
Coding the Matrix

*Linear Algebra
through Applications to Computer Science*

Edition 1

PHILIP N. KLEIN

Brown University

The companion website is at codingthematrix.com. There you will find, in digital form, the data, examples, and support code you need to solve the problems given in the book. Auto-grading will be provided for many of the problems.

Contents

0	The Function (and other mathematical and computational preliminaries)	1
0.1	Set terminology and notation	1
0.2	Cartesian product	2
0.3	The function	2
0.3.1	Functions versus procedures, versus computational problems	4
0.3.2	The two computational problems related to a function	5
0.3.3	Notation for the set of functions with given domain and co-domain	6
0.3.4	Identity function	6
0.3.5	Composition of functions	6
0.3.6	Associativity of function composition	7
0.3.7	Functional inverse	8
0.3.8	Invertibility of the composition of invertible functions	10
0.4	Probability	12
0.4.1	Probability distributions	12
0.4.2	Events, and adding probabilities	14
0.4.3	Applying a function to a random input	14
0.4.4	Perfect secrecy	16
0.4.5	Perfect secrecy and invertible functions	18
0.5	<i>Lab: Introduction to Python—sets, lists, dictionaries, and comprehensions</i>	19
0.5.1	<i>Simple expressions</i>	20
0.5.2	<i>Assignment statements</i>	22
0.5.3	<i>Conditional expressions</i>	22
0.5.4	<i>Sets</i>	23
0.5.5	<i>Lists</i>	28
0.5.6	<i>Tuples</i>	33
0.5.7	Other things to iterate over	34
0.5.8	<i>Dictionaries</i>	36
0.5.9	<i>Defining one-line procedures</i>	40
0.6	<i>Lab: Python—modules and control structures—and inverse index</i>	42
0.6.1	<i>Using existing modules</i>	42
0.6.2	<i>Creating your own modules</i>	43
0.6.3	<i>Loops and conditional statements</i>	44
0.6.4	<i>Grouping in Python using indentation</i>	45

0.6.5	<i>Breaking out of a loop</i>	46
0.6.6	<i>Reading from a file</i>	46
0.6.7	<i>Mini-search engine</i>	46
0.7	Review questions	48
0.8	Problems	48
1	The Field	51
1.1	Introduction to complex numbers	51
1.2	Complex numbers in Python	52
1.3	Abstracting over <i>fields</i>	53
1.4	Playing with \mathbb{C}	54
1.4.1	The absolute value of a complex number	56
1.4.2	Adding complex numbers	57
1.4.3	Multiplying complex numbers by a positive real number	59
1.4.4	Multiplying complex numbers by a negative number: rotation by 180 degrees	60
1.4.5	Multiplying by i : rotation by 90 degrees	61
1.4.6	The unit circle in the complex plane: <i>argument</i> and angle	63
1.4.7	Euler's formula	65
1.4.8	Polar representation for complex numbers	66
1.4.9	The First Law of Exponentiation	67
1.4.10	Rotation by τ radians	67
1.4.11	Combining operations	69
1.4.12	Beyond two dimensions	69
1.5	Playing with $GF(2)$	69
1.5.1	Perfect secrecy revisited	70
1.5.2	Network coding	72
1.6	Review questions	74
1.7	Problems	75
2	The Vector	79
2.1	What is a vector?	80
2.2	Vectors are functions	81
2.2.1	Representation of vectors using Python dictionaries	83
2.2.2	Sparsity	83
2.3	What can we represent with vectors?	84
2.4	Vector addition	86
2.4.1	Translation and vector addition	86
2.4.2	Vector addition is associative and commutative	87
2.4.3	Vectors as arrows	88
2.5	Scalar-vector multiplication	90
2.5.1	Scaling arrows	91
2.5.2	Associativity of scalar-vector multiplication	92
2.5.3	Line segments through the origin	92
2.5.4	Lines through the origin	93
2.6	Combining vector addition and scalar multiplication	94

2.6.1	Line segments and lines that don't go through the origin	94
2.6.2	Distributive laws for scalar-vector multiplication and vector addition . . .	96
2.6.3	First look at convex combinations	97
2.6.4	First look at affine combinations	99
2.7	Dictionary-based representations of vectors	99
2.7.1	Setter and getter	100
2.7.2	Scalar-vector multiplication	101
2.7.3	Addition	102
2.7.4	Vector negative, invertibility of vector addition, and vector subtraction . .	103
2.8	Vectors over $GF(2)$	104
2.8.1	Perfect secrecy re-revisited	104
2.8.2	All-or-nothing secret-sharing using $GF(2)$	105
2.8.3	<i>Lights Out</i>	106
2.9	Dot-product	111
2.9.1	Total cost or benefit	112
2.9.2	Linear equations	113
2.9.3	Measuring similarity	115
2.9.4	Dot-product of vectors over $GF(2)$	119
2.9.5	Parity bit	120
2.9.6	Simple authentication scheme	120
2.9.7	Attacking the simple authentication scheme	122
2.9.8	Algebraic properties of the dot-product	123
2.9.9	Attacking the simple authentication scheme, revisited	125
2.10	Our implementation of <code>Vec</code>	126
2.10.1	Syntax for manipulating <code>Vecs</code>	126
2.10.2	The implementation	127
2.10.3	Using <code>Vecs</code>	127
2.10.4	Printing <code>Vecs</code>	128
2.10.5	Copying <code>Vecs</code>	128
2.10.6	From list to <code>Vec</code>	128
2.11	Solving a triangular system of linear equations	129
2.11.1	Upper-triangular systems	129
2.11.2	Backward substitution	130
2.11.3	First implementation of backward substitution	131
2.11.4	When does the algorithm work?	133
2.11.5	Backward substitution with arbitrary-domain vectors	133
2.12	<i>Lab: Comparing voting records using dot-product</i>	134
2.12.1	<i>Motivation</i>	134
2.12.2	<i>Reading in the file</i>	135
2.12.3	<i>Two ways to use dot-product to compare vectors</i>	135
2.12.4	<i>Policy comparison</i>	136
2.12.5	<i>Not your average Democrat</i>	136
2.12.6	<i>Bitter Rivals</i>	137
2.12.7	<i>Open-ended study</i>	137
2.13	Review Questions	138

