

Table of Contents

Table of Contents	1
Home	10
Software	10
Links	10
Index	11
Contact	14
Acknowledgements	14
1. Introduction	14
1.1 Overview	14
1.1.1 About this livebook	14
1.1.2 Outline	15
1.2 Optimization Models	15
1.2.1 Gauss' bet	15
1.2.2 Functions and maps	16
1.2.3 Standard forms	17
1.2.4 Nomenclature	17
1.2.5 Hard vs. easy problems	18
1.2.6 Problem classes	18
1.2.7 Historical background	20
1.2.8 Exercises	20
2. Linear Algebra	21
2.1 Vectors	21
2.1.1 Basics	21
2.1.2 Scalar product, norms and angles	23
2.1.3 Projection on a line	24
2.1.4 Orthogonalization	25
2.1.5 Hyperplanes and half-spaces	26
2.1.6 Linear functions	27
2.1.7 Application in data visualization	28
2.1.8 Exercises	30
2.2 Matrices	31
2.2.1 Basics	31
2.2.2 Matrix products	32
2.2.3 Special classes of matrices	34
2.2.4 QR decomposition	35
2.2.5 Matrix inverses	37
2.2.6 Linear maps	38
2.2.7 Matrix norms	38
2.2.8 Applications	40
2.2.9 Exercises	40
2.3 Linear Equations	40
2.3.1 Motivating example	41
2.3.2 Existence and unicity of solutions	42
2.3.3 Solving linear equations	43
2.3.4 Applications	44
Temperature distribution	44
Trilateration	44
Estimation of traffic flows	45
2.3.5 Exercises	45
2.4 Least-Squares	45
2.4.1 Ordinary least-squares	46
2.4.2 Variants	47
2.4.3 Kernels for least-squares	48
2.4.4 Applications	49
Linear regression	49
Times series prediction	50

2.4.5 Exercises	50
2.5 Eigenvalues	52
2.5.1 Quadratic functions and symmetric matrices	52
2.5.2 Spectral theorem	53
2.5.3 Positive semi-definite matrices	54
2.5.4 Principal component analysis	55
2.5.5 Applications	57
2.5.6 Exercises	57
2.6 Singular Values	58
2.6.1 The SVD theorem	59
2.6.2 Matrix properties via SVD	59
2.6.3 Solving linear systems via SVD	61
2.6.4 Least-squares and SVD	61
2.6.5 Low-rank approximations	62
2.6.6 Applications	63
2.6.7 Exercises	63
3. Convex Models	64
3.1 Convex Optimization	65
3.3.1 Convex sets	65
3.3.2 Convex functions	66
3.3.3 Convex problems	67
3.3.4 Algorithms	68
3.3.5 Disciplined convex programming	70
3.3.6 Exercises	71
3.2 LP and QP	72
3.2.1 Polyhedra	72
3.2.2 Standard forms	73
3.2.3 Minimizing polyhedral functions	74
3.2.4 Cardinality minimization	76
3.2.5 Applications	77
Binary classification	77
3.2.6 Exercises	77
3.3 SOCP	79
3.3.1. The second-order cone	79
3.3.2 Standard forms	80
3.3.3 Group sparsity	80
3.3.4 Applications	81
Minimum surface area	81
Image restoration	82
Antenna array design	83
Image annotation	83
3.3.5 Exercises	83
3.4 Robust LP	84
3.4.1 Tractable cases	85
3.4.2 Chance programming	87
3.4.3 Affine recourse	87
3.4.4 Applications	88
Robust inventory control	88
Robust antenna array design	88
Robust supervised learning	88
Affine recourse in cash-flow management	90
3.4.5 Exercises	90
3.5 GP	90
3.5.1 Posynomials	90
3.5.2 Standard forms	92
3.5.3 Applications	93
3.5.4 Exercises	93
3.6 SDP	93
3.6.1 From LP to conic problems	93
3.6.2 Linear Matrix Inequalities	94
3.6.3 Standard forms	95
3.6.4 Applications	96

3.6.5 Exercises	96
3.7 Non-Convex Models	96
3.7.1 Overview	97
3.7.2 Coordinate descent	97
3.7.3 Linearization	97
3.7.4 Approximation via convexity	97
3.7.5 Discrete optimization	97
3.7.6 Applications	97
3.7.7 Exercises	97
4. Duality	97
4.1 Weak Duality	97
4.1.2 Geometric interpretation	98
4.1.3 Minimax inequality	99
4.2 Strong Duality	100
4.2.1 Slater condition	101
4.2.2 Optimality	101
4.2.3 Sensitivity	102
4.3.4 Applications	102
4.3.5 Exercises	103
5. Case Studies	103
5.1 Senate Voting	103
5.1.1 Senate voting data	104
5.1.2 Projections	104
5.1.3 PCA	106
5.1.4 Sparse PCA	108
5.2 Antenna Arrays	108
5.2.1 Physics of antenna arrays	108
5.2.2 Diagram shaping	108
5.2.3 Least-Squares design	109
5.2.4 SOCP design	110
5.3 Digital Circuit Design	112
5.3.1 The problem	112
5.3.2 GP model	113
5.3.3 Example	113
5.3.4 Notes and references	114
5.4 Localization	114
5.4.1 The problem	114
5.4.2 Consistency	114
5.4.3 Inner approximations	116
5.4.4 Outer approximations	116
5.5 Asset Management	119
5.5.1 The problem	119
5.5.2 LP formulation	120
5.5.3 Robustness	121
5.5.4 Affine recourse	122
Appendix	123
Examples	123
Temperatures at different airports	123
Bag-of-words representation of text	124
Document similarity	124
A basis in R ³	124
Dimension of an affine subspace	124
Beer-Lambert Law in Absorption Spectrometry	125
Absorption spectrometry: using measurements	125
QR decomposition: example	126
Two orthogonal vectors	126
An hyperplane in 3D	126
Power law model fitting	127
Gradient of a linear function	127
Linearization of a non-linear function	128

Network flow	128
Image Compression	128
A 2x2 orthogonal matrix	128
Incidence matrix of a network	129
Navigation by range measurement	129
Singular value decomposition of a 4x5 matrix	129
SVD: a 4x4 example	130
A two-dimensional toy optimization problem	130
A toy 2D optimization problem: geometry	130
Nomenclature of a toy 2D optimization problem	131
Eigenvalue decomposition of a 2x2 symmetric matrix	131
Quadratic functions in two variables	132
range of a 4x5 matrix via its SVD	132
Protein folding	132
Crew allocation in airline operations	133
Theorems and proofs	133
Cauchy-Schwartz Inequality	133
Dimension of hyperplanes	133
Spectral theorem	134
Rayleigh Quotients Theorem	134
Optimal set of LS	135
Definitions	135
Sample and weighted mean, expected value	135
Sample variance and standard deviation	136
Gram matrix	136
Rate of return of a financial portfolio	136
Solving triangular systems of equations: backwards substitu	137
Linear and Affine Maps	137
Vector norms	137
Dual Norm	138
Lines	138
Euclidean projection on a set	138
Log-Sum-Exp (LSE) function	139
Power laws	139
Linear and Affine Maps	139
Gradient of a function	140
Hessian of a function	141
Sample covariance matrix	142
Projection of a vector on a line	142
State-space models of linear dynamical systems	143
Functions and maps	144
Determinant of a square matrix	145
Permutation matrices	146
Global vs local minima	146
Backwards substitution for solving triangular linear system	146
Laplacian matrix of a graph	147
Edge weight matrix of a graph	147
Sample average of vectors	147
Homeworks	147
Homework 1	147
Homework 2	147
Standard Forms of SOCP	150
SOCP > Second-order cone Standard f	150
Group sparsity and the -norm trick	150
SOCP > Second-order cone	150
Applications of SOCP	151
SOCP > Second-order cone Standard for	151
Exercises	151
SOCP > Exercises	151
Second-order cones	151
A. A single seco	151
Motivations and Standard Forms	153
Robust LP > Motivations and s	153
Exercises	153
Robust LP > Exercises	153
Posynomials	154
GP > Posynomials Standard Forms Applications	154
Standard Forms of GPGP	155
> Posynomials Standard Forms App	155
Some Applications of GPGP	156
> Posynomials Standard Forms	156
Exercises	156
Geometric Programming > Exercises	156
From LP to Conic Optimization	156
SDP > Conic Problems LMIs Sta	156
Linear Matrix Inequalities	157
SDP > Conic problems LMIs Sta	157

Standard Forms of SDP	SDP > Conic problems LMs Standard	158
Some Applications of SDP	SDP > Conic problems LMs Stand	159
Exercises	Semi-Definite Programming > Exercises	159
A convex and a non-convex set	The intuitive idea of a convex	160
Convex and conic hull	Convex hull of a finite set of points	160
Second-order cone	Definition The set in is a convex cone, ca	161
Semidefinite cone	Positive semidefinite matrices A symmetric	162
Projection of a convex set on a subspace	The projection of a	162
Log-Determinant Function and Properties	The log-determinant	162
Convexity of quadratic functions in two variables	We return	163
Square-to-linear function		163
Negative Entropy Function and Properties	The negative entrop	164
Schur Complement Lemma	Lemma: Schur Complement Let be a symme	164
Proving convexity via monotonicity	Consider the function , w	164
Local and Global Optima in Convex Optimization	Theorem: Glob	165
Minimization of a convex quadratic function	Here we consider	165
Optimality condition for a convex, unconstrained problem	Con	166
Optimality condition for a convex, linearly constrained pro	Optimality condition for a convex, linearly constrained pro	166
A half-space in	Consider the set in defined by a single aff	167
Graph, epigraph, level and sub-level sets of a function	Graph, epigraph, level and sub-level sets of a function	167
Component-wise inequality convention for vectors	If are two	168
A polyhedron in	Consider the set in defined by a three affi	168
Probability simplex in	The probability simplex in is the po	168
A Drug Production Problem	Problem description A company produ	169
Projection on a polyhedron	Recall that a polyhedron is an in	170
A linear program in 2D	Consider the optimization problem The	170
A Quadratic Program with Two Variables	Consider the problem	171
LP and QP in conic form	The LP can always be represented in	171
LP Formulation of - and -norm Regression	The -norm regressio	172
Cardinality of a vector	The cardinality of a vector is the n	172
Linear Binary Classification	LP and QP > Applications > Clas	172
Piece-wise constant fitting	We observe a noisy time-series (175
Network Flows	Network FlowsLP and QP > Applications> Back Networks Ne	175
LP Relaxations of Boolean Problems	LP and QP > Applications	176
Mean-Variance Trade-Off in Portfolio Optimization	LP and QP	176
Filter Design	Finite Impulse Response filtersFilters. A sing	177
Magnitude constraints on affine complex vectors		179
SOCPs include LPs as Special Case	The linear program (LP)can	179
Facility Location	Consider the problem of locating a warehou	179
Robust Least-Squares	We start from the least-squares problem	180
Separation of Ellipsoids	We consider the following geometric	181
SOCPs include QPs as Special Case	The quadratic program (QP)	182
SOCPs include QCQPs as a Special Case	The quadratically cons	182
Minimum Surface Area	SOCOP > Applications > Minimum Surface A	182
Total Variation Image Restoration	SOCOP > Applications > Back	184
Sensor Network Localization	SOCOP > Applications > Back Sen	184
Uncertainty in the Drug Production Problem	Return to the dru	184
Antenna Array Design	SOCOP > SOC inequalities Standard Form	185
Implementation Errors in an Antenna Array Design Problem	Implementation Errors in an Antenna Array Design ProblemRet	185
Robust LP Solution to the Drug Production Problem	Return to	186
Scalar Product and Norms	Vectors, Matrices > Vectors Scala	186
Strong Duality for QP	Primal ProblemConsider the QP where ,	188
Strong Duality for LP	Consider the LP	189
Minimum Euclidean distance to a subspace: strong duality		189
Projection of a vector on a line	A lineThe line in passing t	190
A simple matrix	Matrices > Basics > Example Consider the mat	190
A two-dimensional toy optimization problem	As a toy example	190
Senate Voting Data Matrix	Matrices > Basics > Example The da	191
Gram matrix	Consider -vectors	192
Single factor model of financial price data	Consider a data	192
The QR decomposition of a matrix	Basic ideaCase when the mat	193
Full Rank Matrices	TheoremA matrix isfull column rank if and	194
Rank-nullity theorem	Rank-nullity theoremThe nullity (dimens	194

Orthogonal complement of a subspace	Let be a subspace of	194
Fundamental theorem of linear algebra	Fundamental theorem of	195
Image Compression via Least-Squares	We can use least-squares	195
Set of solutions to the least-squares problem	via QR decomp	195
Auto-Regressive (AR) models for time-series prediction	A pop	196
Portfolio Optimization via Linearly Constrained Least-Square	196	
Portfolio Optimization via Linearly Constrained Least-Square	197	
Absorption spectrometry: using measurements at different li	198	
Largest singular value norm of a matrix	For a matrix , we de	198
Control of a unit mass	Consider the problem if transferring	198
A squared linear function	Symmetric Matrices > Definitions >	198
Control of a unit mass	Consider the problem if transferring	199
Control of a unit mass	Consider the problem if transferring	199
A symmetric matrix	Symmetric Matrices > Definitions > Exampl	199
A symmetric matrix	Symmetric Matrices > Definitions > Exampl	199
Hessian of a quadratic function	Symmetric Matrices > Definit	199
Hessian of a quadratic function	Symmetric Matrices > Definit	200
Representation of a two-variable quadratic function	Symmetri	200
Representation of a two-variable quadratic function	Symmetri	200
Quadratic Approximation of the Log-Sum-Exp Function	Symmetri	200
A diagonal matrix and its associated quadratic form	Symmetri	201
Quadratic Approximation of the Log-Sum-Exp Function	Symmetri	201
A squared linear function	Symmetric Matrices > Definitions >	201
Theorem on positive semi-definite forms and eigenvalues	Theo	201
Laplacian matrix of a graph	Another important symmetric matr	202
Singular value decomposition (SVD)	theoremTheorem: Singular	202
Nullspace of a matrix via its SVD	Returning to this example,	202
Nullspace of a matrix via its SVD	Returning to this example,	202
Linear Equations: Definition and Main Issues	Linear Equation	203
Pseudo-inverse of a matrix via its SVD	Returning to this exa	203
Low-rank approximation of a matrix via its SVD	Returning to	204
Pseudo-Inverse of a Matrix	The pseudo-inverse of a matrix is	204
Optimal set of Least-Squares via SVD	Theorem: optimal set of	205
Applications of SVD: image compression	SVD > Applications >	205
Applications of SVD: market data analysis	SVD > Applications	207
Solution set of a linear equation	TheoremThe solution set of	208
Solution Concepts	Linear Equations > Definitions Propertie	208
Properties	Linear Equations > Definitions Properties Sol	209
Edge weight matrix of a graph	A symmetric matrix is a way to	211
Incidence matrix of a network	Mathematically speaking, a net	211
Definitions	Least-Squares > Definitions Solution Sensiti	211
Rate of return of a financial portfolio	The rate of return (211
Linear regression	A popular example of least-squares problem	211
Digital Circuit Design: Problem GP	> Standard Forms Applic	212
Applications	Matrices > Basics Matrix products Linear ma	213
Vectors and Matrices	Vectors are collections of numbers arra	213
Definitions	Least-Squares > Definitions Solution Sensiti	213
A simple Example of A PSD Matrix	SDP > LMIs > ExampleConside	213
A simple Example of A PSD Matrix	SDP > LMIs > ExampleConside	213
Noname		214
Noname (2)		214
Other Standard Forms	Up BackInequality formAs seen here, a	215
Other Standard Forms	Up BackInequality formAs seen here, a	215
Noname (3)		216
Bounding Portfolio Risk with Incomplete Covariance Informat		216
Boolean Quadratic Programming	SDP > Conic problems LMIs	217
obust Stability of Linear Dynamical Systems	SDP > Conic prob	218
Noname (4)		219
Noname (5)		220
Noname (6)		221
Noname (7)		222
Noname (8)		222
Noname (9)		223

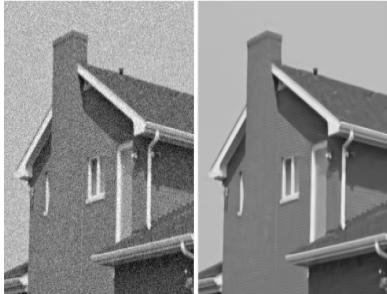
Name (10)	224
Quadratic functions in two variables	224
Two examples of quadrat	224
Name (11)	224
Name (12)	225
Name (13)	225
Name (14)	226
Name (15)	226
Auto-regressive models for time series prediction	226
TTomography	227
Trace of a matrix	227
Triangle inequality	227
Triangular m	227
NNorms: general definition, for vectors, for matrices; see	227
OOptimal point, optimal value, optimal set	227
Orthogonal: vecto	227
Vectors	227
Matrices > Vectors Scalar products Matr	227
Matrices	228
Vectors, Matrices > Vectors Scalar products Mat	228
Name (16)	230
Name (17)	231
Name (18)	232
Name (19)	233
Name (20)	234
Name (21)	235
Name (22)	236
Name (23)	237
Name (24)	238
Name (25)	239
Name (26)	240
AAbsorbtion spectrometry	241
Affine function	241
Affine setAngle betw	241
B Backward substition method for solving triangular systems	241
C CAT scan imaging	241
Cauchy-Schwartz inequality	241
Cardinality of	241
DDeterminant of a square matrix	242
Dimension of an affine set	242
Do	242
EEigenvalue decomposition (EVD) of a square, symmetric matr	242
Spectral Theorem	242
Symmetric Matrices > Definitions Spectra	242
Functions and Maps	243
Optimization Models > Gauss' Bet Functi	243
Sample and weighted mean, expected value	244
The sample mean (or	244
FFeasible point, feasible setFirst-order approximation	244
Frobe	244
GGeometric program (GP)	245
Global optimum	245
Gradient of a function	245
HHalf-space	245
Hessian of a function	245
Hyperplane	245
IImage compression	245
Incidence matrix of a graph	245
Independent se	245
KKernel matrix, kernel trick	245
LLaplacian of a graph	245
Lagragian of an optimization problem	245
MMaps	245
MatrixMatrix-vector product	245
Sample Mean	245
Minimax inequali	245
P Permutation matrix	246
Point-wise maximum of functions	246
Polyhedr	246
Name (27)	246
Name (28)	246
Name (29)	247
Costs of a Water Tank	248
GP > Posynomials > Example: Water Tank	248
Signal to Interference plus Noise Ratio (SINR) in Wireless	248
Why do we take the log in GP?For a posynomial with valueswh	249
Optimization of a Water Tank	249
GP > Posynomials Standard For	249
Optimization of a Water Tank: Convex Form of the GPGP > Pos	250
Mean and Covariance Matrix of a Random Variableif is a vect	251
Name (30)	251
Name (31)	252
Name (32)	253
Name (33)	253
Name (34)	254
Name (35)	254
Name (36)	255
Cardinality minimization: the -norm trickLP and QP> Half-Sp	256
Name (37)	257
Name (38)	258
Sum of largest components in a vectorDefinitionConvexityEff	258
Name (39)	259
Name (40)	260
Surface AreaConsider a surface in that is described by a fu	261

Name (41)	261
JJacobian matrix of a non-linear map	262
QQuadratic program (QP)Quadratic form, quadratic function	262
RRange of a matrixRank of a matrixRate of return of a finan	262
S Sample mean, sample standard deviation, sample covariance	262
UUnconstrained optimizationUnitary matrix (see also Orthogo	263
V VectorsSample Variance	263
W	263
X	263
Y	263
Z	263
OOptimal point, optimal value, optimal setOrthogonal: vecto	263
Special MatricesMatrices > Basics Matrix products Speci	263
BasicsVectors > Basics Scalar product, Norms Projection	265
Sample variance and standard deviationThe sample variance o	267
Algorithms for Convex OptimizationConvex Optimization > Con	267
DefinitionsSymmetric Matrices > Definitions Spectral theo	269
Spectral Theorem Symmetric Matrices > Definitions Spectra	270
Slater Condition for Strong DualityStrong Duality > Slater	271
Sample covariance matrixDefinitionFor a vector , the sample	272
Name (42)	273
Name (43)	273
Name (44)	273
Name (45)	274
Name (46)	274
Name (47)	275
Name (48)	276
Name (49)	277
Name (50)	278
Name (51)	279
Linearly Constrained Least-Squares ProblemsLeast-Squares >	280
Some Limitations of OLSLeast-Squares > Ordinary least-squar	281
Sensitivity AnalysisLeast-Squares > Ordinary LS Variants	281
Regularized Least-Squares ProblemLeast-Squares > Definition	282
Name (52)	282
Name (53)	283
This section under development	284
Name (54)	284
Half-SpacesLP and QP > Half-Spaces Polyhedra Standard F	285
Name (55)	286
Name (56)	286
Name (57)	287
Name (58)	288
Name (59)	289
Name (60)	290
Name (61)	291
Digital Circuit Optimization via Geometric ProgrammingS	292
A orthogonal matrixThe matrix is orthogonal.The vector is t	292
Senate Voting: Scoring SenatorsWe consider the data matrix	292
Network flowWe describe a flow (of goods, traffic, charge,	293
Senate Voting Data MatrixThe data consists of the votes of	293
A simple matrixConsider the matrix The matrix can be interp	294
Dimension of an affine subspaceThe set in defined by the li	294
Basis in The set of three vectors in : is not independent,	294
Representing temperatures at different airports as a vector	295
Sample variance and standard deviationThe sample variance o	295
Sample covariance matrixThe average of given real numbers i	295
Name (62)	296
Sample and weighted average, expected valueThe sample avera	296
Visualization of High-Dimensional DataVectors, Matrices > A	296
Robust Principal Component AnalysisSDP > Conic problems L	297
Sparse Principal Component AnalysisSDP > Conic problems L	297
Standard Forms and GeometryUp NextLinear ProgramsA linear	297

Visualization of High-Dimensional DataMatrices > Application	298
Digital Circuit Design via GPGP > Applications > Circuit Design	299
Digital Circuit Design: GP RepresentationGP > Standard Form	299
Noname (63)	300
Noname (64)	300
Noname (65)	301
Two orthogonal vectorsThe two vectors in : are orthogonal,	302
Noname (66)	302
Noname (67)	303
ExampleSOCP > SOC inequalities Standard Forms Applications	304
Physics of Antenna ArraysSOCP > SOC inequalities Standard	305
Image Annotation via Group SparsitySOCP > Applications > Ba	306
Digital Circuit Design: Notes and References GP > Application	306
Digital Circuit Design: ExampleGP > Standard Forms Applications	306
Senate Voting: Visualizing Senators on a Plane	306
Senate Voting: Scoring SenatorsMatrices > Matrix products >	307
Antenna Array DesignSOCP > Applications > Back Antenna Ar	308

This livebook offers an introduction to optimization models and their applications, with emphasis on numerically tractable problems, such as linear or constrained least-squares optimization. The livebook platform allows users to post comments and ask questions. As an author, I will not guarantee answers, but I will do my best!

This work elaborates on an earlier [hypertextbook](#), which is no longer maintained. It provides a less complete, shorter view of the recently published book "[Optimization Models](#)", by Calafiore and El Ghaoui, Cambridge University Press, 2014.



The image on the left is taken from an application of optimization to image denoising (J. Mairal, F. Bach, J. Ponce, G. Sapiro and A. Zisserman. [Non-Local Sparse Models for Image Restoration](#). International Conference on Computer Vision (ICCV), 2009).

Note: if you wish to cite this livebook, here is a BibTex entry:

```
@misc{elghaouilivebook2015,
key={ELG},
title={Optimization models and applications},
author={Laurent {El Ghaoui}},
year="2015",
note={Livebook visited July 2015},
howpublished="\url{http://livebooklabs.com/keeppies/c5a5868ce26b8125}"}
```

Software

The following is a non-exhaustive list of software packages for scientific computing and optimization.

- As a nice alternative to matlab, look at [Julia](#).
- The matlab-based software [CVX](#) allows for rapid prototyping of convex models.
- For a python-based version, see [CVXPY](#).
- A convenient matlab-based modelling language for advanced modeling and solution of convex and nonconvex optimization problems is [YALMIP](#).
- [MOSEK](#) is a highly efficient commercial software package that offers free student licences.

Links A more detailed and extensive coverage of the same topics (linear algebra and optimization) can be found in the book:

G. Calafiore and L. El Ghaoui. [Optimization Models](#). Cambridge University Press. September 2014.

The list below is far from exhaustive, and refers to material we have found useful in developing this livebook.

Linear algebra

- [Introduction to Linear Algebra](#), by G. Strang. A reference on linear algebra, with emphasis on concepts.
- [Linear Algebra and Applications](#), by D. Lay. I haven't fully checked this one, but seems to contain interesting examples.
- [Matrix Computations](#) by Golub and Van Loan. The reference on numerical linear algebra.
- [Linear Algebra Close to Earth](#) by J. Khoury. Nice web-based material, with lots of illustrative examples.
- [Linear Algebra](#), by J. Hefferon introduces the concepts and examples.
- [Course Reader for EE 263](#) contains S. Boyd's lecture notes for a graduate-level class in linear algebra.
- [A Tutorial on Principal Components Analysis](#), by L. Smith.

Optimization

- [Convex Optimization](#), by Boyd and Vandenberghe. A reference on convex optimization, at graduate level.
- [Optimization methods in finance](#), by Gerard Cornuejols and Reha Tutuncu. Covers applications of optimization in finance.
- For other optimization-related books, consult this [list](#).

Other links

- [Optimization Online](#).
- [Optimization journals](#).

Index

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#) | [Z](#)

A

- [Absorbtion spectrometry](#)
- [Affine function](#)
- [Affine set](#)
- [Angle](#) between vectors
- [Auto-regressive models](#) for time-series prediction

B

- [Backward substition](#) method for solving [triangular systems](#) of linear equations
- [Bag-of-word](#) representation of text
- [Basis](#) of a subspace
- [Beer-Lambert law](#)

C

- [CAT scan imaging](#)
- [Cauchy-Schwartz inequality](#)
- [Cardinality of a vector](#)
- [Cardinality minimization](#), see also l_1 -norm
- [Circuit design via Geometric Programming](#)
- [Combinational logic](#) in circuit design
- [Componentwise inequality](#) between vectors
- [Condition number](#) of a matrix
- [Convex function](#)
- [Convex optimization problem](#)
- [Covariance matrix](#)
- [CVX](#)

D

- [Determinant](#) of a square matrix
- [Dimension of an affine set](#)
- [Domain of a function](#)
- Duality: [weak](#), [strong](#)
- [Dual norm](#)
- [Dual problem](#)
- [Dual function](#)
- [Dual norm](#), see also [norm](#)
- [Dyad](#)

E

- [Eigenvalue](#) decomposition (EVD) of a square, symmetric matrix
- [Epigraph of a function](#)
- [Expected value](#) of a random variable

F

- [Feasible point, feasible set](#)

- [First-order approximation](#)
- [Frobenius norm of a matrix](#)
- [Function](#)
- [Fundamental theorem of linear algebra](#)

G

- [Geometric program](#) (GP)
- [Global optimum](#)
- [Gradient of a function](#)
- [Gradient methods](#), for solving convex programs
- [Gram matrix](#)
- [Gram-Schmidt \(GS\) procedure](#)
- [Graph of a function](#)

H

- [Half-space](#)
- [Hessian of a function](#)
- [Hyperplane](#)

I

- [Image compression](#)
- [Incidence matrix of a graph](#)
- [Independent set of vectors](#)
- [Interior-point methods](#), for solving convex programs
- [Inverse](#) of a matrix

J

- [Jacobian matrix](#) of a non-linear map

K

- [Kernel matrix](#), kernel trick

L

- [Laplacian of a graph](#)
- [Lagragian](#) of an optimization problem
- [Laplace formula](#) for the determinant of a matrix
- [Largest singular value](#) (LSV) norm of a matrix
- [Least-squares](#)
- [Leibnitz formula](#) for the determinant of a matrix
- [Left inverse](#) of a matrix
- [Linear function](#)
- [Linear matrix inequality](#)
- [Linear program](#) (LP)
- [Linear regression](#)
- [Local optimum](#)
- [Log-return](#) of a financial asset
- [Log-sum-exp function](#)
- [\$l_p\$ norms](#) of a vector: l_1 norm, l_2 -norm (also called Euclidean norm), l_∞ norm.
- [\$l_0\$ norm](#), or cardinality, of a vector.

M

- [Maps](#)
- [Matrix](#)
- [Matrix-vector product](#)

- Sample [Mean](#)
- [Minimax inequality](#)

N

- Norms: [general definition](#), for [vectors](#), for [matrices](#); see also dual norm
- [Nullspace](#) of a matrix

O

- [Optimal point, optimal value, optimal set](#)
- Orthogonal: [vectors](#), [matrices](#)

P

- [Permutation matrix](#)
- [Point-wise maximum](#) of functions
- [Polyhedron](#)
- [Polyhedral function](#)
- [Positive-definite](#)
- [Power laws](#)
- [Principal Component Analysis](#) (PCA)
- [Probability simplex](#)
- Projection: on a [line](#)
- [Pseudo-inverse of a matrix](#)

Q

- [Quadratic program](#) (QP)
- [Quadratic form, quadratic function](#)

R

- [Range of a matrix](#)
- [Rank of a matrix](#)
- [Rate of return](#) of a financial asset
- [Rayleigh quotient](#)
- Regression: [Linear](#)
- [Right inverse](#) of a matrix
- [Robust linear program](#)

S

- [Sample mean, sample standard deviation, sample covariance matrix](#)
- Scalar product: for [vectors](#), for [matrices](#)
- [Second-order approximation of a function](#)
- [Second-order cone](#)
- [Second-order cone program](#) (SOCP)
- [Semidefinite program](#) (SDP)
- [Singular value](#) of a general matrix
- [Singular value decomposition](#) (SVD)
- [Slater's condition for strong duality](#)
- [Spectral theorem](#), or symmetric eigenvalue decomposition (SED) theorem
- [Symmetric matrix](#)

T

- [Tomography](#)
- [Trace of a matrix](#)
- [Triangle inequality](#)
- [Triangular matrices](#)

- [Triangular systems of linear equations](#), see also [backward substitution](#) algorithm

U

- [Unconstrained optimization](#)
- [Unitary](#) matrix (see also [Orthogonal](#) matrix)

V

- [Vectors](#)
- Sample [Variance](#)

WXYZ

Contact

For comments or suggestions on this livebook, send email to [L_El_Ghaoui \(elghaoui@livebooklabs.com\)](mailto:L_El_Ghaoui@livebooklabs.com). You can also post comments or ask questions using the right-side panel.

Acknowledgements

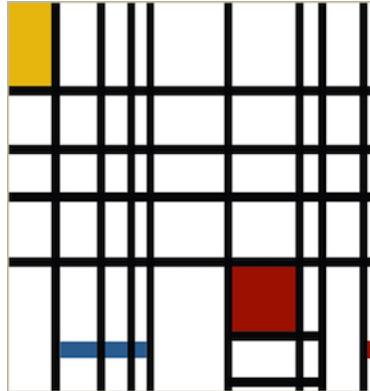
I would like to acknowledge the help of the following people for creating this livebook.

- Stephen Boyd for making available his extensive material, lecture notes, source code, and general guidance.
- Lieven Vandenberghe was also instrumental by providing code, advice and material on convex optimization.
- Eduardo Lopez for help, comments and guidance.
- The developers of the livebooklabs platform, Alexey Goder and Alex Kurzhanskiy, and their team.
- The students enrolled in EE 127A Spring 2009, and the TA at that time, John Duchi, for their help, comments, and patience during the first version of the course. Thanks for the help and comments from the TA for the Spring 2011 installment, Aude Hofleitner.
- This work is supported in part by a grant from [The MathWorks, Inc.](#).

1. Introduction

- 1.1 [Overview](#)
- 1.2 [Optimization models](#)

Overview



- [About this livebook](#)
- [Outline](#)

About this livebook Course focus

This livebook focuses on tractable optimization problems, that is, problems that can be reliably and efficiently solved on a computer. Specifically we study *convex optimization*, with emphasis on problems involving convex quadratic constraints and objectives.

Philosophy and main message

We follow the philosophy that has guided the development of linear algebra. While nearly every practical engineering problem is non-linear, the impact of linear algebra algorithms on engineering has been immense so far, and continues to be. The reason is that there is a strong interplay, in engineering modeling, between what we'd like to do (model systems as accurately as possible) and what we can do (analyze or design simple models). Thus, while non-linear, complex models may be attractive, when concerned with practical analysis or design problems, engineers often rely on simpler (linear) models to perform computations, or find approximate solutions.

Linear algebra has provided a nice trade-off between computer tractability and model accuracy. We view *convex optimization* as an extension of linear algebra, as it holds the promise of providing as reliable and efficient solutions, with much wider application range.

Course goals

Our goal is to help develop a sense of what convex optimization is, and how it can be used in practical contexts. We would like to empower students to use new convex optimization software such as CVX, which allow to quickly prototype convex optimization models.

We review some theory, including weak duality, which allows to use convex optimization to develop approximations to hard, non-convex problems.

Optimization models by nature try to fit the solution to the available data. In general, the solution might exhibit an extreme sensitivity to changes in the problem data. This appears to make any attempt at optimization a dangerous proposition. One of main messages of this course is that there exist remedies to this sensitivity issue, from simple to sophisticated, which allow to compute very robust solutions at moderate drop in performance.

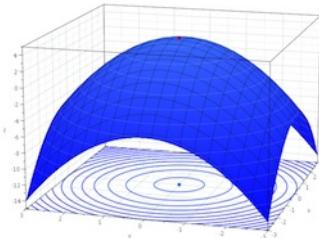
This course does not provide details on algorithms, although we briefly discuss important classes of algorithms along our way. Students are encouraged to use the wonderful modeling software [CVX](#) to get used to convex optimization.

Outline

The lectures are divided in five parts.

- [**Part 1**](#): In this *introduction*, we discuss optimization models and some basic definitions.
- [**Part 2**](#) is entirely devoted to *linear algebra*, from basic concepts such as vectors and matrices to more advanced ideas such as singular value decomposition and principal component analysis. Least-squares and some variants is included in this part, as it can be solved with the traditional tools of linear algebra.
- [**Part 3**](#): We are then equipped to introduce *convex optimization problems*. We describe briefly what convex optimization is, and how convex problems are solved in practice. We emphasize the practical importance of the notion of “disciplined convex programming” on which convex modeling software such as CVX rests. We then review a number of “standard” convex models, and applications. One topic in this part explores difficulties and some solution approaches for optimization problems with uncertain data.
- [**Part 4**](#) is a gentle introduction to the key notion of *duality*. We show how the approach of weak duality allows to use convex optimization to approximately solve non-convex problems. The notion of strong duality, which applies to most convex problems, allows to derive rigorous optimality conditions, and algorithms. Duality has many other applications, ranging from decentralized algorithms for large-scale convex problems, to dimensionality reduction for problems with large number of variables and small number of constraints, or vice-versa.
- [**Part 5**](#) collects a few practical *case studies* that are evoked in the previous parts, and more. The applications range from image processing, to circuit design, antenna array design, and portfolio optimization.

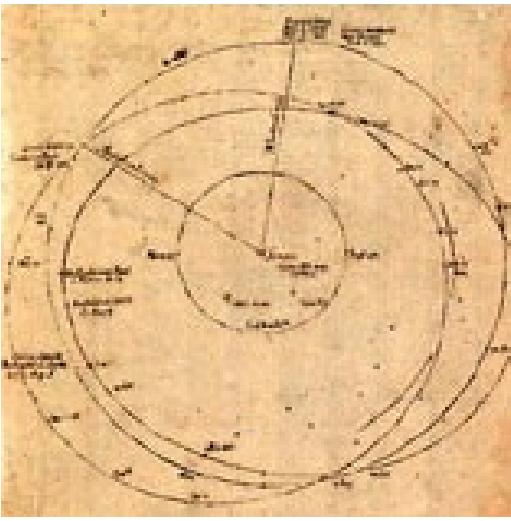
Optimization Models



Optimization refers to a branch of applied mathematics concerned with the minimization or maximization of a certain function, possibly under constraints. The birth of the field can be perhaps traced back to an astronomy problem solved by the young Gauss in the 1850s. It matured later with advances in physics, notably mechanics, where natural phenomena were described as the result of the minimization of a certain “energy” function. Optimization has evolved towards the study and application of algorithms to solve mathematical problems on computers.

To day, the field contributes to many disciplines, ranging from statistics, dynamical systems and control, complexity theory, algorithms. It is applied to a widening array of contexts, including machine learning and information retrieval, engineering design, economics, finance, and management.

Gauss' Bet In the early 1800's, astronomers had just discovered a new planetoid, Ceres, when the object became impossible to track due to the glare of the sun. Surely the object would reappear some time soon; but where to look in the vast sky? The problem of predicting the location of the object, based on location data gathered during the past 40 days, became the challenge of the day for the astronomy community.



The problem raised the interest of a young mathematician, [Carl Friedrich Gauss](#), who had invented the method of [least-squares](#) at age 18. He used it (together with several other approximations) to predict the location of Ceres. His computations were so accurate as to allow the astronomer Franz Xaver von Zach to quickly relocate Ceres. The figure on the left shows a draft of Gauss' orbital map. For more on this story, see [here](#) or [here](#).

Note that there is a controversy as to the identity of the inventor of the Least-Squares method, as [Legendre](#) has published it earlier than Gauss (in 1805), but Gauss claimed to have discovered it years before (see [here](#)).

Functions and Maps

Functions. In this course we define *functions* as objects which take an argument in \mathbf{R}^n , and return a value in \mathbf{R} . We use the notation $f : \mathbf{R}^n \rightarrow \mathbf{R}$,

to refer to a function with “input” space \mathbf{R}^n . The “output” space for functions is \mathbf{R} .

Example: The function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$ with values

$$f(x) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

gives the distance from the point (x_1, x_2) to a given point (y_1, y_2) .

Domain. We allow functions to take infinity values. The *domain* of a function f , denoted $\text{dom } f$, is defined as the set of points where the function is finite.

Example: Define the logarithm function as the function $f : \mathbf{R} \rightarrow \mathbf{R}$, with values $f(x) = \log x$ if $x > 0$, and $-\infty$ otherwise. The domain of the function is thus \mathbf{R}_{++} (the set of positive reals).

Two functions can differ not by their formal expression, but because they have different domains. For example,

$$f(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ +\infty & \text{otherwise,} \end{cases}, \quad g(x) = \begin{cases} 1/x & \text{if } x > 0 \\ +\infty & \text{otherwise,} \end{cases}$$

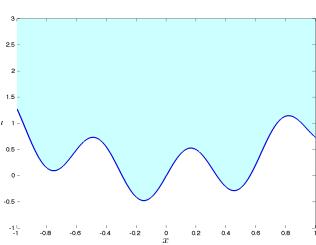
have the same formal expression inside their respective domains. However, they are not the same functions, since their domain is different.

Maps We reserve the term *map* to refer to vector-valued functions. That is, maps are functions which return more than a single value. We use the notation $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$,

to refer to a map with input space \mathbf{R}^n and output space \mathbf{R}^m . The *components* of the map f are the (scalar-valued) functions $f_i, i = 1, \dots, m$.

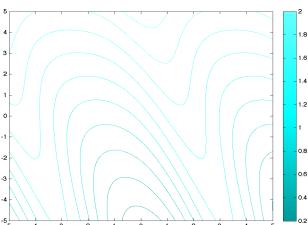
Example: the map $f : \mathbf{R}^2 \rightarrow \mathbf{R}^2$, with values $f(x) = (x_1^2 + 3x_2, x_1 x_2 - 4)$.

Graph and Epigraph Consider a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$. We can define two sets relevant to f : the *graph* and the *epigraph*. Both are subsets of \mathbf{R}^{n+1} . Graph. The *graph* of f is the set of input-output pairs that f can attain, that is: $G(f) := \{(x, f(x)) \in \mathbf{R}^{n+1} : x \in \mathbf{R}^n\}$. Epigraph. The *epigraph*, denoted $\text{epi } f$, describes the set of input-output pairs that f can achieve, as well as “anything above” (*epi* in Greek means “above”). $\text{epi } f := \{(x, t) \in \mathbf{R}^{n+1} : x \in \mathbf{R}^n, t \geq f(x)\}$.



The function $f : \mathbf{R} \rightarrow \mathbf{R}$, with domain $(-1, 1)$ and value inside the domain $f(x) = x^2 + (1/2) \sin(10x)$. The graph corresponds to the points in blue, and the epigraph is in light blue. The epigraph extends to infinity above the graph. Points outside the domain are not shown.

Level and Sub-level Sets Level and sub-level sets correspond to the notion of contour of a function. Both are indexed on some scalar value t . Level sets. A *level set* is simply the set of points that achieve exactly some value for the function f . For $t \in \mathbf{R}$, the t -level set of the function f is defined as $L_t(f) := \{x \in \mathbf{R}^n : x \in \mathbf{R}^n, t = f(x)\}$. Sub-level sets. A related notion is that of *sub-level set*. This is now the set of points that achieve at most a certain value for f , or below. $S_t(f) := \{x \in \mathbf{R}^n : x \in \mathbf{R}^n, t \geq f(x)\}$.



Level and sub-level sets of a function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$, with domain \mathbf{R}^2 itself, and values on the domain given by

$$f(x) = \log(e^{\sin(x_1 + 0.3x_2 - 0.1)} + e^{2x_2 + 0.7}).$$

Standard forms

Functional form
An optimization problem is a problem of the form

$$p^* := \min_x f_0(x) \text{ subject to } f_i(x) \leq 0, \quad i = 1, \dots, m,$$

where

- $x \in \mathbf{R}^n$ is the *decision variable*;
- $f_0 : \mathbf{R}^n \rightarrow \mathbf{R}$ is the *objective function*, or *cost*;
- $f_i : \mathbf{R}^n \rightarrow \mathbf{R}, i = 1, \dots, m$ represent the *constraints*;
- p^* is the *optimal value*.

In the above, the term “subject to” is sometimes replaced with a short-hand colon notation.

Often the above is referred to as a “mathematical program”. The term “programming” (or “program”) does not refer to a computer code. It is used mainly for historical purposes. We will use the more rigorous (but less popular) term “optimization problem”.

Example: [An optimization problem in two variables](#).

Epigraph form

In optimization, we can always assume that the objective is a linear function of the variables. This can be done via the *epigraph* representation of the problem, which is based on adding a new scalar variable t :

$$p^* = \min_{x,t} t \text{ subject to } f_0(x) - t \leq 0, \quad f_i(x) \leq 0, \quad i = 1, \dots, m.$$

At optimum, $t = p^*$. In the above, the objective function is $\tilde{f} : \mathbf{R}^{n+1} \rightarrow \mathbf{R}$, with values $\tilde{f}_0(x, t) = t$.

We can picture this as follows. Consider the *sub-level sets* of the objective function, which are of the form $\{x : t \geq f_0(x)\}$ for some $t \in \mathbf{R}$. The problem amounts to finding the smallest t for which the corresponding sub-level set intersects the set of points that satisfy the constraints.

Example: [Geometric view of the optimization problem in two variables](#).

Other standard forms

Sometimes we single out equality constraints, if any:

$$\min_x f_0(x) \text{ subject to } h_i(x) = 0, \quad i = 1, \dots, p, \quad f_i(x) \leq 0, \quad i = 1, \dots, m,$$

where h_i 's are given. Of course, we may reduce the above problem to the standard form above, representing each equality constraint by a pair of inequalities.

Sometimes, the constraints are described abstractly via a set condition, of the form $x \in \mathbf{X}$ for some subset \mathbf{X} of \mathbf{R}^n . The corresponding notation is

$$\min_{x \in \mathbf{X}} f_0(x)$$

Some problems come in the form of maximization problems. Such problems are readily cast in standard form via the expression

$$\max_{x \in \mathbf{X}} f_0(x) = -\min_{x \in \mathbf{X}} g_0(x),$$

where $g_0 := -f_0$.

Nomenclature

Consider the optimization problem

$$(P) \quad \min_x f_0(x) : f_i(x) \leq 0, \quad i = 1, \dots, m. \quad \text{Feasible set}$$

The *feasible set* of problem (P) is defined as

$$\mathbf{X} := \{x \in \mathbf{R}^n : f_i(x) \leq 0, \quad i = 1, \dots, m\}.$$

A point x is said to be *feasible* for problem (P) if it belongs to the feasible set \mathbf{X} , that is, it satisfies the constraints.

Example: In the [toy optimization problem](#), the feasible set is the “box” in \mathbf{R}^2 , described by $-1 \leq x_1 \leq 2, 0 \leq x_2 \leq 3$.

The feasible set may be empty, if the constraints cannot be satisfied simultaneously. In this case the problem is said to be *infeasible*.

What is a solution?

In an optimization problem, we are usually interested in computing the optimal value of the objective function, and also often a *minimizer*, which is a vector which achieves that value, if any.

Feasibility problems

Sometimes an objective function is not provided. This means that we are just interested in finding a feasible point, or determine that the problem is infeasible. By convention, we set f_0 to be a constant

in that case, to reflect the fact that we are indifferent to the choice of a point x as long as it is feasible.

Optimal value

The *optimal value* of the problem is the value of the objective at optimum, and we denote it by p^* :

$$p^* := \min_x f_0(x) : f_i(x) \leq 0, \quad i = 1, \dots, m.$$

Example: In the [toy optimization problem](#), the optimal value is $p^* = -10.2667$.

Optimal set

The *optimal set* (or, set of solutions) of problem (P) is defined as the set of feasible points for which the objective function achieves the optimal value:

$$\mathbf{X}^{\text{opt}} := \{x \in \mathbf{R}^n : f_0(x) = p^*, \quad f_i(x) \leq 0, \quad i = 1, \dots, m\}.$$

We take the convention that the optimal set is empty if the problem is not feasible.

A standard notation for the optimal set is via the $\arg \min$ notation:

$$\mathbf{X}^{\text{opt}} = \arg \min_{x \in \mathbf{X}} f_0(x).$$

A point x is said to be *optimal* if it belongs to the optimal set. Optimal points may not exist, and the optimal set may be empty. This can be due to the fact that the problem is infeasible. Or, it may be due to the fact that the optimal value is only attained in the limit.

Example: The problem

$$\min_x e^{-x}$$

has no optimal points, as the optimal value $p^* = 0$ is only attained in the limit $x \rightarrow +\infty$.

If the optimal set is not empty, we say that the problem is *attained*.

