: Comparing Parametric and Nonparametric Models.” Electronic preprint, SSRN/2164297. URL <http://ssrn.com/abstract=2164297>.
- Spanos, Aris (2011). “A Frequentist Interpretation of Probability for Model-based Inductive Inference.” *Synthese*, 190. URL [http://www.econ.vt.edu/faculty/2008vitas\\_research/Spanos/1Spanos-2011-Synthese.pdf](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/>.
- Spector, Phil, Jerome Friedman, Robert Tibshirani and Thomas Lumley (2013). *acepack: ace() and avas() for selecting regression transformations*. URL <http://CRAN.R-project.org/package=acepack>. R package version 1.3-3.3.
- 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.
- Spivak, Michael (1965). *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*. Menlo Park, California: Benjamin Cummings.
- 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>.
- Stigler, Stephen M. (1986). *The History of Statistics: The Measurement of Uncertainty before 1900*. Cambridge, Massachusetts: Harvard University Press.

- Stone, M. (1974). "Cross-validatory choice and assessment of statistical predictions." *Journal of the Royal Statistical Society B*, **36**: 111–147. URL <http://www.jstor.org/stable/2984809>.
- 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/factorialanalysi032965mbp>.
- 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>.
- Tutz, Gerhard (2012). *Regression for Categorical Data*. Cambridge, England: Cambridge University Press.
- van der Vaart, A. W. (1998). *Asymptotic Statistics*. Cambridge, England: Cambridge University Press.
- Vapnik, Vladimir N. (2000). *The Nature of Statistical Learning Theory*. Berlin: Springer-Verlag, 2nd edn.
- Vidyasagar, Mathukumalli (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.
- Watson, Geoffrey S. (1964). "Smooth Regression Analysis." *Sanhkyā*, 26: 359–372. URL <http://www.jstor.org/stable/25049340>.
- Weisberg, Sanford (1985). *Applied Linear Regression*. New York: Wiley, 2nd edn.
- White, Halbert (1994). *Estimation, Inference and Specification Analysis*. Cambridge, England: Cambridge University Press.
- 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 (1956). "Nonlinear Prediction and Dynamics." In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability* (Jerzy Neyman, ed.), vol. 3, pp. 247–252. Berkeley: University of California Press. URL <http://projecteuclid.org/euclid.bsmsp/1200502197>.
- (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>.