2.14	Problems	138
3	The Vector Space	143
3.1	Linear combination	143
3.1.1	Definition of linear combination	143
3.1.2	Uses of linear combinations	144
3.1.3	From coefficients to linear combination	146
3.1.4	From linear combination to coefficients	147
3.2	Span	148
3.2.1	Definition of span	148
3.2.2	A system of linear equations implies other equations	149
3.2.3	Generators	151
3.2.4	Linear combinations of linear combinations	152
3.2.5	Standard generators	153
3.3	The geometry of sets of vectors	155
3.3.1	The geometry of the span of vectors over \mathbb{R}	155
3.3.2	The geometry of solution sets of homogeneous linear systems	157
3.3.3	The two representations of flats containing the origin	159
3.4	Vector spaces	160
3.4.1	What's common to the two representations?	160
3.4.2	Definition and examples of vector space	161
3.4.3	Subspaces	162
3.4.4	*Abstract vector spaces	164
3.5	Affine spaces	164
3.5.1	Flats that don't go through the origin	164
3.5.2	Affine combinations	166
3.5.3	Affine spaces	168
3.5.4	Representing an affine space as the solution set of a linear system	170
3.5.5	The two representations, revisited	171
3.6	Linear systems, homogeneous and otherwise	176
3.6.1	The homogeneous linear system corresponding to a general linear system	176
3.6.2	Number of solutions revisited	178
3.6.3	Towards intersecting a plane and a line	179
3.6.4	Checksum functions	179
3.7	Review questions	181
3.8	Problems	182
4	The Matrix	185
4.1	What is a matrix?	185
4.1.1	Traditional matrices	185
4.1.2	The matrix revealed	187
4.1.3	Rows, columns, and entries	188
4.1.4	Our Python implementation of matrices	189
4.1.5	Identity matrix	190
4.1.6	Converting between matrix representations	190

4.1.7	<code>matutil.py</code>	191
4.2	Column space and row space	192
4.3	Matrices as vectors	193
4.4	Transpose	193
4.5	Matrix-vector and vector-matrix multiplication in terms of linear combinations	194
4.5.1	Matrix-vector multiplication in terms of linear combinations	194
4.5.2	Vector-matrix multiplication in terms of linear combinations	195
4.5.3	Formulating <i>expressing a given vector as a linear-combination</i> as a matrix-vector equation	197
4.5.4	Solving a matrix-vector equation	198
4.6	Matrix-vector multiplication in terms of dot-products	200
4.6.1	Definitions	200
4.6.2	Example applications	201
4.6.3	Formulating a system of linear equations as a matrix-vector equation	204
4.6.4	Triangular systems and triangular matrices	206
4.6.5	Algebraic properties of matrix-vector multiplication	207
4.7	Null space	208
4.7.1	Homogeneous linear systems and matrix equations	208
4.7.2	The solution space of a matrix-vector equation	210
4.7.3	Introduction to error-correcting codes	211
4.7.4	Linear codes	212
4.7.5	The Hamming Code	212
4.8	Computing sparse matrix-vector product	213
4.9	The matrix meets the function	214
4.9.1	From matrix to function	214
4.9.2	From function to matrix	215
4.9.3	Examples of deriving the matrix	215
4.10	Linear functions	218
4.10.1	Which functions can be expressed as a matrix-vector product	218
4.10.2	Definition and simple examples	219
4.10.3	Linear functions and zero vectors	221
4.10.4	What do linear functions have to do with lines?	221
4.10.5	Linear functions that are one-to-one	222
4.10.6	Linear functions that are onto?	223
4.10.7	A linear function from \mathbb{F}^C to \mathbb{F}^R can be represented by a matrix	224
4.10.8	Diagonal matrices	225
4.11	Matrix-matrix multiplication	225
4.11.1	Matrix-matrix multiplication in terms of matrix-vector and vector-matrix multiplication	226
4.11.2	Graphs, incidence matrices, and counting paths	229
4.11.3	Matrix-matrix multiplication and function composition	233
4.11.4	Transpose of matrix-matrix product	236
4.11.5	Column vector and row vector	237
4.11.6	Every vector is interpreted as a column vector	238
4.11.7	Linear combinations of linear combinations revisited	238

4.12	Inner product and outer product	239
4.12.1	Inner product	239
4.12.2	Outer product	240
4.13	From function inverse to matrix inverse	240
4.13.1	The inverse of a linear function is linear	240
4.13.2	The matrix inverse	241
4.13.3	Uses of matrix inverse	243
4.13.4	The product of invertible matrices is an invertible matrix	244
4.13.5	More about matrix inverse	246
4.14	<i>Lab: Error-correcting codes</i>	247
4.14.1	<i>The check matrix</i>	248
4.14.2	<i>The generator matrix</i>	248
4.14.3	<i>Hamming's code</i>	248
4.14.4	<i>Decoding</i>	249
4.14.5	<i>Error syndrome</i>	249
4.14.6	<i>Finding the error</i>	249
4.14.7	<i>Putting it all together</i>	250
4.15	<i>Lab: Transformations in 2D geometry</i>	252
4.15.1	<i>Our representation for points in the plane</i>	252
4.15.2	<i>Transformations</i>	253
4.15.3	<i>Image representation</i>	254
4.15.4	<i>Loading and displaying images</i>	256
4.15.5	<i>Linear transformations</i>	256
4.15.6	<i>Translation</i>	256
4.15.7	<i>Scaling</i>	257
4.15.8	<i>Rotation</i>	257
4.15.9	<i>Rotation about a center other than the origin</i>	258
4.15.10	<i>Reflection</i>	258
4.15.11	<i>Color transformations</i>	258
4.15.12	<i>Reflection more generally</i>	258
4.16	Review questions	258
4.17	Problems	259
5	The Basis	269
5.1	Coordinate systems	269
5.1.1	René Descartes' idea	269
5.1.2	Coordinate representation	270
5.1.3	Coordinate representation and matrix-vector multiplication	270
5.2	First look at lossy compression	271
5.2.1	Strategy 1: Replace vector with closest sparse vector	271
5.2.2	Strategy 2: Represent image vector by its coordinate representation	272
5.3	Two greedy algorithms for finding a set of generators	274
5.3.1	Grow algorithm	275
5.3.2	Shrink algorithm	275
5.3.3	When greed fails	276