Suboptimality

The ϵ -*suboptimal set* is defined as

$$\mathbf{X}_\epsilon := \{x \in \mathbf{R}^n : f_i(x) \leq 0, \quad i = 1, \dots, m, \quad f_0(x) \leq p^* + \epsilon\}.$$

(With our notation, $\mathbf{X}_0 = \mathbf{X}_{\text{opt}}$.) Any point in the ϵ -suboptimal set is termed ϵ -suboptimal.

This set allows to characterize points which are close to being optimal (when ϵ is small). Usually practical algorithms are only able to compute suboptimal solutions, and never reach true optimality.

Example: [Nomenclature of the two-dimensional toy problem](#).

Local vs. global optimal points

A point z is *locally optimal* if there is a value $R > 0$ such that z is optimal for problem

$$\min_x f_0(x) : f_i(x) \leq 0, \quad i = 1, \dots, m, \quad \|z - x\|_2 \leq R$$

In other words, a local minimizer z minimizes f_0 , but only for nearby points on the feasible set. So the value of the objective function is *not* necessarily the optimal value of the problem.

The term *globally optimal* (or, optimal for short) is used to distinguish points in the optimal set from local minima.

Example: [a function with local minima](#).

Local optima may be described as the curse of general optimization, as most algorithms tend to be trapped in local minima if they exist.

Hard vs. Easy Problems

Not all optimization problems are created equal. Some problem classes (such as finding a solution to a set of linear equalities or inequalities) are easy, that is, they can be solved in a reasonable amount of time and memory on a computer. Others are inescapably hard: the *travelling salesman problem*, which involves finding a path among a combinatorial number of choices, is a famous example.

The field of complexity theory gives a rough, but very useful, guideline in deciding the “hardness” of a problem. In complexity theory, the focus is on how the worst-case computing time grows as the problem size grows. Problem size can be complicated notion, but we can roughly think of it as the number of variables and constraints in an optimization problem with (obviously the cost of evaluating the objective and constraints for a given choice of the variable also counts).

In the late 80s, it was believed that what made an optimization problem hard was its non-linear components. In other words: linear problems are easy; non-linear ones are hard. A major result, due to Nesterov and Nemirovski (1989), showed that this is not the case: the big watershed is between convex and non-convex problems. Roughly speaking, convex problems are easy (and that includes linear programming problems); non-convex ones are hard. Of course, this statement needs to be qualified. Not *all* convex problems are easy, but a (reasonably large) subset. Conversely, some non-convex problems are actually easy; for example some path planning problems can be solved in linear time. In this class, we will explore the contours of the set of convex problems that are easy to solve.

Some classes of optimization problemsLeast-squares

The least-squares problem is

$$\min_x \sum_{i=1}^m \left(\sum_{j=1}^n A_{ij} x_j - b_i \right)^2,$$

where A_{ij} , b_i , $1 \leq i \leq m$, $1 \leq j \leq n$, are given numbers, and $x \in \mathbf{R}^n$ is the variable.

The problem was posed and solved by Gauss (1777-1855), who used the method it to [predict the trajectory](#) of the planetoid Ceres. Least-squares problems arise in many situations, for example in statistical estimation problems such as linear regression. In image compression this problem also arises, in which case b_i 's contain a pixel representation of a given image, and for every j , the numbers A_{ij} , $i = 1, \dots, m$ contain representations of “basic” images; the sums inside the squares represents a linear combination of these basic images that is supposed to approximate as well as possible the original image contained in b_i 's.

Example: [linear regression via least-squares](#).

Linear programming

This problem is often referred to with the acronym LP, and has the form

$$\min \sum_{j=1}^n c_j x_j : \sum_{j=1}^n A_{ij} x_j \leq b_i, \quad i = 1, \dots, m,$$

where c_j , b_i and A_{ij} , $1 \leq i \leq m$, $1 \leq j \leq n$, are given real numbers. This corresponds to the case where the functions f_i ($i = 0, \dots, m$) in the [standard problem](#) are all affine (that is, linear plus a constant term).

This problem was introduced by G. Dantzig in the 40's in the context of logistical problems arising in military operations. This model of computation is perhaps the most widely used optimization problem today.

Example: [Linear classification](#).

Quadratic programming

Quadratic programming problems (QP's for short) are an extension of linear programming, which involve a sum-of-squares function in the objective. The linear program above is modified to be

$$\min \sum_{j=1}^n (x_i^2 + c_j x_j) : \sum_{j=1}^n A_{ij} x_j \leq b_i, \quad i = 1, \dots, m.$$

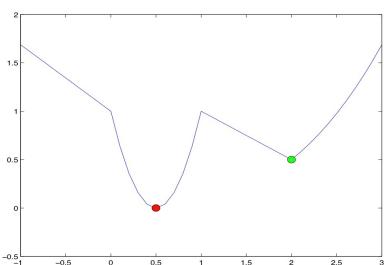
These problems can be thought of as a generalization of both the least-squares and linear programming problems.

QP's are popular in many areas, such as finance, where the linear term in the objective refers to the expected negative return on an investment, and the squared term corresponds to the risk (or variance of the return). This model was introduced in the 50's by H. Markowitz (who was then a colleague of Dantzig at the RAND Corporation), to model investment problems. Markowitz won the Nobel prize in Economics in 1990, mainly for this work.

Nonlinear optimization

Nonlinear optimization is perhaps the largest class of optimization problem. In general such problems are very hard to solve. (In fact, this class comprises combinatorial optimization: if a variable x_i is required to be boolean, we can model this as a single non-linear constraint $x_i^2 = x_i$.)

One of the reasons for which non-linear problems are hard to solve is the issue of *local minima*. We will define this notion rigorously, but the picture below provides an intuitive idea:

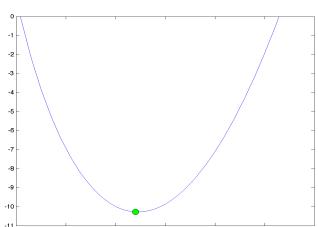


When trying to minimize general non-linear functions, algorithms may be trapped in so-called “local” minima (in green), which do not correspond to the true minimal value of the objective function (in red).

Example: [Protein folding](#).

Convex optimization

Convex optimization is a generalization of QP where the objective and constraints involve “bowl-shaped”, or convex, functions. Not all convex problems are easy to solve, but many of them are. One of the reasons for which this is (approximately) true is that, contrarily to general non-linear optimization problems, convex ones do not suffer from the “curse of local minima” mentioned above.



When trying to minimize convex (that is, bowl-shaped) functions, specialized algorithms will always converge to a global minimum, irrespective of the starting point, provided some (weak) assumptions on the function hold.

Combinatorial optimization

In combinatorial optimization, some (or all) the variables are boolean (or integers), reflecting discrete choices to be made.

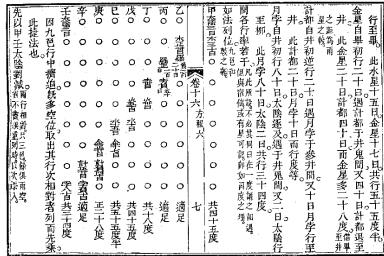
Example: [Crew allocation for airline operations](#).

Combinatorial optimization problems are in general extremely hard to solve. Often, they can be approximately solved with linear or convex programming.

History of Optimization

Early stages: birth of linear algebra

The roots of optimization, as a field concerned with algorithms for solving numerical problems, can perhaps be traced back to the earliest known appearance of a system of linear equations in ancient China. Indeed, the art termed *fangcheng* (often translated as “rectangular arrays”) was used as early as 300 BC to solve puzzles which amounted to linear systems. Records of algorithms identical to Gauss elimination for solving such systems appear in the treatise *Nine Chapters of Mathematical Methods*, around 100 BC.



The picture on the left pictures a 9×9 matrix found in the treatise (with a reversed convention for the column's order).

It is believed that many of the early Chinese results in linear algebra found gradually their way to Europe.

Optimization as a theoretical tool

Gauss (c. 1800) built on early results (and his own contributions) in linear algebra to develop a method for solving least-squares problems, which relied on solving an associated linear system (the famous *normal equations*). But this early algorithmic result remained an exception in the optimization landscape in 18th century Europe, as most of the development of the field remained at a theoretical level.

The notion of optimization problem was crucial to the development of theoretical mechanics and physics between the 17th and 19th century. Around 1750, Maupertuis introduced and later Euler formalized the *principle of least action*, according to which the motion of natural systems could be described as a minimization problem involving a certain cost function called “energy”. This optimization-based (or, *variational*) approach is the foundation of classical mechanics. The Italian mathematician Giuseppe Lodovico (Luigi) Lagrangia (1736–1813), also known as [Lagrange](#), was a key player in this development and his name is associated with a central notion in optimization, duality.

While optimization *theory* played a central role in physics, it is only with the advent of computers that it could start making its mark in practical applications, and venture in fields other than physics.

Advent of linear programming

Dantzig (1914-2005), working in the 40s on Pentagon-related logistical problems, started investigating the numerical solution to linear inequalities. Extending the scope of linear algebra (linear equalities) to inequalities seemed useful, and his efforts led to the famous *simplex algorithm* for solving such systems. Another important early contributor to the field of *linear programming* is the Soviet/Russian mathematician Leonid Kantorovich.

In the 60s-70s, a lot of attention was devoted to non-linear optimization problems. Methods to find local minima were found. In the meantime, researchers recognized that these methods could fail to find global minima, or even converge. Hence the notion that, while linear optimization was numerically tractable, non-linear optimization was not in general. This had concrete practical consequences: linear programming solvers could be reliably used for day-to-day operations (for example, for airline crew management); but non-linear solvers needed an expert to baby-sit them.

In the field of mathematics, the 60s saw the development of convex analysis, which would later serve as an important theoretical basis for progress in optimization.

Advent of convex programming

Most of the research in optimization in the United States in the 60s-80s focussed on nonlinear optimization algorithms, and contextual applications. The availability of large computers made that research possible and practical.

In the Soviet Union at that time, the focus was more towards optimization theory, perhaps due to more restricted access to computing resources. Since non-linear problems are hard, Soviet researchers went back to the linear programming model, and asked the following (at that point theoretical) question: what makes linear programs easy? Is it really linearity of the objective and constraint functions, or some other more general structure? Are there classes of problems out there that are non-linear but still easy to solve?

Nesterov and Nemirovski discovered in the late 80s that the key property that makes an optimization problem easy is convexity. Their result is not only theoretical but also algorithmic, as they discovered so-called *interior-point methods* for solving convex problems. Since their result, convex optimization has emerged as a powerful tool that generalizes linear algebra and linear programming: it has similar characteristics of reliability (it always converges to the global minimum) and tractability (it does so in reasonable time).

Present

In present times there is a very strong interest in optimization in applications, ranging from engineering design, statistics and machine learning, finance, etc.

Exercises

A. Consider the optimization problems (no assumption of convexity here)

$$\begin{aligned} p_1^* &:= \min_{x \in \mathcal{X}_1} f_0(x), \\ p_2^* &:= \min_{x \in \mathcal{X}_2} f_0(x), \\ p_{13}^* &:= \min_{x \in \mathcal{X}_1 \cap \mathcal{X}_3} f_0(x), \\ p_{23}^* &:= \min_{x \in \mathcal{X}_2 \cap \mathcal{X}_3} f_0(x), \\ \text{where } \mathcal{X}_1 &\subseteq \mathcal{X}_2 \end{aligned}$$

1. Prove that $p_1^* \geq p_2^*$ (i.e., enlarging the feasible set cannot worsen the optimal objective).
2. Prove that, if $p_1^* = p_2^*$, then it holds that

$$p_{13}^* = p_1^* \Rightarrow p_{23}^* = p_2^*.$$

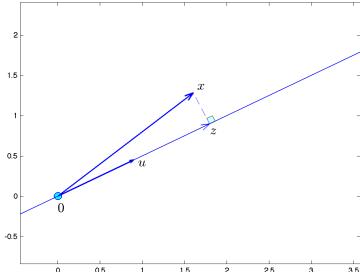
3. Assume that all problems above attain unique optimal solutions. prove that, under such a hypothesis, if $p_1^* = p_2^*$, then it holds that

$$p_{23}^* = p_2^* \Rightarrow p_{13}^* = p_1^*.$$

2. Linear Algebra

- 2.1 [Vectors](#)
- 2.2 [Matrices](#)
- 2.3 [Linear equations](#)
- 2.4 [Least-squares](#)
- 2.5 [Eigenvalues](#)
- 2.6 [Singular values](#)

Vectors



A vector is a collection of numbers arranged in a column or a row, and can be thought of as a [point](#) in space, or as providing a [direction](#). We review basic notions such as independence, span, subspaces, and dimension.

The scalar product allows to define the length of a vector, as well as generalize the notion of angle between two vectors. Via the scalar product, we can view a vector as a [linear function](#). We can also compute the [projection](#) of a vector onto a line defined by another — a basic ingredient in many [visualization](#) techniques for high-dimensional data sets.

Basics Definitions Vectors. Assume we are given a collection of n real numbers, x_1, \dots, x_n . We can represent them as n locations on a line. Alternatively, we can represent the collection as a single [point](#) in a n -dimensional space. This is the *vector* representation of the collection of numbers; each number x_i is called a *component* or *element* of the vector.

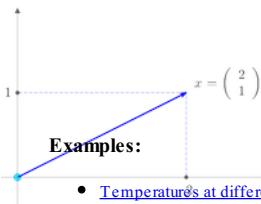
Vectors can be arranged in a column, or a row; we usually write vectors in column format:

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

We denote by \mathbf{R}^n the set of real vectors with n components. If $x \in \mathbf{R}^n$ denotes a vector, we use subscripts to denote components, so that x_i is the i -th component of x . Sometimes the notation $x(i)$ is used to denote the i -th component.

Geometry. A vector represents both a *direction from the origin* and a *point* in the multi-dimensional space \mathbf{R}^n , where each component corresponds to coordinate of the point.

Example: The vector $x = (2, 1)$ in \mathbf{R}^2 . It is both a point (at the tip of the arrow) and a direction from the origin.



Examples:

- [Temperatures at different airports.](#)
- [Bag-of-words representation of text.](#)

Transpose. If x is a column vector, x^T denotes the corresponding row vector, and vice-versa. Hence, if x is the column vector above: $x^T = (x_1 \dots x_n)$.

Sometimes we use the looser, in-line notation $x = (x_1, \dots, x_n)$, to denote a row or column vector, the orientation being understood from context.

Matlab syntax. A column vector $x = (2, 3.1, -4)$ and its transpose y can be declared in Matlab's workspace as follows. Here, no room for loose notation: we use a semicolon to separate the components of a column vector, while we use commas for row vectors.

Matlab syntax: declare and transpose a vector

```
>> x = [2; 3.1; -4]; % declares a column vector using ";".
>> y = x'; % the prime operator ' transposes the vector.
>> y = [2, 3.1, -4]; % can also declare a row vector with commas.
>> x(2) % this produces the second component of x.
>> x([1, 3]) % this produces the 2-vector with the first and the third component of x.
```

Independence

A set of vectors $\{x_1, \dots, x_m\}$ in \mathbf{R}^n , $i = 1, \dots, m$ is said to be *independent* if and only if the following condition on a vector $\lambda \in \mathbf{R}^m$:

$$\sum_{i=1}^m \lambda_i x_i = 0$$

implies $\lambda = 0$. This means that no vector in the set can be expressed as a linear combination of the others.

Example: the vectors $x^1 = [1, 2, 3]$ and $x^2 = [3, 6, 9]$ are *not* independent, since $3x^1 - x^2 = 0$.

Subspace, span, affine sets

Subspaces. A *subspace* of \mathbf{R}^n is a subset that is closed under addition and scalar multiplication. Geometrically, subspaces are “flat” (like a line or plane in 3D) and pass through the origin.

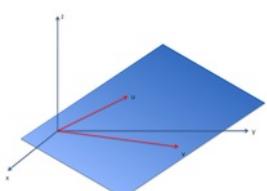
An important result of linear algebra, which we will prove later, says that a subspace \mathbf{S} can always be represented as the *span* of a set of vectors $x_i \in \mathbf{R}^n$, $i = 1, \dots, m$, that is, as a set of the form

$$\mathbf{S} = \text{span}(x_1, \dots, x_m) := \left\{ \sum_{i=1}^m \lambda_i x_i : \lambda \in \mathbf{R}^m \right\}.$$

Affine sets. An *affine set* is a translation of a subspace — it is “flat” but does not necessarily pass through $\mathbf{0}$, as a subspace would. (Think for example of a line, or a plane, that does not go through the origin.) So an affine set \mathbf{A} can always be represented as the translation of the subspace spanned by some vectors:

$$\mathbf{A} = \left\{ x_0 + \sum_{i=1}^m \lambda_i x_i : \lambda \in \mathbf{R}^m \right\},$$

for some vectors x_0, x_1, \dots, x_m . In shorthand notation, we write $\mathbf{A} = x_0 + \mathbf{S}$.



Example: In \mathbf{R}^3 , the span \mathbf{S} of the two vectors

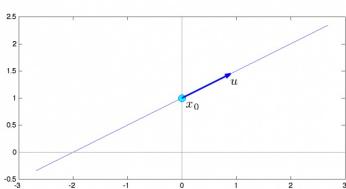
$$u = \begin{bmatrix} -1 \\ 2 \\ 0.5 \end{bmatrix}, \quad v := \begin{bmatrix} 1 \\ 3 \\ 0.1 \end{bmatrix}$$

is the plane passing through the origin pictured in blue.

Special case: lines. When \mathbf{S} is the span of a single non-zero vector, the set \mathbf{A} is called a *line* passing through the point x_0 . Thus, lines have the form

$$\{x_0 + tu : t \in \mathbf{R}\}.$$

where u determines the direction of the line, and x_0 is a point through which it passes.



Example: A line in \mathbf{R}^2 passing through the point

$$x_0 = (2, 0),$$

with direction $u = (0.8944, 0.4472)$.

Basis, dimensionBasis in \mathbf{R}^m . A *basis* of \mathbf{R}^n is a set of n independent vectors. If the vectors u_1, \dots, u_n form a basis, we can express any vector as a linear combination of the u_i 's:

$$x = \sum_{i=1}^n \lambda_i u_i$$

for appropriate numbers $\lambda_1, \dots, \lambda_n$.

The *standard basis* (alternatively, *natural basis*) in \mathbf{R}^n consists of the vectors e_i , where e_i 's components are all zero, except the i -th, which is equal to 1. In \mathbf{R}^3 , we have

$$e_1 := \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad e_2 := \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad e_3 := \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Example: [A basis in \$\mathbf{R}^3\$](#)

Basis of a subspace. The basis of a given subspace $S \subseteq \mathbf{R}^n$ is any *independent* set of vectors whose span is S . If the vectors u_1, \dots, u_r form a basis of S , we can express any vector as a linear combination of the u_i 's:

$$x = \sum_{i=1}^r \lambda_i u_i$$

for appropriate numbers $\lambda_1, \dots, \lambda_r$.

The number of vectors in the basis is actually independent of the choice of the basis (for example, in \mathbf{R}^3 you need two independent vectors to describe a plane containing the origin). This number is called the *dimension* of S . We can accordingly define the dimension of an affine subspace, as that of the linear subspace of which it is a translation.

Examples:

- The dimension of a line is 1, since a line is of the form $x_0 + \text{span}(x_1)$ for some non-zero vector x_1 .
- [Dimension of an affine subspace](#).

Scalar Product, Norms and Angles

The *scalar product* (or, *inner product*, or *dot product*) between two vectors $x, y \in \mathbf{R}^n$ is the scalar denoted $x^T y$, and defined as

$$x^T y = \sum_{i=1}^n x_i y_i.$$

The motivation for our notation above will come [later](#), when we define the matrix-matrix product. The scalar product is also sometimes denoted $\langle x, y \rangle$, a notation which originates in physics.

In matlab, we use a notation consistent with a later definition of [matrix-matrix product](#).

```
Matlab syntax: scalar product

x = [1; 2; 3]; y = [4; 5; 6];
scal_prod = x'*y;
```

Examples:

- [Rate of return of a financial portfolio](#).
- [Sample and weighted average](#).
- [Beer-Lambert law in absorption spectroscopy](#).

Orthogonality

We say that two vectors $x, y \in \mathbf{R}^n$ are *orthogonal* if $x^T y = 0$.

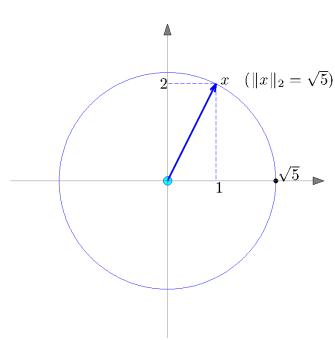
Example: [Two orthogonal vectors in \$\mathbf{R}^3\$](#)

Norms Definition. Measuring the size of a scalar value is unambiguous — we just take the magnitude (absolute value) of the number. However, when we deal with higher dimensions, and try to define the notion of size, or length, of a vector, we are faced with many possible choices. These choices are encapsulated in the notion of *norm*.

Norms are real-valued functions that satisfy a basic set of rules that a sensible notion of size should involve. You can consult the formal definition of a norm [here](#). The norm of a vector v is usually denoted $\|v\|$.

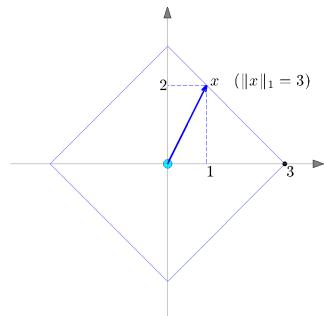
Three popular norms. In this livebook, we focus on the following three popular norms for a vector $x \in \mathbf{R}^n$:

The *Euclidean norm*:



$$\|x\|_2 := \sqrt{\sum_{i=1}^n x_i^2} = \sqrt{x^T x},$$

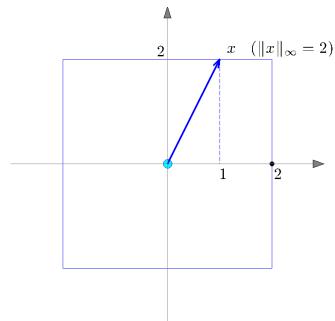
corresponds to the usual notion of distance in two or three dimensions. The set of points with equal L_2 -norm is a circle (in 2D), a sphere (in 3D), or a *hyper-sphere* in higher dimensions.



The L_1 -norm:

$$\|x\|_1 = \sum_{i=1}^n |x_i|,$$

corresponds to the distance travelled on a rectangular grid to go from one point to another.



The L_∞ -norm:

$$\|x\|_\infty := \max_{1 \leq i \leq n} |x_i|,$$

is useful in measuring peak values.

Matlab syntax:

```
>> x = [1; 2; -3];
>> r2 = norm(x,2); % L2-norm
>> r1 = norm(x,1); % L1 norm
>> rinf = norm(x,inf); % L-infinity norm
```

Examples:

- A given vector will in general have different “lengths” under different norms. For example, the vector $x = [1, -2, 3]^T$ yields $\|x\|_2 = 3.7417$, $\|x\|_1 = 6$, and $\|x\|_\infty = 3$.
- [Sample standard deviation](#).

Cauchy-Schwartz inequality

The Cauchy-Schwartz inequality allows to bound the scalar product of two vectors in terms of their Euclidean norm.

Theorem: Cauchy-Schwartz inequality

For any two vectors $x, y \in \mathbf{R}^n$, we have

$$x^T y \leq \|x\|_2 \cdot \|y\|_2.$$

The above inequality is an equality if and only if x, y are collinear. In other words:

$$\max_{x : \|x\|_2 \leq 1} x^T y = \|y\|_2,$$

with optimal x given by $x^* = y/\|y\|_2$ if y is non-zero.

For a proof, see [here](#). The Cauchy-Schwartz inequality can be generalized to other norms, using the concept of [dual norm](#).

Angles between vectors

When none of the vectors x, y is zero, we can define the corresponding angle as θ such that

$$\cos \theta = \frac{x^T y}{\|x\|_2 \|y\|_2}.$$

Applying the Cauchy-Schwartz inequality above to (x, y) and $(x, -y)$ we see that indeed the number above is in $[-1, 1]$.

The notion above generalizes the usual notion of angle between two directions in two dimensions, and is useful in measuring the similarity (or, closeness) between two vectors. When the two vectors are orthogonal, that is, $x^T y = 0$, we obtain that their angle is $\theta = 90^\circ$.

Example:

- [Similarity of two documents](#).

Projection on a line

Definition

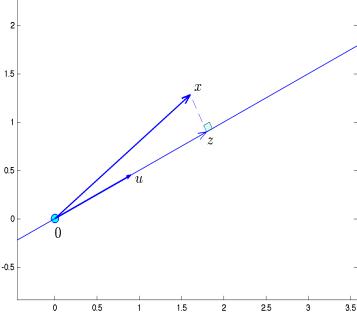
Consider the line in \mathbf{R}^n passing through $x_0 \in \mathbf{R}^n$ and with direction $u \in \mathbf{R}^n$:

$$\{x_0 + tu : t \in \mathbf{R}\},$$

The projection of a given point x on the line is a vector z located on the line, that is closest to x (in Euclidean norm). This corresponds to a simple optimization problem:

$$\min_t \|x - x_0 - tu\|_2.$$

This particular problem is part of a general class of optimization problems known as [least-squares](#). It is also a special case of a [Euclidean projection](#) on a general set.



Projection of the vector $x = (1.6, 2.28)$ on a line passing through the origin ($x_0 = 0$) and with (normalized) direction $u = (0.8944, 0.4472)$. At optimality the “residual” vector $x - z$ is orthogonal to the line, hence $z = tu$, with $t = x^T u = 2.0035$. Any other point on the line is farther away from the point x than its projection z is.

The scalar $t = u^T x$, i.e. the scalar product between x and u , is the component of x along the normalized direction u .

Closed-form expression

Assuming that u is normalized, so that $\|u\|_2 = 1$, the objective function of the projection problem reads, after squaring:

$$\|x - x_0 - tu\|_2^2 = t^2 - 2tu^T(x - x_0) + \|x - x_0\|_2^2 = (t - u^T(x - x_0))^2 + \text{constant}.$$

Thus, the optimal solution to the projection problem is

$$t^* = u^T(x - x_0),$$

and the expression for the projected vector is

$$z^* = x_0 + t^*u = x_0 + u^T(x - x_0)u.$$

The scalar product $u^T(x - x_0)$ is the component of $x - x_0$ along u .

In the case when u is not normalized, the expression is obtained by replacing u with its scaled version $u/\|u\|_2$:

$$z^* = x_0 + \frac{u^T(x - x_0)}{u^T u}u.$$

Interpreting the scalar product

We can now interpret the scalar product between two non-zero vectors x, u , by applying the previous derivation to the projection of x on the line of direction u passing through the origin. If u is normalized ($\|u\|_2 = 1$), then the projection of x on u is $z^* = (u^T x)u$. Its length is $\|z^*\|_2 = |u^T x|$. (See above figure.)

In general, the scalar product $u^T x$ is simply the component of x along the normalized direction $u/\|u\|_2$ defined by u .

Orthogonalization: the Gram-Schmidt procedure

A basis $(u_i)_{i=1}^n$ is said to be *orthogonal* if $u_i^T u_j = 0$ if $i \neq j$. If in addition, $\|u_i\|_2 = 1$, we say that the basis is *orthonormal*.

Example: An orthonormal basis in \mathbf{R}^3 . The collection of vectors $\{u_1, u_2\}$, with

$$u_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad u_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix},$$

forms an orthonormal basis of \mathbf{R}^2 .

What is orthogonalization?

Orthogonalization refers to a procedure that finds an orthonormal basis of the span of given vectors.

Given vectors $a_1, \dots, a_k \in \mathbf{R}^n$, an orthogonalization procedure computes vectors $q_1, \dots, q_n \in \mathbf{R}^n$ such that

$$S := \text{span}\{a_1, \dots, a_k\} = \text{span}\{q_1, \dots, q_r\},$$

where r is the dimension of S , and

$$q_i^T q_j = 0 \quad (i \neq j), \quad q_i^T q_i = 1, \quad 1 \leq i, j \leq r$$

That is, the vectors (q_1, \dots, q_r) form an orthonormal basis for the span of the vectors a_1, \dots, a_k .

Basic step: projection on a line

A basic step in the procedure consists in projecting a vector on a line passing through zero. Consider the line

$$L(q) := \{tq : t \in \mathbf{R}\},$$

where $q \in \mathbf{R}^n$ is given, and normalized ($\|q\|_2 = 1$).

The *projection* of a given point $a \in \mathbf{R}^n$ on the line is a vector v located on the line, that is closest to a (in Euclidean norm). This corresponds to a simple optimization problem:

$$\min_t \|a - tq\|_2.$$

The vector $a_{\text{proj}} := t^* q$, where t^* is the optimal value, is referred to as the *projection* of a on the line $L(q)$. As seen [here](#), the solution of this simple problem has a closed-form expression:

$$a_{\text{proj}} = (q^T a)q.$$

Note that the vector x can now be written as a sum of its projection and another vector that is orthogonal to the projection:

$$a = (a - a_{\text{proj}}) + a_{\text{proj}} = (a - (q^T a)q) + (q^T a)q,$$

where $a - a_{\text{proj}} = a - (q^T a)q$ and $a_{\text{proj}} = (q^T a)q$ are orthogonal. The vector $a - a_{\text{proj}}$ can be interpreted as the result of removing the component of a along q .

Gram-Schmidt procedure

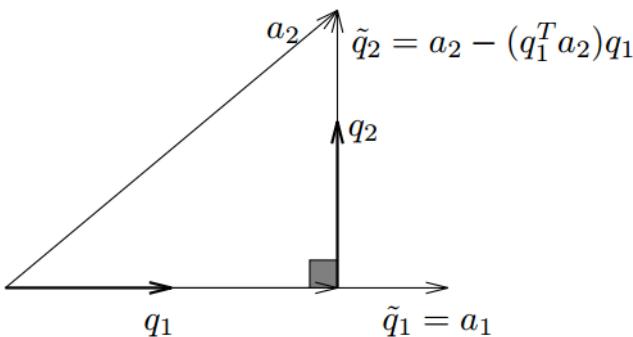
The Gram-Schmidt procedure is a particular orthogonalization algorithm. The basic idea is to first orthogonalize each vector w.r.t. previous ones; then normalize result to have norm one.

Case when the vectors are independent

Let us assume that the vectors a_1, \dots, a_n are linearly independent. The GS algorithm is as follows.

Gram-Schmidt procedure:

1. set $\tilde{q}_1 = a_1$
2. normalize: set $q_1 = \tilde{q}_1 / \|\tilde{q}_1\|_2$
3. remove component of q_1 in a_2 : set $\tilde{q}_2 = a_2 - (a_2^T q_1)q_1$
4. normalize: set $q_2 = \tilde{q}_2 / \|\tilde{q}_2\|_2$
5. remove components of q_1, q_2 in a_3 : set $\tilde{q}_3 = a_3 - (a_3^T q_1)q_1 - (a_3^T q_2)q_2$
6. normalize: set $q_3 = \tilde{q}_3 / \|\tilde{q}_3\|_2$
7. etc.



The image of the left shows the GS procedure applied to the case of two vectors in two dimensions. We first set the first vector to be a normalized version of the first vector a_1 . Then we remove the component of a_2 along the direction q_1 . The difference is the (un-normalized) direction \tilde{q}_2 , which becomes q_2 after normalization. At the end of the process, the vectors q_1, q_2 have both unit length and are orthogonal to each other.

The GS process is well-defined, since at each step $\tilde{q}_i \neq 0$ (otherwise this would contradict the linear independence of the a_i 's).

General case: the vectors may be dependent

It is possible to modify the algorithm to allow it to handle the case when the a_i 's are not linearly independent. If at step i , we find $\tilde{q}_i = 0$, then we directly jump at the next step.

Modified Gram-Schmidt procedure:

1. set $r = 0$

2. for $i = 1, \dots, n$:

1. set $\tilde{q} = a_i - \sum_{j=1}^r (q_j^T a_i) q_j$

2. if $\tilde{q} \neq 0$, $r = r + 1$; $q_r = \tilde{q}/\|\tilde{q}\|_2$.

On exit, the integer r is the dimension of the span of the vectors a_1, \dots, a_k .

Hyperplanes and half-spaces

Hyperplanes

A *hyperplane* is a set described by a single scalar product equality. Precisely, an hyperplane in \mathbf{R}^n is a set of the form

$$\mathbf{H} = \{x : a^T x = b\},$$

where $a \in \mathbf{R}^n$, $a \neq 0$, and $b \in \mathbf{R}$ are given. When $b = 0$, the hyperplane is simply the set of points that are orthogonal to a ; when $b \neq 0$, the hyperplane is a translation, along direction a , of that set.

If $x_0 \in \mathbf{H}$, then for any other element $x \in \mathbf{H}$, we have

$$b = a^T x_0 = a^T x.$$

Hence, the hyperplane can be characterized as the set of vectors x such that $x - x_0$ is orthogonal to a :

$$\mathbf{H} = \{x : a^T(x - x_0) = 0\}.$$

Hyperplanes are affine sets, of dimension $n - 1$ (see the proof [here](#)). Thus, they generalize the usual notion of a plane in \mathbf{R}^3 . Hyperplanes are very useful because they allow to separate the whole space in two regions. The notion of half-space formalizes this.

Example: [A hyperplane in \$\mathbf{R}^3\$](#) .

Projection on a hyperplane

Consider the hyperplane $\mathbf{H} = \{x : a^T x = b\}$, and assume without loss of generality that a is normalized ($\|a\|_2 = 1$). We can represent \mathbf{H} as the set of points x such that $x - x_0$ is orthogonal to a , where x_0 is any vector in \mathbf{H} , that is, such that $a^T x_0 = b$. One such vector is $x_{\text{proj}} := ba$.

By construction, x_{proj} is the *projection* of 0 on \mathbf{H} . That is, it is the point on \mathbf{H} closest to the origin, as it solves the projection problem

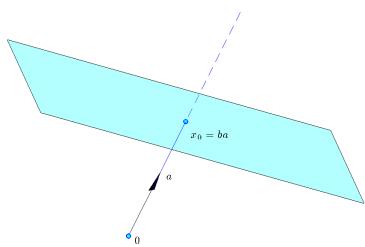
$$\min_x \|x\|_2 : x \in \mathbf{H}.$$

Indeed, for any $x \in \mathbf{H}$, using the Cauchy-Schwartz inequality:

$$\|x_0\|_2 = |b| = |a^T x| \leq \|a\|_2 \cdot \|x\|_2 = \|x\|_2,$$

and the minimum length $|b|$ is attained with $x_{\text{proj}} = ba$.

Geometry of hyperplanes



Geometrically, an hyperplane $\mathbf{H} = \{x : a^T x = b\}$, with $\|a\|_2 = 1$, is a translation of the set of vectors orthogonal to a . The direction of the translation is determined by a , and the amount by b .

Precisely, $|b|$ is the length of the closest point x_0 on \mathbf{H} from the origin, and the sign of b determines if \mathbf{H} is away from the origin along the direction a or $-a$. As we increase the magnitude of b , the hyperplane is shifting further away along $\pm a$, depending on the sign of b . In the image on the left, the scalar b is positive, as x_0 and a point to the same direction.

Half-spaces

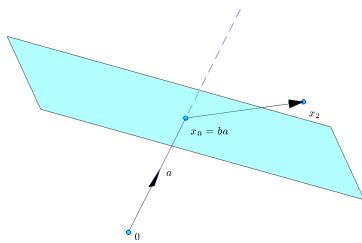
A half-space is a subset of \mathbf{R}^n defined by a single inequality involving a scalar product. Precisely, an half-space in \mathbf{R}^n is a set of the form

$$\mathbf{H} = \{x : a^T x \geq b\},$$

where $a \in \mathbf{R}^n$, $a \neq 0$, and $b \in \mathbf{R}$ are given.

Geometrically, the half-space above is the set of points such that $a^T(x - x_0) \geq 0$, that is, the angle between $x - x_0$ and a is acute (in $[-90^\circ, +90^\circ]$). Here x_0 is the point closest to the origin on the hyperplane defined by the equality $a^T x = b$. (When a is normalized, as in the picture, $x_0 = ba$.)

The half-space $\{x : a^T x \geq b\}$ is the set of points such that $x - x_0$ forms an acute angle with a , where x_0 is the projection of the origin on the boundary of the half-space.



Linear Functions and Maps

Linear and affine functions. Linear functions are functions which preserve scaling and addition of the input argument. Affine functions are “linear plus constant” functions.

Formal definition, linear and affine functions. A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is *linear* if and only if f preserves scaling and addition of its arguments:

- for every $x \in \mathbf{R}^n$, and $\alpha \in \mathbf{R}$, $f(\alpha x) = \alpha f(x)$; and
- for every $x_1, x_2 \in \mathbf{R}^n$, $f(x_1 + x_2) = f(x_1) + f(x_2)$.

A function f is *affine* if and only if the function $\tilde{f} : \mathbf{R}^n \rightarrow \mathbf{R}$ with values $\tilde{f}(x) = f(x) - f(0)$ is linear. \diamond

An alternative characterization of linear functions is given [here](#).

Examples: Consider the functions $f_1, f_2, f_3 : \mathbf{R}^2 \rightarrow \mathbf{R}$ with values

$$\begin{aligned} f_1(x) &= 3.2x_1 + 2x_2, \\ f_2(x) &= 3.2x_1 + 2x_2 + 0.15, \\ f_3(x) &= 0.001x_2^2 + 2.3x_1 + 0.3x_2. \end{aligned}$$

The function f_1 is linear; f_2 is affine; and f_3 is neither. \diamond

Connection with vectors via the scalar product

The following shows the connection between linear functions and scalar products.

Theorem: *Representation of affine function via the scalar product.*

A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is affine if and only if it can be expressed via a scalar product:

$$f(x) = a^T x + b,$$

for some unique pair (a, b) , with $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$, given by $a_i = f(e_i) - f(0)$, with e_i the i -th unit vector in \mathbf{R}^n , $i = 1, \dots, n$, and $b = f(0)$. The function is linear if and only if $b = 0$.

The theorem shows that a vector can be seen as a (linear) function from the “input” space \mathbf{R}^n to the “output” space \mathbf{R} . Both points of view (matrices as simple collections of numbers, or as linear functions) are useful.

Gradient of an affine function

The gradient of a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ at a point x , denoted $\nabla f(x)$, is the vector of first derivatives with respect to x_1, \dots, x_n (see [here](#) for a formal definition and examples). When $n = 1$ (there is only one input variable), the gradient is simply the derivative.

An affine function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, with values $f(x) = a^T x + b$ has a very simple gradient: the constant vector a . That is, for an affine function f , we have for every x :

$$\nabla f(x) = a.$$

Example: [gradient of a linear function](#).

Interpretations

The interpretation of a, b are as follows.

- The $b = f(0)$ is the constant term. For this reason, it is sometimes referred to as the *bias*, or *intercept* (as it is the point where f intercepts the vertical axis if we were to plot the graph of the function).
- The terms $a_j, j = 1, \dots, n$, which correspond to the gradient of f , give the *coefficients of influence* of x_j on f . For example, if $a_1 \gg a_3$, then the first component of x has much greater influence on the value of $f(x)$ than the third.

Example: [Beer-Lambert law in absorption spectrometry](#).

First-order approximation of non-linear functions

Many functions are non-linear. A common engineering practice is to approximate a given non-linear map with a linear (or affine) one, by taking derivatives. This is the main reason for linearity to be such an ubiquitous tool in engineering.

One-dimensional case. Consider a function of *one* variable $f : \mathbf{R} \rightarrow \mathbf{R}$, and assume it is differentiable everywhere. Then we can approximate the values function at a point x near a point x_0 as follows: $f(x) \simeq l(x) := f(x_0) + f'(x_0)(x - x_0)$,

where $f'(x)$ denotes the derivative of f at x .

Multi-dimensional case. With more than one variable, we have a similar result. Let us approximate a differentiable function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ by a linear function l , so that f and l coincide up and including to the first derivatives. The corresponding approximation l is called the *first-order approximation* to f at x_0 .

The approximate function l must be of the form

$$l(x) = a^T x + b,$$

where $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$. Our condition that l coincides with f up and including to the first derivatives shows that we must have

$$\nabla l(x) = a = \nabla f(x_0), \quad a^T x_0 + b = f(x_0),$$

where $\nabla f(x_0)$ the gradient of f at x_0 . Solving for a, b we obtain the following result:

Theorem: *First-order expansion of a function.*

The *first-order approximation* of a differentiable function f at a point x_0 is of the form

$$f(x) \approx l(x) = f(x_0) + \nabla f(x_0)^T (x - x_0)$$

where $\nabla f(x_0) \in \mathbf{R}^n$ is the gradient of f at x_0 .

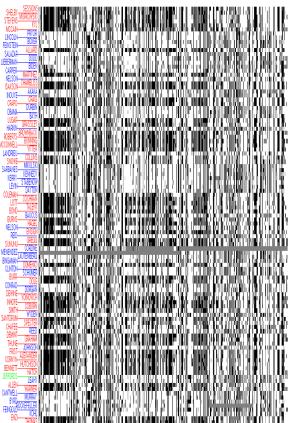
Example: [a linear approximation to a non-linear function](#).

Other sources of linear models

Linearity can arise from a simple change of variables. This is best illustrated with a specific example.

Example: [Power laws](#).

Application: data visualization by projection on a lineSenate voting data



In this section, we are focussing on a data set containing the votes of US Senators. This dataset can be represented as a collection of $n = 100$ vectors $x_j, j = 1, \dots, n$ in \mathbf{R}^m , with $m = 645$ the number of bills, and $n = 100$ the number of Senators. Thus, x_j contains all the votes of Senator j , and the i -th component of x_j contains the vote of that Senator on bill i .

Senate voting matrix: This image shows the votes of the $n = 100$ Senators in the 2004-2006 US Senate, for a total of $m = 645$ bills. "Yes" votes are represented as 1's, "No" as -1's, and the other votes are recorded as 0. Each row represents the votes of a single Senator, and each column contains the votes of all Senators for a particular bill. The vectors $x_j, j = 1, \dots, m = 645$ can be read as rows in the picture.

Source: [VoteWorld](#).

Visualization of high-dimensional data via projection

As seen in the picture above, simply plotting the raw data is often not very informative.

We can try to visualize the data set, by projecting each data point (each row or column of the matrix) on (say) a one-, two- or three-dimensional space. Each "view" corresponds to a particular projection, that is, a particular one-, two- or three-dimensional subspace on which we choose to project the data. Let us detail what it means to project on a *one-dimensional* set, that is, on a line.

[Projecting on a line](#) allows to assign a single number, or "score", to each data point, via a scalar product. We choose a (normalized) direction $u \in \mathbf{R}^m$, and a scalar $v \in \mathbf{R}$. This corresponds to the affine "scoring" function $f : \mathbf{R}^m \rightarrow \mathbf{R}$, which, to a generic data point $x \in \mathbf{R}^m$, assigns the value

$$f(x) = u^T x + v.$$

We thus obtain a vector of values $f \in \mathbf{R}^n$, with components $f_j = u^T x_j + v, j = 1, \dots, n$. It is often useful to center these scores around zero. This can be done by choosing v such that

$$0 = \sum_{j=1}^n (u^T x_j + v) = u^T \left(\sum_{j=1}^n x_j \right) + n \cdot v,$$

The zero-mean condition implies $v = -u^T \hat{x}$, where

$$\hat{x} := \frac{1}{n} \sum_{j=1}^n x_j \in \mathbf{R}^m$$

is the vector of [sample averages](#) of the different data points. The vector \hat{x} can be interpreted as the "average response" across data points (the average vote across Senators in our running example). The values of our scoring function can now be expressed as

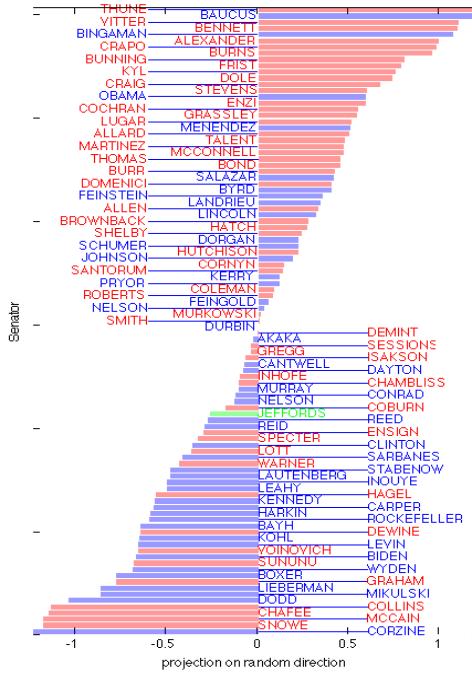
$$f(x) = u^T (x - \hat{x}).$$

In order to be able to compare the relative merits of different directions, we can assume, without loss of generality, that the direction vector u is normalized (so that $\|u\|_2 = 1$).

Note that our definition of $f(x)$ above is consistent with the idea of projecting the data points $(x_i - \hat{x})$ on the line passing through the origin and with normalized direction u . Indeed, the component of $x_j - \hat{x}$ on the line is $u^T(x_j - \hat{x})$.

In the Senate voting example above, a particular projection (that is, a direction in \mathbf{R}^m) corresponds to assigning a “score” to each Senator, and thus represent all the Senators as a single value on a line. We will project the data along a vector in the “bill” space, which is \mathbf{R}^m . That is, we are going to form linear combinations of the bills, so that the $m = 642$ votes for each Senator is reduced to a single number, or “score”. Since we centered our data, the average score (across Senators) is zero.

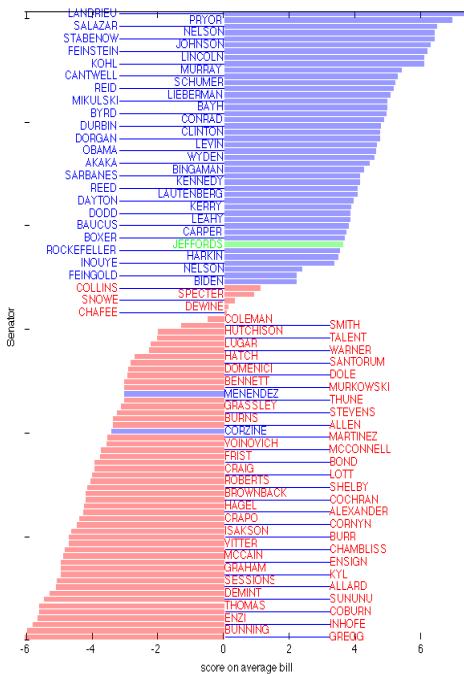
Example: Projection on a random direction



Scores obtained with random direction: This image shows the values of the projections of the Senator's votes $x_j - \hat{x}$ (that is, with average across Senators removed) on a (normalized) “random bill” direction. Each score is a number given by $u_{\text{rand}}(x_j - \hat{x})$, with u_{rand} a random vector (normalized to have unit Euclidean norm). We show the party affiliation, with the green color corresponding to the Independent Senator Jeffords. This projection shows no obvious structure; in particular it does not reveal the party affiliation.