5.4	Minimum Spanning Forest and $GF(2)$	277
5.4.1	Definitions	278
5.4.2	The Grow and Shrink algorithms for <i>Minimum Spanning Forest</i>	279
5.4.3	Formulating <i>Minimum Spanning Forest</i> in linear algebra	280
5.5	Linear dependence	282
5.5.1	The Superfluous-Vector Lemma	282
5.5.2	Defining linear dependence	283
5.5.3	Linear dependence in <i>Minimum Spanning Forest</i>	284
5.5.4	Properties of linear (in)dependence	285
5.5.5	Analyzing the Grow algorithm	287
5.5.6	Analyzing the Shrink algorithm	287
5.6	Basis	288
5.6.1	Defining basis	288
5.6.2	The standard basis for \mathbb{F}^D	292
5.6.3	Towards showing that every vector space has a basis	292
5.6.4	Any finite set of vectors contains a basis for its span	292
5.6.5	Can any linearly independent subset of vectors belonging to \mathcal{V} be extended to form a basis for \mathcal{V} ?	293
5.7	Unique representation	294
5.7.1	Uniqueness of representation in terms of a basis	294
5.8	Change of basis, first look	295
5.8.1	The function from representation to vector	295
5.8.2	From one representation to another	295
5.9	Perspective rendering	296
5.9.1	Points in the world	296
5.9.2	The camera and the image plane	297
5.9.3	The camera coordinate system	299
5.9.4	From the camera coordinates of a point in the scene to the camera coordinates of the corresponding point in the image plane	300
5.9.5	From world coordinates to camera coordinates	302
5.9.6	... to pixel coordinates	303
5.10	Computational problems involving finding a basis	304
5.11	The Exchange Lemma	305
5.11.1	The lemma	305
5.11.2	Proof of correctness of the Grow algorithm for MSF	306
5.12	<i>Lab: Perspective rectification</i>	306
5.12.1	<i>The camera basis</i>	309
5.12.2	<i>The whiteboard basis</i>	310
5.12.3	<i>Mapping from pixels to points on the whiteboard</i>	310
5.12.4	<i>Mapping a point not on the whiteboard to the corresponding point on the whiteboard</i>	311
5.12.5	<i>The change-of-basis matrix</i>	312
5.12.6	<i>Computing the change-of-basis matrix</i>	313
5.12.7	<i>Image representation</i>	317
5.12.8	<i>Synthesizing the perspective-free image</i>	317

5.13	Review questions	319
5.14	Problems	319
6	Dimension	330
6.1	The size of a basis	330
6.1.1	The Morphing Lemma and its implications	330
6.1.2	Proof of the Morphing Lemma	331
6.2	Dimension and rank	334
6.2.1	Definitions and examples	334
6.2.2	Geometry	336
6.2.3	Dimension and rank in graphs	337
6.2.4	The cardinality of a vector space over $GF(2)$	338
6.2.5	Any linearly independent set of vectors belonging to \mathcal{V} can be extended to form a basis for \mathcal{V}	339
6.2.6	The Dimension Principle	339
6.2.7	The Grow algorithm terminates	340
6.2.8	The Rank Theorem	341
6.2.9	Simple authentication revisited	343
6.3	Direct sum	344
6.3.1	Definition	344
6.3.2	Generators for the direct sum	346
6.3.3	Basis for the direct sum	346
6.3.4	Unique decomposition of a vector	347
6.3.5	Complementary subspaces	348
6.4	Dimension and linear functions	350
6.4.1	Linear function invertibility	350
6.4.2	The largest invertible subfunction	351
6.4.3	The Kernel-Image Theorem	353
6.4.4	Linear function invertibility, revisited	354
6.4.5	The Rank-Nullity Theorem	355
6.4.6	Checksum problem revisited	355
6.4.7	Matrix invertibility	356
6.4.8	Matrix invertibility and change of basis	357
6.5	The annihilator	358
6.5.1	Conversions between representations	358
6.5.2	The annihilator of a vector space	361
6.5.3	The Annihilator Dimension Theorem	363
6.5.4	From generators for \mathcal{V} to generators for \mathcal{V}° , and vice versa	363
6.5.5	The Annihilator Theorem	364
6.6	Review questions	365
6.7	Problems	365

7	Gaussian elimination	374
7.1	Echelon form	375
7.1.1	From echelon form to a basis for row space	376
7.1.2	Rowlist in echelon form	377
7.1.3	Sorting rows by position of the leftmost nonzero	378
7.1.4	Elementary row-addition operations	379
7.1.5	Multiplying by an elementary row-addition matrix	380
7.1.6	Row-addition operations preserve row space	381
7.1.7	Basis, rank, and linear independence through Gaussian elimination	383
7.1.8	When Gaussian elimination fails	383
7.1.9	Pivoting, and numerical analysis	384
7.2	Gaussian elimination over $GF(2)$	384
7.3	Using Gaussian elimination for other problems	385
7.3.1	There is an invertible matrix M such that MA is in echelon form	386
7.3.2	Computing M without matrix multiplications	387
7.4	Solving a matrix-vector equation using Gaussian elimination	390
7.4.1	Solving a matrix-vector equation when the matrix is in echelon form—the invertible case	391
7.4.2	Coping with zero rows	391
7.4.3	Coping with irrelevant columns	391
7.4.4	Attacking the simple authentication scheme, and improving it	392
7.5	Finding a basis for the null space	393
7.6	Factoring integers	394
7.6.1	First attempt at factoring	395
7.7	<i>Lab: Threshold Secret-Sharing</i>	396
7.7.1	<i>First attempt</i>	397
7.7.2	<i>Scheme that works</i>	398
7.7.3	<i>Implementing the scheme</i>	399
7.7.4	<i>Generating mathbfu</i>	399
7.7.5	<i>Finding vectors that satisfy the requirement</i>	400
7.7.6	<i>Sharing a string</i>	400
7.8	<i>Lab: Factoring integers</i>	401
7.8.1	<i>First attempt to use square roots</i>	401
7.8.2	<i>Euclid's algorithm for greatest common divisor</i>	402
7.8.3	<i>Using square roots revisited</i>	402
7.9	Review questions	409
8	The Inner Product	417
8.1	The <i>fire engine</i> problem	417
8.1.1	Distance, length, norm, inner product	418
8.2	The inner product for vectors over the reals	419
8.2.1	Norms of vectors over the reals	419
8.3	Orthogonality	421
8.3.1	Properties of orthogonality	422
8.3.2	Decomposition of \mathbf{b} into parallel and perpendicular components	423

8.3.3	Orthogonality property of the solution to the <i>fire engine</i> problem	425
8.3.4	Finding the projection and the closest point	426
8.3.5	Solution to the <i>fire engine</i> problem	427
8.3.6	*Outer product and projection	428
8.3.7	Towards solving the higher-dimensional version	429
8.4	<i>Lab: machine learning</i>	430
8.4.1	<i>The data</i>	430
8.4.2	<i>Supervised learning</i>	431
8.4.3	<i>Hypothesis class</i>	431
8.4.4	<i>Selecting the classifier that minimizes the error on the training data</i>	432
8.4.5	<i>Nonlinear optimization by hill-climbing</i>	433
8.4.6	<i>Gradient</i>	435
8.4.7	<i>Gradient descent</i>	437
8.5	Review questions	438
8.6	Problems	438
9	Orthogonalization	440
9.1	Projection orthogonal to multiple vectors	441
9.1.1	Orthogonal to a set of vectors	441
9.1.2	Projecting onto and orthogonal to a vector space	442
9.1.3	First attempt at projecting orthogonal to a list of vectors	443
9.2	Projecting orthogonal to <i>mutually orthogonal</i> vectors	445
9.2.1	Proving the correctness of <code>project_orthogonal</code>	446
9.2.2	Augmenting <code>project_orthogonal</code>	448
9.3	Building an orthogonal set of generators	450
9.3.1	The <code>orthogonalize</code> procedure	450
9.3.2	Proving the correctness of <code>orthogonalize</code>	452
9.4	Solving the Computational Problem <i>closest point in the span of many vectors</i>	454
9.5	Solving other problems using <code>orthogonalize</code>	455
9.5.1	Computing a basis	455
9.5.2	Computing a subset basis	456
9.5.3	<code>augmented_orthogonalize</code>	456
9.5.4	Algorithms that work in the presence of rounding errors	457
9.6	Orthogonal complement	457
9.6.1	Definition of orthogonal complement	457
9.6.2	Orthogonal complement and direct sum	458
9.6.3	Normal to a plane in \mathbb{R}^3 given as span or affine hull	459
9.6.4	Orthogonal complement and null space and annihilator	460
9.6.5	Normal to a plane in \mathbb{R}^3 given by an equation	460
9.6.6	Computing the orthogonal complement	461
9.7	The <i>QR</i> factorization	462
9.7.1	Orthogonal and column-orthogonal matrices	462
9.7.2	Defining the <i>QR</i> factorization of a matrix	463
9.7.3	Requiring <i>A</i> to have linearly independent columns	463
9.8	Using the <i>QR</i> factorization to solve a matrix equation $A\mathbf{x} = \mathbf{b}$	464

9.8.1	The square case	464
9.8.2	Correctness in the square case	465
9.8.3	The least-squares problem	466
9.8.4	The coordinate representation in terms of the columns of a column-orthogonal matrix	467
9.8.5	Using <code>QR_solve</code> when A has more rows than columns	468
9.9	Applications of least squares	468
9.9.1	Linear regression (Line-fitting)	468
9.9.2	Fitting to a quadratic	469
9.9.3	Fitting to a quadratic in two variables	470
9.9.4	Coping with approximate data in the <i>industrial espionage</i> problem	471
9.9.5	Coping with approximate data in the <i>sensor node</i> problem	472
9.9.6	Using the method of least squares in the machine-learning problem	473
9.10	Review questions	474
9.11	Problems	475
10	Special Bases	483
10.1	Closest k -sparse vector	483
10.2	Closest vector whose representation with respect to a given basis is k -sparse	484
10.2.1	Finding the coordinate representation in terms of an orthonormal basis	485
10.2.2	Multiplication by a column-orthogonal matrix preserves norm	485
10.3	Wavelets	487
10.3.1	One-dimensional “images” of different resolutions	487
10.3.2	Decomposing \mathcal{V}_n as a direct sum	489
10.3.3	The wavelet bases	490
10.3.4	The basis for \mathcal{V}_1	491
10.3.5	General n	492
10.3.6	The first stage of wavelet transformation	492
10.3.7	The subsequent levels of wavelet decomposition	493
10.3.8	Normalizing	496
10.3.9	The backward transform	496
10.3.10	Implementation	496
10.4	Polynomial evaluation and interpolation	496
10.5	Fourier transform	499
10.6	Discrete Fourier transform	501
10.6.1	The Laws of Exponentiation	501
10.6.2	The n stopwatches	501
10.6.3	Discrete Fourier space: Sampling the basis functions	503
10.6.4	The inverse of the Fourier matrix	504
10.6.5	The Fast Fourier Transform Algorithm	506
10.6.6	Deriving the FFT	507
10.6.7	Coding the FFT	509
10.7	The inner product for the field of complex numbers	509
10.8	Circulant matrices	512
10.8.1	Multiplying a circulant matrix by a column of the Fourier matrix	513