Projection on the “all-ones” vector

Clearly, not all directions are “good”, in the sense of producing informative plots. [Here](#), we discuss a general principle that allows to choose an “informative” direction. But for this data set, a good guess could be to choose the direction that corresponds to the “average bill”. That is, we choose the direction u to be the parallel to the vector of ones in \mathbf{R}^m , scaled appropriately so that its Euclidean norm is one.



Scores obtained with all-ones direction: This image shows the values of the projections of the Senator's votes $x_j - \hat{x}$ (that is, with average across Senators removed) on a (normalized) “average bill” direction. Each score is a number given by $u_{\text{avg}}(x_j - \hat{x})$, with u_{avg} a vector with elements all equal to $1/\sqrt{m}$ (it is normalized to unit Euclidean norm). This projection reveals clearly the party affiliation of each senator, although the information was not available to the projection algorithm.

The interpretation is that the behavior of senators on an “average bill” almost fully determines her or his party affiliation. Note the range of the plot (about $[-6, 6]$), which is higher than that of the random vector (about $[-1, 1]$), allowing a better spread of scores.

Exercises Subspaces 1. Consider the set S of points such that $x_1 + 2x_2 + 3x_3 = 0$, $3x_1 + 2x_2 + x_3 = 0$.

Show that S is a subspace. Determine its dimension, and find a basis for it.

2. Consider the set in \mathbf{R}^3 , defined by the equation

$$P := \{x \in \mathbf{R}^3 : x_1 + 2x_2 + 3x_3 = 1\}.$$

- a. Show that the set P is an affine subspace of dimension 2. To this end, express it as $x^0 + \text{span}(x^1, x^2)$, where $x^0 \in P$, and x^1, x^2 are independent vectors.
b. Find the minimum Euclidean distance from 0 to the set P . Find a point that achieves the minimum distance. (*Hint:* using the [Cauchy-Schwartz inequality](#), prove that the minimum-distance point is proportional to $a := (1, 2, 3)$.)

1. Find the projection z of the vector $x = (2, 1)$ on the line that passes through $x_0 = (1, 2)$ and with direction given by the vector $u = (1, 1)$.

2. Find the [Euclidean projection](#) of a point $x_0 \in \mathbf{R}^n$ on a hyperplane $\mathbf{P} = \{x : a^T x = b\}$, where $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$ are given.

3. Determine the angle between the following two vectors:

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad y = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}.$$

Are these vectors linearly independent?

Orthogonalization

1. Let $x, y \in \mathbf{R}^n$ be two unit-norm vectors, that is, such that $\|x\|_2 = \|y\|_2 = 1$. Show that the vectors $x - y$ and $x + y$ are orthogonal. Use this to find an orthogonal basis for the subspace spanned by x and y .

Generalized Cauchy-Schwartz inequalities 1. Show that the following inequalities hold for any vector x : $\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty$.

1. Show that following inequalities hold for any vector x :

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2.$$

Hint: use the [Cauchy-Schwartz inequality](#) for the second inequality.

2. In a generalized version of the above inequalities, show that for any non-zero vector x ,

$$1 \leq \text{Card}(x) \leq \frac{\|x\|_1^2}{\|x\|_2^2},$$

where $\text{Card}(x)$ is the *cardinality* of the vector x , defined as the number of non-zero elements in x . For which vectors x is the upper bound attained?

$$f(x) = x_m - \frac{1}{n} \sum_{i=1}^n x_i.$$

Linear functions 1. For a n -vector x , with $n = 2m - 1$ odd, we define the median of x as x_m . Now consider the function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, with values

Express f as a scalar product, that is, find $a \in \mathbf{R}^n$ such that $f(x) = a^T x$ for every x . Find a basis for the set of points x such that $f(x) = 0$.

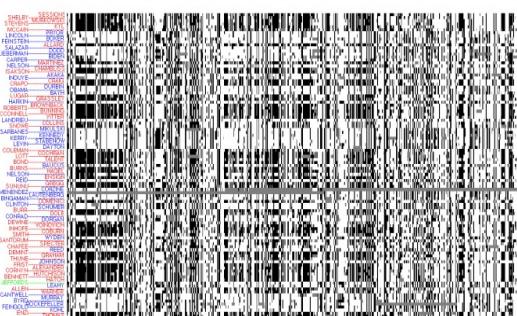
2. For $\alpha \in \mathbf{R}^2$, we consider the “power law” function $f : \mathbf{R}_{++}^2 \rightarrow \mathbf{R}$, with values

$$f(x) = x_1^{\alpha_1} x_2^{\alpha_2}.$$

Justify the statement: “the coefficients α_i provide the ratio between the relative error in f to a relative error in x_i ”.

3. Find the gradient of the function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$ that gives the distance from a given point $p \in \mathbf{R}^2$ to a point $x \in \mathbf{R}^2$.

Matrices



Matrices are collections of vectors of same size, organized in a [rectangular array](#). The image shows the matrix of votes of the 2004-2006 US Senate. ([Source](#))

Via the [matrix-vector product](#), we can interpret matrices as [linear maps](#) (vector-valued functions), which act from an “input” space to an “output” space, and preserve addition and scaling of the inputs. Linear maps arise everywhere in engineering, mostly via a process known as [linearization](#) (of non-linear maps). Matrix norms are then useful to measure how the map amplifies or decreases the norm of specific inputs.

We review a number of prominent classes of matrices. [Orthogonal matrices](#) are an important special case, as they generalize the notion of rotation in the ordinary two- or three-dimensional spaces; they preserve (Euclidean) norms and angles. The [QR decomposition](#), which proves useful in solving linear equations and related problems, allows to write any matrix as a two-term product, involving an orthogonal matrix and a triangular matrix.

Basics

Matrices as collections of column vectors

Matrices can be viewed simply as a collection of (column) vectors of same size, that is, as a collection of points in a multi-dimensional space.

Matrices can be described as follows: given n vectors a_1, \dots, a_n in \mathbf{R}^m , we can define the $m \times n$ matrix A with a_j 's as columns:

$$A = \begin{pmatrix} a_1 & \dots & a_n \end{pmatrix}.$$

Geometrically, A represents n points in a m -dimensional space. The notation $\mathbf{R}^{m \times n}$ denotes the set of $m \times n$ matrices.

With our convention, a column vector in \mathbf{R}^n is thus a matrix in $\mathbf{R}^{n \times 1}$, while a row vector in \mathbf{R}^n is a matrix in $\mathbf{R}^{1 \times n}$.

Transpose

The notation A_{ij} denotes the element of A sitting in row i and column j . The *transpose* of a matrix A , denoted by A^T , is the matrix with (i, j) element $A_{ji}, i = 1, \dots, m, j = 1, \dots, n$.

Matrices as collections of rows

Similarly, we can describe a matrix in row-wise fashion: given m vectors b_1, \dots, b_m in \mathbf{R}^n , we can define the $m \times n$ matrix B with the transposed vectors b_i^T as rows:

$$B = \begin{pmatrix} b_1^T \\ \vdots \\ b_m^T \end{pmatrix}.$$

Geometrically, B represents m points in a n -dimensional space.

Matlab syntax:

```
>> A = [1 2 3; 4 5 6]; % a 2x3 matrix
>> B = A'; % this is the transpose of matrix A
>> B = [1 4; 2 5; 3 6]; % B can also be declared that way
>> C = rand(4,5); % a random 4x5 matrix
```

Examples:

- [A simple \$3 \times 2\$ matrix.](#)
- [Arc-node incidence matrix of a network.](#)
- [Matrix of votes in the US Senate, 2004-2006.](#)

Sparse Matrices

In many applications, one has to deal with very large matrices that are *sparse*, that is, they have many zeros. It often makes sense to use a *sparse storage convention* to represent the matrix.

One of the most common formats involves only listing the non-zero elements, and their associated locations (i, j) in the matrix. In Matlab, the function `sparse` allows to create a sparse matrix based on that information. The user needs also to specify the size of the matrix (it cannot infer it from just the above information).

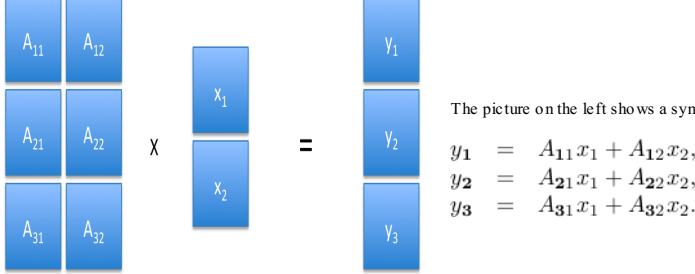
Matlab syntax:

```
>> S = sparse([3 2 3 4 1], [1 2 2 3 4], [1 2 3 4 5], 4, 4) % creates a 4x4 matrix
>> S = [0 0 0 5; 0 2 0 0; 1 3 0 0; 0 0 4 0]; % the same matrix with ordinary convention
```

Matrix Products

Matrix-vector product definition. We define the *matrix-vector product* between a $m \times n$ matrix and a n -vector x , and denote by Ax , the m -vector with i -th component

$$(Ax)_i = \sum_{j=1}^n A_{ij} x_j, \quad i = 1, \dots, m.$$



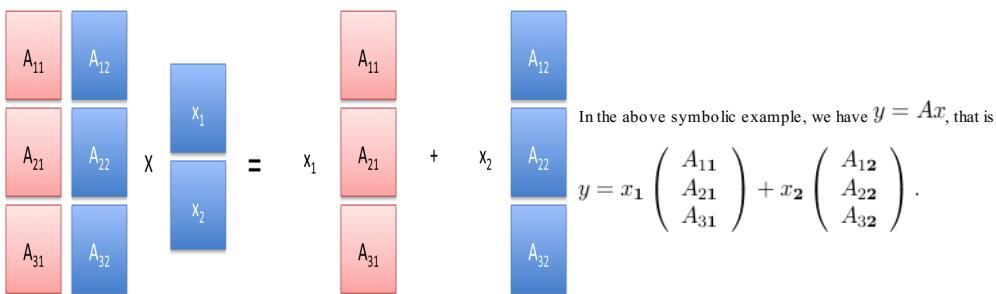
The picture on the left shows a symbolic example with $n = 2$ and $m = 3$. We have $y = Ax$, that is:

$$\begin{aligned} y_1 &= A_{11}x_1 + A_{12}x_2, \\ y_2 &= A_{21}x_1 + A_{22}x_2, \\ y_3 &= A_{31}x_1 + A_{32}x_2. \end{aligned}$$

Interpretation as linear combinations of columns. If the columns of A are given by the vectors $a_i, i = 1, \dots, n$, so that $A = (a_1, \dots, a_n)$, then Ax can be interpreted as a linear combination

$$Ax = \sum_{i=1}^n x_i a_i.$$

of these columns, with weights given by the vector x :



In the above symbolic example, we have $y = Ax$, that is:

$$y = x_1 \begin{pmatrix} A_{11} \\ A_{21} \\ A_{31} \end{pmatrix} + x_2 \begin{pmatrix} A_{12} \\ A_{22} \\ A_{32} \end{pmatrix}.$$

Examples:

[Network flow](#).

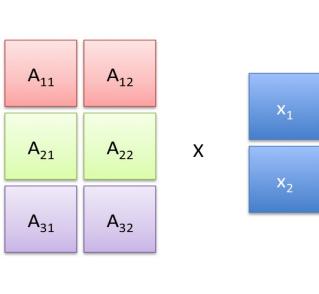
[Image compression](#).

Interpretation as scalar products with rows. Alternatively, if the rows of A are the row vectors $a_i^T, i = 1, \dots, m$:

$$A = \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix},$$

then Ax is the vector with elements $a_i^T x, i = 1, \dots, m$:

$$Ax = \begin{pmatrix} a_1^T x \\ \vdots \\ a_m^T x \end{pmatrix}.$$



In this symbolic example, we have $y = Ax$, that is:

$$\begin{aligned} y_1 &= \begin{pmatrix} A_{11} \\ A_{12} \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = A_{11}x_1 + A_{12}x_2, \\ y_2 &= \begin{pmatrix} A_{21} \\ A_{22} \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = A_{21}x_1 + A_{22}x_2, \\ y_3 &= \begin{pmatrix} A_{31} \\ A_{32} \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = A_{31}x_1 + A_{32}x_2. \end{aligned}$$

Example: [Absorption spectrometry: using measurements at different light frequencies](#).

Left product. If $z \in \mathbf{R}^m$, then the notation $z^T A$ is the row vector of size n equal to the transpose of the column vector $A^T z \in \mathbf{R}^n$. That is:

Example: Return to the [network example](#), involving a $m \times n$ incidence matrix. We note that, by construction, the columns of A sum to zero, which can be compactly written as $\mathbf{1}^T A = 0$, or $A^T \mathbf{1} = 0$.

Matlab syntax

The product operator in Matlab is `*`. If the sizes are not consistent, Matlab will produce an error.

Matlab syntax:

```
>> A = [1 2; 3 4; 5 6]; % 3x2 matrix
>> x = [-1; 1]; % 2x1 vector
>> y = A*x; % result is a 3x1 vector
>> z = [-1; 0; 1]; % 3x1 vector
>> y = z'*A; % result is a 1x2 (i.e., row) vector
```

Matrix-matrix productDefinition. We can extend matrix-vector product to matrix-matrix product, as follows. If $A \in \mathbf{R}^{m \times n}$ and $B \in \mathbf{R}^{n \times p}$, the notation AB denotes the $m \times p$ matrix with i, j

$$(AB)_{ij} = \sum_{k=1}^n A_{ik}B_{kj}.$$

element given by

Transposing a product changes the order, so that $(AB)^T = B^T A^T$.

Column-wise interpretation. If the columns of B are given by the vectors $b_i, i = 1, \dots, n$, so that $B = [b_1, \dots, b_n]$ then AB can be written as $AB = A(b_1 \dots b_n) = (Ab_1 \dots Ab_n)$.

In other words, AB results from transforming each column b_i of B into Ab_i .

Row-wise interpretation

The matrix-matrix product can also be interpreted as an operation on the rows of A . Indeed, if A is given by its rows $a_i^T, i = 1, \dots, m$, then AB is the matrix obtained by transforming each one of these rows via B , into $a_i^T B, i = 1, \dots, m$:

$$AB = \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix} B = \begin{pmatrix} a_1^T B \\ \vdots \\ a_m^T B \end{pmatrix}$$

(Note that $a_i^T B$ s are indeed row vectors, according to our matrix-vector rules.)

Block Matrix Products

Matrix algebra generalizes to blocks, provided block sizes are consistent. To illustrate this, consider the matrix-vector product between a $m \times n$ matrix A and a n -vector x , where A, x are partitioned in blocks, as follows:

$$A = \begin{pmatrix} A_1 & A_2 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

where A_i is $m \times n_i, x_i \in \mathbf{R}^{n_i}, i = 1, 2, n_1 + n_2 = n$. Then

$$Ax = A_1 x_1 + A_2 x_2.$$

Symbolically, it's as if we would form the "scalar" product between the "row vector" (A_1, A_2) and the column vector (x_1, x_2) :

Likewise, if a $n \times p$ matrix B is partitioned into two blocks B_i , each of size n_i , $i = 1, 2$, with $n_1 + n_2 = n$, then

$$AB = \begin{pmatrix} A_1 & A_2 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = A_1 B_1 + A_2 B_2.$$

Again, symbolically we apply the same rules as for the scalar product — except that now the result is a matrix.

Example: Gram matrix.

Finally, we can consider so-called *outer* products. Consider the case for example when A is a $m \times n$ matrix partitioned row-wise into two blocks A_1, A_2 , and B is a $n \times p$ matrix that is partitioned column-wise into two blocks B_1, B_2 :

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}, \quad B = \begin{pmatrix} B_1 & B_2 \end{pmatrix}.$$

Then the product $C = AB$ can be expressed in terms of the blocks, as follows:

$$C = AB = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \begin{pmatrix} B_1 & B_2 \end{pmatrix} = \begin{pmatrix} A_1 B_1 & A_1 B_2 \\ A_2 B_1 & A_2 B_2 \end{pmatrix}.$$

The *trace* of a square $n \times n$ matrix A , denoted by $\text{Tr } A$, is the sum of its diagonal elements: $\text{Tr } A = \sum_{i=1}^n A_{ii}$.

Some important properties:

- *Trace of transpose*: The trace of a square matrix is equal to that of its transpose.
- *Commutativity under trace*: for any two matrices $A \in \mathbf{R}^{m \times n}$ and $B \in \mathbf{R}^{n \times m}$, we have $\text{Tr}(AB) = \text{Tr}(BA)$.

Matlab syntax

```
>> A = [1 2 3; 4 5 6; 7 8 9]; % 3x3 matrix
>> tr = trace(A); % trace of A
```

Scalar product between matrices

We can define the scalar product between two $m \times n$ matrices A, B via

$$\langle A, B \rangle := \text{Tr}(A^T B) = \sum_{i=1}^m \sum_{j=1}^n A_{ij} B_{ij}$$

The above definition is symmetric: we have

$$\langle A, B \rangle = \text{Tr}(A^T B) = \text{Tr}(A^T B)^T = \text{Tr}(B^T A) = \langle B, A \rangle.$$

Our notation is consistent with the definition of the [scalar product between two vectors](#), where we simply view a vector in \mathbf{R}^n as a matrix in $\mathbf{R}^{n \times 1}$. We can interpret the matrix scalar product as the vector scalar product between two long vectors of length mn each, obtained by stacking all the columns of A, B on top of each other.

Matlab syntax

```
>> A = [1 2; 3 4; 5 6]; % 3x2 matrix
>> B = randn(3,2); % random 3x2 matrix
>> scal_prod = trace(A'*B); % scalar product between A and B
>> scal_prod = A(:)'*B(:); % this is the same as the scalar product between the
                           % vectorized forms of A, B
```

Special Matrices [Matrices](#) > [Basics](#) | [Matrix products](#) | Special matrices | [QR](#) | [Matrix inverses](#) | [Linear maps](#) | [Matrix norms](#) | [Applications](#)

- Square Matrices
 - Identity and diagonal matrices
 - Triangular matrices
 - Symmetric matrices
 - Orthogonal Matrices
- Dyads

Some special square matrices

Square matrices are matrices that have the same number of rows as columns. The following are important instances of square matrices.

Identity matrix

The $n \times n$ identity matrix (often denoted I_n , or simply I , if context allows), has ones on its diagonal and zeros elsewhere. It is square, diagonal and symmetric. This matrix satisfies $A \cdot I_n = A$ for every matrix A with n columns, and $I_n \cdot B = B$ for every matrix B with n rows.

Matlab syntax

```
>> I3 = eye(3); % the 3x3 identity matrix
>> A = eye(3,4); % a 3x4 matrix having the 3x3 identity in its first 3 columns
```

Diagonal matrices

Diagonal matrices are square matrices A with $A_{ij} = 0$ when $i \neq j$. A diagonal $n \times n$ matrix A can be denoted as $A = \text{diag}(a)$, with $a \in \mathbf{R}^n$ the vector containing the elements on the diagonal. We can also write

$$A = \begin{pmatrix} a_1 & & \\ & \ddots & \\ & & a_r \end{pmatrix},$$

where by convention the zeros outside the diagonal are not written.

Matlab syntax

```
>> A = diag([1 2 3]); % a diagonal matrix with 1,2,3 on the diagonal
>> A = spdiags([1 2 3]',0,3,3); % the same matrix declared as a sparse matrix
```

Symmetric matrices

Symmetric matrices are square matrices that satisfy $A_{ij} = A_{ji}$ for every pair (i, j) . An entire [section](#) is devoted to symmetric matrices.

Triangular matrices

A square matrix $A \in \mathbf{R}^{m \times n}$ is *upper triangular* if $A_{ij} = 0$ when $i < j$. Here are a few examples:

$$A_1 = \begin{pmatrix} 1 & -1 \\ 0 & 2 \\ 0 & 0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 3 & 8 & 3 \\ 0 & 6 & -1 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 0 & 8 & 3 \\ 0 & 0 & -1 \end{pmatrix}.$$

A matrix is lower triangular if its transpose is upper triangular. For example:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 8 & -9 & 0 \\ 1 & -2 & 3 \end{pmatrix}.$$

Orthogonal (or, *unitary*) matrices are square matrices, such that the columns form an orthonormal basis. If $U = [u_1, \dots, u_n]$ is an orthogonal matrix, then

$$u_i^T u_j = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $U^T U = I_n$. Similarly, $UU^T = I_n$.

Orthogonal matrices correspond to rotations or reflections across a direction: they preserve length and angles. Indeed, for every vector x ,

$$\|Ux\|_2^2 = (Ux)^T(Ux) = x^T U^T U x = x^T x = \|x\|_2^2.$$

Thus, the underlying linear map $x \rightarrow Ux$ preserves the length (measured in Euclidean norm). This is sometimes referred to as the *rotational invariance* of the Euclidean norm.

In addition, angles are preserved: if x, y are two vectors with unit norm, then the angle θ between them satisfies $\cos \theta = x^T y$, while the angle θ' between the rotated vectors $x' = Ux, y' = Uy$ satisfies $\cos \theta' = (x')^T y'$. Since

$$(Ux)^T(Uy) = x^T U^T U y = x^T y,$$

we obtain that the angles are the same. (The converse is true: any square matrix that preserves lengths and angles is orthogonal.)

Geometrically, orthogonal matrices correspond to rotations (around a point) or reflections (around a line passing through the origin).

Examples:

- [A \$2 \times 2\$ orthogonal matrix](#).
- [Permutation matrices](#).

Dyads

Dyads are a special class of matrices, also called rank-one matrices, for reasons seen later.

Definition

A matrix $A \in \mathbf{R}^{m \times n}$ is a *dyad* if it is of the form $A = uv^T$ for some vectors $u \in \mathbf{R}^m, v \in \mathbf{R}^n$. The dyad acts on an input vector $x \in \mathbf{R}^n$ as follows:

$$Ax = (uv^T)x = (v^T x)u.$$

In terms of the associated linear map, for a dyad, the output always points in the *same* direction u in output space (\mathbf{R}^m), no matter what the input x is. The output is thus always a simple scaled version of u . The amount of scaling depends on the vector v , via the linear function $x \rightarrow v^T x$.

Example: [Single-factor models of financial data](#)

Normalized dyads

We can always *normalize* the dyad, by assuming that both u, v are of unit (Euclidean) norm, and using a factor to capture their scale. That is, any dyad can be written in *normalized* form:

$$A = uv^T = (\|u\|_2 \cdot \|v\|_2) \cdot \left(\frac{u}{\|u\|_2}\right) \left(\frac{v}{\|v\|_2}\right)^T = \sigma \tilde{u} \tilde{v}^T,$$

where $\sigma > 0$, and $\|\tilde{u}\|_2 = \|\tilde{v}\|_2 = 1$.

The QR decomposition of a matrix

Basic idea

The basic goal of the QR decomposition is to *factor* a matrix as a product of two matrices (traditionally called Q, R , hence the name of this factorization). Each matrix has a simple structure which can be further exploited in dealing with, say, [linear equations](#).

The QR decomposition is nothing else than the [Gram-Schmidt procedure](#) applied to the columns of the matrix, and with the result expressed in matrix form. Consider a $m \times n$ matrix $A = (a_1, \dots, a_n)$, with each $a_i \in \mathbf{R}^m$ a column of A .

Case when A is full column rank

Assume first that the a_i 's (the columns of A) are linearly independent. Each step of the G-S procedure can be written as

$$a_i = (a_i^T q_1) q_1 + \dots + (a_i^T q_{i-1}) q_{i-1} + \|\tilde{q}_i\|_2 q_i, \quad i = 1, \dots, n.$$

We write this as

$$a_i = \sum_{j=1}^i r_{ji} q_j, \quad i = 1, \dots, n$$

where $r_{ji} = a_i^T q_j$, ($1 \leq j \leq i-1$) $r_{ij} = (a_i^T q_j)$, ($1 \leq j \leq i-1$) and $r_{ii} = \|\tilde{q}_i\|_2$.

Since the q_j 's are unit-length and normalized, the matrix $Q = (q_1, \dots, q_n)$ satisfies $Q^T Q = I_n$. The QR decomposition of a $m \times n$ matrix A thus allows to write the matrix in *factored* form:

$$A = QR, \quad Q = \begin{pmatrix} q_1 & \dots & q_n \end{pmatrix}, \quad R = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ 0 & r_{22} & & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & & 0 & r_{nn} \end{pmatrix}$$

where Q is a $m \times n$ matrix with $Q^T Q = I_n$, and R is $n \times n$, upper-triangular.

Matlab syntax

```
>> [Q,R] = qr(A,0); % A is a mxn matrix, Q is mxn orthogonal, R is nxn upper triangular
```

Example: [QR decomposition of a 6x4 matrix](#).

Case when the columns are not independent

When the columns of A are not independent, at some step of the G-S procedure we encounter a zero vector \tilde{q}_j , which means a_j is a linear combination of a_{j-1}, \dots, a_1 . The [modified Gram-Schmidt procedure](#) then simply skips to the next vector and continues.

In matrix form, we obtain $A = QR$, with $Q \in \mathbf{R}^{m \times r}$, $r = \text{Rank}(A)$, and R has an upper staircase form, for example:

$$R = \begin{pmatrix} * & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{pmatrix}.$$

(This is simply an upper triangular matrix with some rows deleted. It is still upper triangular.)

We can permute the columns of R to bring forward the first non-zero elements in each row:

$$R = \begin{pmatrix} R_1 & R_2 \end{pmatrix} P^T, \quad \left(\begin{array}{c|c} R_1 & R_2 \end{array} \right) := \left(\begin{array}{ccc|ccc} * & * & * & * & * & * \\ 0 & * & 0 & * & * & * \\ 0 & 0 & * & 0 & 0 & * \end{array} \right),$$

where P is a [permutation matrix](#) (that is, its columns are the unit vectors in some order), whose effect is to permute columns. (Since P is orthogonal, $P^{-1} = P^T$.) Now, R_1 is square, upper triangular, and *invertible*, since none of its diagonal elements is zero.

The QR decomposition can be written

$$AP = Q \begin{pmatrix} R_1 & R_2 \end{pmatrix},$$

where

1. $Q \in \mathbf{R}^{m \times r}$, $Q^T Q = I_r$;

2. r is the *rank* of A ;

3. R_1 is $r \times r$ upper triangular, invertible matrix;

4. R_2 is a $r \times (n - r)$ matrix;

5. P is a $m \times m$ permutation matrix.

Matlab syntax

```
>> [Q,R,inds] = qr(A,0); % here inds is a permutation vector such that A(:,inds) = Q*R
```

Full QR decomposition

The full QR decomposition allows to write $A = QR$ where $Q \in \mathbf{R}^{m \times m}$ is *square* and *orthogonal* ($Q^T Q = QQ^T = I_m$). In other words, the columns of Q are an orthonormal basis for the whole output space \mathbf{R}^m , not just for the range of A .

We obtain the full decomposition by appending an $m \times m$ identity matrix to the columns of A : $A \rightarrow [A, I_m]$. The QR decomposition of the augmented matrix allows to write

$$AP = QR = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix},$$

where the columns of the $m \times m$ matrix $Q = [Q_1, Q_2]$ are orthogonal, and R_1 is upper triangular and invertible. (As before, P is a permutation matrix.) In the G-S procedure, the columns of Q_1 are obtained from those of A , while the columns of Q_2 come from the extra columns added to A .

The full QR decomposition reveals the rank of A : we simply look at the elements on the diagonal of R that are not zero, that is, the size of R_1 .

Matlab syntax

```
>> [Q,R] = qr(A); % A is a mxn matrix, Q is mxm orthogonal, R is mxn upper triangular
```

Example: [QR decomposition of a 6x4 matrix](#).

Matrix Inverses [Matrices](#) > [Basics](#) | [Matrix products](#) | [Special matrices](#) | [QR](#) | Matrix inverses | [Linear maps](#) | [Matrix norms](#) | [Applications](#)

- Square full-rank matrices and their inverse
- Full column rank matrices and left inverses
- Full row rank matrices and right inverses

Square full-rank matrices and their inverse

A square $n \times n$ matrix is said to be *invertible* if and only if its columns are independent. This is equivalent to the fact that its rows are independent as well. An equivalent definition states that a matrix is invertible if and only if its [determinant](#) is non-zero.

For invertible $n \times n$ matrices A , there exist a unique matrix B such that $AB = BA = I_n$. The matrix B is denoted A^{-1} and is called the *inverse* of A .

Example: [a simple 2 × 2 matrix](#).

If a matrix R is square, invertible, and *triangular*, we can compute its inverse R^{-1} simply, as follows. We solve n linear equations of the form $Rx_i = e_i$, $i = 1, \dots, n$, with e_i the i -th column of the $n \times n$ identity matrix, using a process known as [backwards substitution](#). Here is an [example](#). At the outset we form the matrix $R^{-1} = [x_1, \dots, x_n]$. By construction, $R \cdot R^{-1} = I_n$.

For general square, invertible matrices A , the QR decomposition allows to compute the inverse. For such matrices, the QR decomposition is of the form $A = QR$, with Q a $n \times n$ orthogonal matrix, and R is upper triangular. Then the inverse is $A^{-1} = R^{-1}Q^T$.

Matlab syntax

```
Ainv = inv(A); % produces the inverse of a square, invertible matrix
```

A useful property is the expression of the inverse of a product of two square, invertible matrices A, B : $(AB)^{-1} = B^{-1}A^{-1}$. (Indeed, you can check that this inverse works.)

Full column rank matrices and left inverses

A $m \times n$ matrix is said to be *full column rank* if its columns are independent. This necessarily implies $m \geq n$.

A matrix A has full column rank if and only if there exist a $n \times m$ matrix B such that $BA = I_n$ (here $n \leq m$ is the small dimension). We say that B is a *left-inverse* of A . To find one left inverse of a matrix with independent columns A , we use the full QR decomposition of A to write

$$A = Q \begin{pmatrix} R_1 \\ 0 \end{pmatrix},$$

where R_1 is $n \times n$ upper triangular and invertible, while Q is $m \times m$ and orthogonal ($Q^T Q = I_m$). We can then set a left inverse B to be

$$B = \begin{pmatrix} R_1^{-1} & 0 \end{pmatrix} Q^T.$$

The particular choice above can be expressed in terms of A directly:

$$B = (A^T A)^{-1} A^T.$$

Note that $A^T A$ is invertible, as it is equal to $R_1^T R_1$.

In general, left inverses are not unique.

Full row rank matrices and right inverses

A $m \times n$ matrix is said to be *full row rank* if its rows are independent. This necessarily implies $m \leq n$.

A matrix A has full row rank if and only if there exist a $n \times m$ matrix B such that $AB = I_m$ (here $m \leq n$ is the small dimension). We say that B is a *right-inverse* of A . We can derive expressions of right inverses by noting that A is full row rank if and only if A^T is full column rank. In particular, for a matrix with independent rows, the full QR decomposition (of A^T) allows to write

$$A = \begin{pmatrix} R_1^T & 0 \end{pmatrix} Q^T,$$

where R_1 is $m \times m$ upper triangular and invertible, while Q is $n \times n$ and orthogonal ($Q^T Q = I_n$). We can then set a right inverse of A to be

$$B = Q \begin{pmatrix} R_1^{-1} \\ 0 \end{pmatrix}.$$

The particular choice above can be expressed in terms of A directly:

$$B = A^T (AA^T)^{-1}.$$

Note that AA^T is invertible, as it is equal to $R_1^T R_1$.

In general, right inverses are not unique.

Linear Maps [Matrices](#) > [Basics](#) | [Matrix products](#) | [Special matrices](#) | [QR](#) | [Matrix inverses](#) | Linear maps | [Matrix norms](#) | [Applications](#)

- Definitions and interpretation
- First-order approximation of non-linear maps

Definition and Interpretation Definition. A map $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ is linear (resp. affine) if and only if every one of its components is. The formal definition we saw [here](#) for functions applies verbatim to maps.

To an $m \times n$ matrix A , we can associate a linear map $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$, with values $f(x) = Ax$. Conversely, to any linear map, we can uniquely associate a matrix A which satisfies $f(x) = Ax$ for every x .

Indeed, if the components of f , $f_i, i = 1, \dots, m$, are linear, then they can be expressed as $f_i(x) = a_i^T x$ for some $a_i \in \mathbf{R}^n$. The matrix A is the matrix that has a_i^T as its i -th row:

$$f(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{pmatrix} = \begin{pmatrix} a_1^T x \\ \vdots \\ a_m^T x \end{pmatrix} = Ax, \quad \text{with } A := \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix} \in \mathbf{R}^{m \times n}.$$

Hence, there is a one-to-one correspondence between matrices and linear maps. This is extending what we saw for vectors, which are in one-to-one correspondence with linear functions.

This is summarized as follows.

Representation of affine maps via the matrix-vector product. A map $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ is affine if and only if it can be expressed via a matrix-vector product:

$$f(x) = Ax + b,$$

for some unique pair (A, b) , with $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$. The function is linear if and only if $b = 0$. \diamond

The result above shows that a matrix can be seen as a (linear) map from the “input” space \mathbf{R}^n to the “output” space \mathbf{R}^m . Both points of view (matrices as simple collections of vectors, or as linear maps) are useful.

Interpretations

Consider an affine map $x \rightarrow y = Ax + b$. An element A_{ij} gives the coefficient of influence of x_j over y_i . In this sense, if $A_{13} >> A_{14}$ we can say that x_3 has much more influence on y_1 than x_4 . Or, $A_{24} = 0$ says that y_2 does not depend at all on x_4 . Often the constant term $b = f(0)$ is referred to as the “bias” vector.

First-order approximation of non-linear maps

Since maps are just collections of functions, we can approximate a map with a linear (or affine) map, just as we did with functions [here](#).

If $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ is differentiable, then we can approximate the (vector) values of f near a given point $x_0 \in \mathbf{R}^n$ by an affine map \tilde{f} :

$$f(x) \approx \tilde{f}(x) := f(x_0) + A(x - x_0),$$

where $A_{ij} = \frac{\partial f_i}{\partial x_j}(x_0)$ is the derivative of the i -th component of f with respect to x_j . (A is referred to as the *Jacobian matrix* of f at x_0 .)

Examples:

- [Navigation by range measurement](#).
- [State-space models of dynamical systems](#).

Matrix Norms [Matrices](#) > [Basics](#) | [Matrix products](#) | [Special matrices](#) | [QR](#) | [Matrix inverses](#) | [Linear maps](#) | Matrix norms | [Applications](#)

- Motivating example
- RMS gain: the Frobenius norm
- Peak gain: the largest singular value norm
- Applications

Motivating example: effect of noise in a linear system

We saw how a matrix (say, $A \in \mathbf{R}^{m \times n}$) induces, via the matrix-vector product, a linear map $x \rightarrow Ax$. Here, x is an input vector and $y = Ax$ is the output. The mapping (that is, A) could represent a linear amplifier with input an audio signal x and output another audio signal y .

Now, assume that there is some noise in the vector x : the actual input is $x + v$, where $v \in \mathbf{R}^n$ is an error vector. This implies that there will be noise in the output as well: the noisy output is $A(x + v)$, so that the error on the output due to noise is Av . How could we quantify the effect of input noise on the output noise?

One approach is to try to measure the norm of the error vector, $\|Av\|$. Obviously, this norm depends on the noise v , which we do not know. So we will assume that v can take values in a set. We need to come up with a single number that captures in some way the different values of $\|Av\|$ when v spans that set. Since scaling v simply scales the norm $\|Av\|$ accordingly, we will restrict the vectors v

to have a certain norm, say $\|v\| = 1$.

Clearly, depending on the choice of the set, the norms we use to measure norm lengths, and how we choose to capture many numbers $\|Av\|$ with one, etc, we will obtain different numbers.

RMS gain: the Frobenius norm

Let us first assume that the noise vector v can take a finite set of directions, specifically the directions represented by the standard basis, e_1, \dots, e_n . Then let us look at the *average* of the squared error norm:

$$\frac{1}{n} \sum_{i=1}^n \|Ae_i\|_2^2 = \frac{1}{n} \sum_{i=1}^n \|a_i\|_2^2,$$

where a_i stands for the i -th column of A . The quantity above can be written as $\frac{1}{n} \|A\|_F^2$, where

$$\|A\|_F := \sqrt{\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2} = \sqrt{\text{Tr}(A^T A)}$$

is the *Frobenius norm* of A .

The function $A \rightarrow \|A\|_F$ turns out to satisfy the [basic conditions of a norm](#) in the matrix space $\mathbf{R}^{m \times n}$. In fact, it is the Euclidean norm of the vector of length nm formed with all the coefficients of A . Further, the quantity would remain the same if we had chosen any orthonormal basis other than the standard one.

The Frobenius norm is useful to measure the RMS (root-mean-square) gain of the matrix, its average response along given mutually orthogonal directions in space. Clearly, this approach does not capture well the variance of the error, only the average effect of noise.

The computation of the Frobenius norm is very easy: it requires about nm flops.

Matlab syntax

```
>> frob_norm = norm(A, 'fro');
```

Peak gain: the largest singular value norm

To try to capture the variance of the output noise, we may take a worst-case approach.

Let us assume that the noise vector is bounded but otherwise unknown. Specifically, all we know about v is that $\|v\|_2 \leq \alpha$, where α is the maximum amount of noise (measured in Euclidean norm). What is then the worst-case (peak) value of the norm of the output noise? This is answered by the optimization problem

$$\max_v \|Av\|_2 : \|v\|_2 \leq \alpha.$$

The quantity

$$\|A\|_{\text{LSV}} := \max_v \|Av\|_2 : \|v\|_2 \leq 1$$

measures the *peak gain* of the mapping A , in the sense that if the noise vector is bounded in norm by α , then the output noise is bounded in norm by $\alpha \|A\|$. Any vector v which achieves the maximum above corresponds to a direction in input space that is maximally amplified by the mapping A .

The quantity $\|A\|_{\text{LSV}}$ is indeed a matrix [norm](#), called the *largest singular value* (LSV) norm, for reasons seen [here](#). It is perhaps the most popular matrix norm.

The computation of the largest singular value norm of a matrix is not as easy as with the Frobenius norm. However, it can be computed with linear algebra methods seen [here](#), in about $\min(n, m)nm$ flops. While it is more expensive to compute than the Frobenius norm, it is also more useful because it goes beyond capturing the average response to noise.

Matlab syntax

```
>> lsv_norm = norm(A);
```

Other norms

Many other matrix norms are possible, and sometimes useful. In particular, we can generalize the notion of peak norm by using different norms to measure vector size in the input and output spaces. For example, the quantity

$$\|A\|_{\infty,1} := \max_v \|Av\|_1 : \|v\|_\infty \leq 1$$

measures the peak gain with inputs bounded in maximum norm, and outputs measured with the l_1 -norm.

The norms we have just introduced, the Frobenius and largest singular value norms, are the most popular ones, and are easy to compute. Many other norms are hard to compute.

ApplicationsDistance between matrices. Matrix norms are ways to measure the size of a matrix. This allows to quantify the difference between matrices.

Assume for example that we are trying to estimate a matrix A , and came up with an estimate \hat{A} . How can we measure the quality of our estimate? One way is to evaluate by how much they differ when they act on the standard basis. This leads to the Frobenius norm.

Another way is to look at the difference in the output:

$$\|Av - \hat{A}v\|_2$$

when v runs the whole space. Clearly, we need to scale, or limit the size, of v , otherwise the difference above may be arbitrarily big. Let's look at the worst-case difference when v satisfies $\|v\|_2 \leq 1$. We obtain

$$\max_v \|Av - \hat{A}v\|_2 : \|v\|_2 \leq 1,$$

which is the largest singular value norm of the difference $A - \hat{A}$.

Direction of maximal variance. Consider a data set described as a collection of vectors a_1, \dots, a_n , with $a_i \in \mathbf{R}^m$. We can gather this data set in a single matrix $A = [a_1, \dots, a_n] \in \mathbf{R}^{m \times n}$

$$\hat{a} := \frac{1}{n} \sum_{i=1}^n a_i = 0.$$

. For simplicity, let us assume that the [average vector](#) is zero:

Let us try to visualize the data set by projecting it on a single line passing through the origin. The line is thus defined by a vector $x \in \mathbf{R}^m$, which we can without loss of generality assume to be of Euclidean norm 1. The data points, when projected on the line, are turned into real numbers $x^T a_i, i = 1, \dots, n$.

It can be argued that a good line to project data on is one which spreads the numbers $x^T a_i$ as much as possible. (If all the data points are projected to numbers that are very close, we will not see anything, as all data points will collapse to close locations.)

We can find a direction in space which accomplishes this, as follows. The average of the numbers is

$$\frac{1}{n} \sum_{i=1}^n a_i^T x = \left(\frac{1}{n} \sum_{i=1}^n a_i \right)^T x = \hat{a}^T x = 0,$$

while their variance is

$$\frac{1}{n} \sum_{i=1}^n (a_i^T x - \hat{a}^T x)^2 = \frac{1}{n} \sum_{i=1}^n (a_i^T x)^2 = \frac{1}{n} x^T A A^T x = \frac{1}{n} \|A^T x\|_2^2$$

The direction of maximal variance is found by computing the LSV norm of A^T

$$\max_v \{\|A^T v\|_2 : \|v\|_2 \leq 1\} = \|A^T\|_{\text{LSV}}.$$

(It turns out that this quantity is the same as the LSV norm of A itself.)

Applications [Matrices](#) > [Basics](#) | [Matrix products](#) | [Special matrices](#) | [QR](#) | [Matrix inverses](#) | [Linear maps](#) | [Matrix norms](#) | Applications

- [State-space models of linear dynamical systems](#)

Exercises [Matrices](#) > Exercises Matrix products A. Let $f : \mathbf{R}^m \rightarrow \mathbf{R}^k$ and $g : \mathbf{R}^n \rightarrow \mathbf{R}^m$ be two maps. Let $h : \mathbf{R}^n \rightarrow \mathbf{R}^k$ be the composite map $h = f \circ g$, with values $h(x) = f(g(x))$ for $x \in \mathbf{R}^n$. Show that the derivatives of h can be expressed via a matrix-matrix product, as $J_h(x) = J_f(g(x)) \cdot J_g(x)$, where the [Jacobian matrix](#) of h at x is defined as the matrix $J_h(x)$ with (i, j) element $\partial h_i / \partial x_j(x)$.

Special matrices

B. A matrix $P \in \mathbf{R}^{n \times n}$ is a [permutation matrix](#) if it is a permutation of the columns of the $n \times n$ identity matrix.

1. For a $n \times n$ matrix A , we consider the products PA and AP . Describe in simple terms what these matrices look like with respect to the original matrix A .
2. Show that P is orthogonal.
3. Show that P is invertible, and that $P^{-1} = P^T$.

Linear maps, dynamical systems C. Let $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ be a linear map. Show how to compute the (unique) matrix A such that $f(x) = Ax$ for every $x \in \mathbf{R}^n$, in terms of the values of f at appropriate vectors, which you will determine.

D. Consider a discrete-time linear dynamical system (for background, see [here](#)) with state $x \in \mathbf{R}^n$, input vector $u \in \mathbf{R}^p$, and output vector $y \in \mathbf{R}^k$, that is described by the linear equations $x(t+1) = Ax(t) + Bu(t)$, $y(t) = Cx(t)$,

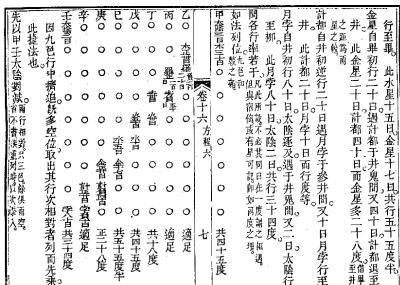
with $A \in \mathbf{R}^{n \times n}$, $B \in \mathbf{R}^{n \times p}$, and $C \in \mathbf{R}^{k \times n}$ given matrices.

1. Assuming that the system has initial condition $x(0) = 0$, express the output vector at time T as a linear function of $u(0), \dots, u(T)$; that is, determine a matrix H such that $y(T) = H\bar{u}(T)$, where $\bar{u}(T) := (u(0), \dots, u(T-1))$ is a vector containing all the inputs up to and including at time $T-1$.
2. What is the interpretation of the range of H ?

Matrix inverses, norms E. Show that a square matrix is invertible if and only if its [determinant](#) is non-zero. You can use the fact that the determinant of a product is a product of the determinant, together with the QR decomposition of the matrix A .

F. Let $A \in \mathbf{R}^{m \times n}$, $B \in \mathbf{R}^{n \times p}$, and let $C := AB \in \mathbf{R}^{m \times p}$. Show that $\|C\| \leq \|A\| \cdot \|B\|$ where $\|\cdot\|$ denotes the [largest singular value norm](#) of its matrix argument.

Linear Equations



Linear equations have been around for thousands of years. The picture on the left shows a 17th century Chinese text that explains the ancient art of *fangcheng* ("rectangular arrays", for more details see [here](#)). Linear equations arise naturally in many areas of engineering, often as simple models of more complicated, non-linear equations. They form the core of linear algebra, and often arise as constraints in optimization problems. They are also an important building block of optimization methods, as many optimization algorithms rely on linear equations.

Linear equations can be expressed as $Ax = y$, where $x \in \mathbf{R}^n$ is the unknown, $y \in \mathbf{R}^m$ is a given vector, and $A \in \mathbf{R}^{m \times n}$ is a matrix. The set of solutions is an [affine set](#). In turn, *any* affine set can be described as the set of solutions to an affine equation.

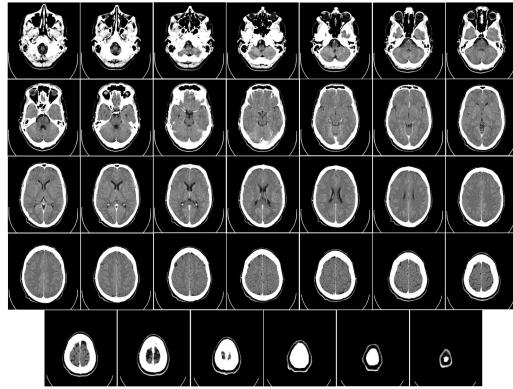
The issue of [existence and unicity](#) of solutions lead to important notions attached to the associated matrix A . The *nullspace*, which contains the input vectors that are crushed to zero by the associated linear map $x \rightarrow Ax$; the *range*, which contains the set of output vectors that is attainable by the linear map; and its dimension, the *rank*. There is a variety of [solution methods](#) for linear equations; we describe how the [QR decomposition](#) of a matrix can be used in this context.