10.8.2	Circulant matrices and change of basis	515
10.9	<i>Lab: Using wavelets for compression</i>	515
10.9.1	<i>Unnormalized forward transform</i>	517
10.9.2	<i>Normalization in the forward transform</i>	518
10.9.3	<i>Compression by suppression</i>	519
10.9.4	<i>Unnormalizing</i>	519
10.9.5	<i>Unnormalized backward transform</i>	519
10.9.6	<i>Backward transform</i>	520
10.9.7	<i>Auxiliary procedure</i>	520
10.9.8	<i>Two-dimensional wavelet transform</i>	521
10.9.9	<i>Forward two-dimensional transform</i>	522
10.9.10	<i>More auxiliary procedures</i>	524
10.9.11	<i>Two-dimensional backward transform</i>	524
10.9.12	<i>Experimenting with compression of images</i>	525
10.10	Review Questions	526
10.11	Problems	527
11	The Singular Value Decomposition	529
11.1	Approximation of a matrix by a low-rank matrix	529
11.1.1	The benefits of low-rank matrices	529
11.1.2	Matrix norm	530
11.2	The <i>trolley-line-location</i> problem	531
11.2.1	Solution to the trolley-line-location problem	532
11.2.2	Rank-one approximation to a matrix	536
11.2.3	The best rank-one approximation	536
11.2.4	An expression for the best rank-one approximation	537
11.2.5	The closest one-dimensional affine space	539
11.3	Closest dimension- k vector space	539
11.3.1	A <i>Gedanken</i> algorithm to find the singular values and vectors	540
11.3.2	Properties of the singular values and right singular vectors	541
11.3.3	The singular value decomposition	542
11.3.4	Using right singular vectors to find the closest k -dimensional space	544
11.3.5	Best rank- k approximation to A	547
11.3.6	Matrix form for best rank- k approximation	548
11.3.7	Number of nonzero singular values is rank A	548
11.3.8	Numerical rank	549
11.3.9	Closest k -dimensional affine space	549
11.3.10	Proof that U is column-orthogonal	550
11.4	Using the singular value decomposition	551
11.4.1	Using SVD to do least squares	552
11.5	PCA	552
11.6	<i>Lab: Eigenfaces</i>	553
11.7	Review questions	556
11.8	Problems	556

12 The Eigenvector	560
12.1 Modeling discrete dynamic processes	560
12.1.1 Two interest-bearing accounts	560
12.1.2 Fibonacci numbers	561
12.2 Diagonalization of the Fibonacci matrix	564
12.3 Eigenvalues and eigenvectors	565
12.3.1 Similarity and diagonalizability	567
12.4 Coordinate representation in terms of eigenvectors	569
12.5 The Internet worm	570
12.6 Existence of eigenvalues	572
12.6.1 Positive-definite and positive-semidefinite matrices	572
12.6.2 Matrices with distinct eigenvalues	573
12.6.3 Symmetric matrices	574
12.6.4 Upper-triangular matrices	575
12.6.5 General square matrices	576
12.7 Power method	577
12.8 Markov chains	578
12.8.1 Modeling population movement	578
12.8.2 Modeling Randy	579
12.8.3 Markov chain definitions	580
12.8.4 Modeling spatial locality in memory fetches	580
12.8.5 Modeling documents: Hamlet in Wonderland	581
12.8.6 Modeling lots of other stuff	582
12.8.7 Stationary distributions of Markov chains	583
12.8.8 Sufficient condition for existence of a stationary distribution	583
12.9 Modeling a web surfer: PageRank	583
12.10*The determinant	584
12.10.1 Areas of parallelograms	584
12.10.2 Volumes of parallelepipeds	586
12.10.3 Expressing the area of a polygon in terms of areas of parallelograms	587
12.10.4 The determinant	590
12.10.5*Characterizing eigenvalues via the determinant function	592
12.11*Proofs of some eigentheorems	593
12.11.1 Existence of eigenvalues	593
12.11.2 Diagonalization of symmetric matrices	594
12.11.3 Triangularization	596
12.12 <i>Lab: Pagerank</i>	598
12.12.1 <i>Concepts</i>	598
12.12.2 <i>Working with a Big Dataset</i>	601
12.12.3 <i>Implementing PageRank Using The Power Method</i>	602
12.12.4 <i>The Dataset</i>	604
12.12.5 <i>Handling queries</i>	604
12.12.6 <i>Biasing the pagerank</i>	605
12.12.7 <i>Optional: Handling multiword queries</i>	606
12.13 Review questions	606