Motivating Example: CAT Scan Imaging Overview

Tomography means reconstruction of an image from its sections. The word comes from the greek “*tomos*” (“slice”) and “*graph*” (“description”). The problem arises in many fields, ranging from astronomy to medical imaging.

Computerized Axial Tomography (CAT) is a medical imaging method that processes large amounts of two-dimensional X-ray images in order to produce a three-dimensional image. The goal is to picture for example the *tissue density* of the different parts of the brain, in order to detect anomalies (such as brain tumors).

Typically, the X-ray images represent “slices” of the part of the body (such as the brain) that is examined. Those slices are indirectly obtained via axial measurements of X-ray attenuation, as explained below. Thus, in CAT for medical imaging, we use axial (line) measurements to get two-dimensional images (slices), and from that scan of images, we may proceed to digitally reconstruct a three-dimensional view. Here, we focus on the process that produces a single two-dimensional image from axial measurements.



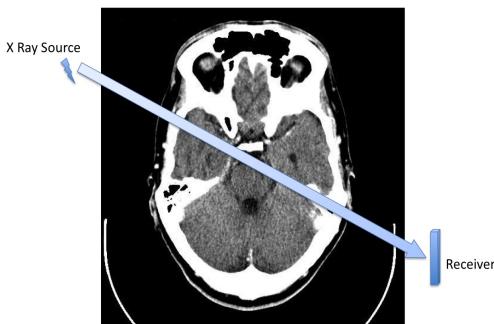
A collection of “slices” of a human brain obtained by CAT scan. The pictures offer an image of the *density* of tissue in the various parts of the brain.

Each slice is actually *reconstructed* image obtained by a tomography technique explained below. The collection of slices can in turn be used to form a full three-dimensional representation of the brain.

Source: [Wikipedia entry](#).

From 1D to 2D: axial tomography

In CAT-based medical imaging, a number of X-rays are sent through the tissues to be examined along different directions, and their intensity after they have traversed the tissues is captured by a camera. For each direction, we record the attenuation of the X-ray, by comparing the intensity of the X-ray at the source, I_{source} , to the intensity after the X-ray has traversed the tissues, at the receiver's end, I_{rec} .



A single slice obtained by CAT scan. Each slice is a *reconstructed* image obtained by recording the attenuation of x-rays through the tissues along a vast number of directions. The picture shows the X-Ray source and the corresponding receiver used to measure attenuation along a specific direction.

Linear equations for a single slice

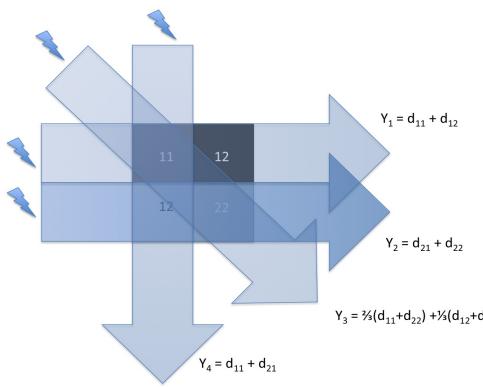
Similar to the [Beer-Lambert law](#) of optics, it turns out that, to a reasonable degree of approximation, the log-ratio of the intensities at the source and at the receiver is linear in the densities of the tissues traversed.

11	12
12	22

We consider a discretized version of a square slice image, with $N \times N$ pixels of gray scale values. On the example pictured here, we simply have $N = 2$, and a total of four pixels. Each pixel can be represented by an index pair (i, j) , with $1 \leq i, j \leq N$. The gray scale values represent the density of the tissues, with d_{ij} the density at pixel (i, j) .

$$y = \log \frac{I_{\text{rec}}}{I_{\text{source}}} = \sum_{(i,j) \in T} A_{ij} d_{ij},$$

With the discretization, the linear relationship between intensities log-ratios and densities can be expressed as $y = \log \frac{I_{\text{rec}}}{I_{\text{source}}} = \sum_{(i,j) \in T} A_{ij} d_{ij}$, where T denotes the indices of pixel areas traversed by the X-ray, d_{ij} the density in the (i, j) area, and A_{ij} the proportion of the area within the pixel that is traversed by the ray.



Linear relationship between the observed log-intensity ratios y_1, y_2, y_3, y_4 and (unobserved) densities $d_{ij}, 1 \leq i, j \leq 2$ within the four pixels. The slanted arrow corresponds to an X ray which traverses about $2/3$ of the area of pixels $(1, 1)$ and $(2, 2)$, and $1/3$ of the areas of the pixels $(1, 2)$ and $(2, 1)$.

Thus, we can relate the vector $\mathbf{d} \in \mathbf{R}^{N^2}$ to the observed intensity log-ratio vector $\mathbf{y} \in \mathbf{R}^m$ in terms of a linear equation $\mathbf{y} = \mathbf{Ad}$, where $\mathbf{A} \in \mathbf{R}^{n \times m}$, with $n := N^2$. Note that depending on the number of pixels used, and the number of measurements, the matrix \mathbf{A} can be quite large. In general, the matrix is *wide*, in the sense that it has (many) more columns than rows ($n \gg m$). Thus, the above system of equations is usually undetermined.

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 2/3 & 1/3 & 1/3 & 2/3 \\ 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} d_{11} \\ d_{12} \\ d_{21} \\ d_{22} \end{pmatrix}.$$

In the example pictured above, we have

Issues

The above example motivates us to address the problems of *solving* linear equations. It also raises the issue of existence (do we have enough measurements to find the densities?) and unicity (if a solution exists, is it unique?).

Existence, Unicity

Consider the linear equation in $\mathbf{x} \in \mathbf{R}^n$: $\mathbf{Ax} = \mathbf{y}$, where $\mathbf{A} \in \mathbf{R}^{m \times n}$ and $\mathbf{y} \in \mathbf{R}^m$ are given, and $\mathbf{x} \in \mathbf{R}^n$ is the variable.

The set of solutions to the above equation, if it is not empty, is an affine subspace. That is, it is of the form $\mathbf{x}_0 + \mathbf{L}$ where \mathbf{L} is a subspace.

We'd like to be able to

- determine if a solution exists;
- if so, determine if it is unique;
- compute a solution \mathbf{x}_0 if one exists;
- find an orthonormal basis of the subspace \mathbf{L} .

Existence: range and rank of a matrixRange. The *range* (or, *image*) of a $m \times n$ matrix \mathbf{A} is defined as the following subset of \mathbf{R}^m : $\mathbf{R}(\mathbf{A}) := \{\mathbf{Ax} : \mathbf{x} \in \mathbf{R}^n\}$. The range describes the vectors $\mathbf{y} = \mathbf{Ax}$ that can be attained in the output space by an arbitrary choice of a vector \mathbf{x} in the input space. The range is simply the span of the columns of \mathbf{A} .

If $\mathbf{y} \notin \mathbf{R}(\mathbf{A})$, we say that the linear equation $\mathbf{Ax} = \mathbf{y}$ is *infeasible*. The set of solutions to the linear equation is empty.

The Matlab function `orth` accepts a matrix \mathbf{A} as input, and returns a matrix, the columns of which span the range of the matrix \mathbf{A} , and are mutually orthogonal. Hence, $\mathbf{U}^T \mathbf{U} = \mathbf{I}_r$, where r is the dimension of the range. One algorithm to obtain the matrix \mathbf{U} is the [Gram-Schmidt procedure](#).

Matlab syntax:

```
>> U = orth(A); % columns of U span the range of A, and U'*U = identity
```

Example: [An infeasible linear system](#).

Rank. The dimension of the range is called the *rank* of the matrix. As we will see later, the rank cannot exceed any one of the dimensions of the matrix \mathbf{A} : $r \leq \min(m, n)$. A matrix is said to be *full rank* if $r = \min(m, n)$.Matlab syntax

```
r = rank(A); % r is the rank of A
```

Note that the rank is a very “brittle” notion, in that small changes in the entries of the matrix can dramatically change its rank. Random matrices, such as ones generated using the Matlab command `rand`, are full rank. We will develop [here](#) a better, more numerically reliable notion.

Examples:

- Range and rank of a simple matrix.
- [Rank-one matrices](#).
- [Rank properties of the arc-node incidence matrix](#).

Full row rank matrices

The matrix \mathbf{A} is said to be *full row rank* (or, *onto*) if the range is the whole output space, \mathbf{R}^m . The name “full row rank” comes from the fact that the rank equals the row dimension of \mathbf{A} . Since the rank is always less than the smallest of the number of columns and rows, a $m \times n$ matrix of full row rank has necessarily less rows than columns (that is, $m \leq n$).

An equivalent condition for \mathbf{A} to be full row rank is that the square, $m \times m$ matrix \mathbf{AA}^T is invertible, meaning that it has full rank, m . [Proof](#).

Unicity: nullspace of a matrixNullspace. The *nullspace* (or, *kernel*) of a $m \times n$ matrix \mathbf{A} is the following subspace of \mathbf{R}^n : $\mathbf{N}(\mathbf{A}) := \{\mathbf{x} \in \mathbf{R}^n : \mathbf{Ax} = 0\}$. The nullspace describes the ambiguity in \mathbf{x} given $\mathbf{y} = \mathbf{Ax}$: any $\mathbf{z} \in \mathbf{N}(\mathbf{A})$ will be such that $\mathbf{A}(\mathbf{x} + \mathbf{z}) = \mathbf{y}$, so \mathbf{x} cannot be determined by the sole knowledge of \mathbf{y} if the nullspace is not reduced to the singleton $\{0\}$.

The Matlab function `null` accepts a matrix \mathbf{A} as input, and returns a matrix, the columns of which span the nullspace of the matrix \mathbf{A} , and are mutually orthogonal. Hence, $\mathbf{U}^T \mathbf{U} = \mathbf{I}_p$, where p is the

dimension of the nullspace.

Matlab syntax

```
U = null(A); % columns of U span the nullspace of A, and U'*U = I
```

Example: The nullspace of the matrix

$$A = \begin{pmatrix} 1 & -1 \\ \alpha & -2 \end{pmatrix}$$

is $\{0\}$ when $\alpha \neq 2$. Otherwise, it is the set

Nullity

The *nullity* of a matrix is the dimension of the nullspace. The *rank-nullity theorem* states that the nullity of a $m \times n$ matrix A is $n - r$, where r is the rank of A .

Full column rank matrices

The matrix A is said to be *full column rank* (or, *one-to-one*) if its nullspace is the singleton $\{0\}$. In this case, if we denote by a_i the n columns of A , the equation

$$(Ax =) \sum_{i=1}^n a_i x_i = 0$$

has $x = 0$ as the unique solution. Hence, A is one-to-one if and only if its columns are independent. Since the rank is always less than the smallest of the number of columns and rows, a $m \times n$ matrix of full column rank has necessarily less columns than rows (that is, $m \geq n$).

The term “one-to-one” comes from the fact that for such matrices, the condition $y = Ax$ uniquely determines x , since $Ax_1 = y$ and $Ax_2 = y$ implies $A(x_1 - x_2) = 0$, so that the solution is unique: $x_1 = x_2$. The name “full column rank” comes from the fact that the rank equals the column dimension of A .

An equivalent condition for A to be full column rank is that the square, $n \times n$ matrix $A^T A$ is invertible, meaning that it has full rank, n . ([Proof](#))

Example: [Nullspace of a transpose incidence matrix](#).

Fundamental facts

Two important results about the nullspace and range of a matrix.

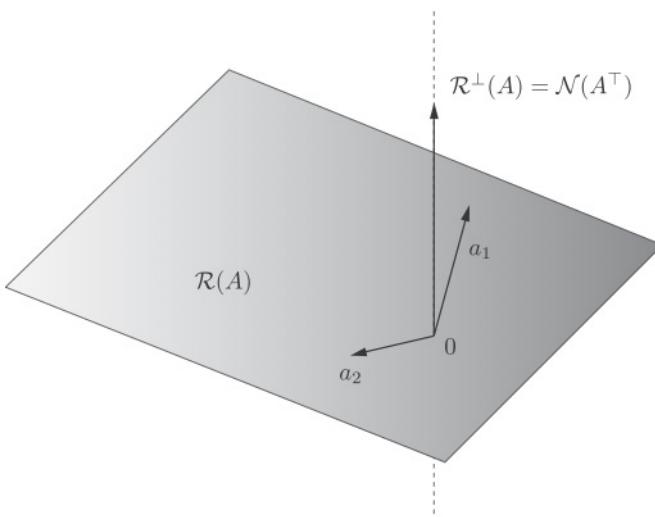
Rank-nullity theorem ([Proof](#))

The nullity (dimension of the nullspace) and the rank (dimension of the range) of a $m \times n$ matrix A add up to the column dimension of A , n .

Another important result involves the definition of the [orthogonal complement](#) of a subspace.

Fundamental theorem of linear algebra ([Proof](#))

The range of a matrix is the orthogonal complement of the nullspace of its transpose. That is, for a $m \times n$ matrix A : $\mathbf{R}(A)^\perp = \mathbf{N}(A^T)$.



The figure provides a sketch of the proof: consider a 3×2 matrix, and denote by $a_i \in \mathbf{R}^3$ ($i = 1, 2$) its rows, so that $A = (a_1 \ a_2)$, $A^T = \begin{pmatrix} a_1^T \\ a_2^T \end{pmatrix}$. Then $A^T x = 0$ if and only if $a_i^T x = 0$, $i = 1, 2$. In words: x is in the nullspace of A^T if and only if it is orthogonal to the vectors a_i , $i = 1, 2$. But those two vectors span the range of A , hence x is orthogonal to any element in the range.

Consider the problem of solving a system of linear equations $Ax = y$, where $A \in \mathbf{R}^{m \times n}$ and $y \in \mathbf{R}^m$ are given.

The basic idea in the solution algorithm starts with the observation that in the special case when A is [upper triangular](#), that is, $A_{ij} = 0$ when $i > j$, then the system can be easily solved by a process known as [backward substitution](#). In backward substitution we simply start solving the system by eliminating the last variable first, then proceed to solve backwards. The process is illustrated in this [example](#), and described in generality [here](#).

The QR decomposition of a matrix

The [QR decomposition](#) allows to express any $m \times n$ matrix A as the product $A = QR$ where Q is $m \times m$ and orthogonal (that is, $Q^T Q = I_m$) and R is upper triangular. For more details on this, see [here](#).

Once the QR factorization of A is obtained, we can solve the system by first pre-multiplying with Q both sides of the equation: $QRx = y \iff Rx = Q^T y$. This is due to the fact that $Q^T Q = I_m$. The new system $Rx = Q^T y$ is triangular and can be solved by [backwards substitution](#). For example, if A is full column rank, then R is invertible, so that the solution is unique, and given by $x = R^{-1} Q^T y$.

Let us detail the process now.

$$AP = QR = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix}, \text{ where}$$

Using the full QR decomposition we start with the full [QR decomposition](#) of A with column permutations:

- $Q = [Q_1, Q_2]$ is $m \times m$ and orthogonal ($Q^T Q = I_m$);
- Q_1 is $m \times r$, with orthonormal columns ($Q^T Q = I_r$);
- Q_2 is $m \times (m - r)$, with orthonormal columns ($Q^T Q = I_{m-r}$);
- r is the [rank](#) of A ;
- R_1 is $r \times r$ upper triangular, and invertible;
- R_2 is a $r \times (n - r)$ matrix;
- P is a $m \times m$ permutation matrix (thus, $P^T = P^{-1}$).
- The zero submatrices in the bottom (block) row of R have $m - r$ rows.

Using $A = QRP^T$, we can write $Rz = Q^T y$, where $z := P^T x$. Let's look at the equation in z in expanded form:

$$\begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} Q_1^T y \\ Q_2^T y \end{pmatrix}.$$

We see that unless $Q_2^T y = 0$, there is no solution. Let us assume that $Q_2^T y = 0$. We have then $R_1 z_1 + R_2 z_2 = Q_1^T y$, which is a set of r linear equations in n variables.

A particular solution is obtained upon setting $z_2 = 0$, which leads to a triangular system in z_1 , with an invertible triangular matrix R_1 . Hence $z_1 = R_1^{-1} Q_1^T y$, which corresponds to a particular solution x_0 to $Ax = y$.

$$x_0 := P \begin{pmatrix} R_1^{-1} Q_1^T y \\ 0 \end{pmatrix}.$$

Set of solutions

We can also generate all the solutions, by noting that z_2 is a free variable. We have

$$x = P \begin{pmatrix} R_1^{-1} Q_1^T (y - R_2 z_2) \\ 0 \end{pmatrix} = x_0 + L z_2,$$

$$L := -P \begin{pmatrix} R_1^{-1} Q_1^T R_2 \\ 0 \end{pmatrix}.$$

where The set of solutions is the affine set $x_0 + \text{range}(L)$.

Applications

- [Temperature distribution](#).
- [Trilateration](#) by distance measurements.
- [Traffic flow estimation](#).

Temperature distribution

See [here](#).

Trilateration

alt text

In many applications, such as GPS, it is of interest to infer the location of an emitter (for example, a cell phone) from the measurement of distances to known points. These distances are obtained by estimating the differences in time of arrival of a wave front originating from the emitter. In trilateration, only three points are used. We then have to find the intersection of three spheres. The problem can then be reduced to solving a linear equation followed by a quadratic equation in one variable. The multilateration problem, which allows for more than three points, provides more accurate measurements. ([Source](#).)

Denote by $\square, i = 1, 2, 3$ the three known points and by R_i the measured distances to the emitter. Mathematically the problem is to solve, for a point $x \in \mathbf{R}^3$, the equations $|x - x^i|^2 = R_i^2$, $i = 1, 2, 3$. We write them out: $x^T x - 2x^T x^i + |x^i|^2 = R_i^2, ;; i = 1, 2, 3$.

Let $t := (1/2)x^T x$. The equations above imply that $t - x^T x^i = \text{gamma}_i := (1/2)(R_i^2 - |x^i|^2), ;; i = 1, 2, 3$. Using matrix notation, with $X = [x_1, x_2, x_3]$ the matrix of points, and $\mathbf{1}$ the vector of ones:

Let us assume that the square matrix X is full-rank, that is, invertible. The equation above implies that \square . In words: the point lies in a line passing through \square and with direction \square .

We can then solve the equation in t : $x(t)^T x(t) = 2t$. This equation is quadratic in t : $(v^T v)t^2 + 2((v^T x^0) - 1)t + |x^0|^2 = 0$ and can be solved in closed-form. The spheres intersect if and only if there is a real, non-negative solution t . Generically, if the spheres have a non-empty intersection, there are two positive solutions, hence two points in the intersection. This is understandable geometrically: the intersection of two spheres is a circle, and intersecting a circle with a third sphere produces two points. The line joining the two points is the line \square , as identified above.

Estimation of traffic flows

The basic traffic flow estimation problem involves inferring the amount of cars going through links based on information on the amount of cars passing through neighbouring links.

For the simple problem above, we simply use the fact that at each intersection, the incoming traffic has to match the outgoing traffic. This leads to the linear equations

$$\square$$

We can write this in matrix format: $Ax = y$, with

$$\square$$

The matrix A is nothing else than the [incidence matrix](#) associated with the graph that has the intersections as nodes and links as edges.

Exercises

A. Determine the nullspace, range and rank of a $m \times n$ matrix of the form

$$A = \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix},$$

where $S = \text{diag}(\sigma_1, \dots, \sigma_r)$, with $\sigma_1 \geq \dots \geq \sigma_r > 0$, and $r \leq \min(m, n)$. In the above, the zeroes are in fact matrices of zeroes with appropriate sizes.

B. Consider the matrix $A = uv^T$ with $u \in \mathbf{R}^m, v \in \mathbf{R}^n$.

1. What is the size of A ?
2. Determine the nullspace, the range, and the rank of A .

C. Consider the linear equation in $x \in \mathbf{R}^n$

$$Ax = y$$

where $A \in \mathbf{R}^{m \times n}, y \in \mathbf{R}^m$. Answer the following questions to the best of your knowledge.

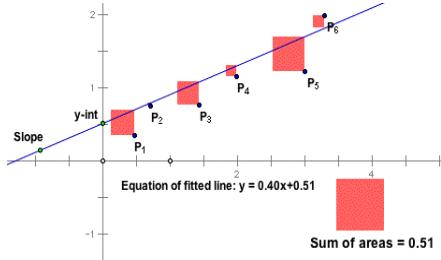
1. The time required to solve the general system depends on the sizes m, n and the entries of A . Provide a rough estimate of that time as a function of m, n only. You may assume that m, n are of the

same order.

2. Assume now that $m = n$, $A = D + uvT$, where D is diagonal, invertible, and $u, v \in \mathbf{R}^n$. How would you exploit this structure to solve the above linear system, and what is a rough estimate of the complexity of your algorithm?

3. What if A is upper-triangular?

Least-Squares and Variants



The ordinary least-squares (OLS) problem is a particularly simple optimization problem that involves the minimization of the Euclidean norm of a “residual error” vector that is affine in the decision variables.

The problem is one of the most ubiquitous optimization problems in engineering and applied sciences. It can be used for example to fit a straight line through points, as in the figure on the left. The least-squares approach then amounts to minimize the sum of the area of the squares with side-length equal to the vertical distances to the line.

We discuss a few variants amenable to the linear algebra approach: regularized least-squares, linearly-constrained least-squares. We also explain how to use “[kernels](#)” to handle problems involving non-linear curve fitting and prediction using non-linear functions.

Ordinary Least-Squares Problem Definition

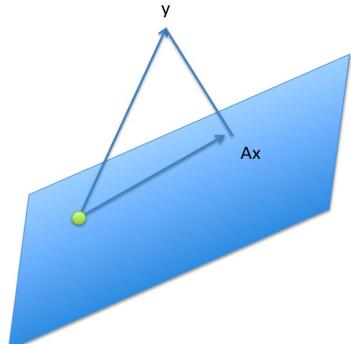
The Ordinary Least-Squares (OLS, or LS) problem is defined as

$$\min_x \|Ax - y\|_2^2$$

where $A \in \mathbf{R}^{m \times n}$, $y \in \mathbf{R}^m$ are given. Together, the pair (A, y) is referred to as the *problem data*. The vector y is often referred to as “measurement” or “output” vector, and the data matrix A as the “design” or “input” matrix. The vector $r := y - Ax$ is referred to as the *residual error* vector.

Note that the problem is equivalent to one where the norm is not squared. Taking the squares is done for convenience of the solution.

Interpretations Interpretation as projection on the range.



We can interpret the problem in terms of the columns of A , as follows. Assume that $A = [a_1, \dots, a_n]$, where $a_j \in \mathbf{R}^m$ is the j -th column of A , $j = 1, \dots, n$. The problem reads

$$\min_x \left\| \sum_{j=1}^n x_j a_j - y \right\|_2.$$

In this sense, we are trying to find the best approximation of y in terms of a linear combination of the columns of A . Thus, the OLS problem amounts to *project* (find the minimum Euclidean distance) the vector y on the span of the vectors a_j 's (that is to say: the range of A).

As seen in the picture, at optimum the *residual vector* $Ax - y$ is orthogonal to the range of A .

Example: [Image compression via least-squares](#).

Interpretation as minimum distance to feasibility. The OLS problem is usually applied to problems where the linear $Ax = y$ is not *feasible*, that is, there is no solution to $Ax = y$.

The OLS can be interpreted as finding the smallest (in Euclidean norm sense) perturbation of the right-hand side, δy , such that the linear equation

$$Ax = y + \delta y$$

becomes feasible. In this sense, the OLS formulation implicitly assumes that the data matrix A of the problem is known exactly, while only the right-hand side is subject to perturbation, or measurement errors. A more elaborate model, *total least-squares*, takes into account errors in both A and y .

Interpretation as regression. We can also interpret the problem in terms of the rows of A , as follows. Assume that $A^T = [a_1^T, \dots, a_m^T]$, where $a_i \in \mathbf{R}^n$ is the i -th row of A , $i = 1, \dots, m$. The problem reads

$$\min_x \sum_{i=1}^m (y_i - a_i^T x)^2.$$

In this sense, we are trying to fit of each component of y as a linear combination of the corresponding input a_i , with x as the coefficients of this linear combination.

Examples:

- [Linear regression](#).
- [Auto-regressive models for time series prediction](#).
- [Power law model fitting](#).

Solution via QR decomposition (full rank case)

Assume that the matrix $A \in \mathbf{m} \times \mathbf{n}$ is tall ($m \geq n$) and full column rank. Then the solution to the problem is unique, and given by

$$x^* = (A^T A)^{-1} A^T y.$$

This can be seen by simply taking the [gradient](#) (vector of derivatives) of the objective function, which leads to the optimality condition $A^T(Ax - y) = 0$. Geometrically, the residual vector $Ax - y$ is orthogonal to the span of the columns of A , as seen in the picture above.

We can also prove this via the [QR decomposition](#) of the matrix A : $A = QR$ with Q a $m \times n$ matrix with orthonormal columns ($Q^T Q = I_n$) and R a $n \times n$ upper-triangular, invertible matrix. Noting that

$$\|Ax - y\|_2^2 = x^T A^T Ax - 2x^T A^T y + y^T y = x^T R^T Rx - 2x^T R^T Q^T y + y^T y = \|Rx - Q^T y\|_2^2 + y^T(I - QQ^T)y,$$

and exploiting the fact that R is invertible, we obtain the optimal solution $x^* = R^{-1}Q^T y$. This is the same as the formula above, since

$$(A^T A)^{-1} A^T y = (R^T Q^T QR)^{-1} R^T Q^T y = (R^T R)^{-1} R^T Q^T y = R^{-1} Q^T y.$$

Thus, to find the solution based on the QR decomposition, we just need to implement two steps:

1. Rotate the output vector: set $\bar{y} = Q^T y$.
2. Solve the triangular system $Rx = \bar{y}$ by [backwards substitution](#).

In Matlab, the backslash operator finds the (unique) solution when A is full column rank.

Matlab syntax

`>> x = A\y;`

Optimal solution and optimal set

Recall that the [optimal set](#) of an minimization problem is its set of minimizers. For least-squares problems, the optimal set is an affine set, which reduces to a singleton when A is full column rank.

In the general case (A not necessarily tall, and /or not full rank) then the solution may not be unique. If x^0 is a particular solution, then $x = x^0 + z$ is also a solution, if z is such that $Az = 0$, that is, $z \in \mathbf{N}(A)$. That is, the nullspace of A describes the *ambiguity* of solutions. In mathematical terms:

$$\arg \min_x \|Ax - b\|_2 = x^0 + \mathbf{N}(A).$$

The formal expression for the set of minimizers to the least-squares problem can be found again via the QR decomposition. This is shown [here](#).

Variants of the Least-Squares Problem

Linearly constrained least-squares Definition. An interesting variant of the ordinary least-squares problem involves equality constraints on the decision variable x :

$$\min_x \|Ax - y\|_2^2 : Cx = d,$$

where $C \in \mathbf{R}^{p \times n}$, and $d \in \mathbf{R}^p$ are given.

Example: [Minimum-variance portfolio](#).

Solution. We can express the solution by first computing the nullspace of C . Assuming that the feasible set of the constrained LS problem is not empty, that is, d is in the range of C , this set can be expressed as $\mathbf{X} = \{x_0 + Nz : z \in \mathbf{R}^k\}$,

where k is the dimension of the nullspace of C , N is a matrix whose columns span the nullspace of C , and x_0 is a particular solution to the equation $Cx = d$.

Expressing x in terms of the free variable z , we can write the constrained problem as an unconstrained one:

$$\min_z \|\tilde{A}z - \tilde{y}\|_2,$$

where $\tilde{A} := AN$, and $\tilde{y} = y - Ax_0$.

Minimum-norm solution to linear equations

A special case of linearly constrained LS is

$$\min_x \|x\|_2 : Ax = y,$$

in which we implicitly assume that the linear equation $Ax = y$, has a solution, that is, y is in the range of A .

The above problem allows to select a particular solution to a linear equation, in the case when there are possibly many, that is, the linear system $Ax = y$ is under-determined.

As seen [here](#), when A is full row rank, that is, the matrix AA^T is invertible, the above has the closed-form solution

$$x^* = A^T(AA^T)^{-1}y.$$

Example: [Control positioning of a mass](#).

Regularized least-squares

In the case when the matrix A in the OLS problem is not full column rank, the closed-form solution cannot be applied. A remedy often used in practice is to transform the original problem into one where the full column rank property holds.

The *regularized* least-squares problem has the form

$$\min_x \|Ax - y\|_2^2 + \lambda \|x\|_2^2,$$

where $\lambda > 0$ is a (usually small) parameter.

The regularized problem can be expressed as an ordinary least-squares problem, where the data matrix is full column rank. Indeed, the above problem can be written as the ordinary LS problem

$$\min_x \|\tilde{A}x - \tilde{y}\|_2^2,$$

where

$$\tilde{A} := \begin{pmatrix} A \\ \sqrt{\lambda}I_n \end{pmatrix}, \quad \tilde{y} := \begin{pmatrix} y \\ 0 \end{pmatrix}.$$

The presence of the identity matrix in the $(m+n) \times n$ matrix \tilde{A} ensures that it is full (column) rank.

Solution

Since the data matrix in the regularized LS problem has full column rank, the formula seen [here](#) applies. The solution is unique, and given by

$$x^* = (\tilde{A}^T \tilde{A})^{-1} \tilde{A}^T \tilde{y} = (A^T A + \lambda I_n)^{-1} A^T y.$$

For $\lambda = 0$, we recover the ordinary LS expression that is valid when the original data matrix is full rank.

The above formula explains one of the motivations for using regularized least-squares in the case of a rank-deficient matrix A : if $\lambda > 0$, but is small, the above expression is still defined, even if A is rank-deficient.

Weighted regularized least-squares

Sometimes, as in [kernel methods](#), we are led to problems of the form

$$\min_x \|Ax - y\|_2^2 + x^T W x,$$

where W is [positive definite](#) (that is, $x^T W x > 0$ for every non-zero x). The solution is again unique and given by

$$x^* = (A^T A + W)^{-1} A^T y.$$

Kernel Least-Squares Motivations

Consider a linear [auto-regressive model for time-series](#), where y_t is a linear function of y_{t-1}, y_{t-2}

$$y_t = w_1 + w_2 y_{t-1} + w_3 y_{t-2}, \quad t = 1, \dots, T.$$

This writes $y_t = w^T x_t$, with x_t the “feature vectors”

$$x_t := (1, y_{t-1}, y_{t-2}), \quad t = 1, \dots, T.$$

We can fit this model based on historical data via least-squares:

$$\min_w \|X^T w - y\|_2^2$$

The associated *prediction rule* is

$$\hat{y}_{T+1} = w_1 + w_2 y_T + w_3 y_{T-1} = w^T x_{T+1}.$$

We can introduce a non-linear version, where y_t is a quadratic function of y_{t-1}, y_{t-2}

$$y_t = w_1 + w_2 y_{t-1} + w_3 y_{t-2} + w_4 y_{t-1}^2 + w_5 y_{t-1} y_{t-2} + w_6 y_{t-2}^2.$$

This writes $y_t = w^T \phi(x_t)$, with $\phi(x_t)$ the augmented feature vectors

$$\phi(x_t) := (1, y_{t-1}, y_{t-2}, y_{t-1}^2, y_{t-1} y_{t-2}, y_{t-2}^2).$$

Everything the same as before, with x replaced by $\phi(x)$.

It appears that the size of the least-squares problem grows quickly with the degree of the feature vectors. How do we do it in a computationally efficient manner?

The kernel trick

We exploit a simple fact: in the least-squares problem

$$\min_w \|X^T w - y\|_2^2 + \lambda \|w\|_2^2$$

the optimal w lies in the span of the data points (x_1, \dots, x_m) :

$$w = Xv$$

for some vector $v \in \mathbf{R}^m$. Indeed, from the [fundamental theorem of linear algebra](#), every $w \in \mathbf{R}^n$ can be written as the sum of two orthogonal vectors:

$$w = Xv + r$$

where $X^T r = 0$ (that is, r is in the nullspace $\mathcal{N}(X^T)$).

Hence the least-squares problem depends only on $K := X^T X$:

$$\min_v \|Kv - y\|_2^2 + \lambda v^T K v.$$

The matrix K is called the Kernel matrix. It is equal to the [Gram matrix](#) of the data points x_1, \dots, x_m , and provides a measure of similarity between them, in the sense that if they all have the same (unit) Euclidean norm, then K_{ij} is the cosine angle between x_i and x_j .

The prediction rule depends on the scalar products between new point x and the data points x_1, \dots, x_m :

$$w^T x = v^T X^T x = v^T k, \quad k := X^T x = (x_1^T x, \dots, x_m^T x).$$

Once K is formed (this takes $O(n)$), then the training problem has only m variables. When $n \gg m$, this leads to a dramatic reduction in problem size.

Nonlinear case

In the nonlinear case, we simply replace the feature vectors x_i by some “augmented” feature vectors $\phi(x_i)$, with ϕ a non-linear mapping.

This leads to the modified kernel matrix

$$K_{ij} = \phi(x_i)^T \phi(x_j), \quad 1 \leq i, j \leq m.$$

The kernel function associated with mapping ϕ is

$$k(x, z) = \phi(x)^T \phi(z).$$

It provides information about the metric in the feature space, e.g.:

$$\|\phi(x) - \phi(z)\|_2^2 = k(x, x) - 2k(x, z) + k(z, z).$$

The computational effort involved in

1. solving the training problem;
2. making a prediction,

depends only on our ability to quickly evaluate such scalar products. We can't choose k arbitrarily; it has to satisfy the above for some ϕ .

Examples of kernels

A variety of kernels are available. Some are adapted to the structure of data, for example text or images. Here are a few popular choices.

Polynomial kernels. The regression problem with quadratic functions involves feature vectors of the form: $\phi(x) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$.

In fact, given two vectors $x, z \in \mathbf{R}^2$, we have

$$\phi(x)^T \phi(z) = (1 + x^T z)^2.$$

More generally when $\phi(x)$ is the vector formed with all the products between the components of $x \in \mathbf{R}^n$, up to degree d , then for any two vectors $x, z \in \mathbf{R}^n$,

$$\phi(x)^T \phi(z) = (1 + x^T z)^d.$$

The computational effort grows linearly in n .

This represents a dramatic reduction in speed over the “brute force” approach:

1. Form $\phi(x), \phi(z)$;
2. evaluate $\phi(x)^T \phi(z)$. In the above approach the computational effort grows as n^d .

$$k(x, z) = \exp\left(-\frac{\|x - z\|_2^2}{2\sigma^2}\right),$$

Gaussian kernels. The Gaussian kernel function is

where $\sigma > 0$ is a scale parameter. This allows to ignore points that are too far apart. Corresponds to a non-linear mapping ϕ to infinite-dimensional feature space.

Other kernels. There is a large variety (a zoo?) of other kernels, some adapted to structure of data (text, images, etc). Kernels in practice

1. Kernels need to be chosen by the user.
2. The choice is not always obvious; Gaussian or polynomial kernels are popular.
3. We control over-fitting via cross validation (to choose, say, the scale parameter of Gaussian kernel, or degree of polynomial kernel).

Applications of Least-Squares

- [Linear regression via least-squares](#).
- [Auto-Regressive \(AR\) models for time-series prediction](#).

Linear regression via least-squares

Linear regression is based on the idea of fitting a linear function through data points.

In its basic form, the problem is as follows. we are given data (y_i, x_i) , $i = 1, \dots, m$, where $x_i \in \mathbf{R}^n$ is the “input” and y_i is the “output” for the i -th measurement. We seek to find a linear function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ such that $f(x_i)$ are collectively close to the corresponding values y_i .

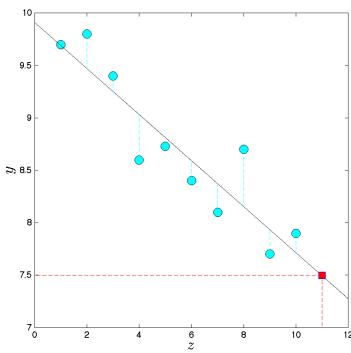
In *least-squares regression*, the way we evaluate how well a candidate function f fits the data is via the (squared) Euclidean norm:

Since a linear function f has the form $f(x) = \theta^T x$ for some , the problem of minimizing the above criterion takes the form

We can formulate this as a least-squares problem:

where

The linear regression approach can be extended to multiple dimensions, that is, to problems where the output in the above problem contains more than one dimension (see [here](#)). It can also be extended to the problem of fitting non-linear curves.



In this example we seek to analyze how customers react to an increase in the price of a given item. We are given two-dimensional data points (x_i, y_i) , $i = 1, \dots, m$. The x_i 's contain the prices of the item, and the y_i 's the average number of customers who buy the item at that price.

The generic equation of a non-vertical line is , where contains the decision variables. The quality of the fit of a generic line is measured via the sum of the squares of the error in the component y (blue dotted lines). Thus, the best least-squares fit is obtained via the least-squares problem

Once the line is found, it can be used to predict the value of the average number of customers buying the item (y) for a new price (x). The prediction is shown in red.

See also:

- [Auto-regressive models for time series prediction](#).
- [The problem of Gauss](#).

Auto-Regressive (AR) models for time-series prediction

A popular model for the prediction of time series is based on the so-called auto-regressive model

$$y_t = \theta_1 y_{t-1} + \dots + \theta_m y_{t-m}, \quad t = 1, \dots, m,$$

where θ_i 's are constant coefficients, and m is the "memory length" of the model. The interpretation of the model is that the next output is a linear function of the past. Elaborate variants of auto-regressive models are widely used for prediction of time series arising in finance and economics.

To find the coefficient vector θ in \mathbf{R}^m , we collect observations $(y_t)_{0 \leq t \leq T}$ (with $T \geq m$) of the time series, and try to minimize the total squared error in the above equation:

$$\min_{\theta} : \sum_{t=m}^T (y_t - \theta_1 y_{t-1} - \dots - \theta_m y_{t-m})^2.$$

This can be expressed as a linear least-squares problem, with appropriate data A, y .

See also: [Linear regression via Least-Squares](#).

Exercises Standard forms A. Regularization for noisy data. Consider a least-squares problem

$$\min_x \|Ax - y\|_2^2$$

in which the data matrix $A \in \mathbf{R}^{m \times n}$ is noisy. Our specific noise model assumes that each row $a_i^T \in \mathbf{R}^n$ has the form $a_i = \hat{a}_i + u_i$, where the noise vector $u_i \in \mathbf{R}^n$ has zero mean and covariance matrix $\sigma^2 I_n$, with σ a measure of the size of the noise. Therefore, now the matrix A is a function of the uncertain vector $u = (u_1, \dots, u_n)$, which we denote by $A(u)$. We will write \hat{A} to denote the matrix with rows \hat{a}_i^T , $i = 1, \dots, m$. We replace the original problem with

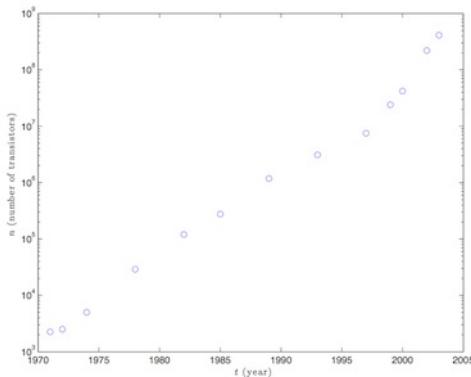
$$\min_x \mathbf{E}_u \|A(u)x - y\|_2^2,$$

where \mathbf{E}_u denotes the expected value with respect to the random variable u . Show that this problem can be written as

$$\min_x \|\hat{A}x - y\|_2^2 + \lambda \|x\|_2^2,$$

where $\lambda \geq 0$ is some regularization parameter, which you will determine. That is, regularized least-squares can be interpreted as a way to take into account uncertainties in the matrix A , in the expected value sense. Hint: compute the expected value of $((\hat{a}_i + u_i)^T x - y_i)^2$, for a specific row index i .

Applications



B. The figure shows the number of transistors in 13 microprocessors as a function of the year of their introduction. The plot suggests that we can obtain a good fit with a function of the form $n(t) = \alpha^{t-t_0}$, where t is the year, $n(t)$ is the number of transistors at year t , and $\alpha > 0$ and t_0 are model parameters. This model results in a straight line if we plot $n(t)$ on a logarithmic scale versus t on a linear scale, as is done in the figure.

[Moore's law](#) describes a long-term trend in the history of computing hardware, and states that the number of transistors that can be placed inexpensively on an integrated circuit has doubled approximately every two years. In this problem, we investigate the validity of the claim via least-squares.

1. Using the [problem data](#), show how to estimate the parameters α, t_0 using least-squares, that is, via a problem of the form $\min_x \|\hat{A}x - y\|_2$. Make sure to define precisely the data A, y and how the variable x relates to the original problem parameters α, t_0 . (Use the notations $n = (n_1, \dots, n_{13}) \in \mathbf{R}^{13}$ for the number of processors, and $t = (t_1, \dots, t_{13}) \in \mathbf{R}^{13}$ for the corresponding years. You can assume that no component of x is zero at optimum.)
2. Is the solution to problem above unique? Justify carefully your answer, and give the expression for the unique solution x^* in terms of A, y .
3. The solution to the problem yields $\alpha = 1.4257, t_0 = 1949.7$. Is this estimate consistent with the so-called Moore's law, which states that the number of transistors per integrated circuit roughly doubles every two years?

C. The [Michaelis-Menten model](#) for enzyme kinetics relates the rate y of an enzymatic reaction, to the concentration x of a substrate, as follows:

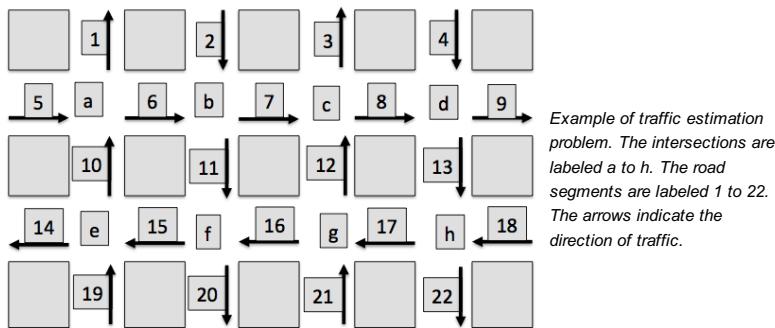
$$y = \frac{\beta_1 x}{\beta_2 + x},$$

where $\beta_i, i = 1, 2$, are parameters.

1. Show that the model can be expressed as a linear relation between the values $1/y$ and $1/x$.
2. Use this expression to fit the parameter β using linear least-squares.
3. The above approach has been found to be quite sensitive to errors in input data. Can you experimentally confirm this opinion? Hint: generate noisy data from parameter values $\beta_1 = 3.4$ and $\beta_2 = 0.4$.

D. Least norm estimation on traffic flow networks.

You want to estimate the traffic (in San Francisco for example, but we'll start with a smaller example). You know the road network as well as the historical average of flows on each road segment.

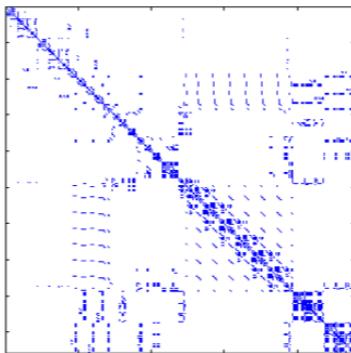


segment	average	measured
1	2047.6	2028
2	2046.0	2008
3	2002.6	2035
4	2036.9	
5	2013.5	2019
6	2021.1	
7	2027.4	
8	2047.1	
9	2020.9	2044
10	2049.2	
11	2015.1	
12	2035.1	
13	2033.3	
14	2027.0	2043
15	2034.9	
16	2033.3	
17	2008.9	
18	2006.4	
19	2050.0	2030
20	2008.6	2025
21	2001.6	
22	2028.1	2045

Table of flows: historical averages q (center column), and some measured flows (right column).

1. We call q_i the flow of vehicles on each road segment $i \in I$. Write down the linear equation that corresponds to the conservation of vehicles at each intersection $j \in J$. Hint: think about how you might represent the road network in terms of matrices, vectors, etc.
2. The goal of the estimation is to estimate the traffic flow on each of the road segment. The flow estimates should satisfy the conservation of vehicles exactly at each intersection. Among the solutions that satisfy this constraint, we are searching for the estimate that is the closest to the historical average, q , in the ℓ_2 -norm sense. The vector q has size I and the i -th element represent the average for the road segment i . Pose the optimization problem.
3. Explain how to solve this problem mathematically. Detail your answer (do not only give a formula but explain where it comes from).
4. Formulate the problem for the small example of the figure above and solve it using the historical average given in the Table of Flows above. What is the flow that you estimate on road segments 1, 3, 6, 15 and 22?
5. Now, assume that besides the historical averages, you are also given some flow measurements on some of the road segments of the network. You assume that these flow measurements are correct and want your estimate of the flow to match these measurements perfectly (besides matching the conservation of vehicles of course). The right column of Table 1 lists the road segments for which we have such flow measurements. Do you estimate a different flow on some of the links? Give the difference in flow you estimate for road segments 1, 3, 6, 15 and 22. Also check that your estimate gives you the measured flow on the road segments for which you have measured the flow.

Eigenvalue Decomposition of Symmetric Matrices



Symmetric matrices are square, with elements that mirror each other across the diagonal. They arise naturally in graph models, and are closely related to quadratic functions.

A fundamental theorem, the *spectral theorem*, shows that we can decompose any symmetric matrix as a three-term product of matrices, involving an orthogonal transformation and a diagonal matrix. The theorem has a direct implication for quadratic functions: it allows us to decompose any quadratic function into a weighted sum of squared linear functions involving vectors that are mutually orthogonal. The weights are called the *eigenvalues* of the symmetric matrix.

The spectral theorem allows in particular to determine when a given quadratic function is “bow-shaped”, that is, convex. The spectral theorem also allows to find directions of maximal variance within a data set. Such directions are useful to visualize high-dimensional data points in two or three dimensions. This is the basis of a visualization method known as *principal component analysis* (PCA).

Definitions: Symmetric matrices and quadratic functions

Symmetric matrices. A square matrix $A \in \mathbf{R}^{n \times n}$ is *symmetric* if it is equal to its transpose. That is, $A_{ij} = A_{ji}$, $1 \leq i, j \leq n$.

The set of symmetric $n \times n$ matrices is denoted \mathbf{S}^n . This set is a subspace of $\mathbf{R}^{n \times n}$.

Examples:

- [A \$3 \times 3\$ example](#).
- [Representation of a weighted, undirected graph](#).
- [Laplacian matrix of a graph](#).

- [Hessian of a function](#).
- [Gram matrix](#) of data points.

$$q(x) = \sum_{i=1}^n \sum_{j=1}^n A_{ij} x_i x_j + 2 \sum_{i=1}^n b_i x_i + c,$$

Quadratic functions. A function $q : \mathbf{R}^n \rightarrow \mathbf{R}$ is said to be a *quadratic function* if it can be expressed as

for numbers A_{ij} , b_i , and c , $i, j \in \{1, \dots, n\}$. A quadratic function is thus an affine combination of the x_i 's and all the “cross-products”, $x_i x_j$. We observe that the coefficient of $x_i x_j$ is $(A_{ij} + A_{ji})$.

The function is said to be a *quadratic form* if there are no linear or constant terms in it: $b_i = 0, c = 0$.

Note that the [Hessian](#) (matrix of second-derivatives) of a quadratic function is constant.

Examples:

- [Quadratic functions of two variables](#).
- [Hessian of a quadratic function](#).

Link between quadratic functions and symmetric matrices. There is a natural relationship between symmetric matrices and quadratic functions. Indeed, any quadratic function $q : \mathbf{R}^n \rightarrow \mathbf{R}$ can be written as

$$q(x) = \begin{pmatrix} x \\ 1 \end{pmatrix}^T \begin{pmatrix} A & b \\ b^T & c \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} = x^T A x + 2b^T x + c,$$

for an appropriate symmetric matrix $A \in \mathbf{S}^n$, vector $b \in \mathbf{R}^n$ and scalar $c \in \mathbf{R}$. Here, A_{ii} is the coefficient of x_i^2 in q ; for $i \neq j$, $2A_{ij}$ is the coefficient of the term $x_i x_j$ in q ; $2b_i$ is that of x_i ; and c is the constant term, $q(0)$. If q is a quadratic form, then $b = 0, c = 0$, and we can write $q(x) = x^T A x$ where now $A \in \mathbf{S}^n$.

Example: [Two-dimensional example](#).

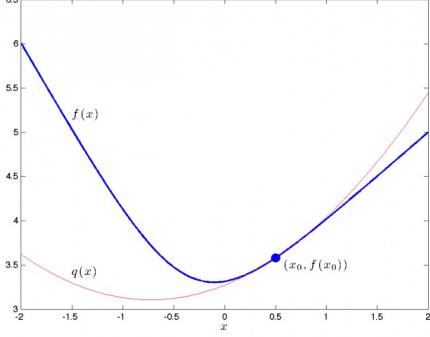
Second-order approximations of non-quadratic functions

We have seen [here](#) how linear functions arise when one seeks a simple, linear approximation to a more complicated non-linear function. Likewise, quadratic functions arise naturally when one seeks to approximate a given non-quadratic function by a quadratic one.

One-dimensional case. If $f : \mathbf{R} \rightarrow \mathbf{R}$ is a twice-differentiable function of a *single* variable, then the *second order approximation* (or, second-order Taylor expansion) of f at a point x_0 is of the form

$$f(x) \approx q(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2,$$

where $f'(x_0)$ is the first derivative, and $f''(x_0)$ the second derivative, of f at x_0 . We observe that the quadratic approximation q has the same value, derivative, and second-derivative as f , at x_0 .



Example: The figure shows a second-order approximation q of the univariate function $f : \mathbf{R} \rightarrow \mathbf{R}$, with values

$$f(x) = \log(\exp(x-3) + \exp(-2x+2)),$$

at the point $x_0 = 0.5$ (in blue).

Multi-dimensional case. In multiple dimensions, we have a similar result. Let us approximate a twice-differentiable function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ by a quadratic function q , so that f and q coincide up and including to the second derivatives.

The function q must be of the form

$$q(x) = x^T A x + 2b^T x + c,$$

where $A \in \mathbf{S}^n$, $b \in \mathbf{R}^n$, and $c \in \mathbf{R}$. Our condition that q coincides with f up and including to the second derivatives shows that we must have

$$\nabla^2 q(x) = 2A = \nabla^2 f(x_0), \quad \nabla q(x) = 2(Ax_0 + b) = \nabla f(x_0), \quad x_0^T A x_0 + 2b^T x_0 + c = f(x_0),$$

where $\nabla^2 f(x_0)$ is the [Hessian](#), and $\nabla f(x_0)$ the gradient, of f at x_0 .

Solving for A, b, c we obtain the following result:

Second-order expansion of a function. The *second-order approximation* of a twice-differentiable function f at a point x_0 is of the form

$$f(x) \approx q(x) = f(x_0) + \nabla f(x_0)^T (x - x_0) + \frac{1}{2} (x - x_0)^T \nabla^2 f(x_0) (x - x_0),$$

where $\nabla f(x_0) \in \mathbf{R}^n$ is the [gradient](#) of f at x_0 , and the symmetric matrix $\nabla^2 f(x_0)$ is the [Hessian](#) of f at x_0 . ◇

Example: [Second-order expansion of the log-sum-exp function](#).

Special symmetric matricesDiagonal matrices. Perhaps the simplest special case of symmetric matrices is the class of diagonal matrices, which are non-zero only on their diagonal.

If $\lambda \in \mathbf{R}^n$, we denote by $\text{diag}(\lambda_1, \dots, \lambda_n)$, or $\text{diag}(\lambda)$ for short, the $n \times n$ (symmetric) diagonal matrix with λ on its diagonal. Diagonal matrices correspond to quadratic functions that

are simple sums of squares, of the form:

$$q(x) = \sum_{i=1}^n \lambda_i x_i^2 = x^T \text{diag}(\lambda)x.$$

Such functions do not have any “cross-terms” of the form $x_i x_j$ with $i \neq j$.

Example: [A diagonal matrix and its associated quadratic form](#).

Symmetric dyads. Another important class of symmetric matrices is that of the form uu^T , where $u \in \mathbf{R}^n$. The matrix has elements $u_i u_j$, and is symmetric. Such matrices are called *symmetric dyads*. (If $\|u\|_2 = 1$, then the dyad is said to be normalized.)

Symmetric dyads corresponds to quadratic functions that are simply squared linear forms:

$$q(x) = (u^T x)^2.$$

Example: [A squared linear form](#).

Spectral Theorem Eigenvalues and eigenvectors of symmetric matrices

Let A be a square, $n \times n$ symmetric matrix. A real scalar λ is said to be an *eigenvalue* of A if there exist a non-zero vector $u \in \mathbf{R}^n$ such that

$$Au = \lambda u.$$

The vector u is then referred to as an *eigenvector* associated with the eigenvalue λ . The eigenvector u is said to be *normalized* if $\|u\|_2 = 1$. In this case, we have

$$u^T Au = \lambda u^T u = \lambda.$$

The interpretation of u is that it defines a direction along A behaves just like scalar multiplication. The amount of scaling is given by λ . (In German, the root “eigen”, means “self” or “proper”). The eigenvalues of the matrix A are characterized by the *characteristic equation*

$$\det(\lambda I - A) = 0,$$

where the notation \det refers to the [determinant](#) of its matrix argument. The function with values $t \rightarrow p(t) := \det(tI - A)$ is a polynomial of degree n called the *characteristic polynomial*.

From the fundamental theorem of algebra, any polynomial of degree n has n (possibly not distinct) complex roots. For symmetric matrices, the eigenvalues are real, since $\lambda = u^T Au$ when $Au = \lambda u$, and u is normalized.

Spectral theorem

An important result of linear algebra, called the *spectral theorem*, or *symmetric eigenvalue decomposition* (SED) theorem, states that for any symmetric matrix, there are exactly n (possibly not distinct) eigenvalues, and they are all real; further, that the associated eigenvectors can be chosen so as to form an orthonormal basis. The result offers a simple way to decompose the symmetric matrix as a product of simple transformations.

Theorem: *Symmetric eigenvalue decomposition* ([Proof](#))

We can decompose any symmetric matrix $A \in \mathbf{S}^n$ with the symmetric eigenvalue decomposition (SED)

$$A = \sum_{i=1}^n \lambda_i u_i u_i^T = U \Lambda U^T, \quad \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n).$$

where the matrix of $U := [u_1, \dots, u_n]$ is orthogonal (that is, $U^T U = U U^T = I_n$), and contains the eigenvectors of A , while the diagonal matrix Λ contains the eigenvalues of A .

The SED provides a decomposition of the matrix in simple terms, namely [dyads](#). We check that in the SED above, the scalars λ_i are the eigenvalues, and u_i 's are associated eigenvectors, since

$$Au_j = \sum_{i=1}^n \lambda_i u_i u_i^T u_j = \lambda_j u_j, \quad j = 1, \dots, n.$$

The eigenvalue decomposition of a symmetric matrix can be efficiently computed with standard software, in time that grows proportionately to its dimension n as n^3 . Here is the matlab syntax, where the first line ensure that matlab knows that the matrix A is exactly symmetric.

Matlab syntax

```
>> A = triu(A)+tril(A',-1);
>> [U,D] = eig(A);
```

Example: [Eigenvalue decomposition of a \$2 \times 2\$ symmetric matrix](#).

Rayleigh quotients

Given a symmetric matrix A , we can express the smallest and largest eigenvalues of A , denoted λ_{\min} and λ_{\max} respectively, in the so-called *variational* form

$$\lambda_{\min}(A) = \min_x \{x^T Ax : x^T x = 1\}, \quad \lambda_{\max}(A) = \max_x \{x^T Ax : x^T x = 1\}.$$

For a proof, see [here](#).

The term “variational” refers to the fact that the eigenvalues are given as optimal values of optimization problems, which were referred to in the past as variational problems. Variational representations exist for all the eigenvalues, but are more complicated to state.

The interpretation of the above identities is that the largest and smallest eigenvalues is a measure of the range of the quadratic function $x \rightarrow x^T Ax$ over the unit Euclidean ball. The quantities above can be written as the minimum and maximum of the so-called *Rayleigh quotient* $x^T Ax / x^T x$.

Historically, David Hilbert coined the term “spectrum” for the set of eigenvalues of a symmetric operator (roughly, a matrix of infinite dimensions). The fact that for symmetric matrices, every eigenvalue lies in the interval $[\lambda_{\min}, \lambda_{\max}]$ somewhat justifies the terminology.

Example: [Largest singular value norm of a matrix.](#)

Positive Semi-Definite Matrices Definitions

For a given symmetric matrix $A \in \mathbf{R}^{n \times n}$, the associated *quadratic form* is the function $q : \mathbf{R}^n \rightarrow \mathbf{R}$ with values

$$q(x) = x^T A x.$$

A symmetric matrix A is said to be *positive semi-definite* (PSD, notation: $A \succeq 0$) if and only if the associated quadratic form q is non-negative everywhere:

$$q(x) \geq 0 \text{ for every } x \in \mathbf{R}^n.$$

It is said to be *positive definite* (PD, notation: $A \succ 0$) if the quadratic form is non-negative, and definite, that is, $q(x) = 0$ if and only if $x = 0$.

It turns out that a matrix is PSD if and only if the eigenvalues of A are non-negative. Thus, we can check if a form is PSD by computing the eigenvalue decomposition of the underlying symmetric matrix.

Theorem: eigenvalues of PSD matrices ([Proof](#)).

A quadratic form $q(x) = x^T A x$, with $A \in \mathbf{S}^n$ is non-negative (resp. positive-definite) if and only if every eigenvalue of the symmetric matrix A is non-negative (resp. positive).

By definition, the PSD and PD properties are properties of the eigenvalues of the matrix only, not of the eigenvectors. Also, if the $n \times n$ matrix A is PSD, then for every matrix B with n rows, the matrix $B^T A B$ also is.

Special cases and examplesSymmetric dyads. Special cases of PSD matrices include symmetric dyads. Indeed, if $A = uu^T$ for some vector $u \in \mathbf{R}^n$, then for every x :
 $q_A(x) = x^T uu^T x = (u^T x)^2 \geq 0$.

More generally if $B \in \mathbf{R}^{m \times n}$, then $A = B^T B$ is PSD, since

$q_A(x) = x^T B^T B x = \|Bx\|_2^2 \geq 0$. Diagonal matrices. A diagonal matrix is PSD (resp. PD) if and only if all of its (diagonal) elements are non-negative (resp. positive). Examples of PSD matrices

- [Covariance matrix](#).
- [Laplacian matrix of a graph](#).
- [Gram matrix](#) of data points.

Square root and Cholesky decomposition

For PD matrices, we can generalize the notion of ordinary square root of a non-negative number. Indeed, if A is PSD, there exist a unique PSD matrix, denoted $A^{1/2}$, such that $A = (A^{1/2})^2$. We can express this matrix square root in terms of the SED of $A = U\Lambda U^T$, as $A^{1/2} = U\Lambda^{1/2}U^T$, where $\Lambda^{1/2}$ is obtained from Λ by taking the square root of its diagonal elements. If A is PD, then so is its square root.

Any PSD matrix can be written as a product $A = LL^T$ for an appropriate matrix L . The decomposition is not unique, and $L = A^{1/2}$ is only a possible choice (the only PSD one). Another choice, in terms of the SED of $A = U^T\Lambda U$, is $L = U^T\Lambda^{1/2}$. If A is positive-definite, then we can choose L to be lower triangular, and invertible. The decomposition is then known as the *Cholesky decomposition* of A .

Ellipsoids

There is a strong correspondence between ellipsoids and PSD matrices.

Definition. We define an ellipsoid to be affine transformation of the unit ball for the Euclidean norm: $\mathbf{E} = \{\hat{x} + Lz : \|z\|_2 \leq 1\}$,

where $L \in \mathbf{R}^{n \times n}$ is an arbitrary non-singular matrix. We can express the ellipsoid as

$$\mathbf{E} = \{x : \|L^{-1}(x - \hat{x})\|_2 \leq 1\} = \{x : (x - \hat{x})^T A^{-1}(x - \hat{x}) \leq 1\},$$

where $A := L^{-T}L^{-1}$ is PD.

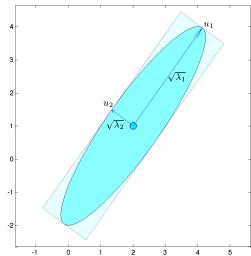
Geometric interpretation via SED. We can interpret the eigenvectors and associated eigenvalues of A in terms of geometrical properties of the ellipsoid, as follows. Consider the SED of A : $A = U\Lambda U^T$, with $U^T U = I$ and Λ diagonal, with diagonal elements positive. The SED of its inverse is $A^{-1} = LL^T = U\Lambda^{-1}U^T$. Let $\tilde{x} = U^T(x - \hat{x})$. We can express the condition $x \in \mathbf{E}$ as

$$\tilde{x}^T \Lambda^{-1} \tilde{x} = \sum_{i=1}^n \frac{\tilde{x}_i^2}{\lambda_i} \leq 1.$$

Now set $\bar{x}_i = \tilde{x}_i / \sqrt{\lambda_i}$, $i = 1, \dots, n$. The above writes $\bar{x}^T \bar{x} \leq 1$; in \bar{x} -space, the ellipsoid is simply a unit ball. In \tilde{x} -space, the ellipsoid corresponds to scaling each \tilde{x} -axis by the square roots of the eigenvalues. The ellipsoid has principal axes parallel to the coordinate axes in \tilde{x} -space. We then apply a rotation and a translation, to get the ellipsoid in the original x -space. The rotation is determined by the eigenvectors of A^{-1} , which are contained in the orthogonal matrix U . Thus, the geometry of the ellipsoid can be read from the SED of the PD matrix $A^{-1} = LL^T$: the eigenvectors give the principal directions, and the semi-axis lengths are the square root of the eigenvalues.

The graph on the left shows the ellipsoid $\{\hat{x} + Lz : \|z\|_2 \leq 1\}$, with

$$\hat{x} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \quad L = \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}.$$



The matrix LL^T admits the SED $LL^T = U^T \Lambda U$, with

$$U = \begin{pmatrix} -0.9871 & 0.1602 \\ 0.1602 & 0.9871 \end{pmatrix}, \quad \Lambda = \begin{pmatrix} 0.6754 & 0 \\ 0 & 13.3246 \end{pmatrix}.$$

We check that the columns of U determine the principal directions, and $\sqrt{\lambda_1} = 3.6503$, $\sqrt{\lambda_2} = 0.8219$ are the semi-axis lengths.

The above shows in particular that an equivalent representation of an ellipsoid is

$$\{x : (x - \hat{x})^T B(x - \hat{x}) \leq 1\}$$

where $B := A^{-1}$ is PD.

It is possible to define *degenerate ellipsoids*, which correspond to cases when the matrix B in the above, or its inverse A , is degenerate. For example, cylinders or slabs (intersection of two parallel half-spaces) are degenerate ellipsoids.

Principal Component Analysis Projection on a line via variance maximization

Consider a data set of n points x_j , $j = 1, \dots, n$ in \mathbf{R}^m . We can represent this data set as a $m \times n$ matrix $X = [x_1, \dots, x_n]$ where each x_j is a m -vector. The *variance maximization problem* is to find a direction $u \in \mathbf{R}^m$ such that the sample variance of the corresponding vector $u^T X = (u^T x_1, \dots, u^T x_n)$ is maximal.

Recall that when u is normalized, the scalar $u^T x$ is the component of x along u , that is, it corresponds to the projection of x on the line passing through 0 and with direction u .

Here, we seek a (normalized) direction u such that the empirical variance of the projected values $u^T x_j$, $j = 1, \dots, n$, is large. If \hat{x} is the vector of averages of the x_j 's, then the average of the projected values is $u^T \hat{x}$. Thus, the direction of maximal variance is one that solves the optimization problem

$$\max_{u : \|u\|_2=1} \frac{1}{n} \sum_{j=1}^n ((x_j - \hat{x})^T u)^2.$$

The above problem can be formulated as

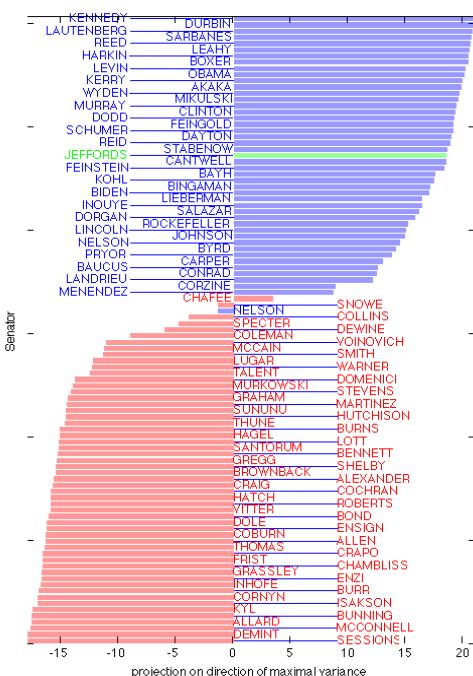
$$\max_{u : \|u\|_2=1} u^T \Sigma u,$$

where

$$\Sigma := \frac{1}{n} \sum_{j=1}^n (x_j - \hat{x})(x_j - \hat{x})^T$$

is the $m \times m$ sample covariance matrix of the data.

We have seen the above problem before, under the name of the Rayleigh quotient of a symmetric matrix. Solving the problem entails simply finding an eigenvector of the covariance matrix Σ that corresponds to the largest eigenvalue.



Maximal variance direction for the Senate voting data. This image shows the scores assigned to each Senator along the direction of maximal variance, $u_{\max}^T (x_j - \hat{x})$, $j = 1, \dots, n$, with u_{\max} a normalized eigenvector corresponding to the largest eigenvalue of the covariance matrix Σ . Republican Senators tend to score negatively, while we find many Democrats on the positive score (obviously the sign themselves don't count here, as we could switch u to $-u$; only the order is important). Hence the direction could be interpreted as revealing the party affiliation. The two Senators that are in the opposite group (especially Sen. Chafee) have indeed sometimes voted against their party.

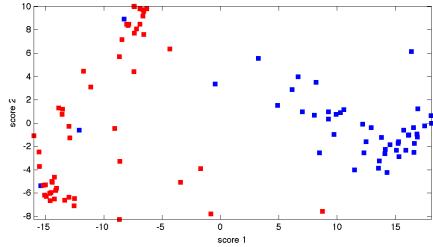
Note that the largest absolute score obtained in this plot is about 18 times bigger than that observed on the projection on a random direction. This is consistent with the fact that the current direction has maximal variance.

The main idea behind *principal component analysis* is to first find a direction that corresponds to maximal variance between the data points. The data is then projected on the hyperplane orthogonal to that direction. We obtain a new data set, and find a new direction of maximal variance. We may stop the process when we have collected enough directions (say, three if we want to visualize the data in 3D).

It turns out that the directions found in this way are precisely the eigenvectors of the data's covariance matrix. The term *principal components* refers to the directions given by these eigenvectors. Mathematically, the process thus amounts to finding the eigenvalue decomposition of a positive semi-definite matrix, the covariance matrix of the data points.

Projection on a plane

The projection to use to obtain, say, a two-dimensional view with the largest variance, is of the form $x \rightarrow Px$, where $P = [u_1, u_2]^T$ is a matrix that contains the eigenvectors corresponding to the first two eigenvalues.



Two-dimensional projection of the Senate voting matrix: This particular planar projection uses the two eigenvectors corresponding to the largest two eigenvalues of the data's covariance matrix. It seems to allow to cluster the Senators along party line, and is therefore more informative than, say, the [plane corresponding to the two smallest eigenvalues](#).

Explained variance

The total variance in the data is defined as the sum of the variances of the individual components. This quantity is simply the [trace](#) of the covariance matrix, since the diagonal elements of the latter contain the variances. If Σ has the EVD $\Sigma = U\Lambda U^T$, where $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ contains the eigenvalues, and U an orthogonal matrix of eigenvectors, then the total variance can be expressed as the *sum* of all the eigenvalues:

$$\text{Tr}\Sigma = \text{Tr}(U\Lambda U^T) = \text{Tr}(U^T U \Lambda) = \text{Tr}\Lambda = \lambda_1 + \dots + \lambda_n.$$

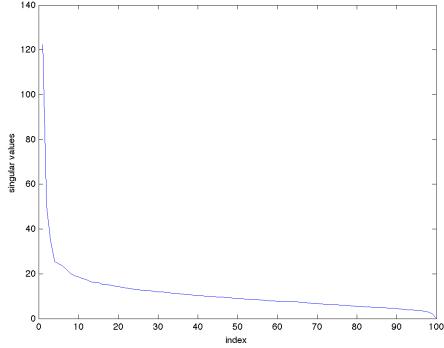
When we project the data on a two-dimensional plane corresponding to the eigenvectors u_1, u_2 associated with the two largest eigenvalues λ_1, λ_2 , we get a new covariance matrix $P\Sigma P^T$, where $P = [u_1, u_2]^T$ the total variance of the projected data is

$$\text{Tr}(P\Sigma P^T) = \lambda_1 + \lambda_2.$$

Hence, we can define the ratio of variance "explained" by the projected data as the ratio:

$$\frac{\lambda_1 + \lambda_2}{\lambda_1 + \dots + \lambda_n}.$$

If the ratio is high, we can say that much of the variation in the data can be observed on the projected plane.



This image shows the eigenvalues of the $m \times m$ covariance matrix of the [Senate voting data](#), which contains the covariances between the votes of each pair of Senators. Clearly, the eigenvalues decrease very fast. One is tempted to say that in this case, the ratio of explained to total variance is almost 90%, which indicates that "most of the information" is contained in the first eigenvalue. Since the corresponding eigenvector almost corresponds to a perfect "party line", we can say that party affiliation explains most of the variance in the Senate voting data.

Applications

- [PCA of Senate voting data](#)
- [Eigenfaces](#). See also [this post](#).
- [Eigenshirts](#). (A fun blog post by Joel Grus.)

Exercises A. Interpretation of covariance matrix. We are given m of points x_1, \dots, x_m in \mathbf{R}^n . Recall that the sample average of the data points is $\widehat{x} = (x_1 + \dots + x_m)/m$, and the sample

$$\Sigma := \frac{1}{m} \sum_{i=1}^m (x_i - \widehat{x})(x_i - \widehat{x})^T,$$

covariance matrix of the data points is given by
We assume that the average and variance of the data projected along a given direction does not change with the direction. In this exercise we will show that the [sample covariance matrix](#) is then proportional to the identity.

We formalize this as follows. To a given normalized direction $w \in \mathbf{R}^n$ ($\|w\|_2 = 1$), we associate the line with direction w passing through the origin, $\mathcal{L}(w) = \{tw : t \in \mathbf{R}\}$. We then consider the projection of the points $x_i, i = 1, \dots, m$, on the line $\mathcal{L}(w)$, and look at the associated coordinates of the points on the line. These *projected values* are given by

$$t_i(w) := \arg \min_t \|tw - x_i\|_2, \quad i = 1, \dots, m.$$

We assume that for any w , the sample average $\hat{t}(w)$ of the projected values $t_i(w), i = 1, \dots, m$, and their sample variance $\sigma^2(w)$, are both *constant*, independent of the direction w (with $\|w\|_2 = 1$). Denote by \hat{t} and σ^2 the (constant) sample average and variance.

Justify your answer to the following as carefully as you can.

1. Show that $t_i(w) = x_i^T w, i = 1, \dots, m$.
2. Show that the sample average of the data points is zero.
3. Show that the sample covariance matrix of the data points, is of the form $\sigma^2 \cdot I$, where I is the identity matrix of order n . Hint: the largest eigenvalue λ_{\max} of the matrix Σ can be written as:

$$\lambda_{\max} = \max_w \{w^T \Sigma w : w^T w = 1\},$$

and a similar expression holds for the smallest eigenvalue.

B. Eigenvalue decomposition of a rank-two matrix. Let $p, q \in \mathbf{R}^n$ be two linearly independent vectors, with unit norm ($\|p\|_2 = \|q\|_2 = 1$). Define the symmetric matrix $A := pq^T + qp^T$. In your derivations, it may be useful to use the notation $c := p^T q$.

1. Show that $p + q$ and $p - q$ are eigenvectors of A , and determine the corresponding eigenvalues.
2. Determine the nullspace and rank of A .
3. Find an eigenvalue decomposition of A . Hint: use the previous two parts.
4. What is the answer to the previous part if p, q are not normalized?

C. Positive-definite matrices, ellipsoids. In this problem we examine the geometrical interpretation of the positive definiteness of a matrix. For each of the following cases determine the shape of the region generated by the constraint $x^T Ax \leq 1$.

$$1. A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

$$2. A = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

$$3. A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

D. Show that if a square, $n \times n$ symmetric matrix A is positive semi-definite, then for every $n \times k$ matrix B , $B^T AB$ is also positive semi-definite. (Here, k is an arbitrary integer.)

E. Drawing an ellipsoid. How would you efficiently draw an ellipsoid in \mathbf{R}^2 , if the ellipsoid is described by a quadratic inequality of the form $E = \{x^T Ax + 2b^T x + c \leq 0\}$,

where A is 2×2 and symmetric, positive-definite, $c \in \mathbf{R}^2$, and $b \in \mathbf{R}^2$? Describe your algorithm as precisely as possible. (You are welcome to provide a matlab code.) Draw the ellipsoid

$$E = \{4x_1^2 + 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 3 \leq 1\}$$

F. BLUE property of least-squares. Consider a system of linear equations in vector x : $Ax = y + v$, where v is a noise vector, and the input is $A \in \mathbf{R}^{m \times n}$, a full rank, tall matrix ($m \geq n$), and $y \in \mathbf{R}^m$. We do not know anything about v , except that it is bounded: $\|v\|_2 \leq \alpha$, with $\alpha \geq 0$ a measure of the level of noise. Our goal is to provide an estimate \hat{x} of x via a *linear estimator*, that is, a function $\hat{x} = By$ with B a $n \times m$ matrix. We restrict attention to *unbiased* estimators, which are such that $\hat{x} = x$ when $v = 0$. This implies that B should be a left inverse of A , that is, $BA = I_n$. A example of linear estimator is obtained by solving the least-squares problem

$$\min_x \|Ax - y\|_2$$

The solution is, when A is full column rank, of the form $x^* = B_{\text{ls}}y$, with $B_{\text{ls}} = (A^T A)^{-1} A^T$. We note that $B_{\text{ls}}A = I$, which means that the LS estimator is unbiased. In this exercise, we show that B_{ls} is the best unbiased linear estimator. (This is often referred to as the BLUE property.)

1. Show that the estimation error of an unbiased linear estimator is $x - \hat{x} = -Bv$.
2. This motivates us to minimize the size of B , say using the [Frobenius norm](#):

$$\min_B \|B\|_F : BA = I.$$

Show that B_{ls} is the best unbiased linear estimator (BLUE), in the sense that it solves the above problem. Hint: Show that any unbiased linear estimator B can be written as $B = B_{\text{ls}} + Z$ with $ZB_{\text{ls}}^T = 0$, and that $BB^T = ZZ^T + B_{\text{ls}}B_{\text{ls}}^T$ is positive semi-definite.

G. Rational-quadratic function. We consider the function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, with values on its domain $\mathcal{X} := \{x : \|x\|_2 \leq 1\}$ given by

$$f(x) = \frac{x^T B x}{1 - x^T A x}.$$

Here, $A, B \in \mathcal{S}^n$ are given symmetric matrices, with $\lambda_{\max}(A) < 1$. In this exercise, we consider the problem

$$f_{\max} := \max_{x \in \mathcal{X}} f(x).$$

1. Show that indeed f is well-defined everywhere on \mathcal{X} .
2. Is the above problem convex, as stated? Proof or counter-example.
3. Show that the problem can be expressed as a problem in one variable, namely

$$f_{\max} = \min_t t : \lambda_{\max}(tA + B) \leq t.$$

Hint: for given $t \geq 0$, express the condition $f(x) \leq t$ for every $x \in \mathcal{X}$, in simple terms.

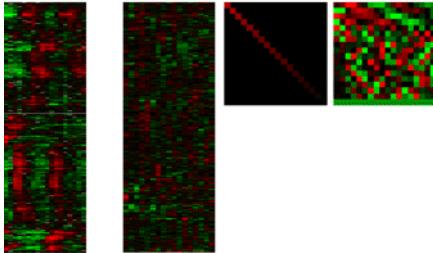
4. Show that the objective value further reduces to

$$f_{\max} = \lambda_{\max}((I - A)^{-1/2}B(I - A)^{-1/2}).$$

5. Explain how to recover an optimal point \underline{x} .

Singular Value Decomposition of General Matrices

$$A = U \cdot W \cdot V^T$$



The singular value decomposition (SVD) generalizes the spectral theorem (available for a square, symmetric matrix), to any non-symmetric, and even rectangular, matrix. The SVD allows to describe the effect of a matrix on a vector (via the matrix-vector product), as a three-step process: a first rotation in the input space; a simple positive scaling that takes a vector in the input space to the output space; and another rotation in the output space. The figure on the left shows the SVD of a matrix of biological data.

The SVD allows to analyze matrices and associated linear maps in detail, and solve a host of special optimization problems, from solving linear equations to linear least-squares. It can also be used to reduce the dimensionality of high-dimensional data sets, by approximating data matrices with low-rank ones. This technique is closely linked to the [principal component analysis](#) method.

The SVD theorem

The SVD theorem

Basic idea. Recall from [here](#) that any matrix $A \in \mathbf{R}^{m \times n}$ with rank one can be written as $A = \sigma u v^T$, where $u \in \mathbf{R}^m, v \in \mathbf{R}^n$, and $\sigma > 0$.

$$A = \sum_{i=1}^r \sigma_i u_i v_i^T,$$

It turns out that a similar result holds for matrices of arbitrary rank r . That is, we can express any matrix $A \in \mathbf{R}^{m \times n}$ with rank one as sum of rank-one matrices u_1, \dots, u_r are mutually orthogonal, v_1, \dots, v_r are also mutually orthogonal, and the σ_i 's are positive numbers called the *singular values* of A . In the above, r turns out to be the rank of A .

Theorem statement. The following important result applies to any matrix A , and allows to understand the structure of the mapping $x \rightarrow Ax$.

Theorem: Singular Value Decomposition (SVD)

An arbitrary matrix $A \in \mathbf{R}^{m \times n}$ admits a decomposition of the form

$$A = \sum_{i=1}^r \sigma_i u_i v_i^T = U \tilde{S} V^T, \quad \tilde{S} := \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix},$$

where $U \in \mathbf{R}^{m \times m}, V \in \mathbf{R}^{n \times n}$ are both orthogonal matrices, and the matrix S is diagonal: $S = \text{diag}(\sigma_1, \dots, \sigma_r)$, where the positive numbers $\sigma_1 \geq \dots \geq \sigma_r > 0$ are unique, and are called the *singular values* of A . The number $r \leq \min(m, n)$ is equal to the rank of A , and the triplet (U, \tilde{S}, V) is called a *singular value decomposition (SVD)* of A . The first r columns of U : $u_i, i = 1, \dots, r$ (resp. V : $v_i, i = 1, \dots, r$) are called *left (resp. right) singular vectors* of A , and satisfy $Av_i = \sigma_i u_i, \quad u_i^T A = \sigma_i v_i, \quad i = 1, \dots, r$.

The proof of the theorem hinges on the spectral theorem for symmetric matrices. Note that in the theorem, the zeros appearing alongside S are really *blocks* of zeros. They may be empty, for example if $r = n$, then there are no zeros to the right of S .

Computing the SVD. The SVD of a $m \times n$ matrix A can be easily computed via a sequence of linear transformations. The complexity of the algorithm, expressed roughly as the number of floating point operations per seconds it requires, grows as $O(nm \min(n, m))$. This can be substantial for large, dense matrices. For sparse matrices, we can speed up the computation if we are interested only in the largest few singular values and associated singular vectors. *Matlab syntax:*

```
>> [U, Stilde, V] = svd(A); % this produces the SVD of A, with Stilde of same size as A
>> [Uk, Sk, Vk] = svds(A, k); % the k largest singular values of A, assuming A is sparse
```

Example: [A 4 × 5 example](#).

Geometry

The theorem allows to decompose the action of A on a given input vector as a three-step process. To get Ax , where $x \in \mathbf{R}^n$, we first form $\tilde{x} := V^T x \in \mathbf{R}^n$. Since V is an orthogonal matrix, V^T is also orthogonal, and \tilde{x} is just a rotated version of x , which still lies in the input space. Then we act on the rotated vector \tilde{x} by scaling its elements. Precisely, the first r elements of \tilde{x} are scaled by the singular values $\sigma_1, \dots, \sigma_r$; the remaining $n - r$ elements are set to zero. This step results in a new vector \tilde{y} which now belongs to the output space \mathbf{R}^m . The final step consists in rotating the vector \tilde{y} by the orthogonal matrix U , which results in $y = U\tilde{y} = Ax$.

Example: Assume A has the simple form

$$A = \begin{pmatrix} 1.3 & 0 \\ 0 & 2.1 \\ 0 & 0 \end{pmatrix},$$

then for an input vector x in \mathbf{R}^2 , Ax is a vector in \mathbf{R}^3 with first component $1.3x_1$, second component $2.1x_2$, and last component zero.

To summarize, the SVD theorem states that any matrix-vector multiplication can be decomposed as a sequence of three elementary transformations: a rotation in the input space, a scaling that goes from the input space to the output space, and a rotation in the output space. In contrast with symmetric matrices, input and output directions are different.

The interpretation allows to make a few statements about the matrix.

Example: [A \$4 \times 4\$ example.](#)

Link with the [spectral theorem](#)

If A admits an SVD, then the matrices AA^T and A^TA has the following SEDs: $AA^T = U\Lambda_m U^T$, $A^TA = V\Lambda_n V^T$, where $\Lambda_m := \tilde{S}\tilde{S}^T = \text{diag}(\sigma_1^2, \dots, \sigma_r^2, 0, \dots, 0)$ is $m \times m$ (so it has $m - r$ trailing zeros), and $\Lambda_n := \tilde{S}^T\tilde{S} = \text{diag}(\sigma_1^2, \dots, \sigma_r^2, 0, \dots, 0)$ is $n \times n$ (so it has $n - r$ trailing zeros). The eigenvalues of AA^T and A^TA are the same, and equal to the squared singular values of A . The corresponding eigenvectors are the left and right singular vectors of A .

This is a method (not the most computationally efficient) to find the SVD of a matrix, based on the SED.

Matrix Properties via SVD

Nulspace Finding a basis for the nullspace. The SVD allows to compute an orthonormal basis for the nullspace of a matrix. To understand this, let us first consider a matrix of the form

$$\tilde{S} = \begin{pmatrix} 1.3 & 0 & 0 \\ 0 & 2.1 & 0 \end{pmatrix},$$

The nullspace of this matrix is readily found by solving the equation $\tilde{S}\tilde{x} = 0$. We obtain that $\tilde{x} = (x_1, x_2, x_3)$ is in the nullspace if and only if the first two components of \tilde{x} are zero: $x_1 = x_2 = 0$.

What about a general matrix A , which admits the SVD as given in the [SVD theorem](#)? Since U is orthogonal, we can pre-multiply the nullspace equation $Ax = 0$ by U^T , and solve in terms of the "rotated" variable $\tilde{x} := V^T x$. We obtain the condition on \tilde{x} :

$$0 = \tilde{S}\tilde{x} = (\sigma_1\tilde{x}_1, \dots, \sigma_r\tilde{x}_r, 0, \dots, 0).$$

The above is equivalent to the first r components of \tilde{x} being zero. Since $x = V\tilde{x}$, this corresponds to the fact that x belongs to the span of the last $n - r$ columns of V . Note that these columns form a set of mutually orthogonal, normalized vectors that span the nullspace: hence they form an [orthonormal basis](#) for it.

Theorem: [nullspace via SVD](#)

The nullspace of a matrix A with SVD

$$A = U\tilde{S}V^T, \quad \tilde{S} := \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix}, \quad S = \text{diag}(\sigma_1, \dots, \sigma_r),$$

where $U \in \mathbf{R}^{m \times m}, V \in \mathbf{R}^{n \times n}$ are both orthogonal matrices, admits the last $n - r$ columns of V as an orthonormal basis.

Example: [Nullspace of a \$4 \times 5\$ matrix](#).

Full column-rank matrices. One-to-one (or, full column rank) matrices are the matrices with nullspace reduced to $\{0\}$. If the dimension of the nullspace is zero, then we must have $n = r$. Thus, full

column rank matrices are ones with SVD of the form $A = U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T$. Range, rank via the SVD Basis of the range. As with the nullspace, we can express the range in terms of the SVD of the matrix A . Indeed the range of A is the set of vectors of the form

$$Ax = U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T x,$$

where $x \in \mathbf{R}^n$. Since V is orthogonal, when x spans \mathbf{R}^n , so does $\tilde{x} := V^T x$. Decomposing the latter vector in two sub-vectors $\tilde{x} = (\tilde{x}_r, \tilde{x}_{n-r})$, we obtain that the range is the set of vectors $U\tilde{y}$, with

$$\tilde{y} = \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \tilde{x}_r \\ \tilde{x}_{n-r} \end{pmatrix} = \begin{pmatrix} S\tilde{x}_r \\ 0 \end{pmatrix},$$

where \tilde{x}_r is an arbitrary vector of \mathbf{R}^r . Since S is invertible, $z := S\tilde{x}_r$ also spans \mathbf{R}^r . We obtain that the range is the set of vectors $U\tilde{y}$, where \tilde{y} is of the form $(z, 0)$ with $z \in \mathbf{R}^r$ arbitrary. This means that the range is the span of the first r columns of the orthogonal matrix U , and that these columns form an orthonormal basis for it. Hence, the number r of dyads appearing in the SVD decomposition is indeed the rank (dimension of the range).

Theorem: [range and rank via SVD](#).

The range of a matrix A with SVD $A = U\tilde{S}V^T$, $\tilde{S} = \text{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0)$ where $U \in \mathbf{R}^{m \times m}, V \in \mathbf{R}^{n \times n}$ are both orthogonal matrices, admits the first r columns of U as an orthonormal basis.

Full row rank matrices. An onto (or, full row rank) matrix has a range $r = m$. These matrices are characterized by an SVD of the form $A = U \begin{pmatrix} S & 0 \end{pmatrix} V^T$.

Example: [Range of a \$4 \times 5\$ matrix](#).

Fundamental theorem of linear algebra

The theorem already mentioned [here](#) allows to decompose any vector into two orthogonal ones, the first in the nullspace of a matrix A , and the second in the range of its transpose. This theorem will be useful for writing [optimality conditions for linearly constrained optimization problems](#).

Theorem: [Fundamental theorem of linear algebra](#)

Let $A \in \mathbf{R}^{m \times n}$. The sets $\mathbf{N}(A)$ and $\mathbf{R}(A^T)$ form an orthogonal decomposition of \mathbf{R}^n , in the sense that any vector $x \in \mathbf{R}^n$ can be written as

$x = y + z$, $y \in \mathbf{N}(A)$, $z \in \mathbf{R}(A^T)$, $y^T z = 0$. In particular, we obtain that the condition on a vector x to be orthogonal to any vector in the nullspace implies that it must be in the range: $x^T y = 0$ whenever $Ay = 0 \iff \exists \lambda \in \mathbf{R}^m : x = A^T \lambda$.

Proof.

Matrix norms, condition number

[Matrix norms](#) are useful to measure the size of a matrix. Some of them can be interpreted in terms of input-output properties of the corresponding linear map; for example, the Frobenius norm measure

the average response to unit vectors, while the largest singular (LSV) norm measures the peak gain. These two norms can be easily read from the SVD.

Frobenius norm. The Frobenius norm can be defined as $\|A\|_F = \sqrt{\text{Tr} A^T A}$. Using the SVD (U, \tilde{S}, V) of A , we obtain

$$\|A\|_F^2 = \text{Tr}(V \tilde{S}^T \tilde{S} V^T) = \text{Tr}(V^T V \tilde{S}^T \tilde{S}) = \text{Tr}(\tilde{S}^T \tilde{S}) = \sum_{i=1}^r \sigma_i^2.$$

Hence the squared Frobenius norm is nothing else than the sum of the squares of the singular values.

Largest singular value norm. An alternate way to measure matrix size is based on asking the maximum ratio of the norm of the output to the norm of the input. When the norm used is the Euclidean norm, the corresponding quantity

$$\|A\|_{LSV} := \max_x \|Ax\|_2 : \|x\|_2 \leq 1.$$

is called the *largest singular value* (LSV) norm. The reason for this wording is given by the following theorem.

Theorem: *largest singular value norm.*

For any matrix A , we have

$$\|A\|_{LSV} = \max_{x : \|x\|_2 \leq 1} \|Ax\|_2 = \sigma_1(A),$$

where $\sigma_1(A)$ is the largest singular value of A . Any left singular vector associated with the largest singular value achieves the maximum in the above.

Example: [Norms of a \$4 \times 5\$ matrix](#).

Condition number. The *condition number* of an invertible $n \times n$ matrix A is the ratio between the largest and the smallest singular values:

$$\kappa(A) = \frac{\sigma_1}{\sigma_n} = \|A\| \cdot \|A^{-1}\|.$$

As seen [here](#), this number provides a measure of the *sensitivity* of the solution of a linear equation to changes in A .

Solving Linear Equations via SVD

Solution set

Consider a [linear equation](#) $Ax = y$, where $A \in \mathbf{R}^{m \times n}$ and $y \in \mathbf{R}^m$ are given. We can completely describe the set of solutions via SVD, as follows. Let us assume that A admits an SVD given [here](#). With $A = U \tilde{S} V^T$, pre-multiply the linear equation by the inverse of U, U^T ; then we express the equation in terms of the rotated vector $\tilde{x} = V^T x$. This leads to

$$\tilde{S} \tilde{x} = \tilde{y},$$

where $\tilde{y} := U^T y$ is the “rotated” right-hand side of the equation. Due to the simple form of \tilde{S} , the above writes

$$\sigma_i \tilde{x}_i = \tilde{y}_i, \quad i = 1, \dots, r, \quad 0 = \tilde{y}_i, \quad i = r + 1, \dots, m.$$

Two cases can occur.

- If the last $m - r$ components of \tilde{y} are not zero, then the above system is infeasible, and the solution set is empty. This occurs when y is not in the range of A .
- If y is in the range of A , then the last set of conditions in the above system hold, and we can solve for \tilde{x} with the first set of conditions: $\tilde{x}_i = \frac{\tilde{y}_i}{\sigma_i}, \quad i = 1, \dots, r$. The last $n - r$ components of \tilde{x} are free. This corresponds to elements in the nullspace of A . If A is full column rank (its nullspace is reduced to $\{0\}$), then there is a *unique* solution.

Pseudo-inverse Definition. The solution set is conveniently described in terms of the *pseudo-inverse* of A , denoted by A^\dagger , and defined via the SVD of A :

$$A = U \tilde{S} V^T, \quad \tilde{S} = \text{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0) \quad \text{as one with same SVD, with non-zero singular values inverted, and the matrix } \tilde{S} \text{ transposed: } A^\dagger = V \tilde{S}^\dagger U^T, \quad \text{with} \\ \tilde{S}^\dagger = \text{diag}(\sigma_1^{-1}, \dots, \sigma_r^{-1}, 0, \dots, 0) \in \mathbf{R}^{m \times n}.$$

The pseudo-inverse of a matrix is always well-defined, and that it has the same size as the transpose A^T . When the matrix is invertible (it is square and full column or row rank: $m = n = r$), then it reduces to the inverse.

Example: [pseudo-inverse of a \$4 \times 5\$ matrix](#).

Matlab syntax

```
>> Adagger = pinv(A);
```

Link with solution set. From the above development, we see that the solution set can be written as $\mathbf{S} = \{A^\dagger y + z : z \in \mathcal{N}(A)\}$, where $\mathcal{N}(A)$ is the nullspace of A . Both A^\dagger and a basis for the nullspace can be computed via the SVD. Case when A is full rank. If A is full column rank, the pseudo-inverse can be written as $A^\dagger = (A^T A)^{-1} A^T$. In that case, A^\dagger is a *left-inverse* of A , since $A^\dagger A = I_m$.

If A is full row-rank, then the pseudo-inverse can be written as $A^\dagger = A^T (A A^T)^{-1}$. In that case, A^\dagger is a *right-inverse* of A , since $A A^\dagger = I_m$.

Sensitivity analysis and condition number

Sensitivity analysis refers to the process of quantifying the impact of changes in the linear equations’ coefficients (the matrix A and vector y), on the solution. To simplify, let us assume that A is square and invertible, and analyze the effects of errors in y only. The *condition number* of the matrix A quantifies this.