12.14	Problems	607
13	The Linear Program	612
13.1	The diet problem	612
13.2	Formulating the diet problem as a linear program	613
13.3	The origins of linear programming	614
13.3.1	Terminology	615
13.3.2	Linear programming in different forms	615
13.3.3	Integer linear programming	616
13.4	Geometry of linear programming: polyhedra and vertices	616
13.5	There is an optimal solution that is a vertex of the polyhedron	620
13.6	An enumerative algorithm for linear programming	620
13.7	Introduction to linear-programming duality	621
13.8	The simplex algorithm	623
13.8.1	Termination	624
13.8.2	Representing the current solution	624
13.8.3	A pivot step	625
13.8.4	Simple example	627
13.9	Finding a vertex	630
13.10	Game theory	632
13.11	Formulation as a linear program	635
13.12	Nonzero-sum games	636
13.13	<i>Lab: Learning through linear programming</i>	636
13.13.1	<i>Reading in the training data</i>	637
13.13.2	<i>Setting up the linear program</i>	638
13.13.3	<i>Main constraints</i>	639
13.13.4	<i>Nonnegativity constraints</i>	640
13.13.5	<i>The matrix A</i>	640
13.13.6	<i>The right-hand side vector \mathbf{b}</i>	640
13.13.7	<i>The objective function vector \mathbf{c}</i>	641
13.13.8	<i>Putting it together</i>	641
13.13.9	<i>Finding a vertex</i>	641
13.13.10	<i>Solving the linear program</i>	641
13.13.11	<i>Using the result</i>	641
13.14	Compressed sensing	642
13.14.1	More quickly acquiring an MRI image	642
13.14.2	Computation to the rescue	643
13.14.3	Onwards	644
13.15	Review questions	644
13.16	Problems	644

Introduction

Tourist on Fifty-Seventh Street,
Manhattan: “Pardon me—could you tell
me how to get to Carnegie Hall?”

Native New Yorker: “Practice, practice!”

There’s a scene in the movie *The Matrix* in which Neo is strapped in a chair and Morpheus inserts into a machine what looks like a seventies-era videotape cartridge. As the tape plays, knowledge of how to fight streams into Neo’s brain. After a very short time, he has become an expert.

I would be delighted if I could strap my students into chairs and quickly stream knowledge of linear algebra into their brain, but brains don’t learn that way. The input device is rarely the bottleneck. Students need lots of practice—but what kind of practice?

No doubt students need to practice the basic numerical calculations, such as matrix-matrix multiplication, that underlie elementary linear algebra and that seem to fill all their time in traditional cookbook-style courses on linear algebra. No doubt students need to find proofs and counterexamples to exercise their understanding of the abstract concepts of which linear algebra is constructed.

However, they also need practice in using linear algebra to think about problems in other domains, and in actually *using* linear-algebra computations to address these problems. These are the skills they most need from a linear-algebra class when they go on to study other topics such as graphics and machine learning. This book is aimed at students of computer science; such students are best served by seeing applications from their field because these are the applications that will be most meaningful for them.

Moreover, a linear-algebra instructor whose pupils are students of computer science has a special advantage: her students are computationally sophisticated. They have a learning modality that most students don’t—they can learn through reading, writing, debugging, and using computer programs.

For example, there are several ways of writing a program for matrix-vector or matrix-matrix multiplication, each providing its own kernel of insight into the meaning of the operation—and the experience of writing such programs is more effective in conveying this meaning and cementing the relationships between the operations than spending the equivalent time carrying out hand calculations.

Computational sophistication also helps students in the more abstract, mathematical aspects

of linear algebra. Acquaintance with object-oriented programming helps a student grasp the notion of a *field*—a set of values together with operations on them. Acquaintance with subtyping prepares a student to understand that some vector spaces are inner product spaces. Familiarity with loops or recursion helps a student understand procedural proofs, e.g. of the existence of a basis or an orthogonal basis.

Computational thinking is the term suggested by Jeannette Wing, former head of the National Science Foundation’s directorate on Computer and Information Science and Engineering, to refer to the skills and concepts that a student of computer science can bring to bear. For this book, computational thinking is the road to mastering elementary linear algebra.

Companion website

The companion website is at codingthematrix.com. There you will find, in digital form, the data, examples, and support code you need to solve the problems given in the book.

Intended audience

This book is accessible to a student who is an experienced programmer. Most students who take my course have had at least two semesters of introductory computer science, or have previously learned programming on their own. In addition, it is desirable that the student has some exposure (in prior semesters or concurrently) in proof techniques such as are studied in a Discrete Math course.

The student’s prior programming experience can be in pretty much any programming language; this book uses Python, and the first two labs are devoted to bringing the student up to speed in Python programming. Moreover, the programs we write in this book are not particularly sophisticated. For example, we provide stencil code that obviates the need for the student to have studied object-oriented programming.

Some sections of the text, marked with *, provide supplementary mathematical material but are not crucial for the reader’s understanding.

Labs

An important part of the book is the labs. For each chapter, there is a lab assignment in which the student is expected to write small programs and use some modules we provide, generally to carry out a task or series of tasks related to an application of the concepts recently covered or about to be covered. Doing the labs “keeps it real”, grounding the student’s study of linear algebra in getting something done, something meaningful in its own right but also illustrative of the concepts.

In my course, there is a lab section each week, a two-hour period in which the students carry out the lab assignment. Course staff are available during this period, not to supervise but to assist when necessary. The goal is to help the students move through the lab assignment efficiently, and to get them unstuck when they encounter obstacles. The students are expected to have prepared by reviewing the previous week’s course material and reading through the lab assignment.

Most students experience the labs as the most fun part of the course—it is where they discover the power of the knowledge they are acquiring to help them accomplish something that

has meaning in the world of computer science.

Programming language

The book uses Python, not a programming language with built-in support for vectors and matrices. This gives students the opportunity to build vectors and matrices out of the data structures that Python does provide. Using their own implementations of vector and matrix provides transparency. Python does provide complex numbers, sets, lists (sequences), and dictionaries (which we use for representing functions). In addition, Python provides *comprehensions*, expressions that create sets, lists, or dictionaries using a simple and powerful syntax that resembles the mathematical notation for defining sets. Using this syntax, many of the procedures we write require only a single line of code.