We start from the linear equation above, which has the unique solution $x = A^{-1} y$. Now assume that y is changed into $y + \delta y$, where δy is a vector that contains the changes in y . Then let us denote by $x + \delta x$ the new solution, which is $A^{-1}(y + \delta y)$. From the equations $\delta x = A^{-1} \delta y$, and via the definition of the largest singular value norm, we obtain:

$$\|\delta x\|_2 \leq \|A^{-1}\| \cdot \|\delta y\|_2, \quad \|y\|_2 \leq \|A\| \cdot \|x\|_2.$$

Combining the two inequalities we further obtain:

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq \kappa(A) \cdot \frac{\|\delta y\|_2}{\|y\|_2},$$

where $\kappa(A)$ is the condition number of A , defined as $\kappa(A) := \|A\| \cdot \|A^{-1}\|$. We can express the condition number as the ratio between the largest and smallest singular values of A :

$$\kappa(A) = \frac{\sigma_1(A)}{\sigma_n(A)}.$$

The condition number gives a bound on the ratio between the relative error in the left-hand side to that of the solution. We can also analyze the effect of errors in the matrix itself on the solution. The condition number turns out to play a crucial role there as well.

Least-squares and SVD

Set of solutions

The following theorem provides all the solutions (optimal set) of a least-squares problem.

Theorem: optimal set of ordinary least-squares ([Proof](#)).

The optimal set of the OLS problem

$$p^* := \min_x \|Ax - y\|_2$$

can be expressed as $\mathbf{X}^{\text{opt}} = A^\dagger y + \mathbf{N}(A)$, where A^\dagger is the *pseudo-inverse* of A , and $A^\dagger y$ is the minimum-norm point in the optimal set. If A is full column rank, the solution is unique, and equal to $x^* = A^\dagger y = (A^T A)^{-1} A^T y$. In general, the particular solution $A^\dagger y$ is the minimum-norm solution to the least-squares problem.

Sensitivity analysis

We consider the situation where $y + \delta y = Ax$, with

- $A \in \mathbf{R}^{m \times n}$ the data matrix (known), with A full column rank (hence $m \geq n$).
- $y \in \mathbf{R}^m$ is the measurement (known).
- $x \in \mathbf{R}^n$ is the vector to be estimated (unknown).
- $\delta y \in \mathbf{R}^m$ is a measurement noise or error (unknown).

We can use OLS to provide an estimate \hat{x}_{OLS} of x . The idea is to seek the smallest vector δy such that the above equation becomes feasible, that is,

$$\min_{x, \delta y} \|\delta y\|_2 : y + \delta y = Ax.$$

This leads to the OLS problem:

$$\min_x \|Ax - y\|_2.$$

Since A is full column rank, its SVD can be expressed as

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T,$$

where $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_m)$ contains the singular values of A , with $\sigma_1 \geq \dots \geq \sigma_m > 0$.

Since A is full column rank, the solution \hat{x}_{OLS} to the OLS problem is unique, and can be written as a linear function of the measurement vector y : $\hat{x}_{\text{OLS}} = A^\dagger y$, with A^\dagger the pseudo-inverse of A . Again, since A is full column rank,

$$A^\dagger = (A^T A)^{-1} A^T = V \begin{pmatrix} \Sigma^{-1} & 0 \end{pmatrix} U^T.$$

The OLS formulation provides an estimate \hat{x} of the input x such that the residual error vector $Ax - y$ is minimized in norm. We are interested in analyzing the impact of perturbations in the vector y , on the resulting solution \hat{x}_{OLS} . We begin by analyzing the absolute errors in the estimate, and then turn to the analysis of relative errors.

Set of possible errors

Let us assume a simple model of potential perturbations: we assume that δy belongs to a unit ball: $\|\delta y\|_2 \leq \alpha$, where $\alpha > 0$ is given. We will assume $\alpha = 1$ for simplicity; the analysis is easily extended to any $\alpha > 0$.

We have

$$\begin{aligned} \delta x := x - \hat{x} &= x - A^\dagger y \\ &= x - A^\dagger(Ax - \delta y) \\ &= (I - A^\dagger A)x + A^\dagger \delta y \\ &= A^\dagger \delta y. \end{aligned}$$

In the above we have exploited the fact that A^\dagger is a left inverse of A , that is, $A^\dagger A = I_n$.

The set of possible errors on the solution δx is then given by $\mathbf{E} = \{A^\dagger \delta y : \|\delta y\|_2 \leq 1\}$, which is an ellipsoid centered at zero, with principal axes given by the singular values of A^\dagger . This ellipsoid can be interpreted as an *ellipsoid of confidence* for the estimate \hat{x} , with size and shape determined by the matrix $A^\dagger (A^\dagger)^T$.

We can draw several conclusions from this analysis:

- The largest absolute error in the solution that can result from a unit-norm, additive perturbation on \mathbf{y} is of the order of $1/\sigma_m$, where σ_m is the smallest singular value of \mathbf{A} .
- The largest relative error is σ_1/σ_m , the condition number of \mathbf{A} .

BLUE property

We now return to the case of an OLS with full column rank matrix \mathbf{A} .

Unbiased linear estimators. Consider the family of linear estimators, which are of the form $\hat{\mathbf{x}} = \mathbf{B}\mathbf{y}$, where $\mathbf{B} \in \mathbf{R}^{n \times m}$. To this estimator, we associate the error $\delta\mathbf{x} = \mathbf{x} - \hat{\mathbf{x}} = \mathbf{x} - \mathbf{B}\mathbf{y} = \mathbf{x} - \mathbf{B}(\mathbf{A}\mathbf{x} - \delta\mathbf{y}) = (\mathbf{I} - \mathbf{B}\mathbf{A})\mathbf{x} + \mathbf{B}\delta\mathbf{y}$. We say that the estimator (as determined by matrix \mathbf{B}) is *unbiased* if the first term is zero: $\mathbf{B}\mathbf{A} = \mathbf{I}$. Unbiased estimators only exist when the above equation is feasible, that is, \mathbf{A} has a [left inverse](#). This is equivalent to our condition that \mathbf{A} be full column rank. Since \mathbf{A}^\dagger is a left-inverse of \mathbf{A} , the OLS estimator is a particular case of an unbiased linear estimator. Best unbiased linear estimator. The above analysis leads to the following question: which is the best unbiased linear estimator? One way to formulate this problem is to assume that the perturbation vector $\delta\mathbf{y}$ is bounded in some way, and try to minimize the possible impact of such bounded errors on the solution.

Let us assume that $\delta\mathbf{y}$ belongs to a unit ball: $\|\delta\mathbf{y}\|_2 \leq 1$. The set of resulting errors on the solution $\delta\mathbf{x}$ is then

$$\mathbf{E} = \{\mathbf{B}\delta\mathbf{y} : \|\delta\mathbf{y}\|_2 \leq 1\},$$

which is an ellipsoid centered at zero, with principal axes given by the singular values of \mathbf{B} . This ellipsoid can be interpreted as an ellipsoid of confidence for the estimate $\hat{\mathbf{x}}$, with size and shape determined by the matrix $\mathbf{B}\mathbf{B}^T$.

It can be shown that the OLS estimator is optimal in the sense that it provides the “smallest” ellipsoid of confidence among all unbiased linear estimators. Specifically:

$$\mathbf{B}\mathbf{B}^T \succeq \mathbf{A}^\dagger(\mathbf{A}^\dagger)^T.$$

This optimality of the LS estimator is referred to as the BLUE (Best Linear Unbiased Estimator) property.

Low-rank approximation of a matrix

Low-rank approximations

We consider a matrix $\mathbf{A} \in \mathbf{R}^{m \times n}$, with SVD given as in the [SVD theorem](#): $\mathbf{A} = \mathbf{U}\tilde{\mathbf{S}}\mathbf{V}^T$, $\tilde{\mathbf{S}} = \text{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0)$, where the singular values are ordered in decreasing order, $\sigma_1 \geq \dots \geq \sigma_r > 0$. In many applications it can be useful to approximate \mathbf{A} with a low-rank matrix.

Example: Assume that \mathbf{A} contains the log-returns of n assets over m time periods, so that each column of \mathbf{A} is a time-series for a particular asset. Approximating \mathbf{A} by a rank-one matrix of the form pq^T , with $p \in \mathbf{R}^m$ and $q \in \mathbf{R}^n$ amounts to model the assets’ movements as all following the same pattern given by the time-profile p , each asset’s movements being scaled by the components in q . Indeed, the (t, i) component of \mathbf{A} , which is the log-return of asset i at time t , then expresses as $p(t)q(i)$.

We consider the *low-rank approximation* problem:

$$\min_X \|A - X\|_F : \text{rank}(X) = k,$$

where k ($1 \leq k < r = \text{rank}(A)$) is given. In the above, we measure the error in the approximation using the Frobenius norm; using the largest singular value norm leads to the same set of solutions X .

Theorem: Low-rank approximation.

A best k -rank approximation $\hat{\mathbf{A}}_k$ is given by zeroing out the $r - k$ trailing singular values of \mathbf{A} , that is $\hat{\mathbf{A}}_k = \mathbf{U}\hat{\mathbf{S}}_k\mathbf{V}^T$, $\hat{\mathbf{S}}_k = \text{diag}(\sigma_1, \dots, \sigma_k, 0, \dots, 0)$. The minimal error is given by the Euclidean norm of the singular values that have been zeroed out in the process:

Sketch of proof: The proof rests on the fact that the Frobenius norm, is invariant by rotation of the input and output spaces, that is, $\|U^T B V\|_F = \|B\|_F$ for any matrix B , and orthogonal matrices U, V of appropriate sizes. Since the rank is also invariant, we can reduce the problem to the case when $A = \tilde{S}$.

Example: [Low-rank approximation of a \$4 \times 5\$ matrix](#).

Link with Principal Component Analysis

Principal Component Analysis operates on the covariance matrix of the data, which is proportional to $\mathbf{A}\mathbf{A}^T$, and sets the principal directions to be the eigenvectors of that (symmetric) matrix. As noted [here](#), the eigenvectors of $\mathbf{A}\mathbf{A}^T$ are simply the left singular vectors of \mathbf{A} . Hence, both methods, the above approximation method and PCA, rely on the same tool, the SVD. The latter is a more complete approach as it also provides the eigenvectors of $\mathbf{A}^T\mathbf{A}$, which can be useful if we want to analyze the data in terms of rows instead of columns.

In particular, we can express the [explained variance](#) directly in terms of the singular values. In the context of visualization, the explained variance is simply the ratio to the total amount of variance in the projected data, to that in the original. More generally, when we are approximating a data matrix by a low-rank matrix, the explained variance compares the variance in the approximation to that in the original data. We can also interpret it geometrically, as the ratio of squared norm of the approximation matrix to that of the original matrix:

$$\frac{\|\hat{\mathbf{A}}_k\|_F^2}{\|\mathbf{A}\|_F^2} = \frac{\sigma_1^2 + \dots + \sigma_k^2}{\sigma_1^2 + \dots + \sigma_n^2}.$$

Applications of SVD

- [Image compression](#).
- [Market data analysis](#).
- [Latent Semantic Indexing](#) (LSI) for web document search (see also [here](#)).
- [Political spectrum analysis](#).
- [3D image deformation using moving least-squares](#).
- [SVD and PCA for gene expression data](#).

Exercises SVD of simple matrices

A. Consider the matrix

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

1. Find the range, nullspace, and rank of A .
2. Find an SVD of A .
3. Determine the set of solutions to the linear equation $Ax = y$, with

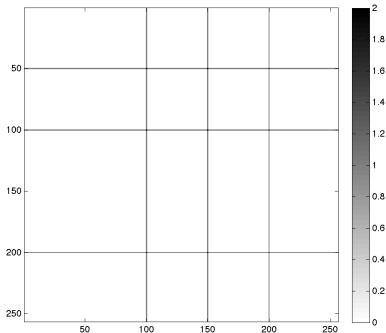
$$y = \begin{pmatrix} 2 \\ 3 \\ 0 \\ 0 \end{pmatrix}, \quad y = \begin{pmatrix} 2 \\ 0 \\ 0 \\ 3 \end{pmatrix}.$$

B. Consider the 2×2 matrix

$$A = \frac{1}{\sqrt{10}} \begin{pmatrix} 2 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \end{pmatrix} + \frac{2}{\sqrt{10}} \begin{pmatrix} -1 \\ 2 \end{pmatrix} \begin{pmatrix} 1 & 1 \end{pmatrix}.$$

1. What is an SVD of A ? Express it as $A = USV^T$, with S the diagonal matrix of singular values ordered in decreasing fashion. Make sure to check all the properties required for U, S, V .
2. Find the semi-axis lengths and principal axes (minimum and maximum distance and associated directions from \mathbf{E} to the center) of the ellipsoid $\mathbf{E}(A) = \{Ax : x \in \mathbf{R}^2, \|x\|_2 \leq 1\}$. Hint: Use the SVD of A to show that every element of $\mathcal{E}(A)$ is of the form $y = U\bar{y}$ for some element \bar{y} in $\mathcal{E}(S)$. That is, $\mathcal{E}(A) = \{U\bar{y} : \bar{y} \in \mathcal{E}(S)\}$. (In other words the matrix U maps $\mathcal{E}(S)$ into the set $\mathcal{E}(A)$.) Then analyze the geometry of the simpler set $\mathcal{E}(S)$.
3. What is the set $\mathcal{E}(A)$ when we append a zero vector after the last column of A , that is A is replaced with $\tilde{A} = (A, 0) \in \mathbf{R}^{2 \times 3}$?
4. Same question when we append a row after the last row of A , that is, A is replaced with $\tilde{A} = [A^T, 0]^T \in \mathbf{R}^{3 \times 2}$. Interpret geometrically your result.

Rank and SVD



C. The image on the left shows a 256×256 matrix A of pixel values. The lines indicate $+1$ values; at each intersection of lines the corresponding matrix element is $+2$. All the other elements are zero.

1. Show that for some permutation matrices P, Q , the permuted matrix $B := PAQ$ has the symmetric form $B = pq^T + qp^T$, for two vectors p, q . Determine P, Q, B and p, q .
2. What is the rank of A ? Hint: find the nullspace of B .
3. Find an SVD of A . Hint: Find an eigenvalue decomposition of S , using the results of an exercise on eigenvalue [here](#).

Procrustes problem

D. The [Orthogonal Procrustes problem](#) is a problem of the form

$$\min_X \|AX - B\|_F : X^T X = I_p,$$

where $\|\cdot\|_F$ denotes the Frobenius norm, and the matrices $A \in \mathbf{R}^{m \times n}, B \in \mathbf{R}^{m \times p}$ are given. Here, the matrix variable $X \in \mathbf{R}^{n \times p}$ is constrained to have orthonormal columns. When $n = m = p$, the problem can be interpreted geometrically as seeking a transformation of points (contained in A) to other points (contained in B) that involves only rotation.

1. Show that the solution of the Procrustes problem above can be found via the SVD of the matrix $A^T B$.

2. Derive a formula for the answer to the constrained least-squares problem $\min_x \|Ax - b\|_2 : \|x\|_2 = 1$, with $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^m$ given.

SVD and projections

E. We consider a set of m data points $x_i \in \mathbf{R}^n, i = 1, \dots, m$. We seek to find a line in \mathbf{R}^n such that the sum of the squares of the distances from the points to the line is minimized. To simplify, we assume that the line goes through the origin.

1. Consider a line that goes through the origin $\mathcal{L} := \{tu : t \in \mathbf{R}\}$, where $u \in \mathbf{R}^n$ is given. (You can assume without loss of generality that $\|u\|_2 = 1$.) Find an expression for the projection of a given point x on \mathcal{L} .
2. Now consider the m points and find an expression for the sum of the squares of the distances from the points to the line \mathcal{L} .
3. Explain how you would find the line via the SVD of the $n \times m$ matrix $X = [x_1, \dots, x_m]$.
4. How would you address the problem without the restriction that the line has to pass through the origin?
5. Solve the same problems as previously by replacing the line by an hyperplane.

SVD and least-squares

F. Consider the matrix A formed as $A = (c_1, c_2, c_3)$, with $c_1 = (1, 2, 8), c_2 = (3, 6, 9), c_3 = c_1 - 4c_2 + \epsilon_1 v, \epsilon_1 = .0000001$, with v a vector chosen randomly in $[-1/2, 1/2]^4$. In addition we define $y = 2c_1 - 7c_2 + \epsilon_2 w, \epsilon_2 = .0001$, with again w chosen randomly in $[-1/2, 1/2]^4$. We consider the associated least-squares problem

$$\min_x \|Ax - y\|_2.$$

1. What is the rank of \bar{A} ?
2. Apply the least-squares formula $x^* = (\bar{A}^T \bar{A})^{-1} \bar{A}^T y$. What is the norm of the residual vector, $r = Ax - y$?
3. Express the least-squares solution in terms of the SVD of \bar{A} . That is, form the [pseudo-inverse](#) of \bar{A} and apply the formula $x^* = \bar{A}^\dagger y$. What is now the norm of the residual?
4. Interpret your results.

G. Consider a least-squares problem

$$\min_x \|Ax - y\|_2$$

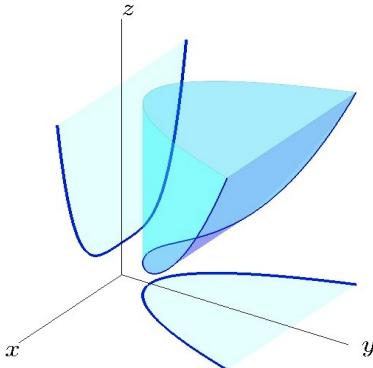
where the data matrix $A \in \mathbf{R}^{m \times n}$ has rank one.

1. Is the solution unique?
2. Show how to reduce the problem to one involving one scalar variable.
3. Express the set of solutions in closed-form.

3. Convex Models

- 3.1 [Convex optimization](#)
- 3.2 [Linear and quadratic programming](#)
- 3.3 [Second-order cone programming](#)
- 3.4 [Robust linear programming](#)
- 3.5 [Geometric programming](#)
- 3.6 [Semi-definite programming](#)
- 3.7 [Some non-convex models](#)

Convex Optimization



The [ordinary least-squares](#) problem can be solved using linear algebra methods. It turns out that we can confidently use this approach in an *iterative algorithm*, to globally minimize “bowl-shaped”, or convex, functions, under convex set constraints.

We first define precisely what is meant by convex sets and functions, and outline a few simple rules that can help in determining if a set or function is convex. The picture on the left illustrates one of these rules, the fact that the [projection of a convex set is convex](#). Then we will define convex optimization problems and single out broad classes of problems that fall in the convex optimization category.

We provide the main ideas behind some convex optimization algorithms, including the so-called interior-point and gradient methods. Our final focus is on *disciplined convex programming* (DCP), a subclass of convex optimization that corresponds exactly to the problems amenable to the software package [CVX](#).

Convex Sets Definitions

A subset C of \mathbf{R}^n is said to be *convex* if and only if it contains the line *segment* between any two points in it:

[Convex sets](#)

Subspaces and affine sets, such as lines, planes and higher-dimensional “flat” sets, are obviously convex, as they contain the entire *line* passing through any two points, not just the line segment. That is, there is no restriction on the scalar λ anymore in the above condition.

Examples:

- [A convex and a non-convex set](#).
- [Convex and conic hull of a set of points](#).

A set is said to be a *convex cone* if it is convex, and has the property that if , then for every $\alpha \geq 0$.

Operations that preserve convexity Intersection. The intersection of a (possibly infinite) family of convex sets is convex. This property can be used to prove convexity for a wide variety of situations.

Examples:

- [The second-order cone](#).
- [The semidefinite cone](#).

Affine transformation. If a map $\boxed{\quad}$ is affine, and \mathbf{C} is convex, then the set $\boxed{\quad}$

is convex. In particular, the projection of a convex set on a subspace is convex.

Example: [Projection of a convex set on a subspace](#).

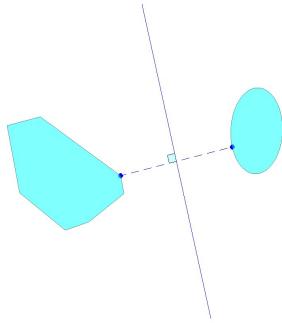
Separation theorems

Separation theorems are one of the most important tools in convex optimization. They convey the intuitive idea that two convex sets that do not intersect can be separated by a straight line.

There are many versions of separation theorems. One of them is the separating hyperplane theorem:

Theorem: Separating hyperplane

If $\mathbf{C}, \boxed{\quad}$ are two convex subsets of \mathbf{R}^n that do not intersect, then there is an hyperplane that separates them, that is, there exist $a \in \mathbf{R}^n, a \neq 0$, and $b \in \mathbf{R}$, such that $a^T x \leq b$ for every $\boxed{\quad}$ and $a^T x \geq b$ for every $\boxed{\quad}$

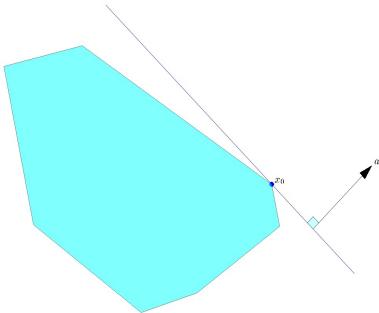


When two convex sets do not intersect, it is possible to find a hyperplane that separates them. In two dimensions, we can picture the hyperplane as a straight line. The proof of the separation theorem relies on finding the two points in each set that are closest to each other.

Another result involves the separation of a set from a point on its boundary:

Theorem: Supporting hyperplane

If $\boxed{\quad}$ is convex and non-empty, then for any x_0 at the boundary of \mathbf{C} , there exist a supporting hyperplane to \mathbf{C} at x_0 , meaning that there exist $a \in \mathbf{R}^n, a \neq 0$, such that $\boxed{\quad}$ for every $\boxed{\quad}$



Convex Functions Definition Domain of a function. The *domain* of a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is the set $\boxed{\quad}$ over which f is well-defined, in other words: $\boxed{\quad}$

Examples:

- The function with values $\boxed{\quad}$ has domain $\boxed{\quad}$.
- The function with values $\boxed{\quad}$ has domain $\boxed{\quad}$ (the set of positive-definite matrices).

Definition of convexity. A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is convex if

1. Its domain $\text{dom } f$ is convex;
2. In addition,

$\forall x, y \in \text{dom } f, \forall \theta \in [0,1]: f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$.

Note that the convexity of the domain is required. For example, the function $f : \mathbf{R} \rightarrow \mathbf{R}$ defined as

$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$

is *not* convex, although it is linear (hence, convex) on its domain $\mathbf{R} \setminus \{0\}$.

We say that a function is *concave* if $-f$ is convex.

Example. The function $f : \mathbf{R} \rightarrow \mathbf{R}$ defined as

$f(x) = \begin{cases} x^2 & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases}$

is convex. However, the function with domain the whole line except $\{0\}$ is not, since that domain is not convex.

Alternate characterizations of convexity

Let $f : \mathbf{R}^n \rightarrow \mathbf{R}$. The following are equivalent conditions for f to be convex.

Epigraph. A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is convex if and only if its *epigraph* $\text{epi } f := \{(x, t) \in \mathbf{R}^{n+1} : t \geq f(x)\}$

is convex.

Example: We can use this result to prove for example, that the largest eigenvalue function $\lambda_{\max}(X)$, which to a given $n \times n$ symmetric matrix X associates its largest eigenvalue, is convex, since the condition $\lambda_{\max}(X) \geq \lambda_{\max}(Y)$ is equivalent to the condition that $\lambda_{\max}(X - Y) \geq 0$.

First-order condition. If f is differentiable (that is, $\text{dom } f$ is open and the gradient exists everywhere on the domain), then f is convex if and only if ∇f is monotone increasing.

The geometric interpretation is that the graph of f is bounded below everywhere by anyone of its tangents.

Restriction to a line. The function f is convex if and only if its restriction to /any/ line is convex, meaning that for every $x_0 \in \mathbf{R}^n$, and $v \in \mathbf{R}^n$, the function $t \mapsto f(x_0 + tv)$ is convex. Note that the "if" part is a direct consequence of the "composition with an affine function" result below.

Example. [Convexity of the log-determinant function](#).

Second-order condition. If f is twice differentiable, then it is convex if and only if its Hessian $\nabla^2 f$ is positive semi-definite everywhere on the domain of f . This is perhaps the most commonly known characterization of convexity.

Examples:

- [Convexity of a quadratic function](#).
- [Convexity of the square-to-linear function](#).
- [Convexity of the log-sum-exp function](#).

Operations that preserve convexityComposition with an affine function. The composition with an affine function preserves convexity: if $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^m$ and $f : \mathbf{R}^m \rightarrow \mathbf{R}$ is convex, then the function $x \mapsto f(Ax + b)$ with values $f(Ax + b)$ is convex. Point-wise maximum. The *pointwise maximum* of a family of convex functions is convex: if $\{f_\alpha\}_{\alpha \in \mathcal{A}}$ is a family of convex functions indexed by \mathcal{A} , then the function $x \mapsto \max_{\alpha \in \mathcal{A}} f_\alpha(x)$ is convex. This is one of the most powerful ways to prove convexity.

Examples:

- *Dual norm*: for a given norm, we define the *dual norm* as the function

$\|x\|_* = \sup_{\|y\|=1} |y^T x|$

This function is convex, as the maximum of convex (in fact, linear) functions (indexed by the vector y). The dual norm earns its name, as it satisfies the properties of a [norm](#).

- *Largest singular value of a matrix*: Another example is the [largest singular value](#) (see also [here](#)) of a matrix A :

$\|A\|_{\text{F}} = \sqrt{\lambda_{\max}(A^T A)}$

Here, each function (indexed by $x \in \mathbf{R}^n$) $\|Ax\|$ is convex, since it is the composition of the Euclidean norm (a convex function) with an affine function $x \mapsto Ax$.

Nonnegative weighted sum. The nonnegative weighted sum of convex functions is convex.

Example: [Negative entropy function](#).

Partial minimum. If f is a convex function in $x = (y, z)$, then the function $x \mapsto \min_y f(x)$ is convex. (Note that joint convexity in x is essential.)

Example: case when f is quadratic: the [Schur complement lemma](#).

Composition with monotone convex functions. The composition with another function does not always preserve convexity. However, if $f = h \circ g$, with g convex and h increasing, then f is convex.

Indeed, the condition $\nabla g \geq 0$ is equivalent to the existence of y such that

$\nabla g(x) = \nabla g(x + y) - \nabla g(y)$

The condition above defines a convex set in the space of y -variables. The epigraph of f is thus the projection of that convex set on the space of (x, t) -variables, hence it is convex.

Example: [proving convexity via monotonicity](#).

More generally, if the functions $g_i : \mathbf{R}^n \rightarrow \mathbf{R}$, $i = 1, \dots, k$ are convex and non-decreasing in each argument, with $\nabla g_i \geq 0$, then

$\nabla(g_1 \circ \dots \circ g_k)(x) = \nabla g_1(x) + \dots + \nabla g_k(x)$

is convex.

For example, if \square 's are convex, then \square also is.

Convex Optimization Problems Definition

The optimization problem in [standard form](#):

is called a *convex optimization problem* if:

- the objective function f_0 is [convex](#);
- the functions defining the inequality constraints, $f_i, i = 1, \dots, m$ are convex;
- the functions defining the equality constraints, $h_i, i = 1, \dots, m$ are [affine](#).

Note that, in the convex optimization model, we do not tolerate equality constraints, unless they are affine.

Optimality Local and global optima. Recall the definition of global and local optima given [here](#).

Theorem: Global vs. local optima in convex optimization ([Proof](#))

For convex problems, any locally optimal point is globally optimal. In addition, the optimal set is convex.

Optimality condition. Consider the optimization problem above, which we write as \square

where \mathbf{X} is the feasible set.

When f_0 is differentiable, then we know that for every \square ,

Then x is optimal if and only if

$$x \in \mathbf{X} \text{ and } \forall y \in \mathbf{X} : \nabla f_0(x)^T(y - x) \geq 0.$$

If \square , then it defines a [supporting hyperplane](#) to the feasible set at x .

When the problem is unconstrained, we obtain the optimality condition:

Note that these conditions are not always feasible, since the problem may not have any minimizer. This can happen for example when the optimal value is only attained in the limit; or, in constrained problems, when the feasible set is empty.

Examples:

- [Minimization of a convex quadratic function](#).
- [Optimality condition for a convex, unconstrained problem](#).
- [Optimality condition for a convex, linearly constrained problem](#).

The optimality conditions given above might be hard to solve. We will return to this issue later.

Maximization of convex functions

Sometimes we would like to *maximize* a convex function over a set \mathbf{S} . Such problems are usually hard to solve.

For example, the problem of maximizing the distance from a given point (say, $\mathbf{0}$) to a point in a polyhedron described as \square is an example of such hard problems.

One important property of convex functions is that their maximum over any set is the same as the maximum over the [convex hull](#) of that set. That is, for any set $\mathbf{S} \subseteq \mathbf{R}^n$ and any convex function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, we have

In the example mentioned above, where we seek to maximize the Euclidean norm from a point (say, the origin) to a point in a polyhedron, if we know the vertices of the polyhedron, that is, we can express the polyhedron as

then the optimal distance is simply \square .

Unfortunately, in practice, finding the vertices (given the original representation of \mathbf{P} as an intersection of hyperplanes) is hard, as the polytope might have an exponential number of vertices.

Algorithms for Convex Optimization From Least-Squares to convex minimization

We have seen how [ordinary least-squares](#) (OLS) problems can be solved using linear algebra (e.g. [SVD](#)) methods. Using OLS, we can minimize convex, quadratic functions of the form

$$q(x) = \frac{1}{2}x^T H x + g^T x$$

when $H = H^T \succeq 0$. This last requirement ensures that the function q is convex. For such convex quadratic functions, as for any convex functions, any local minimum is global. In fact, when $H \succ 0$, then the unique minimizer is $x^* = -H^{-1}g$.

It turns out one can leverage the approach to minimizing more general functions, using an iterative algorithm, based on a *local quadratic* approximation of the function at the current point. The approach can then be extended to problems with constraints, by replacing the original constrained problem with an unconstrained one, in which the constraints are *penalized* in the objective.

Unconstrained minimization: Newton's method

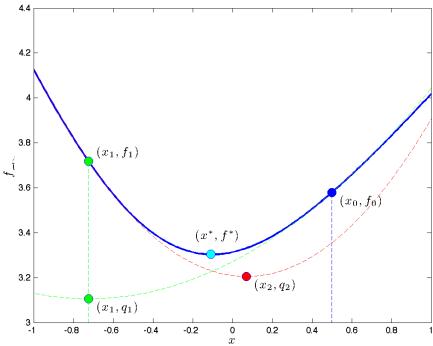
We consider an unconstrained minimization problem, where we seek to minimize a function twice-differentiable function f . Let us assume that the function under consideration is *strictly convex*, which is to say that its Hessian is positive definite everywhere. (If f is not convex, we might run into a [local minima](#).)

For minimizing convex functions, an iterative procedure could be based on a simple quadratic approximation procedure known as Newton's method. We start with initial guess x_0 . At each step t , we update our current guess x_t by minimizing the second-order approximation \tilde{f} of f at x_t , which is the quadratic function (see [here](#))

$$\tilde{f}(x) := f(x_t) + \nabla f(x_t)^T(x - x_t) + \frac{1}{2}(x - x_t)^T \nabla^2 f(x_t)(x - x_t),$$

where $\nabla f(x_t)$ denotes the [gradient](#), and $\nabla^2 f(x_t)$ the [Hessian](#) of f at x_t . Our next guess x_{t+1} will be set to be a solution to the problem of minimizing \tilde{f} . Since the function is strictly convex, we have $\nabla^2 f(x_0) \succ 0$, so that the problem we are solving at each step has a unique solution, which corresponds to the global minimum of \tilde{f} . The basic Newton iteration is thus

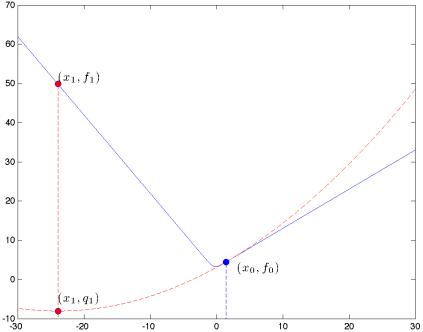
$$x_{t+1} = x_t - (\nabla^2 f(x_t))^{-1} \nabla f(x_t)$$



Two initial steps of Newton's method to minimize the function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$ with domain the whole \mathbf{R}^2 , and values $f(x) = \log(\exp(x-3) + \exp(-2x+2))$.

A first local quadratic approximation at the initial point $x_0 = 0.5$ is formed (dotted line in green). The corresponding minimizer is the new iterate, x_1 . The Newton algorithm proceeds to form a new quadratic approximation of the function at that point (dotted line in red), leading to the second iterate, x_2 . Although x_1 turns out to be further away from the global minimizer x^* (in light blue), x_2 is closer, and the method actually converges quickly.

This idea will fail for general (non-convex) functions. It might even fail for some convex functions. However, for a large class of convex functions, known as *self-concordant functions*, a variation on the Newton method works extremely well, and is guaranteed to find the global minimizer of the function f . For such functions, the Hessian does not vary too fast, which turns out to be a crucial ingredient for the success of Newton's method.



Failure of the Newton method to minimize the above convex function. The initial point is chosen too far away from the global minimizer x^* , in a region where the function is almost linear. As a result, the quadratic approximation is almost a straight line, and the Hessian is close to zero, sending the first iterate of Newton's method to a relatively large negative value. The method quickly diverges in this case, with a second iterate at $x_2 \approx 2e^{30}$.

Constrained minimization: interior-point methods

The method above can be applied to the more general context of convex optimization problems of standard form:

$$\min_x f_0(x) \text{ subject to } f_i(x) \leq 0, \quad i = 1, \dots, m,$$

where every function involved is twice-differentiable, and convex.

The basic idea behind interior-point methods is to replace the constrained problem by an unconstrained one, involving a function that is constructed with the original problem functions. One further idea is to use a *logarithmic barrier*: in lieu of the original problem, we address

$$\min_x f(x) := f_0(x) - \mu \sum_{i=1}^m \log(-f_i(x)),$$

where $\mu > 0$ is a small parameter. The function f turns out to be convex, as long as f_0, \dots, f_m are.

For μ large, solving the above problem results in a point well “inside” the feasible set, an “interior point”. As $\mu \rightarrow 0$ the solution converges to a global minimizer for the original, constrained problem. In fact, the theory of convex optimization says that if we set $\mu = m/\epsilon$, then a minimizer to the above function is ϵ -suboptimal. In practice, algorithms do not set the value of μ so aggressively, and update the value of μ a few times.

For a large class of convex optimization problems, the function f is self-concordant, so that we can safely apply Newton's method to the minimization of the above function. In fact, for a large class of convex optimization problems, the method converges in time polynomial in m .

The interior-point approach is limited by the need to form the gradient and Hessian of the function f above. For extremely large-scale problems, this task may be too daunting.

Gradient methods unconstrained case. Gradient methods offer an alternative to interior-point methods, which is attractive for large-scale problems. Typically, these algorithms need a considerably larger number of iterations compared to interior-point methods, but each iteration is much cheaper to process.

Perhaps the simplest algorithm to minimizing a convex function involves the iteration

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t),$$

where $\alpha_t > 0$ is a parameter. The interpretation of the algorithm is that it tries to decrease the value of the function by taking a step in the direction of the negative gradient. For small enough value of α_t , indeed we have $f(x_{t+1}) \leq f(x_t)$.

Depending on the choice of the parameter α_t (as a function of the iteration number t), and some properties on the function f , convergence can be rigorously proven.

Constrained case. The gradient method can be adapted to constrained problems, via the iteration $x_{t+1} = P(x_t - \alpha_t \nabla f(x_t))$,

where P is the projection operator, which to its argument z associates the point closest (in Euclidean norm sense) to z in \mathbf{C} . Depending on problem structure, this projection may or may not be easy to perform.

Example: In the "sign-constrained least-squares" problem:

$$\min_x \|Ax - y\|_2^2 : x \geq 0,$$

the above algorithm involves the iterations

$$x_{t+1} = \max(0, x_t - \alpha_t A^T (Ax_t - y)).$$

Disciplined Convex Programming and CVX

If an optimization problem turns out to be convex, special-purpose algorithms can be used to solve it efficiently. The difficulty for the modeler is that proving convexity of a problem is a very hard task in general. There is no magic tool that allows a user to plug in any optimization problem and hope that the tool recognizes convexity and exploits it in the solution method.

In *disciplined convex programming*, the set of problems that are allowed is restricted: problems are constructed as convex from the outset. Hence, convexity verification is automatic. In the software package [CVX](#), the objective and constraint functions that are allowed are built from a library of basic examples of convex functions. Calculus rules or transformations then allow to handle functions beyond the ones in the library.

The ruleset

The ruleset corresponds to the [operations that preserve convexity](#), and includes the following.

- *Sum*: if f and g are convex, so is
- *Affine composition*: if f is convex, and A is a matrix and b a vector of compatible size, so is
- *Pointwise maximum*: if are convex, so is
- *Partial minimization*: if is convex, and C is convex, then the function g with values

is convex. (See for example the [case](#) when f is quadratic and convex.)

- *Composition*: if h is convex and increasing, and f is convex, then is convex (there are several similar composition rules).

For example, if CVX encounters a constraint of the form

with f_1, f_2 convex functions in the library, then it adds two new convex constraints

$i = 1, 2$,

and replaces the original constraint with This way, only "basic" constraints are to be handled when solving the problem.

Example

Consider the problem of [sparse linear least-squares](#):

where $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^m$ are given. A CVX code for solving this is almost a verbatim version of the above:

CVX syntax

```
% input: mxn data matrix A, m-vector y
% output: optimal solution x in Rn
cvx_begin
    variable x(n,1)
    minimize( (A*x-y)' * (A*x-y) + norm(x, 1) )
cvx_end
```

CVX recognizes the first term as an affine composition involving the squared function , which is part of the library. The second term is an l_1 -norm, and is also part of the library. CVX knows how to handle this norm, by introducing a new variable , replacing the norm by , and adding the constraints $i = 1, \dots, n$. The problem is replaced with the equivalent form

Using an [interior-point method](#) amounts to handle the inequality constraints via a "barrier" term in the objective. The problem is further replaced with

where $\mu > 0$ is a small parameter. A final step is to add extra variables to as to make sure the objective only contains basic functions:

The problem above has linear equality constraints only. It involves an objective function given as a sum of functions that are in the library only. Hence, its gradient and hessian (the objects needed to

implement an interior-point method) can be easily derived from those of functions in the library.

Some do's and don'ts of CVX

Consider the following variation on the problem above:

where the l_1 -norm is squared.

We can be tempted to write the above problem in CVX as follows.

A wrong CVX implementation

```
cvx_begin
    variable x(n,1)
    minimize( (A*x-y) ** (A*x-y) + norm(x,1)^2 )
cvx_end
```

However, CVX will fail with this expression of the problem, as it cannot recognize the convexity of the squared l_1 -norm function as written. The reason is that the square of a convex function is only convex if that function is non-negative. To square a convex function that is non-negative, we use the `square_pos` function.

CVX implementation

```
cvx_begin
    variable x(n,1)
    minimize( (A*x-y) ** (A*x-y) + square_pos(norm(x,1)) )
end
cvx_end
```

Pros and cons of CVX

This approach does restrict the available models:

- We can't just call a convex optimization solver, hoping for the best; convex optimization is not a "plug & play" or "try my code" method;
- we can't just type in a problem description, hoping it's convex, and that a sophisticated analysis tool will recognize it.

The good news are:

- by learning and following a modest set of atoms and rules, we can specify a problem in a form very close to its natural mathematical form.
- We can simultaneously verify convexity of problem, automatically generate an equivalent problem that is compatible with an interior-point method solver.

ExercisesConvexity A. show that, for x, y both positive scalars

Justify carefully your approach. Use the above result to prove that the function f defined as

is concave.

B. Show that for any numbers x_1, \dots, x_n , we have

C. Consider the set defined by the following inequalities

1. Draw the set. Is it convex?
2. Show that it can be described as a single quadratic inequality of the form for matrix and $c \in \mathbf{R}$ which you will determine.
3. What is the convex hull of this set?

D. Prove Young's inequality, valid for every non-negative numbers a, b , and integers p, q with , :

Hint: use the convexity of the exponential function.

Simple optimization problems

E. Solve the optimization problems below: determine the optimal value and the optimal set.

1. subject to , .
2. subject to , .
3. subject to .
4. subject to .
5. subject to .

F. Consider the optimization problem

Make a sketch of the feasible set. For each of the following objective functions, give the optimal set and the optimal value.

1.
2.

3.
 4.

G. Show the following results.

- For any $A \in \mathbf{R}$, and $\kappa > 0$:

$$\psi_1(A, \kappa) := \max_{\gamma \geq 0} A\gamma - \frac{\kappa}{2}\gamma^2 = \frac{(A_+)^2}{2\kappa},$$

with unique optimal point $\gamma^* = A_+ / \kappa$, where $A_+ := \max\{A, 0\}$.

- For $A, B \in \mathbf{R}$, and $\kappa > 0$, define

$$\psi_2(A, B, \kappa) := \max_{p \geq 0, q \geq 0} Ap + Bq - \frac{\kappa}{2}(q-p)^2.$$

Show that $\psi_2(A, B, \kappa) = +\infty$ if $A + B > 0$, otherwise:

$$\psi_2(A, B, \kappa) = \max(\psi_1(A, \kappa), \psi_1(B, \kappa)) = \frac{1}{2\kappa} \max(0, A, B)^2.$$

Hint: Add one scaling variable and an equality constraint to the problem:

$$p = \gamma \bar{p}, q = \gamma \bar{q},$$

where $\gamma \geq 0$, $\bar{q} - \bar{p} = \pm 1$.

- For the second problem above, determine an optimal pair (p^*, q^*) . Is it unique?

H. We have to decide which items to sell among of a collection of n items, with given selling prices $s_i > 0$, $i = 1, \dots, n$. Selling implies a transaction cost $c_i > 0$ for each item i ; the total transaction cost is the sum of these costs, plus a fixed amount $d > 0$, say, $d = 1$. We are interested in the following decision problem: which items need to be sold, in order to maximize the margin (revenue-to-cost ratio).

1. Show that the problem can be formalized as

$$p^* := \max_{x \in \{0,1\}^n} f(x), \quad f(x) := \frac{s^T x}{1 + c^T x}.$$

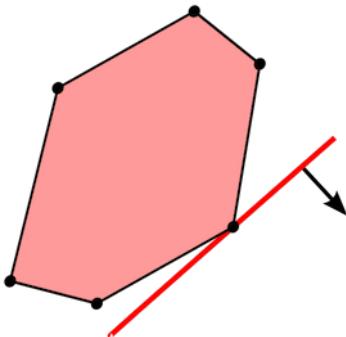
2. Show that the problem admits a convex formulation:

$$p^* = \min_{t \geq 0} t : t \geq \sum_{i=1}^n \max(0, s_i - tc_i).$$

Hint: for given $t \geq 0$, express the condition: $f(x) \leq t$ for every $x \in \{0,1\}^n$, in simple terms.

3. How can you recover an optimal solution x^* , from an optimal value t^* for the above problem?

Linear and Quadratic Programming



A linear program (LP) is an optimization problem in standard form, in which all the functions involved are affine. The feasible set is thus a **polyhedron**, that is, an intersection of half-spaces. Polyhedral functions are functions with a polyhedral epigraph, and include maxima or sums of maxima of linear or affine functions. Such functions can be minimized via LP.

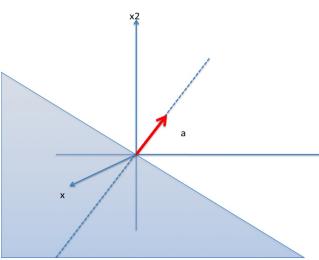
Quadratic programs (QPs) offer an extension of linear programs, in which all the constraint functions involved are affine, and the objective is the sum of a linear function and a positive semi-definite quadratic form. QPs generalize both LPs and ordinary least-squares: the objective function is the same as in ordinary least-squares, and the problem includes polyhedral constraints, just as in LP.

PolyhedraHalf-spacesDefinition. A **half-space** is a set defined by a single affine inequality. Precisely, a half-space in \mathbf{R}^n is a set of the form $\mathbf{H} = \{x : a^T x \leq b\}$,

where $a \in \mathbf{R}^n, b \in \mathbf{R}$. A half-space is a [convex set](#), the boundary of which is a [hyperplane](#).

A half-space separates the whole space in two halves. The complement of the half-space is the open half-space $\{x : a^T x > b\}$.

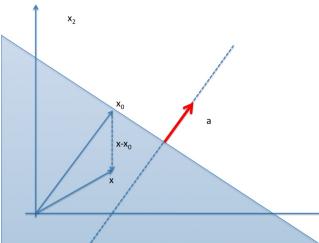
Geometry. A half-space separates the whole space in two halves. The complement of the half-space is the open half-space $\{x : a^T x > b\}$.



When $b = 0$, the half-space

$$\mathbf{H} = \{x : a^T x \leq 0\},$$

is the set of points which form an obtuse angle (between 90° and 270°) with the vector a . The boundary of this set is a subspace, the hyperplane of vectors orthogonal to a .



When $b \neq 0$, the half-space \mathbf{H} can be written as

$$\mathbf{H} = \{x : a^T(x - x_0) \leq 0\},$$

where x_0 is chosen such that $a^T x_0 = b$. For example, $x_0 = b/\|a\|_2^2$ is such a point on the boundary of the half-space (this particular choice corresponds to the [minimum-norm solution](#) to the equation $a^T x = b$). Thus, the half-space above corresponds to the set of points such that $x - x_0$ (shown in dotted) forms an obtuse angle with the vector a . The vector a points outwards from the boundary.

Example: [A half-space in \$\mathbf{R}^2\$](#) .

Link with linear functions

Hyperplanes correspond to [level sets](#) of linear functions.

Half-spaces represent [sub-level sets](#) of linear functions: the half-space above describes the set of points such that the linear function $x \rightarrow a^T x$ achieves the value b , or less. A quick way to check which half of the space the half-space describes is to look at where the origin is: if $b \geq 0$, then $x = 0$ is in the half-space.

Polyhedra Definition. A *polyhedron* is a set described by finitely many affine inequalities. Precisely, a polyhedron is a set of the form

$$\mathbf{P} = \{x : a_i^T x \leq b_i, i = 1, \dots, m\},$$

where $a_i \in \mathbf{R}^n, b_i \in \mathbf{R}, i = 1, \dots, m$.

A polyhedron can be expressed as the intersection of (finitely many) half-spaces:

$$\mathbf{P} = \bigcap_{i=1}^m \{x : a_i^T x \leq b_i\},$$

Geometry. A polyhedron is a convex set, with boundary made up of "flat" boundaries (the technical term is facet). Each facet corresponds to one of the hyperplanes defined by . The vectors a_i are orthogonal to the facets, and point outside the polyhedra.

Note that not every set with flat boundaries can be represented as a polyhedron: the set has to be convex.

Matrix notation. It is often convenient to describe a half-space in matrix notation: $\mathbf{P} = \{x : Ax \leq b\}$,

where $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$ are defined as follows:

$$A = \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix}, b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}.$$

Here, we adopted the [component-wise inequality](#) convention: the notation $Ax \leq b$ means that for every $i = 1, \dots, m$, the corresponding components of Ax and b are ordered: $(Ax)_i = a_i^T x \leq b_i$.

Example: [A polyhedron in \$\mathbf{R}^2\$](#) .

Equality constraints are allowed. Sets defined by affine inequalities and equalities are also polyhedra.

Indeed, consider the set

$$\mathbf{P} = \{x : Ax \leq b, Cx = d\},$$

where $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^m$ define the (component-wise) inequalities, and $C \in \mathbf{R}^{p \times n}, d \in \mathbf{R}^p$ define the equalities.

The set above can be expressed as an "inequalities-only" polyhedron:

$$\mathbf{P} = \{x : Ax \leq b, Cx \leq d, -Cx \leq -d\},$$

which can be put in the standard form for polyhedra, with augmented matrices and right-hand side vector.

Example: [The probability simplex](#).

Standard Forms of Linear and Quadratic Programming
Linear and quadratic programming problems
A *linear program* (or LP, for short) is an optimization problem with linear objective and affine inequality constraints. In the standard form introduced [here](#): $\min_x f_0(x) : f_i(x) \leq 0, i = 1, \dots, m$,

the objective function f_0 , and the constraint functions $f_i, i = 1, \dots, m$, are all [affine](#). (Since affine functions are linear plus constant, and constant terms in the objective do not matter, we can always assume that f_0 is actually linear.)

Examples:

- [A drug production problem](#).

- [A cash-flow management problem.](#)

Quadratic programs. A *quadratic program* (or QP, for short) is an optimization problem in the standard form above, where:

- the *constraint* functions $f_i, i = 1, \dots, m$, are all [affine](#), as in LP;
- the *objective* function f_0 is quadratic convex, that is, its values can be expressed as

$$f_0(x) = c^T x + x^T Q x$$

for some vector $c \in \mathbf{R}^n$ and $Q = Q^T \succeq 0$ (Q is [positive-semidefinite](#): it is [symmetric](#), and everyone of its eigenvalues is non-negative). LPs are special cases of QPs, in which the matrix Q is zero.

Example: [Projection on a polyhedron.](#)

Geometry. Each constraint in an LP is a single affine inequality in the decision vector x . Hence, each constraint says that the decision vector x should lie in a [half-space](#). Taken together, the constraints require that x should lie in the intersection of those half-spaces, that is, a [polyhedron](#).

An LP is to minimize a linear function over a polyhedron. Geometrically, this means “going as far as possible in the direction $-C$ ”, where C is the vector that defines the (linear) objective function. The geometry of a QP is to minimize a convex (bowl-shaped) quadratic function over a polyhedron.

Examples:

- [An LP in two variables.](#)
- [A QP in two variables.](#)

Standard forms Standard inequality form. As seen [here](#), a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is affine if and only if it can be expressed via the scalar product, as $f(x) = a^T x - b$

for some appropriate vector a and scalar b .

Hence, a linear program can be expressed as

$$\min_x c^T x : a_i^T x \leq b_i, \quad i = 1, \dots, m,$$

for some appropriate vectors $c, a_1, \dots, a_m \in \mathbf{R}^n$ and scalars $b_1, \dots, b_m \in \mathbf{R}$.

We can use the [component-wise inequality convention](#) and put the above problem into the *inequality standard form*:

$$\min_x c^T x : Ax \leq b,$$

where

$$A := \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix} \in \mathbf{R}^{m \times n}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \mathbf{R}^m.$$

Similarly, for QP a standard form is

$$\min_x c^T x + x^T Q x : Ax \leq b, \quad \text{Equality constraints. In LP or QP models, we can have equality constraints as well. For example, the LP } \min_x c^T x : Ax \leq b, \quad Cx = d,$$

where $C \in \mathbf{R}^{p \times n}, d \in \mathbf{R}^p$ define the equality constraints, can be put in standard inequality form, as

$$\min_x c^T x : Ax \leq b, \quad Cx \leq d, \quad -Cx \leq -d. \quad \text{Conic form. A problem of the form } \min_x c^T x : Ax = b, \quad x \geq 0$$

is an LP. Conversely, any LP can be put in the above form ([Proof](#)). A similar result holds for QP. We call this representation a “conic” form. The term “conic” comes from the fact that the above problem is part of a more general class known as conic problems, in which the sign constraints on the variables are replaced with $x \in \mathbf{K}$, where \mathbf{K} is a convex cone.

The above form is useful to develop theory and algorithms for LP, as it puts all the “data” of the problem into the linear objective and the linear *equality* constraints, while the inequalities have a very simple, data-independent structure.

CVX implementation Basic implementation. Consider the QP with equality and inequality constraints $\boxed{\quad}$

In CVX, we use component-wise inequalities to specify the affine inequality constraints. Note that equality constraints need to be coded with the double inequality sign $\boxed{\quad}$

CVX implementation

```
cvx_begin
    variable x(n,1)
    minimize( c'*x+x'*Q*x )
    subject to
        A*x <= b;
        Cx == d;
cvx_end
pstar = cvx_optval; % optimal value of the problem
```

CVX will only solve the problem if Q is positive semi-definite; otherwise, it will fail.

Some do's and don'ts of CVX. Consider the case when $\boxed{\quad}$, the identity. Then $\boxed{\quad}$. We can be tempted to write the above problem in CVX as follows. A wrong CVX implementation

```
cvx_begin
    variable x(n,1)
    minimize( c'*x+norm(x,2)^2 )
    subject to
        A*x <= b; Cx == d;
cvx_end
```

However, CVX will fail with this expression of the problem, as it cannot recognize the convexity of the squared norm function. The reason is that the square of a convex function is only convex if that

function is non-negative. To square a function that is non-negative, we use the `square_pos` function.

CVX implementation

```
cvx_begin
    variable x(n,1)
    minimize( c'*x+square_pos(norm(x,2)) )
    subject to
        A*x <= b; Cx == d;
cvx_end
```

Alternatively, we can use the function `sum_square`, replacing the last term in the objective in the above with `sum_square(x)`.

Minimizing Polyhedral Functions Polyhedral functions Definition. We say that a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is *polyhedral* if its [epigraph](#) is a polyhedron.

That is, a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is polyhedral if and only if the [epigraph](#)

$$\text{epi } f := \{(x, t) \in \mathbf{R}^{n+1} : t \geq f(x)\}$$

can be expressed as a polyhedron: there exist a matrix and a vector such that

Maxima of affine functions. Polyhedral functions include in particular, functions that can be expressed as a maximum of a finite number of affine functions:

where $a_i \in \mathbf{R}^n$, $b_i \in \mathbf{R}$, $i = 1, \dots, m$. Indeed, the epigraph of f :

can be expressed as the polyhedron

Example. The l_∞ -norm function, with values , is polyhedral, as it can be written as the maximum of affine functions:

Sums of maxima of affine functions. Polyhedral functions include more general functions. For example, a function that can be expressed as a sum of functions themselves expressed as maxima of affine functions:

for appropriate vectors and scalars , is polyhedral.

Indeed, the condition is equivalent to the existence of a vector $u \in \mathbf{R}^p$ such that

Hence, is the projection (on the space of (x, t) -variables) of a polyhedron, which is itself a polyhedron. Note however that representing this polyhedron in terms of a set of affine inequalities involving (x, t) only, is complicated.

Example. The l_1 -norm function, with values , is polyhedral, as it can be written as the sum of maxima of affine functions:

Minimization of polyhedral functions

Using LPs, we can minimize polyhedral functions, under polyhedral constraints.

Indeed, consider the problem

with f polyhedral. We can solve the problem as the LP

Minimization of maxima of affine functions. For example, assume that f is defined as the maximum of linear functions as above. The problem

can be expressed as the LP (in variables $x \in \mathbf{R}^n$ and

The above problem is indeed an LP:

- The objective of the problem is linear in the variables (x, t) ;
- the constraints are ordinary inequalities involving affine functions.

Minimization of a sum of maxima of affine functions. We can formulate the problem of minimizing the function f with values

under polytopic constraints, as an LP.

We use the same trick as before, introducing a new variable for each max-linear function that appears in the function f . We obtain the LP representation

Example. [l₁- and l_∞-norm regression problems](#).

CVX implementation

In [CVX](#), there is no need to introduce new variables in order to transform a polyhedral function minimization. We can directly minimize the function expressed as a maximum of affine functions.

The snippet below implements the problem

where c_1, \dots, c_m are given vectors in \mathbf{R}^n . We assume that exists in the workspace.

CVX implementation
<pre>cvx_begin variable x(n,1) minimize(max(C'*x)) subject to A*x <= b; cvx_end</pre>

For minimizing l_1 - or l_∞ -norms, we can simply rely on the corresponding matlab functions, as they are overloaded in CVX. The following snippet solves

where $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$ exist in the workspace.

CVX implementation
<pre>cvx_begin variable x(n,1) minimize(norm(A*x-b,inf) + norm(x,1)) cvx_end</pre>

Cardinality minimization: the l_1 -norm trick

Many problems in engineering and scientific computing can be cast as

$$\min_x \mathbf{Card}(x) : x \in \mathbf{P},$$

where \mathbf{P} is a [polyhedron](#) (or, more generally, a convex set), and $\mathbf{Card}(x)$ denotes the [cardinality](#) (number of non-zero elements) of the vector x . Such problems seek a “sparse” solution, one with many zeroes in it.

A related problem is a penalized version of the above, where we seek to trade-off an objective function against cardinality:

$$\min_x f(x) + \lambda \mathbf{Card}(x) : x \in \mathbf{P},$$

where f is some (usually convex) “cost” function, and $\lambda > 0$ is a penalty parameter.

Cardinality minimization is a hard problem in general. In fact, the cardinality function is non-convex, as the one-dimensional case (x is a scalar) shows: the function is equal to one everywhere, except at zero, where it is zero. It appears in many areas, such as [classification](#).

The l_1 -norm heuristic

The l_1 -norm heuristic consists in replacing the (non-convex) cardinality function $\mathbf{Card}(x)$ with a polyhedral (hence, convex) one, involving the l_1 -norm. This heuristic leads to replace the problem at the top with

$$\min_x \|x\|_1 : x \in \mathbf{P},$$

which is an LP (provided \mathbf{P} is a polyhedron).

If \mathbf{P} is described via affine inequalities, as $\mathbf{P} = \{x : Ax \leq b\}$, with A a matrix and b a vector existing in the matlab workspace, then the following CVX snippet solves the above problem.

CVX implementation

cvx_begin
<pre>variable x(n,1) minimize(norm(x,1)) subject to Ax <= b; cvx_end</pre>

Why does it work?

The l_1 -norm heuristic often works well, especially if the variable x is bounded.

Often the variable is not bounded *a priori*, but its size is penalized, say by a squared l_2 -norm. Consider an optimization problem of the form

where is a given integer that bounds the number of non-zeroes in the variable. Invoking the [Cauchy-Schwartz inequality](#) between x and the vector y a n -vector with 1's where x is non-zero, and 0's elsewhere, we obtain (with the vector with elements , $i = 1, \dots, n$):

The above implies a lower bound on the Euclidean norm of any vector x with cardinality k :

$$\|x\|_2^2 \geq \frac{1}{k} \|x\|_1^2$$

Hence, instead of solving the difficult problem above, we can solve with the above *lower bound*:

The above problem is convex. In fact, it can be written as the QP, by adding new variables:

As often, CVX allows a direct implementation of the problem, but care must be taken to use the `square_pos` function to model the squared l_1 -norm, as discussed in [do's and don'ts of CVX](#).

CVX implementation

```
cvx_begin
    variable x(n)
    minimize( c'*x + (1/k^2)*square_pos(norm(x,1)) )
cvx_end
```

Applications

- [Piece-wise constant fitting](#).
- [Sparse classification](#).

Applications of LP and QP

- [Linear binary classification](#)
- [LP relaxations of boolean problems](#)
- [Network flows](#)
- [Portfolio optimization](#)
- [Filter design](#)

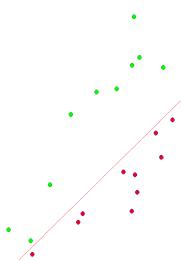
Linear classification

Consider the problem of finding a line which separates two data sets in two dimensions. These data sets may represent measurements about two kinds of populations (the first set might contain spam emails, while the second might contain legitimate ones). Each axis represents the frequency (say) of certain keywords in the email at hand— in our case, we look at two keywords, but in practice we might want to involve thousands of possible keywords.

We formalize this as follows. Each data point is given by its coordinates $\boxed{\quad}$ and has a label $y_i \in \{-1, 1\}$ which determines whether it is spam or not. A (possibly vertical) line in \mathbb{Z} -space, is parametrized by a vector $\boxed{\quad}$, via the equation $\boxed{\quad} = \boxed{\quad}$. Such a line is said to correctly classify these two sets if all data points with $y_i = +1$ fall on one side (hence $\boxed{\quad} > 0$) and all the others on the other side (hence $\boxed{\quad} < 0$). Hence, the affine inequalities on x

guarantee correct classification. The above can be used as constraints in an optimization problem, to derive an “optimal” line.

Once a line is found, a new point (for which we do not have a label) can be classified, by checking on which side of the line it falls. This is further discussed [here](#).



The image shows a line which separates two sets of points in \mathbb{R}^2 . An optimization problem can be formed to find such a line, even in more than two dimensions.

Exercises Standard forms

A. *Squared objective*. Consider the two problems

$\boxed{\quad}$

and

$\boxed{\quad}$

1. Show by a counterexample that these problems do not necessarily share the same set of optimal points.
2. Formulate the second problem as a linear program in standard form.
3. Show that if $\boxed{\quad}$ (that is, every component of C is non-negative), then the problems are equivalent.

B. *Least-squares and QP*. Consider a least-squares problem with linear inequality constraints

$\boxed{\quad}$

where A, C are given matrices (of row size equal to n , the number of variables) and $\boxed{\quad}$ are given vectors.

1. Write the problem as a QP in standard form.
2. Is the converse true? In other words, can we always transform a QP in standard form, into a problem with the form above? Discuss.

C. For given n -vectors , consider the problem

Proof or counter-example: the above problem is (a) an LP; (b) a QP; (c) An ordinary least-squares problem.

D. We consider, for $\lambda > 0$, $Q = Q^T$ a $n \times n$ positive-definite matrix, a n -vector c , and a $m \times n$ matrix A , the QP

$$\max_x c^T x - \frac{\lambda}{2} x^T Q x : Ax \leq 0$$

We would like to solve the problem for many values of $\lambda > 0$. We will make use of a second problem

$$\max_z c^T z : Az \leq 0, z^T Q z \leq 1.$$

1. Show that the optimal value of the problem is attained and unique. Denote it by $x(\lambda)$.
2. Show that the first problem can be reduced to the second problem. Make sure to clarify the relationship between z and x at optimum. Justify carefully your steps. Hint: set $x = tz$, with t a non-negative scalar, z a vector such that $z^T Q z = 1$, and optimize over $t \geq 0$.
3. Show that $x(\lambda) = x(1)/\lambda$ for any $\lambda > 0$, and discuss.

E. Formulating problems as LPs or QPs. Formulate the problem

$$p_j^* = \min_x f_j(x)$$

for different functions f_j , $j = 1, \dots, 5$, with values given in Table 2, as QPs or LPs, or, if you cannot, explain why. In our formulations, we always use $x \in \mathbf{R}^n$ as the variable, and assume that $A \in \mathbf{R}^{m \times n}$, $y \in \mathbf{R}^m$, and $k \in \{1, \dots, m\}$ are given. If you obtain an LP or QP formulation, make sure to put the problem in standard form, stating precisely what the variables, objective and constraints are.

$$\begin{aligned} f_1(x) &= \|Ax - y\|_\infty + \|x\|_1 \\ f_2(x) &= \|Ax - y\|_2^2 + \|x\|_1 \\ f_3(x) &= \|Ax - y\|_2^2 - \|x\|_1 \\ f_4(x) &= \|Ax - y\|_2^2 + \|x\|_2^2 \\ f_5(x) &= \sum_{i=1}^k |Ax - y|_{[i]} + \|x\|_2^2 \end{aligned}$$

Table 2: Table of the values of different functions f . Here, $|z|_{[i]}$ denotes the element in a vector z that has the i -th largest magnitude.

Applications

F. Computing the convex hull. In this exercise, you will devise an algorithm that, given a set of distinct n points $x_i \in \mathbf{R}^2$, $i = 1, \dots, n$, plots the convex hull \mathcal{C} of the points. This well-studied problem arises a lot in computational geometry, see [here](#). You will use the data matrix

where each column represents a point.

1. Find a point x_0 in the interior of the convex hull. Hint: think about the average of the points.
2. Show how to formulate the following problem as a linear optimization problem: given a direction v , find the largest t such that .
3. Using CVX, plot the convex hull of the points in the matrix X given above. Hint: use the previous part with many directions v .
4. Comment on the accuracy needed when you scan the possible angular directions.
5. Can you come up with a more efficient method to solve this problem? Discuss.

G. A control problem. We consider a dynamical system with one input and one output, and described by a linear, time-invariant relationship:

where is the order of the system, and is called the *impulse response*. We consider an input design problem, where we seek to find a sequence of inputs which tracks a given desired output signal over a given time horizon , with N given. Specifically, we seek inputs should be such that the following three conditions hold. (i) The peak deviation between $y(t)$ and

is minimized; (ii) the inputs should be zero for , where is given; (iii) the inputs should satisfy amplitude and slew rate constraints:

where , are given. Show that the problem can be formulated as a linear program. Write a CVX code that plots the results given problem data.

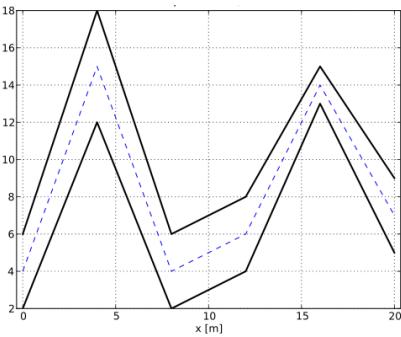
H. An advertising problem. A company wants to promote its newly developed product by launching an advertising campaign. There are four advertising options to choose from: TV Spot, Newspaper, Radio (prime time), and Radio (afternoon); these options are labelled T, N, P, A , respectively. The table below provides, for each type of advertising, the audience reached, the cost and maximum number of ads per week.

Advertising options	TV Spot (T)	Newspaper (N)	Radio (P) (prime time)	Radio (A) (afternoon)
Audience Reached (per ad)	5000	8500	2400	2800
Cost (per ad)	\$800	\$925	\$290	\$380
Max Ads (per week)	12	5	25	20

The company has a budget of \$8000 per week and seeks to maximize audience reached. However, the company also wants 5 or more radio spots per week and cannot spend more than \$1800 on radio per week. Let T, N, P, A be the decision variables corresponding to the numbers of ads chosen weekly by the company. (You can ignore the fact that these variables should be integers.)

Formulate this as a linear programming problem, making sure to incorporate all the constraints in the formulation.

I. A slalom problem. A two-dimensional skier must slalom down a slope, by going through parallel gates of known position (x_i, y_i) , and of width c_i , $i = 1, \dots, n$. The initial position (x_0, y_0) is given, as well as the final one, (x_{n+1}, y_{n+1}) . Here, the x-axis represents the direction down the slope, from left to right.

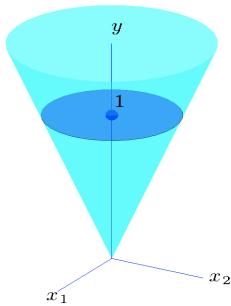


i	x_i	y_i	c_i
0	0	4	N/A
1	4	5	3
2	8	4	2
3	12	6	2
4	16	5	1
5	20	7	2
6	24	4	N/A

Geometry and parameters for the slalom problem.

1. Find the path that minimizes the total length of the path. Your answer should come in the form of an optimization problem.
2. Try solving the problem numerically, with the data given in the table above.

Second-Order Cone Programming



The second-order cone in \mathbf{R}^3 is the set of vectors (x_1, x_2, y) with

$$y \geq \sqrt{x_1^2 + x_2^2}.$$

The picture shows part of the cone, and a slice (at $y = 1$). This set is called an "ice-cream cone".

Second-order cone programming (SOCP) is a generalization of linear and quadratic programming that allows for affine combination of variables to be constrained inside second-order cones. The SOCP model includes as special cases problems with convex quadratic objective and constraints.

SOCP models are particularly useful in geometry problems, as well as in linear programs where the data is imprecisely known. We explore this further [here](#).

The second-order cone

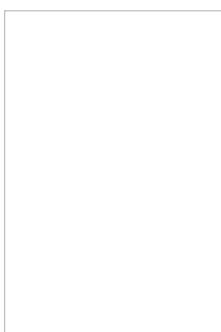
The second-order cone in \mathbf{R}^{p+1} is defined as [link](#)

$$\mathbf{K}_p := \{(x, y) \in \mathbf{R}^{p+1} : \|x\|_2 \leq y\}.$$

This set is convex, since it is the intersection of (an infinite number of) half-spaces:

$$\mathbf{K}_p = \bigcap_{u : \|u\|_2 \leq 1} \{(x, y) \in \mathbf{R}^{p+1} : x^T u \leq y\}.$$

It is a cone, since it is invariant by scaling; if $x \in \mathbf{K}_p$, so does αx for any $\alpha \geq 0$.



The second-order cone in \mathbf{R}^3 . The set contains points (x_1, x_2, y) such that

$$y \geq \sqrt{x_1^2 + x_2^2}.$$

The set actually extends to infinity upwards. When we "slice" the set by imposing $y = 1$, the corresponding intersection is the circle of center $(0, 0, 1)$ and radius one (in dark blue).

Example: [Magnitude constraints on affine complex vectors](#).

Rotated second-order cone

The rotated second-order cone in \mathbf{R}^{p+2} is the set

$$\mathbf{K}_p := \{(x, y, z) \in \mathbf{R}^{p+2} : x^T x \leq yz, \quad y \geq 0, \quad z \geq 0\}.$$

Note that the rotated second-order cone in \mathbf{R}^{p+2} can be expressed as a linear transformation (actually, a rotation) of the (plain) second-order cone in \mathbf{R}^{p+2} , since

$$\|x\|_2^2 \leq yz, \quad y \geq 0, \quad z \geq 0 \iff \left\| \begin{pmatrix} 2x \\ y-z \end{pmatrix} \right\|_2 \leq y+z.$$

This is, $(x, y, z) \in \mathbf{K}_p^r$ if and only if $(w, y+z) \in \mathbf{K}_{p+1}$, where $w = (2x, y-z)$. Indeed, the right-hand side is equivalent to

$$(y+z)^2 \geq 4x^T x + (y-z)^2, \quad y+z \geq 0,$$

which in turn is equivalent to $yz \geq x^T x$ and $y+z \geq 0$. Since the two conditions $yz \geq 0, y+z \geq 0$ hold if and only if $y \geq 0, z \geq 0$, this concludes the proof.

The above proves that rotated second-order cones are also convex.

Rotated second-order cone constraints are useful to describe quadratic convex inequalities. Precisely, if $Q = Q^T \succeq 0$, the constraint

$$c^T x + x^T Q x \leq t$$

is equivalent to the existence of w, y, z such that

$$w^T w \leq yz, \quad z = 1, \quad w = Q^{1/2} x, \quad y = t - c^T x,$$

where $Q^{1/2}$ is the [square-root](#) of the PSD matrix Q . In the space of (x, w, y, z) -variables, the above constraints represent the intersection of a rotated second-order cone with affine sets.

Second-order cone inequalities

A **second-order cone** (SOC) inequality on a vector $x \in \mathbf{R}^n$ states that a vector (y, t) that is some affine combination of x belongs to a second-order cone.

This is a constraint of the form

$$\|Ax + b\|_2 \leq c^T x + d,$$

where $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^m, c \in \mathbf{R}^n$, and d is a scalar.

Example: the constraint $0.01\sqrt{(3x_1 - 2x_2 + 5)^2 + (6x_1 + 2x_2 - 9)^2} \leq 4x_1 + 2x_2 + 5$.

Standard Forms of SOCP Second-order cone programs: standard formsInequality form. A **second-order cone program** (or SOCP, for short) is an optimization problem of the form

$$\min_x c^T x : \|A_i x + b_i\|_2 \leq c_i^T x + d_i, \quad i = 1, \dots, m,$$

where $A_i \in \mathbf{R}^{p_i \times n}$'s are given matrices, $b_i \in \mathbf{R}^{p_i}$, $c_i \in \mathbf{R}^n$ vectors, and d_i 's scalars.

The problem is convex, since the constraint functions of the corresponding [standard form](#)

$$f_i(x) = \|A_i x + b_i\|_2 - (c_i^T x + d_i), \quad i = 1, \dots, m,$$

are.

Examples.

- [Linear program as SOCP](#).
- [Facility location](#).
- [Robust least-squares](#).
- [Separation of ellipsoids](#).

$$\min_x c^T x : (A_i x + b_i, c_i^T x + d_i) \in \mathbf{K}_{p_i}, \quad i = 1, \dots, m.$$

Conic form. We can put the above problem in the so-called “conic” format

Since the cones \mathbf{K}_{p_i} are convex, and the mappings $x \rightarrow (A_i x + b_i, c_i^T x + d_i)$ are affine, the feasible set is convex.

Rotated second-order cone constraints. Since the rotated second-order cone can be expressed as some linear transformation of an ordinary second-order cone, we can include rotated second-order cone constraints, as well as ordinary linear inequalities or equalities, in the formulation. This allows to formulate LPs and QPs as special cases of SOCP.

Examples:

- [Quadratic program as SOCP](#).
- Logarithmic Chebyshev approximation

Quadratically constrained quadratic programming Definition. A quadratically constrained quadratic programming (QCQP for short) is a problem of the form

$$\min_x a_0^T x + x^T Q_0 x : x^T Q_i x + a_i^T x \leq b_i, \quad i = 1, \dots, m,$$

where $Q_i \succeq 0, i = 1, \dots, m$. This condition ensures that the problem is convex. QCQPs contain LPs and QPs as special case.

Examples:

SOCP formulation. We can formulate QCQPs as SOCPs, by introducing new variables and affine equality constraints. ([Proof](#)).

Group sparsity and the l_1/l_2 -norm trick

Group sparsity problem

In many applications, such as [image annotation](#), we would like to encourage the occurrences of whole blocks of zeros in the decision vector. Assume for example that the decision vector is partitioned into two disjoint groups $x = (x_1, \dots, x_k)$, with $x_i \in \mathbf{R}^{n_i}, i = 1, \dots, k$, with $n = n_1 + \dots + n_k$. We seek to minimize the number of non-zero blocks x_i , while satisfying some

constraint, say $x \in \mathbf{P}$, with \mathbf{P} a given polytope.

The l_1/l_2 -norm heuristic

We can use the [l₁-norm trick](#) on the vector of Euclidean norms $(\|x_1\|_2, \dots, \|x_k\|_2)$. (This will encourage many of the blocks x_i to be zero; those blocks that are not zero will not, in general, be sparse.)

This leads to the problem

$$\min_x \sum_{i=1}^k \|x_i\|_2 : x \in \mathbf{P}.$$

The above is a second-order cone program. This is easily seen after introducing a new vector variable

$$\min_{x,y} \sum_{i=1}^k y_i : x \in \mathbf{P}, y_i \geq \|x_i\|_2, i = 1, \dots, p.$$

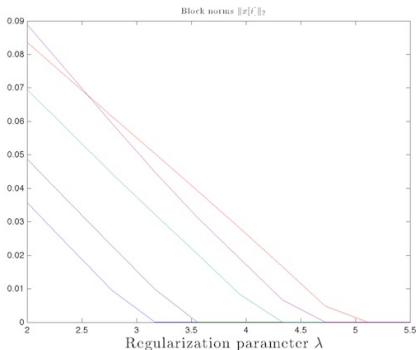
The following CVX snippets actually uses the above representation for the least-squares problem with a block-norm penalty:

$$\min_x \|Ax - b\|_2 + \sum_{i=1}^k \|x_i\|_2.$$

In our implementation we assume that each block x_i is of the same length k .

CVX syntax: least-squares with group sparsity

```
cvx_begin
    variable x(n);
    variable y(k);
    minimize( norm(A*x-b,2) + lambda*sum( y ) )
    subject to
        for i=1:k,
            y(i) >= norm(x((i-1)*p+(1:p)),2);
        end
    cvx_end
```



In this example, we solve the problem for a random instance of the problem (with $n = 100$, $m = 1000$, $k = 5$), and various values of the parameter λ . As λ grows, we observe that the block norms go to zero one by one. Thus the parameter allows to control the sparsity of the block norms.

Applications of SOCP

- [Minimum surface area](#).
- [Total variation image restoration](#).
- [Antenna array design](#).
- [Image annotation via group sparsity](#).
- Sensor network localization.

Minimum Surface Area The minimum surface area problem Consider a surface in \mathbf{R}^3 that is described by a function from the square $C := [0, 1] \times [0, 1]$ to \mathbf{R} . The corresponding [surface area](#) is

$$A(f) := \int_C \sqrt{1 + \|\nabla f(x, y)\|_2^2} dx dy.$$

The *minimum surface area* problem is to find the function f which minimizes the area $A(f)$, subject to boundary values. To be specific, we will assume that we are given values of f on the left and right side of the square, that is

$$f(x, 0) = l(x), \quad f(x, 1) = r(x), \quad x \in [0, 1],$$

where $l : \mathbf{R} \rightarrow \mathbf{R}$ and $r : \mathbf{R} \rightarrow \mathbf{R}$ are two given functions.

The above is an infinite-dimensional problem, in the sense that the variable is a function, not a finite-dimensional vector.

Discretization

We can discretize the square with a square grid, with points (ih, jh) , $0 \leq i, j \leq K$, where K is an integer, and where $h = 1/K$ the (uniform) spacing of the grid. We represent the variable of our problem, f , as a matrix $F \in \mathbf{R}^{(K+1) \times (K+1)}$, with elements $F_{ij} = f(ih, jh)$. Similarly, we represent the boundary conditions as vectors of length L, R ,

To approximate the gradient, we start from the first-order expansion of a function of two variables, valid for some small increment h :

$$\frac{\partial F}{\partial x}(x, y) \approx \frac{1}{h}(f(x + h, y) - f(x, y)).$$

We obtain that the gradient of F at a grid point can be approximated as

$$\nabla F(ih, jh) \approx \begin{pmatrix} K(F_{i+1,j} - F_{i,j}) \\ K(F_{i,j+1} - F_{i,j}) \\ 1 \end{pmatrix}, \quad 0 \leq i, j \leq K - 1. \quad \text{SOCP formulation}$$

The discretized version of our problem is thus

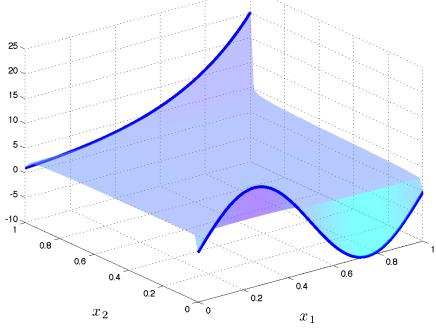
$$\min_F \frac{1}{K^2} \sum_{0 \leq i, j \leq K-1} \left\| \begin{pmatrix} K(F_{i+1,j} - F_{i,j}) \\ K(F_{i,j+1} - F_{i,j}) \\ 1 \end{pmatrix} \right\|_2 : F(i, 0) = l(ih), \quad F(i, 1) = r(ih), \quad 0 \leq i \leq K.$$

The CVX syntax for this problem can be as follows.

CVX syntax

```
>> % input: left_vals and right_vals, two row vectors of length K+1
>> h = 1/K;
cvx_begin
    variables F(K+1,K+1)
    variables T(K,K)
    minimize( sum(T(:)) )
    subject to
        for j = 1:K, for i = 1:K,
            norm([K*(F(i+1,j)-F(i,j)); K*(F(i,j+1)-F(i,j)); 1],2) <= T(i,j);
        end, end
        F(1,:) == left_vals;
        F(K+1,:) == right_vals;
cvx_end
```

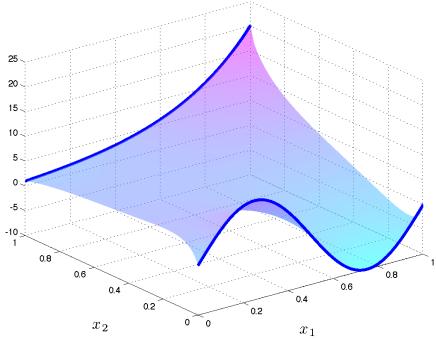
Examples



Minimal surface area joining two curves (in dark blue).

It is interesting to compare the minimal surface area with one that is obtained by squaring the norms. This corresponds to the QP

$$\min_F \frac{1}{K^2} \sum_{0 \leq i, j \leq K-1} \left\| \begin{pmatrix} K(F_{i+1,j} - F_{i,j}) \\ K(F_{i,j+1} - F_{i,j}) \\ 1 \end{pmatrix} \right\|_2^2 : F(i, 0) = l(ih), \quad F(i, 1) = r(ih), \quad 0 \leq i \leq K.$$



A similar problem where surface area is replaced with a least-squares criterion. We observe that, in contrast to the solution above, this new surface has much less “constant” values. In effect the minimal surface area formulation tends to assign zero values to the gradient more aggressively.

Total Variation Image Restoration

The image restoration problem

Digital images always contain noise. In image restoration, the problem is to filter out the noise. Early methods involved least-squares but the solutions exhibited the ‘‘ringing’’ phenomenon, with spurious oscillations near edges in the restored image. To address this phenomenon, one may add to the objective of the least-squares problem a term which penalizes the variations in the image.

We may represent a given (noisy) image as function from the square $C := [0, 1] \times [0, 1]$ to \mathbf{R} . We define the *image restoration* problem as minimizing, over functions $\hat{f} : C \rightarrow \mathbf{R}$, the

objective

$$\int_C \|\nabla \hat{f}(x)\|_2 dx + \lambda \int_C (\hat{f}(x) - f(x))^2 dx,$$

where the function \hat{f} is our estimate. The first term penalizes functions which exhibit large variations, while the second term accounts for the distance from the estimate to the noisy image, f .

The above is an infinite-dimensional problem, in the sense that the variable is a function, not a finite-dimensional vector.

Discretization

We can discretize the square with a square grid, as follows:

$$x_{ij} = \begin{pmatrix} \frac{i}{K} \\ \frac{j}{K} \end{pmatrix}, \quad 0 \leq i, j \leq K.$$

We represent the data of our problem, f , as a matrix $F \in \mathbf{R}^{(K+1) \times (K+1)}$, with elements $F_{ij} = f(x_{ij})$. Similarly, we represent the variable \hat{f} of our problem with a $(K+1) \times (K+1)$ matrix \hat{F} , which contains the values of \hat{f} at the grid points x_{ij} .

To approximate the gradient, we start from the first-order expansion of a function of two variables, valid for some small increment h :

$$\frac{\partial \hat{f}}{\partial x}(x, y) \approx \frac{1}{h}(f(x+h, y) - f(x, y))$$

Applying this to a grid point, with the small increment set to $h = 1/K$, we obtain that the gradient of \hat{f} at a grid point can be approximated as

$$\nabla \hat{f}(x_{ij}) \approx G_{ij} := \begin{pmatrix} K(\hat{F}_{i+1,j} - \hat{F}_{i,j}) \\ K(\hat{F}_{i,j+1} - \hat{F}_{i,j}) \end{pmatrix}, \quad 1 \leq i, j \leq K$$

with the convention that the terms involved are zero on the boundary (that is, if either i or j is n).

SOCOP formulation

The discretized version of our problem is thus

$$\min_{\hat{F}} \frac{1}{K^2} \sum_{0 \leq i, j \leq K} \left(\left\| \begin{pmatrix} K(\hat{F}_{i+1,j} - \hat{F}_{i,j}) \\ K(\hat{F}_{i,j+1} - \hat{F}_{i,j}) \end{pmatrix} \right\|_2 + \lambda(\hat{F}_{ij} - F_{ij})^2 \right).$$

Antenna array design

See [here](#).

Image annotation and group sparsity

See [here](#).

Exercises Second-order cones A. A single second-order cone constraint. Draw the set of points $x \in \mathbf{R}^2$ such that

$$\rho \|x\|_2 \leq 2x_1 + x_2 - 3,$$

where $\rho = 0.3, 1, 3, 5$. In general, for which values of ρ , if any, is this set an ellipsoid? When is it not empty?

B. Squaring and second-order cones. When considering a second-order cone constraint, a temptation might be to square it in order to obtain a classical convex quadratic constraint. This might not always work. Consider the constraint

$$x_1 + 2x_2 \geq \|x\|_2,$$

and its squared counterpart:

$$(x_1 + 2x_2)^2 \geq \|x\|_2^2.$$

Is the set defined by the second inequality convex? Discuss.

C. A robust constraint. Let $\hat{u} \in \mathbf{R}^n$ and $v \in \mathbf{R}$, $\rho \geq 0$ be given. Consider the following condition on a given $x \in \mathbf{R}^n$:

$$u^T x \leq v \text{ for every } u \in \mathbf{R}^n \text{ such that } \|u - \hat{u}\|_2 \leq \rho.$$

1. Show that this condition is equivalent to a second-order cone condition on x .
2. What is the geometrical interpretation of this condition?
3. Plot the set of $x \in \mathbf{R}^2$ that satisfy the condition with $\hat{u} = (2, 1)$, $v = 3$, and $\rho = 2$.

Standard forms D. A complicated function. We would like to minimize the function $f : \mathbf{R}^3 \rightarrow \mathbf{R}$, with values:

$$f(x) = \max \left(x_1 + x_2 - \min(\min(x_1 + 2, x_2 + 2x_1 - 5), x_3 - 6), \frac{(x_1 - x_3)^2 + 2x_2^2}{1 - x_1} \right),$$

with the constraint $x_1 < 1$.

1. Explain precisely how to formulate the problem as an SOCP in standard form. Hint: Remember that $\forall z \in \mathbf{R}^n$, for all $a, b \in \mathbf{R}$ non negative, the constraint $\|z\|_2^2 \leq ab$, $a \geq 0$, $b \geq 0$ is equivalent to a second order cone constraint.
2. Write a CVX code that solves the problem. Hint: use the function `quad_over_lin.m`.

E. SOCP and QCQP. Quadratically constrained quadratic programs are minimization problems involving convex quadratic objectives and constraints. [This section](#) shows that QCQP's can be expressed as SOCP's. Is the converse true, in other words, can we always write an SOCP as a QCQP? Discuss.

F. Robust least-squares. For a given matrix $A \in \mathbf{R}^{m \times n}$ and a vector $b \in \mathbf{R}^m$, we consider the robust least-squares problem

$$\min_x \max_{\Delta : \|\Delta\| \leq \rho} \|(A + \Delta)x - b\|_2,$$

where $\|\Delta\|$ denotes the largest singular value of the matrix Δ , and $\rho \geq 0$ is given. This problem arises when we seek to solve a least-squares problem in which the matrix A is subject to an additive perturbation, and we would like to find a solution that minimizes the worst-case residual error.

1. Prove that for any given vectors $x, z \in \mathbf{R}^n$, we have $\|\Delta x + z\|_2 \leq \rho \|x\|_2 + \|z\|_2$ whenever $\|\Delta\| \leq \rho$.
2. Show that the bound you found in part (a) is attained, by exhibiting a matrix Δ that achieves the bound. Hint: use a rank-one matrix constructed with x, z .
3. Formulate the robust least-squares problem as an SOCP in standard form.
4. Solve the problem in the case when the bound on the perturbation Δ is a component-wise bound, of the form $\max_{i,j} |\Delta_{ij}| \leq \rho$.

Applications

G. k -Ellipses. Consider k points x_1, \dots, x_k in \mathbf{R}^2 . For a given positive number d , we define the k -ellipse with radius d as the set of points $x \in \mathbf{R}^2$ such that the sum of the distances from x to the points x_i is equal to d .

1. How do k -ellipses look like when $k = 1$ or $k = 2$?
2. Express the problem of computing the *geometric median*, which is the point that minimizes the sum of the distances to the points $x_i, i = 1, \dots, k$, as an SOCP in standard form.
3. Using CVX, write a code with input $X = (x_1, \dots, x_k) \in \mathbf{R}^{2 \times k}$ that plots the corresponding k -ellipse.

H. Sparse Classification for word imaging. The image of a given query word in a given corpus of text news can be defined as a short list of other words with which this query is strongly associated. To be easily understandable, the list should be extremely short with respect to the number of terms present in the corpus.

One way to obtain a word image is to use l_1 -norm penalization in a classification algorithm, where indicator of the query word's appearance in each headline is used as that headline's label/response ($y \in \{1, -1\}^m$), and the indicators for all other words are used as predictors/features ($X = [x_1, \dots, x_m] \in \mathbf{R}^{n \times m}$). The few features (words) that are then predictive of the appearance of the query term correspond to that query's image in the corpus.

As explained [here](#), we consider the following optimization problem, also called l_1 -norm Support Vector Machine (L1-SVM):

$$\min_{w \in \mathbf{R}^n, b \in \mathbf{R}} \sum_{i=1}^m [1 - y_i(w^T x_i + b)]_+ + \lambda \|w\|_1$$

where $z_+ := \max(0, z)$. By imposing a sparsity constraint on the weight vector, we can single out the few words that are most able to predict the presence or absence of a query word in any document. These selected words are then considered the list of words comprising the query word's image.

We look at the Word Imaging problem in a small-scale setting. Our original data is the headlines of New York Times between Jan 1 2006 and Dec 31 2006: there are 84612 headlines and 160,624 distinct words in total. To make the problem accessible to you, we preprocessed the text data and down-sampled (with special care) both the number of headlines and that of distinct words to 997 and 1045 respectively. We try to obtain the image of the query word "Microsoft". The label y , the predictor X and the dictionary of terms $dict$ are defined in `sparseSVM.mat`. Note that y_j indicates whether "Microsoft" shows up in the j -th headline, X_{ij} indicates whether j -th headline contains i -th word, and $dict_i$ is the i -th word.

1. Show how to formulate L1-SVM as a linear program.
2. Using CVX, write a MATLAB program to solve the problem of minimizing the approximate loss for the given data. What are the top 20 words (i.e. top 20 features with highest coefficients) that predict the presence of the query word "Microsoft"? Does the result make sense to you? Experiment with $\lambda \in \{0.1, 0.5, 1, 5, 10\}$ and see how the list of top words changes.
3. Consider now the Support Vector Machine setup where L2 regularization is used (call it the L2-SVM)

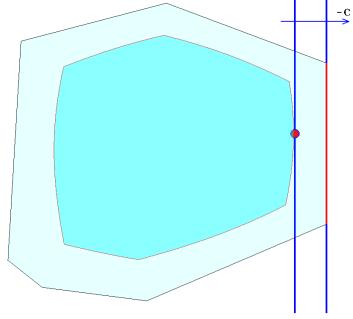
$$\min_{w \in \mathbf{R}^n, b \in \mathbf{R}} \sum_{i=1}^m [1 - y_i(w^T x_i + b)]_+ + \lambda \|w\|_2$$

Show how to formulate the above as an SOCP.

5. Follow the procedure from part 2 to solve this problem using the same lambdas. Discuss your results.
6. Compare the top 20 features extracted by the L1-SVM and the L2-SVM for each lambda. How do they compare across the lambdas? Which formulation makes more sense to you? Discuss.

Robust Linear Programming

Robust linear programming addresses linear programming problems where the data is uncertain, and a solution which remains feasible despite that uncertainty, is sought.



The robust counterpart to an LP is not an LP in general, but is always convex. The figure on the left illustrates the feasible set of the “robust counterpart” of an LP after we take into account uncertainty in the facets’ directions.

We focus on specific models of uncertainty that lead to [tractable solutions](#), and describe a few [applications](#) of the method. We further explore the concept of [affine recourse](#), which is used in dynamic decision problems under uncertainty.

Tractable Cases
Scenario uncertainty Uncertainty model. In the scenario uncertainty model, the uncertainty on a coefficient vector a is described by a *finite* set of points: $\mathbf{U} = \{a^1, \dots, a^K\}$,

where each vector $a^k \in \mathbf{R}^n, k = 1, \dots, K$ corresponds to a particular “scenario”.

The robust counterpart to a half-space constraint:

$$\forall a \in \mathbf{U}, \quad a^T x \leq b,$$

can be simply expressed as a set of K affine inequalities:

$$(a^k)^T x \leq b, \quad k = 1, \dots, K.$$

Note that the scenario model actually enforces more than feasibility at the “scenario” points a^k . In fact, for any a that is in the [convex hull](#) of the set \mathbf{U} , the robust counterpart holds. Indeed, if the above holds, then for any set of nonnegative weights $\lambda_1, \dots, \lambda_K$ summing to one, we have

$$\sum_{k=1}^K \lambda_k (a^k)^T x \leq b, \quad k = 1, \dots, K.$$

With a scenario uncertainty, the robust counterpart to the original LP:

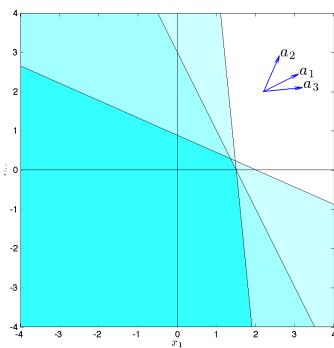
$$\min_x c^T x : \forall a_i \in \mathbf{U}_i, \quad a_i^T x \leq b_i, \quad i = 1, \dots, m,$$

with $\mathbf{U}_i = \{a_i^1, \dots, a_i^{K_i}\}, i = 1, \dots, m$, becomes

$$\min_x c^T x : (a_i^k)^T x \leq b, \quad k = 1, \dots, K_i, \quad i = 1, \dots, m,$$

This is an LP, with a total of $K_1 + \dots + K_m$ constraints, where K_i is the number of elements in the finite set \mathbf{U}_i , and m is the number of constraints in the original (nominal) LP.