Students are not expected to know Python at the beginning; the first two labs form an introduction to Python, and the examples throughout the text reinforce the ideas.

Vector and Matrix representations

The traditional concrete representation for a vector is as a sequence of field elements. This book uses that representation but also uses another, especially in Python programs: a vector as a function mapping a finite set D to a field. Similarly, the traditional representation for a matrix is as a two-dimensional array or grid of field elements. We use this representation but also use another: a matrix as a function from the Cartesian product $R \times C$ of two finite sets to a field.

These more general representations allow the vectors and matrices to be more directly connected to the application. For example, it is traditional in information retrieval to represent a document as a vector in which, for each word, the vector specifies the number of occurrences of the word in the document. In this book, we define such a vector as a function from the domain D of English words to the set of real numbers. Another example: when representing, say, a 1024×768 black-and-white image as a vector, we define the vector as a function from the domain $D = \{1, \dots, 1024\} \times \{1, \dots, 768\}$ to the real numbers. The function specifies, for each pixel (i, j) , the image intensity of that pixel.

From the programmer's perspective, it is certainly more convenient to directly index vectors by strings (in the case of words) or tuples (in the case of pixels). However, a more important advantage is this: having to choose a domain D for vectors gets us thinking about the application from the vector perspective.

Another advantage is analogous to that of type-checking in programs or unit-checking in physical calculations. For an $R \times C$ matrix A , the matrix-vector product $A\mathbf{x}$ is only legal if \mathbf{x} is a C -vector; the matrix-matrix product AB is only legal if C is the set of row-labels of B . These constraints further reinforce the meanings of the operations.

Finally, allowing arbitrary finite sets (not just sequences of consecutive integers) to label the elements helps make it clear that the order of elements in a vector or matrix is not always (or even often) significant.

Fundamental Questions

The book is driven not just by applications but also by fundamental questions and computational problems that arise in studying these applications. Here are some of the fundamental questions:

- How can we tell whether a solution to a linear system is unique?
- How can we find the number of solutions to a linear system over $GF(2)$?
- How can we tell if a set \mathcal{V} of vectors is equal to the span of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$?
- For a system of linear equations, what other linear equations are implied?
- How can we tell if a matrix is invertible?
- Can every vector space be represented as the solution set of a homogeneous linear system?

Fundamental Computational Problems

There are a few computational problems that are central to linear algebra. In the book, these arise in a variety of forms as we examine various applications, and we explore the connections between them. Here are examples:

- Find the solution to a matrix equation $M\mathbf{x} = \mathbf{b}$.
- Find the vector \mathbf{x} minimizing the distance between $M\mathbf{x}$ and \mathbf{b} .
- Given vector \mathbf{b} , find the closest vector to \mathbf{b} whose representation in a given basis is k -sparse.
- Find the solution to a matrix inequality $M\mathbf{x} \leq \mathbf{b}$.
- Given a matrix M , find the closest matrix to M whose rank is at most k .

Multiple representations

The most important theme of this book is the idea of multiple different representations for the same object. This theme should be familiar to computer scientists. In linear algebra, it arises again and again:

- Representing a vector space by generators or by homogeneous linear equations.
- Different bases for the same vector space.
- Different data structures used to represent a vector or a matrix.
- Different decompositions of a matrix.

Multiple fields

In order to illustrate the generality of the ideas of linear algebra and in order to address a broader range of applications, the book deals with three different fields: the real numbers, the complex numbers, and the finite field $GF(2)$. Most examples are over the real numbers because they are most familiar to the reader. The complex numbers serve as a warm-up for vectors since they can be used to represent points in the plane and transformations on these points. The complex numbers also come up in the discussion of the finite Fourier transform and in eigenvalues. The finite field $GF(2)$ comes up in many applications involving information, such as encryption, authentication, checksums, network coding, secret-sharing, and error-correcting codes.

The multiple codes help to illustrate the idea of an inner-product space. There is a very simple inner product for vectors over the reals; there is a slightly more complicated inner product for vectors over the complex numbers; and there is no inner-product for vectors over a finite field.

Acknowledgements

Thanks to my Brown University colleague John F. Hughes, computer scientist and recovering mathematician. I have learned a great deal from our conversations, and this book owes much to him.

Thanks to my Brown University colleague Dan Abramovich, mathematician, who has shared his insights on the tradeoffs between abstraction and simplicity of presentation.

Thanks to the students who have served as teaching assistants for the Brown University course on which this book is based and who have helped prepare some of the problems and labs, especially Sarah Meikeljohn, Shay Mozes, Olga Ohrimenko, Matthew Malin, Alexandra Berke, Anson Rosenthal, and Eli Fox-Epstein.

Thanks to Rosemary Simpson for her work on the index for this book.

Thanks to the creator of xkcd, Randall Munroe, for giving permission to include some of his work in this book.

Thanks, finally, to my family for their support and understanding.

Chapter 0

The Function (and other mathematical and computational preliminaries)

Later generations will regard
Mengenlehre [set theory] as a disease
from which one has recovered.
attributed to Poincaré

The basic mathematical concepts that inform our study of vectors and matrices are sets, sequences (lists), functions, and probability theory.

This chapter also includes an introduction to Python, the programming language we use to (i) model the mathematical objects of interest, (ii) write computational procedures, and (iii) carry out data analyses.

0.1 Set terminology and notation

The reader is likely to be familiar with the idea of a *set*, a collection of mathematical objects in which each object is considered to occur at most once. The objects belonging to a set are its *elements*. We use curly braces to indicate a set specified by explicitly enumerating its elements. For example, $\{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$ is the set of suits in a traditional deck of cards. The order in which elements are listed is not significant; a set imposes no order among its elements.

The symbol \in is used to indicate that an object belongs to a set (equivalently, that the set *contains* the object). For example, $\heartsuit \in \{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$.

One set S_1 is *contained in* another set S_2 (written $S_1 \subseteq S_2$) if every element of S_1 belongs to S_2 . Two sets are equal if they contain exactly the same elements. A convenient way to prove that two sets are equal consists of two steps: (1) prove the first set is contained in the second, and (2) prove the second is contained in the first.

A set can be infinite. In Chapter 1, we discuss the set \mathbb{R} , which consists of all real numbers, and the set \mathbb{C} , which consists of all complex numbers.

If a set S is not infinite, we use $|S|$ to denote its *cardinality*, the number of elements it contains. For example, the set of suits has cardinality 4.

0.2 Cartesian product

One from column A, one from column B.

The *Cartesian product* of two sets A and B is the set of all pairs (a, b) where $a \in A$ and $b \in B$.

Example 0.2.1: For the sets $A = \{1, 2, 3\}$ and $B = \{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$, the Cartesian product is

$$\{(1, \heartsuit), (2, \heartsuit), (3, \heartsuit), (1, \spadesuit), (2, \spadesuit), (3, \spadesuit), (1, \clubsuit), (2, \clubsuit), (3, \clubsuit), (1, \diamondsuit), (2, \diamondsuit), (3, \diamondsuit)\}$$

Quiz 0.2.2: What is the cardinality of $A \times B$ in Example 0.2.1 (Page 2)?

Answer

$$|A \times B| = 12.$$

Proposition 0.2.3: For finite sets A and B , $|A \times B| = |A| \cdot |B|$.

Quiz 0.2.4: What is the cardinality of $\{1, 2, 3, \dots, 10, J, Q, K\} \times \{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$?

Answer

We use Proposition 0.2.3. The cardinality of the first set is 13, and the cardinality of the second set is 4, so the cardinality of the Cartesian product is $13 \cdot 4$, which is 52.

The Cartesian product is named for René Descartes, whom we shall discuss in Chapter 6.

0.3 The function

Mathematicians never die—they just lose function.

Loosely speaking, a function is a rule that, for each element in some set D of possible inputs, assigns a possible output. The output is said to be the *image* of the input under the function

and the input is a *pre-image* of the output. The set D of possible inputs is called the *domain* of the function.

Formally, a *function* is a (possibly infinite) set of pairs (a, b) no two of which share the same first entry.

Example 0.3.1: The doubling function with domain $\{1, 2, 3, \dots\}$ is

$$\{(1, 2), (2, 4), (3, 6), (4, 8), \dots\}$$

The domain can itself consist of pairs of numbers.

Example 0.3.2: The multiplication function with domain $\{1, 2, 3, \dots\} \times \{1, 2, 3, \dots\}$ looks something like this:

$$\{((1, 1), 1), ((1, 2), 2), \dots, ((2, 1), 2), ((2, 2), 4), ((2, 3), 6), \dots\}$$

For a function named f , the image of q under f is denoted by $f(q)$. If $r = f(q)$, we say that q maps to r under f . The notation for “ q maps to r ” is $q \mapsto r$. (This notation omits specifying the function; it is useful when there is no ambiguity about which function is intended.)

It is convenient when specifying a function to specify a *co-domain* for the function. The co-domain is a set from which the function’s output values are chosen. Note that one has some leeway in choosing the co-domain since not all of its members need be outputs.

The notation

$$f : D \longrightarrow F$$

means that f is a function whose domain is the set D and whose *co-domain* (the set of possible outputs) is the set F . (More briefly: “a function from D to F ”, or “a function that maps D to F .”)

Example 0.3.3: Caesar was said to have used a cryptosystem in which each letter was replaced with the one three steps forward in the alphabet (wrapping around for X, Y, and Z).^a Thus the plaintext MATRIX would be encrypted as the cyphertext PDWULA. The function that maps each plaintext letter to its cyphertext replacement could be written as

$$A \mapsto D, B \mapsto E, C \mapsto F, D \mapsto G, W \mapsto Z, X \mapsto A, Y \mapsto B, Z \mapsto C$$

This function’s domain and co-domain are both the alphabet $\{A, B, \dots, Z\}$.

^aSome imaginary historians have conjectured that Caesar’s assassination can be attributed to his use of such a weak cryptosystem.

Example 0.3.4: The cosine function, \cos , maps from the set of real numbers (indicated by \mathbb{R})

- [download Bad Seeds: The True Story of Toronto's Galloway Boys Street Gang online](#)
- [read The Hunter's Oath \(James Bishop, Book 3\)](#)
- [download online Israel: the First Hundred Years, Volume 3: Israeli Politics and Society since 1948](#)
- [The Gate book](#)
- [click A History of 1970s Experimental Film: Britain's Decade of Diversity](#)
- [download online Easy English Step-By-Step for ESL Learners pdf, azw \(kindle\), epub](#)

- <http://www.celebritychat.in/?ebooks/Bad-Seeds--The-True-Story-of-Toronto-s-Galloway-Boys-Street-Gang.pdf>
- <http://transtrade.cz/?ebooks/Introduction-to-Business-Statistic--6th-Edition-.pdf>
- <http://fortune-touko.com/library/Autograf--New-York-City-s-Graffiti-Writers.pdf>
- <http://transtrade.cz/?ebooks/The-Gate.pdf>
- <http://cavalldecartro.highlandagency.es/library/A-History-of-1970s-Experimental-Film--Britain-s-Decade-of-Diversity.pdf>
- <http://www.1973vision.com/?library/Not-for-Tourists-Guide-to-Philadelphia--8th-Edition-.pdf>