The scenario model is attractive for its simplicity. However, the number of scenarios can result in too large a problem.



A robust half-space constraint with scenario uncertainty, with three scenarios. The constraint represents a polyhedron (dark blue).

Example: [Robust LP solution to the drug production problem](#).

Box uncertainty models Definition. The box uncertainty model assumes that every coefficient vector a_i lies in a “box”, or more generally, a *hyper-rectangle* in \mathbf{R}^n , but is otherwise unknown. In its simplest case, the uncertainty model has the following form: $\mathbf{U} = \{a : \|a - \hat{a}\|_\infty \leq \rho\}$,

where $\rho \geq 0$ is a measure of the size of the uncertainty, and \hat{a} represents a “nominal” vector. This describes a “box” of half-diameter ρ around the center \hat{a} .

Note that the robust counterpart to a half-space constraint

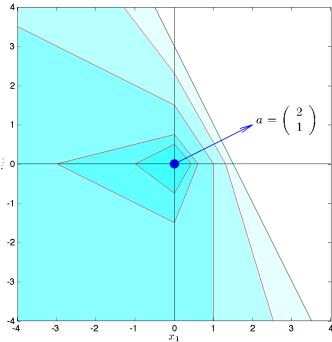
$$\forall a \in \mathbf{U}, \quad a^T x \leq b,$$

can be handled as a scenario model, with “scenarios” the vectors $a^k := \hat{a} + \rho v^k$, where v^k represents one of the 2^n vertices of the unit box (that is, vectors with ± 1 's as components). Indeed, enforcing the constraint $(a^k)^T x \leq b$ implies that $a^T x \leq b$ holds for every a in the convex hull of the a^k 's, which is precisely the box \mathbf{U} . This approach is not practical, as there are an exponential number of vertices, hence of constraints, to deal with.

Robust counterpart: LP formulation. The robust counterpart is better handled after one realizes that the robust counterpart constraint above is equivalent to $b \geq \max_{a \in \mathbf{U}} a^T x$.

The maximization problem has a simple form ([proof](#)):

$$\max_{a \in \mathbf{U}} a^T x = \hat{a}^T x + \rho \|x\|_1.$$



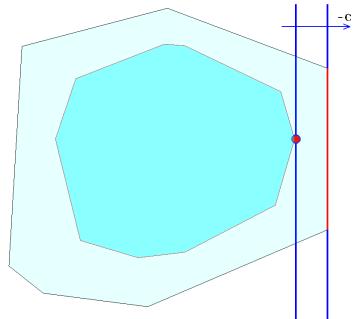
A robust half-space constraint with box uncertainty, with different uncertainty levels. Such sets can be described by a finite (but exponential in the dimension) number of inequalities.

With a box uncertainty model, defined as $\mathbf{U}_i = \{a : \|a - \hat{a}_i\|_\infty \leq \rho\}, i = 1, \dots, m$, the robust counterpart to the original LP:

$$\min_x c^T x : \forall a_i \in \mathbf{U}_i, a_i^T x \leq b_i, i = 1, \dots, m,$$

is also an LP (in [polyhedral form](#))

$$\min_x c^T x : \hat{a}_i^T x + \rho \|x\|_1 \leq b_i, i = 1, \dots, m.$$



A robust LP with box uncertainty. The robust feasible set (darker blue) is inside the feasible set for the nominal LP (lighter blue); both feasible sets are polyhedral. The objective is to minimize $c^T x$, where the vector $-c$ is shown. The nominal LP has many optimal points (red line), which means a solution might be very sensitive to data changes (such as if we change the direction of the objective slightly). Although in this case, the robust LP has a unique solution (red dot), it is still an LP, so it might be sensitive to changes in the objective vector.

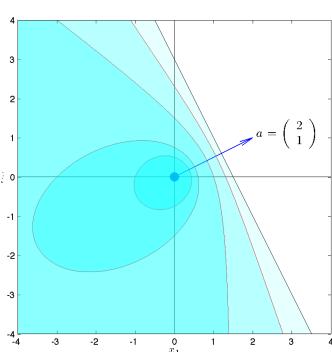
Ellipsoidal uncertainty Form of the model. The *ellipsoidal uncertainty* model has the form $\mathbf{U} = \{a = \hat{a} + Ru : \|u\|_2 \leq 1\}$,

where $R \in \mathbf{R}^{n \times p}$ is a matrix which describes the “shape” of the ellipsoid around its center, which is \hat{a} . If $R = \rho I$ for some $\rho \geq 0$, then the above is a simple sphere of radius ρ around \hat{a} ; we refer to this special case as the *spherical uncertainty* model.

Ellipsoidal uncertainty models are useful to “couple” uncertainties across different components of the coefficient vector a . This is contrast with the previous “box” models, which allow uncertainties to take their largest values independently of each other.

$$b \geq \max_{a \in \mathbf{U}} a^T x = \hat{a}^T x + \|R^T x\|_2,$$

where we have exploited the [Cauchy-Schwartz inequality](#). (See also [here](#).)



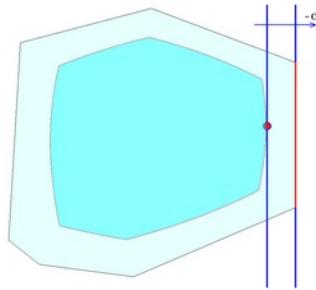
A robust half-space constraint with spherical uncertainty, with different uncertainty levels.

With an ellipsoidal uncertainty model, defined as $\mathbf{U}_i = \{\hat{a}_i + R_i u : \|u\|_2 \leq 1\}, i = 1, \dots, m$, the robust counterpart to the original LP:

$$\min_x c^T x : \forall a_i \in \mathbf{U}_i, a_i^T x \leq b_i, i = 1, \dots, m,$$

becomes an SOCP:

$$\min_x c^T x : \hat{a}_i^T x + \|R_i^T x\|_2 \leq b_i, i = 1, \dots, m.$$



A robust LP with spherical uncertainty. The robust feasible set (darker blue) is inside the (polyhedral) feasible set for the nominal LP (lighter blue). The objective is to minimize $c^T x$, where the vector $-c$ is shown. The nominal LP has many optimal points (red line), which means a solution might be very sensitive to data changes (such as if we change the direction of the objective slightly). In contrast, the solution to the robust LP is unique (red dot), irrespective of the choice of the objective. As a result, it is not very sensitive to changes in the objective or other problem data.

Chance Programming

Chance constraints

Consider a vector $a \in \mathbf{R}^n$, and a scalar $b \in \mathbf{R}$. When a is a random variable, the constraint $a^T x \leq b$ is not deterministic. One way to deal with randomness is to enforce the constraint in probability. Specifically, assume that the distribution on a is known, we can try to enforce

$$\mathbf{Prob}\{a : a^T x \leq b\} \geq 1 - \epsilon$$

where ϵ is small.

Both these formulations assume that the distribution of the random variables (a or A) is known.

When the distribution on the random variables is only partially known, we may consider worst-case variants:

$$\inf_{\pi \in \mathcal{P}} \mathbf{Prob}_{\pi}\{a : a^T x \leq b\} \geq 1 - \epsilon,$$

where \mathcal{P} is a (known) set of distributions, π is a distribution of the random variables (a), and \mathbf{Prob}_{π} refers to the probability under distribution π . A typical example is when \mathcal{P} refers to the set of distributions with given mean and covariance matrix. In this case, the bound above is related to the concept of [Chebyshev](#) bounds.

In general, chance constraints are intractable numerically. Evaluating the probability of a simple object (such as a hyperplane), under a known distribution, is in general extremely difficult. There are useful exceptions, as in the Gaussian case, seen below. Surprisingly, the second (worst-case) version of chance constraints is sometimes more easily dealt with than the first.

A more general version is when we start with a constraint $Ax \leq b$, where A is a random matrix. The matrix case is substantially more difficult and not dealt with here.

Tractable cases

Gaussian assumption. When the distribution on a is Gaussian, and with a single affine constraint, a simplification occurs. Consider the top condition, where we assume that a is a Gaussian vector, with given mean \bar{a} and covariance matrix C . The condition turns out to reduce to

$$\bar{a}^T x + \kappa(\epsilon) \sqrt{x^T C x} \leq b,$$

with $\kappa(\epsilon) = \Phi^{-1}(1 - \epsilon)$, and Φ the conditional density of the normal distribution. Note that $\kappa(\epsilon) > 0$ when $\epsilon \in [0, 1/2]$. (For example, $\kappa(0.01) = 2.83$) The above can be interpreted as a condition on the mean value, with an added "risk premium" involving the standard deviation of the random variable $a^T x$. The parameter $\kappa(\epsilon)$ is a "safety parameter" that grows as ϵ decreases.

The above is a second-order cone constraint. Indeed, we can factorize C as $C = R^T R$, with R a (possibly rectangular) matrix, and rewrite the above as

$$\kappa(\epsilon) \|Rx\|_2 \leq b - Ax.$$

Chebyshev bounds. Assume that the distribution on a is only partially known, via its given mean \bar{a} and covariance matrix C . The constraint [above](#) writes

$$1 - \epsilon \leq \mathbf{Prob}_{\pi}\{a : a^T x \leq b\} \text{ whenever } E_{\pi}(a) = \bar{a}, E_{\pi}(a - \bar{a})(a - \bar{a})^T = C.$$

It turns out that the above can be written in an equivalent, simple way, as a constraint of the form [above](#), with

$$\kappa(\epsilon) = \sqrt{\frac{1 - \epsilon}{\epsilon}}.$$

Note that we have now $\kappa(\epsilon) > \Phi^{-1}(1 - \epsilon)$. As expected, when the distribution is not Gaussian but only known through its mean and covariance, the safety parameter needs to be increased.

Affine Recourse

Basic idea

LP with right-hand side uncertainty. Consider a "nominal" linear program:

$$\min_x c^T x : Ax \leq b.$$

We assume that b is affected by an uncertainty vector $u \in \mathcal{U} \subseteq \mathbf{R}^p$, in affine fashion. Precisely:

$$b(u) = \widehat{b} + Bu,$$

where $\widehat{b} \in \mathbf{R}^m$ and $B \in \mathbf{R}^{m \times p}$ given.

Standard robust counterpart. The standard robust counterpart is of the form

$$\min_{\widehat{x}} c^T \widehat{x} : \forall u \in \mathcal{U}, Ax \leq b(u).$$

The above is often expressible in a standard convex form, such as LP or SOCP. For example, if $\mathcal{U} = [-\rho, \rho]^m$ is a box (this corresponds to an interval uncertainty on the vector b), then the robust counterpart above writes

$$\min_{\widehat{x}} c^T \widehat{x} : A\widehat{x} + \rho|B|\mathbf{1} \leq \widehat{b},$$

where $\mathbf{1}$ is the vector of ones in \mathbf{R}^p , and matrix $|B|$ contains the absolute values of the elements of B .

The interpretation of the above approach is simple: we simply replace the original vector b by its "worst-case" counterpart $b_{wc} := \widehat{b} - \rho|B|\mathbf{1}$.

Example: [cash-flow management](#) with [uncertain liability vector: robust LP model](#). In this problem, the vector b represents liabilities, that is, cash that a company expects to receive or to have to pay at each end of the next six months. It is reasonable to try to guard against possible changes in the liability profile.

Affinely Adjustable Robust counterpart (AARC)

In many applications, especially decision problems with multiple time periods, the uncertainty is available to some decision variables (e.g., tomorrow's prices become ultimately known). We seek an affinely adjusted robust solution, that is, a linear feedback. We look for a *functional* solution of the form

$$x(u) = \widehat{x} + Xu,$$

where $\widehat{x} \in \mathbf{R}^n$ and $X \in \mathbf{R}^{n \times p}$ are both variables. Often, we need to introduce linear equality constraints on X . In dynamic situations for example, the different components of vector u correspond to different time periods; the equality constraints mean that the function $u \rightarrow x(u)$ must be causal (it does not depend on future variables). In that case, X must have a particular structure, such as upper-triangular. We cover these situations by imposing a constraint on X of the form $X \in \mathcal{X}$, where $\mathcal{X} \subseteq \mathbf{R}^{n \times p}$ is a given affine set.

The affinely adjustable robust counterpart is

$$\min_{(\widehat{x}, X) \in \mathcal{X}} \max_{u \in \mathcal{U}} c^T x(u) : \forall u \in \mathcal{U}, Ax(u) \leq b(u).$$

The above problem is tractable in many situations. For example, if $\mathcal{U} = [-\rho, \rho]^m$ is a box (this corresponds to an interval uncertainty on the vector b), then the AARC is

$$\min_{(\widehat{x}, X) \in \mathcal{X}} c^T \widehat{x} - \rho \|X^T c\|_1 : A\widehat{x} + \rho|AX - B|\mathbf{1} \leq \widehat{b}.$$

As expected, we recover the standard robust counterpart upon setting $X = 0$.

Example: [cash-flow management with uncertain liability vector: AARC model](#).

Applications of Robust LP

- [Robust inventory control](#).
- [Robust antenna array design](#).
- [Robust supervised learning](#).
- [Affine recourse for cash-flow management](#).

Robust inventory control

In progress. See [here](#).

Robust Antenna Array Design

See [here](#).

Robust Supervised Learning

Supervised learning

Many supervised learning problems (e.g. classification, regression) can be written as

$$\min_w \mathcal{L}(X^T w)$$

where \mathcal{L} is convex, and X contains the data.

Example: [Support Vector Machine](#). The Support Vector Machine (SVM) [classification problem](#) is

$$\min_{w,b} \sum_{i=1}^m (1 - y_i(z_i^T w + b))_+$$

where for a scalar ξ , ξ_+ denotes its positive part $\max(0, \xi)$, and:

- $Z := [z_1, \dots, z_m] \in \mathbf{R}^{n \times m}$ contains the *data points* z_1, \dots, z_m .
- $y \in \{-1, 1\}^m$ contain the *labels*.
- (w, b) contains the classifier parameters, allowing to classify a new point z via the rule

$$y = \text{sgn}(z^T w + b).$$

Defining $X = [x_1, \dots, x_m]$ with $x_i = (z_i, 1) \in \mathbf{R}^{n+1}$, $i = 1, \dots, m$, we obtain the generic learning problem above, with the "hinge loss" function $\mathcal{L}(\xi) = (1 - \xi)_+ = \max(0, 1 - \xi)$.

The above can be written as an LP.

Penalization. Often, optimal values and solutions of optimization problems are sensitive to data. A common approach to deal with sensitivity is via penalization, for example:

$$\min_x \mathcal{L}(X^T w) + \|Wx\|_2^2 \quad (W = \text{weighting matrix}).$$

Example: [Regularized least-squares](#).

Robustness to data uncertainty

Assume the data matrix is only *partially* known, and address the robust optimization problem:

$$\min_{w,b} \max_{U \in \mathcal{U}} \sum_{i=1}^m (1 - y_i((z_i + u_i)^T w + b))_+,$$

where $U = [u_1, \dots, u_m]$ and $\mathcal{U} \subseteq \mathbf{R}^{n \times m}$ is a set that describes additive uncertainty in the data matrix.

Measurement-wise, spherical uncertainty. Assume

$$\mathcal{U} = \{U = [u_1, \dots, u_m] \in \mathbf{R}^{n \times m} : \|u_i\|_2 \leq \rho\},$$

where $\rho > 0$ is given. The above model says that each data point is not actually known, but lives in a sphere of radius ρ around the "nominal" point z_i .

The robust SVM then reduces to

$$\min_{w,b} \sum_{i=1}^m (1 - y_i(z_i^T w + b) + \rho \|w\|_2)_+.$$

Contrast with the classical SVM, which involves a l_2 -norm regularization term:

$$\min_{w,b} \sum_{i=1}^m (1 - y_i(z_i^T w + b))_+ + \lambda \|w\|_2^2.$$

where $\lambda > 0$ is a penalty parameter. With spherical uncertainty, the robust SVM is similar to (although different than) classical SVM.

Separable data. Assume that the data is separable, let us impose that, instead of separating the points z_i , we separate the spheres of radius ρ around them. We will seek to maximize the robustness level ρ subject to the robust separability conditions

$$\forall u_i \text{ with } \|u_i\|_2 \leq \rho : y_i((z_i + u_i)^T w + b) \geq 0, \quad i = 1, \dots, m.$$

As seen [here](#), the above can be written as

$$y_i(z_i^T w + b) \geq \rho \|w\|_2, \quad i = 1, \dots, m.$$

Now let us maximize the uncertainty level (the radius of the spheres) subject to the above conditions. The resulting problem is not convex jointly in (w, b, ρ) . But we can always scale w, b, ρ so that $\rho \|w\|_2 = 1$. Then, maximizing ρ is the same as minimizing $\|w\|_2$, and we are led to the problem

$$\min_{w,b} \|w\|_2 : y_i(z_i^T w + b) \geq 1, \quad i = 1, \dots, m.$$

The above leads to the so-called "maximum margin" classifier. The margin is the distance between the convex hulls of each class of points, and is equal to twice the optimal radius, that is, $2/\|w\|_2$.

Thus, in the case of separable data, we recover exactly the classical maximum-marg classifier.

- The figure shows a maximally robust classifier for separable data, with spherical uncertainties around each data point. In this case, the robust counterpart reduces to the classical maximum-marg classifier problem. At optimum, the margin is the amount of separation between the two data sets, that is, twice the optimal radius, which is inversely proportional to the Euclidean norm of the classifier vector.

Interval uncertainty. Assume

$$\mathcal{U} = \{U = [u_1, \dots, u_m] \in \mathbf{R}^{n \times m} : \|u_i\|_\infty \leq \rho\},$$

where $\rho > 0$ is given. The robust SVM reduces to

$$\min_{w,b} \sum_{i=1}^m (1 - y_i(z_i^T w + b) + \rho \|w\|_1)_+$$

The ℓ_1 -norm term encourages sparsity, and may not regularize the solution. The separable case provides an intuitive explanation as to why the ℓ_1 -norm encourages sparser classifiers.

- Maximally robust classifier for separable data, with box uncertainties around each data point. This uncertainty model encourages sparsity of the solution, which in 2D corresponds to a vertical (or horizontal) separation boundary.

Affine recourse in cash-flow management

See [here](#).

Exercises A. We consider a resource allocation problem of the form $\max_{w \in \mathcal{W}} \min_{r \in \mathcal{E}} r^T w$, where $\mathcal{W} := \{w \in \mathbf{R}_+^n : w_1 + \dots + w_n = 1\}$, and

$$\mathcal{E} := \{\hat{r} + Du : \|u\|_2 \leq 1\}.$$

Here, $\hat{r} \in \mathbf{R}^n$ and $D = \text{diag}(\sigma_1, \dots, \sigma_n)$ are given, with $\sigma \in \mathbf{R}^n$, $\sigma > 0$.

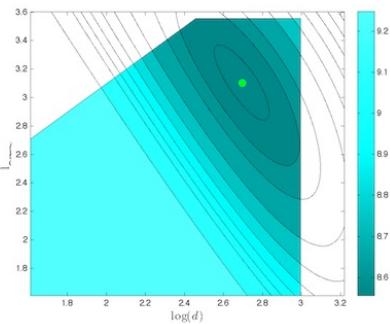
The above problem appears when trying to allocate resources to various revenue-generating processes (which could be ads, financial investments, physical sensors, etc). The revenue vector r is unknown but bounded, and the goal is to maximize the worst-case total revenue, $\min_{r \in \mathcal{E}} r^T w$.

1. Describe the shape of \mathcal{E} in simple geometrical terms, detailing its "size" and "orientation".
2. Use the [Cauchy-Schwartz inequality](#) to prove that

$$\min_{r \in \mathcal{E}} r^T w = \hat{r}^T w - \|Du\|_2.$$

3. Express the problem as an SOCP in standard format, involving the variable w and one extra scalar variable.

Geometric Programming



Geometric programming (GP) is an optimization model where the variables are non-negative, and the objective and constraints are sums of powers of those variables, with non-negative weights. Although GPs are not convex, we can transform them, via a change of variables, into convex problems. In its convex form, GP can be seen as a natural extension of LP.

GP arise naturally in the context of geometric design, or with models of processes that are well approximated with [power laws](#). They also arise (via their convex representation) when trying to fit discrete probability models in the context of classification.

Posynomials Monomials Definition. A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is a *monomial* if its domain is \mathbf{R}_{++}^n (the set of vectors with positive components) and its values take the form $f(x) = cx_1^{a_1} \dots x_n^{a_n}$,

where $c > 0$ and $a \in \mathbf{R}^n$. We can write in short-hand notation:

$$f(x) = cx^a,$$

where we follow the *power law notation*: by convention, for two vectors $x, a \in \mathbf{R}^n$, x^a is the product $x_1^{a_1} \dots x_n^{a_n}$.

Example. The function with domain $\{x \in \mathbf{R}^2 : x_1 > 0, x_2 > 0\}$ and values $f(x) = 3x_1^{1.2}x_2^{-0.003}$ is a monomial, while $-f$ is not.

Log-linearity and power laws. Monomials are closely related to linear or affine functions: indeed, if f is a monomial in variable x , then $\log f$ is affine in the vector $\log x := (\log x_1, \dots, \log x_n)$. Hence monomial functions could be called "log-linear", although the term is not widely used.

Just as linear models are important in (approximate) models between general variables, monomials play an ubiquitous role for modeling relationships between *positive* variables, such as prices, concentrations, energy, or geometric data such as length, area and volume, etc. Like their linear counterpart, power laws can be [easily fitted](#) to experimental data, via least-squares methods.

Examples:

- The *Stephan law* of physics states that the power radiated by a body evolves as AT^4 , where A is the surface area of the body, and T the temperature; this is a monomial in variables A, T .
- In industrial engineering, the *experience curve* relates the cost C_1 of producing a unit for the first time, to the cost of producing the unit for the n -th time, as $C_n = C_1 n^{-a}$, where n is the cumulative number of unit produced, and exponent a depends on the industry. Thus C is a monomial in C_1 and n .

Posynomials

A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is a *posynomial* if its domain is \mathbf{R}_{++} (the set of vectors with positive components) and its values take the form of a *non-negative sum of monomials*:

$$f(x) = \sum_{k=1}^K c_k f_k(x),$$

where $c_k \geq 0$, $k = 1, \dots, K$, and each f_k is a monomial.

The values of a posynomial can be always written as

$$f(x) = \sum_{k=1}^K c_k x_1^{a_{k1}} \dots x_n^{a_{kn}},$$

for some $c \in \mathbf{R}_{+}^K$, and $A = (a_{kj}) \in \mathbf{R}^{K \times n}$. If we denote by a_k , $k = 1, \dots, K$ the k -th row of A , then we can write in short-hand notation

$$f(x) = \sum_{k=1}^K c_k x^{a_k},$$

where we follow the above power law notation to define what x^a means when x, a are two vectors of the same size.

Examples:

- [Construction and operating costs for a water tank](#).
- [Signal-to-noise ratios in wireless communications](#).

Generalized Posynomials. A *generalized posynomial* function is any function obtained from posynomials using addition, multiplication, pointwise maximum, and raising to constant positive power. For example, the function $f : \mathbf{R}_{++}^3 \rightarrow \mathbf{R}$ with values

is a generalized posynomial.

Convex Representation

Monomials and (generalized) posynomials are not convex. However, with a simple transformation of the variables, we can transform them into convex ones.

Convex representation of posynomials. Consider a posynomial function f . Instead of the original (positive) variables, we use the new variable $y_i = \log x_i$, $i = 1, \dots, n$. We then take the logarithm of the function f . Let us look at the effect of such transformations on monomials and posynomials.

- For a monomial f with values $f(x) = cx_1^{a_1} \dots x_n^{a_n}$, where $a \in \mathbf{R}^n$ and $c > 0$, we have

$$\log f(x) = a^T y + b,$$

where $y_i = \log x_i$, $i = 1, \dots, n$, and $b = \log c$. Our transformation yields an affine function.

- For a posynomial f with values

$$f(x) = \sum_{k=1}^K c_k x^{a_k},$$

where $c > 0$, we have

$$\log f(x) = \log \left(\sum_{k=1}^K e^{a_k^T y + b_k} \right),$$

where $b_k := \log c_k$, $k = 1, \dots, K$. The above can be written

$$\log f(x) = \text{lse}(Ay + b),$$

where A is the $K \times n$ matrix with rows a_1, \dots, a_K , $b \in \mathbf{R}^K$, and lse is the [log-sum-exp function](#). Recall that this function is [convex](#).

Thus, we can view a posynomial as the log-sum-exp function of an affine combination of the logarithm of the original variables. Since the lse function is convex, this transformation will allow us to use convex optimization to optimize posynomials.

Remark: [Why do we take the log?](#)

Convex representation of generalized posynomials. Adding variables, and with the logarithmic change of variables seen above, we can also transform generalized posynomial inequalities into convex ones.

For example, consider the posynomial $f : \mathbf{R}_{++}^n \rightarrow \mathbf{R}$ with values

$$f(x) = \max(f_1(x), f_2(x)),$$

where f_1, f_2 are two posynomials. For $t > 0$, the constraint

$$f(x) \leq t$$

can be expressed as two posynomial constraints in (x, t) .

Likewise, for $t > 0$, consider the power constraint

$$f_1(x)^\alpha \leq t,$$

with f an ordinary posynomial and $\alpha > 0$. Since $\alpha > 0$, the above is equivalent to

$$f(x) \leq t^{1/\alpha},$$

which in turn is equivalent to the posynomial constraint

$$g(x, t) := t^{-1/\alpha} f(x) \leq 1.$$

Hence, by adding as many variables as necessary, we can express a generalized posynomial constraint as a set of ordinary ones.

Standard Forms of GP Geometric Programs

A geometric program (GP for short) is a problem with generalized posynomial objective and inequality constraints, and (possibly) monomial equality constraints.

In standard form, a GP can be written as

$$\min_x f_0(x) : f_i(x) \leq 1, \quad i = 1, \dots, m, \quad h_j(x) = 1, \quad j = 1, \dots, p,$$

where f_0, \dots, f_m are generalized posynomials and h_1, \dots, h_p are monomials.

Assuming for simplicity that the generalized posynomials involved are ordinary posynomials, we can express a GP explicitly, in the so-called standard form:

$$\begin{aligned} \min_x \sum_{k=1}^{K_0} c_{k0} x^{a_{k0}} : \quad & \sum_{k=1}^{K_i} c_{ki} x^{a_{ki}} \leq 1, \quad i = 1, \dots, m, \\ & g_j x^{f_j} = 1, \quad j = 1, \dots, p. \end{aligned}$$

where a_0, \dots, a_m and c_1, \dots, c_m are vectors in \mathbf{R}^n , and $c_i > 0$, $k = 1, \dots, m$, $g > 0$ are vectors with positive components. In the above, we follow the [power law notation](#).

Converting the problem into the standard form when the original formulation involves generalized posynomials entails adding new variables and constraints (see [here](#)).

Example: [Optimization of a water tank](#).

Convex Form GPs in standard form are not convex, but a simple transformation allows to put them into an equivalent, convex form. The idea, already seen in the context of [posynomial functions](#), is to take logarithms of the original variables. Again we assume that the posynomials involved are not generalized posynomials, but ordinary ones.

Consider a posynomial constraint of the form

$$f(x) := \sum_{k=1}^K c_k x^{a_k} \leq 1.$$

where the vector C has positive components: $c > 0$. We can express this constraint in terms of the new variable $y \in \mathbf{R}^n$, defined as $y_i = \log x_i, i = 1, \dots, n$:

$$\log f(x) = \text{lse}(Ay + b) := \log \left(\sum_{k=1}^K e^{a_k^T y + b_k} \right) \leq 0,$$

where $b_k := \log c_k, k = 1, \dots, K$, A is the $K \times n$ matrix with rows a_1, \dots, a_K , and **lse** is the [log-sum-exp function](#).

Note that if the posynomial was actually a monomial, there would be only one term in the **lse** function, and the constraint would reduce to an ordinary affine inequality.

Since the log-sum-exp function is [convex](#), the above constraint is actually convex in the new variable y .

By this argument, the GP

$$\begin{aligned} \min_x \sum_{k=1}^{K_0} c_{k0} x^{a_{k0}} : \quad & \sum_{k=1}^{K_i} c_{ki} x^{a_{ki}} \leq 1, \quad i = 1, \dots, m, \\ & g_j x^{f_j} = 1, \quad j = 1, \dots, p. \end{aligned}$$

can be expressed as a convex problem

$$\min_x \text{lse}(A_0 y + b_0) : \quad \text{lse}(A_i y + b_i) \leq 0, \quad i = 1, \dots, m, \\ Fy + g = 0,$$

for appropriate matrices A_0, \dots, A_m, C and vectors b_0, \dots, b_m, d . Precisely, we set $b_i = \log c_i$, and A_i to be the matrices with rows $a_{ki}, k = 1, \dots, K_i, i = 0, \dots, m$; the matrix F contains the vectors f_j as rows, $j = 1, \dots, p$.

Example: [Convex form of the water tank optimization problem](#).

CVX Syntax

With the above convex representation of a GP, we can directly invoke CVX with the `logsumexp_sdp` function. CVX knows it is convex. The extension “`sdp`” at the end of the function refers to the fact that the function is actually approximated via a technique based on [semidefinite programming](#) (SDP).

In CVX there is actually no need to transform the problem into the standard convex format. All you need to do is let CVX know that the problem is a GP. This is done by replacing the command `cvx_begin` with `cvx_begin gp`. This is illustrated [here](#).

Some Applications of GP

- [Digital circuit design.](#)

Exercises

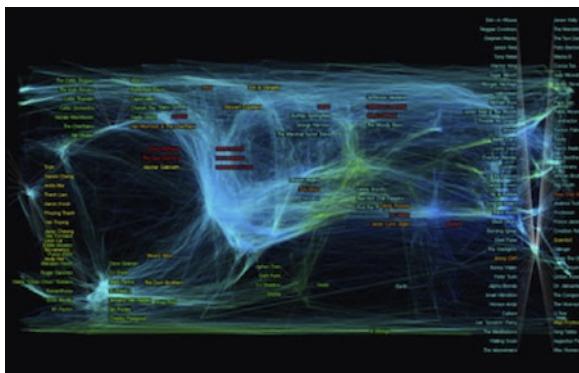
A. We seek to optimize the shape of a box-shaped structure with height h , width w , and depth d . Due to cost concerns, we have an upper limit A_{wall} on the total wall area $2(hw + hd)$, another upper limit A_{floor} on the floor area wd . For structural reasons, we have lower and upper bounds α, β on the aspect ratio h/w , and similar bounds γ, δ on the aspect ratio w/d . Subject to these constraints, we wish to maximize the volume of the structure, hwd .

1. Express the problem in the standard form for an optimization problem.
2. Express this problem as a geometric program (GP) in standard GP form. (*Note:* you are not asked to put it in the standard convex form for GPs.)

Semidefinite Programming

Semidefinite programming (SDP) is an optimization model where the objective is linear, and the constraints involve affine combinations of symmetric matrices that are required to be positive semi-definite. SDPs include as special cases LPs, when all the symmetric matrices involved are diagonal; and SOCPs, when the symmetric matrices have a special “arrow” form. General SDPs are currently one of the most powerful forms of convex optimization.

SDPs arise in a wide range of applications. For example, they can be used as sophisticated *relaxations* (approximations) of non-convex problems, such as boolean problems with quadratic objective. They are also useful in the context of analyzing the stability, or more generally, the time-behavior, of linear dynamical systems subject to perturbations. They can also allow to solve data visualization problems, in particular those where sparsity constraints are imposed on the vectors on which the data is projected.



The image shows a graph showing similarities between musicians based on Yahoo! music data. A method based on semidefinite programming, known as *maximum variance unfolding*, was used to represent the similarity data in two dimensions.

Outline

- [From LP to conic problems](#)
- [Linear matrix inequalities](#)
- [Standard forms of SDPs](#)
- [Applications](#)
- [Exercises](#)

From LP to Conic Optimization

In the late 1980s, researchers were trying to generalize linear programming. At that time, LP was known to be solvable efficiently. Theory even showed that LPs could be solved in time roughly cubic in the number of variables or constraints. The new interior-point methods for LP had just become available, and their excellent practical performance matched the theoretical complexity bounds. It seemed however that, beyond linear problems, one encountered a wall. Except for a few special problem classes, such as QP, it appeared that as soon as a problem contained non-linear terms, one would no longer hope the nice practical and theoretical efficiency allowed to LP.

In previous decades, it had been noted that convex problems could be efficiently solved in theory (under some mild conditions), however the known methods were extremely slow in practice. It seemed however that to harness the power of interior-point methods and apply it to problems other than LP, one had to look closely at convex optimization.

A breakthrough occurred by rethinking the role of the set of non-negative vectors, which is the basic object in LP. In a standard form, an LP can be written as

$$\min_x c^T x : Ax = b, \quad x \in \mathbf{R}_+^n,$$

where \mathbf{R}_+^n is the set of non-negative vectors. Researchers asked: “what are the basic characteristics of \mathbf{R}_+^n that make interior-point methods work so well? In other words, are there other sets that could be used in place of \mathbf{R}_+^n , and still allow efficient methods?” It turns out that it is always possible to restrict attention to cones, that is, sets that are invariant by positive scaling of their elements.

At first glance, one can replace the set \mathbf{R}_+^n by any convex cone, say \mathbf{K} ; the new problem

$$\min_x c^T x : Ax = b, \quad x \in \mathbf{K}$$

is convex. But, as mentioned above, convexity alone is not sufficient to guarantee the existence of a practically efficient algorithm to solve it. It turns out that there are a few convex cones \mathbf{K} that are amenable to an efficient extension of interior-point methods.

One if the second-order cone, or any combination of second-order cones (arising when, say some variables are in a cone, others in another, and all are coupled by affine equalities). The SOCP class was the first beneficiary of the new class of algorithms.

Another class of problems involve the cone of positive semi-definite matrices, which leads to the class of semi-definite programs (SDP). SDPs involve a linear objective, matrix variables, and affine equality constraints, and positive semi-definiteness constraints on the (matrix) variables.

For both SOCP and SDP, the interior-point methods that work so well for LP can be extended. This comes of course at a price: the size of SOCPs that can be solved in reasonable time is smaller than for LP, and the complexity of SDPs is even higher. What we gain in the process is a much wider expressive power of the model.

Linear Matrix Inequalities Definition Positive Semi-Definite Matrices. Recall from [here](#) that a $n \times n$ symmetric matrix F is *positive semi-definite* (PSD) if and only if every one of its eigenvalues is non-negative. We use the notation $F \succeq 0$ to mean that F is PSD.

An alternative condition for F to be PSD is that the associated quadratic form is non-negative:

$$\forall z \in \mathbf{R}^n : z^T F z \geq 0.$$

The set of PSD matrices is convex, since the conditions above represent (an infinite number of) ordinary linear inequalities on the elements of the matrix F .

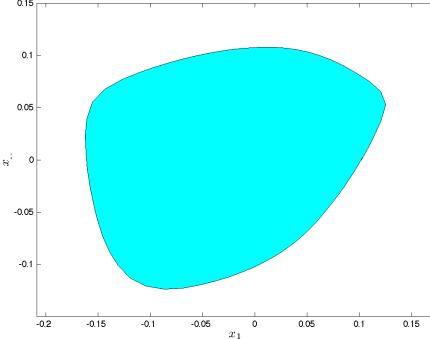
Examples:

- [A simple 3 × 3 example](#).
- For any vector $v \in \mathbf{R}^n$, the dyad $F = vv^T$ is PSD, since the associated quadratic form is $q(x) = x^T(vv^T)x = (v^T x)^2 \geq 0$.
- More generally, for any rectangular matrix A , the “square” matrix $F = A^T A$ is PSD.
- The converse is true: any PSD matrix can be factored as $A^T A$ for some appropriate matrix A .

$$F_0 + \sum_{i=1}^m x_i F_i \succeq 0,$$

Standard form. A *linear matrix inequality* is a constraint of the form

where the $n \times n$ matrices F_0, \dots, F_m are symmetric.



An LMI in two variables:

$$F(x) := x_1 F_1 + x_2 F_2 \preceq I,$$

where F_1, F_2 are the two 5×5 symmetric matrices

$$F_1 = \begin{pmatrix} -1.3 & -4.2 & -0.1 & 2.1 & -1 \\ -4.2 & -0.1 & -1.7 & -4.5 & 0.9 \\ -0.1 & -1.7 & 2.3 & -4.4 & -0.4 \\ 2.1 & -4.5 & -4.4 & 3.3 & -1.7 \\ -1 & 0.9 & -0.4 & -1.7 & 4.7 \end{pmatrix},$$

$$F_2 = \begin{pmatrix} 1.6 & 3.9 & 1.6 & -5.3 & -4 \\ 3.9 & -1.8 & -4.7 & 1 & 0.29 \\ 1.6 & -4.7 & -1.3 & 1.6 & -2.6 \\ -5.3 & 1 & 0.16 & 2.7 & 2.6 \\ -4 & 0.29 & -2.6 & 2.6 & -3.4 \end{pmatrix}.$$

The matrices F_0, \dots, F_m are referred to as the coefficient matrices. Sometimes, these matrices are not explicitly defined. That is, if $F : \mathbf{R}^m \rightarrow \mathbf{S}^n$ is an affine map that takes its values in the set of symmetric matrices of order n , then $F(x) \succeq 0$ is an LMI.

An alternate form for LMIs is as the intersection of the positive semi-definite cone with an affine set:

$$X \in \mathcal{A}, \quad X \succeq 0,$$

where \mathcal{A} is affine. The form we have seen before, and the above one, are equivalent, in the sense that we can always transform one into the other (at the expense possibly of adding new variables and constraints).

LMIs and Convex Sets

Let us denote by \mathbf{X} the set of points $x \in \mathbf{R}^m$ that satisfy the above LMI.

$$\mathbf{X} := \left\{ x : F_0 + \sum_{i=1}^m x_i F_i \succeq 0 \right\}.$$

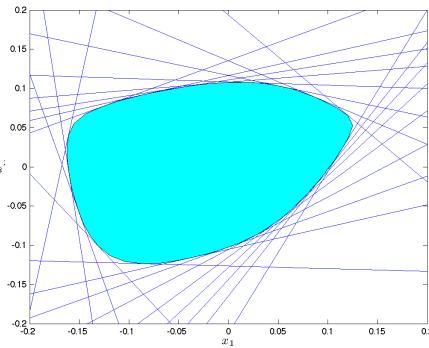
The set \mathbf{X} is convex. Indeed, we have $F(x) \succeq 0$ if and only if

$$\forall z \in \mathbf{R}^n : z^T F(x) z \geq 0.$$

Since

$$z^T F(x) z = f_0(z) + \sum_{i=1}^m x_i f_i(z)$$

with $f_i(z) := z^T F_i z$, $i = 0, \dots, m$, we obtain that the condition on x : $F(x) \succeq 0$ can be represented as an intersection of (an infinite number of) half-space conditions.



Supporting hyperplanes: The picture shows how the set seen above can be interpreted as the intersection of a number of half-spaces. Each half-space corresponds to an affine inequality of the form $z^T(I - F(x))z \geq 0$, for a specific vector z . The vectors z are chosen so that the line touches the boundary of the LMI set, so that it defines a [supporting hyperplane](#).

Multiple LMIs. We can combine multiple LMIs into one. Consider two affine maps from \mathbf{R} to a space of symmetric matrices of order $n_1, n_2, F_1 : \mathbf{R}^m \rightarrow \mathbf{S}^{n_1}, F_2 : \mathbf{R}^m \rightarrow \mathbf{S}^{n_2}$. Then the two LMIs $F_1(x) \succeq 0, F_2(x) \succeq 0$

are equivalent to *one* LMI, involving a larger matrix of size $(n_1 + n_2) \times (n_1 + n_2)$:

$$\begin{pmatrix} F_1(x) & 0 \\ 0 & F_2(x) \end{pmatrix} \succeq 0.$$

This corresponds to intersecting the two LMI sets.

Special Cases

LMIs include as special cases the following.

Ordinary affine inequalities. Consider single affine inequality in $x \in \mathbf{R}^n, a^T x \leq b$,

where $a \in \mathbf{R}^n, b \in \mathbf{R}$. (The above set describes a half-space.) The above is a trivial special case of LMI, where the coefficient matrices are scalar: $F_0 = b, F_i = -a_i, i = 1, \dots, n$.

Using the result above on multiple LMIs, we obtain that the set of ordinary affine inequalities

$$a_i^T x \leq b_i, \quad i = 1, \dots, m$$

can be cast as a single LMI $F(x) \succeq 0$, where

$$F(x) = \begin{pmatrix} b_1 - a_1^T x & & \\ & \ddots & \\ & & b_m - a_m^T x \end{pmatrix}.$$

Second-order cone inequalities. [Second-order cone \(SOC\) inequalities](#) can be represented as LMIs. To see this, let us start with the “basic”

$$\begin{pmatrix} t & y^T \\ y & tI_n \end{pmatrix} = \begin{pmatrix} t & y_1 & \cdots & y_n \\ y_1 & t & & \\ \vdots & & \ddots & \\ y_n & & & t \end{pmatrix} \succeq 0.$$

SOC $\|y\|_2 \leq t$, with $y \in \mathbf{R}^n$ and $t \in \mathbf{R}$. The SOC can be represented as

Indeed, we check that for every $z \in \mathbf{R}^n, \alpha \in \mathbf{R}$, we have

$$\begin{pmatrix} \alpha \\ z \end{pmatrix}^T \begin{pmatrix} t & y^T \\ y & tI_n \end{pmatrix} \begin{pmatrix} \alpha \\ z \end{pmatrix} = \|tz + \alpha y\|_2^2 + \alpha^2(t - \|y\|_2^2) \geq 0$$

if and only if $\|y\|_2 \leq t$.

More generally, a second-order cone inequality of the form

$$\|Ax + b\|_2 \leq c^T x + d,$$

with $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^m, c \in \mathbf{R}^n, d \in \mathbf{R}^n$, can be written as the LMI

$$\begin{pmatrix} c^T x + d & (Ax + b)^T \\ (Ax + b) & (c^T x + d)I_n \end{pmatrix} \succeq 0.$$

The proof relies on the [Schur complement lemma](#).

Standard Forms of SDP

Standard Inequality Form

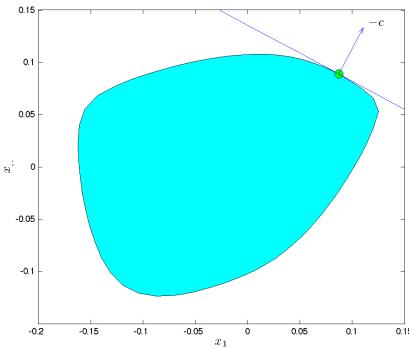
A *semidefinite program* (SDP) is a problem of minimizing a linear function over an LMI constraint. In standard inequality form, an SDP is written as

$$\min_x c^T x : F(x) := F_0 + x_1 F_1 + \dots + x_m F_m \succeq 0,$$

where F_0, \dots, F_m are given symmetric matrices, $c \in \mathbf{R}^n$, and $x \in \mathbf{R}^m$ is a vector variable.

The above problem generalizes the [LP in inequality form](#):

$$\min c^T x : a_i^T x \leq b_i, \quad i = 1, \dots, m.$$



An SDP in two variables: $\min_x \ c^T x$ subject to $F(x) := x_1 F_1 + x_2 F_2 \preceq I$, where $c = ()$, and F_1, F_2 are the two 5×5 symmetric matrices

$$F_1 = \begin{pmatrix} -1.3 & -4.2 & -0.1 & 2.1 & -1 \\ -4.2 & -0.1 & -1.7 & -4.5 & 0.9 \\ -0.1 & -1.7 & 2.3 & -4.4 & -0.4 \\ 2.1 & -4.5 & -4.4 & 3.3 & -1.7 \\ -1. & 00.9 & -0.4 & -1.7 & 4.7 \\ 1.6 & 3.9 & 1.6 & -5.3 & -4. \\ 3.9 & -1.8 & -4.7 & 1. & 02.9 \\ 1.6 & -4.7 & -1.3 & 1.6 & -2.6 \\ -5.3 & 1. & 01.6 & 2.7 & 2.6 \\ -4. & 02.9 & -2.6 & 2.6 & -3.4 \end{pmatrix},$$

$$F_2 = \begin{pmatrix} -1.3 & -4.2 & -0.1 & 2.1 & -1 \\ -4.2 & -0.1 & -1.7 & -4.5 & 0.9 \\ -0.1 & -1.7 & 2.3 & -4.4 & -0.4 \\ 2.1 & -4.5 & -4.4 & 3.3 & -1.7 \\ -1. & 00.9 & -0.4 & -1.7 & 4.7 \\ 1.6 & 3.9 & 1.6 & -5.3 & -4. \\ 3.9 & -1.8 & -4.7 & 1. & 02.9 \\ 1.6 & -4.7 & -1.3 & 1.6 & -2.6 \\ -5.3 & 1. & 01.6 & 2.7 & 2.6 \\ -4. & 02.9 & -2.6 & 2.6 & -3.4 \end{pmatrix}.$$

Standard Conic Form

Recall that we can define the scalar product between two matrices $X, Y \in \mathbf{R}^{m \times n}$ as

$$\langle X, Y \rangle = \text{Tr}(X^T Y).$$

If X, Y are both square, and symmetric, then the scalar product is simply the trace of the product.

In standard conic form, an SDP is written as

$$\min_X \langle C, X \rangle : X \succeq 0, \quad \langle A_i, X \rangle = b_i, \quad i = 1, \dots, m,$$

where A_1, \dots, A_m and C are given symmetric $n \times n$ matrices, and $X \in \mathbf{S}^n$ is the matrix variable.

The above generalizes the standard conic form for LP, which can be written as

$$\min_x c^T x : Ax = b, \quad x \geq 0.$$

Just as in LP, the standard inequality and standard conic forms are equivalent, in the sense that we can always convert one into the other, possibly at the expense of introducing new variables and constraints.

CVX syntax

In CVX, a constraint $X \succeq 0$, when X is a symmetric $n \times n$ matrix variable, is encoded with the `semidefinite` assignment. It is important to let CVX know X is symmetric, when declaring it as a variable.

CVX syntax

```
cvx_begin
variables X(n,n) symmetric;
minimize( trace(C*X) )
subject to
for i = 1:m,
trace(A(i)*X) == b(i);
end
X == semidefinite(n);
cvx_end
```

Some Applications of SDP

- [Bounding portfolio risk with incomplete covariance information](#)
- [Boolean quadratic programming](#)
- [Matrix completion](#)
- [Robust stability of linear dynamical systems](#)

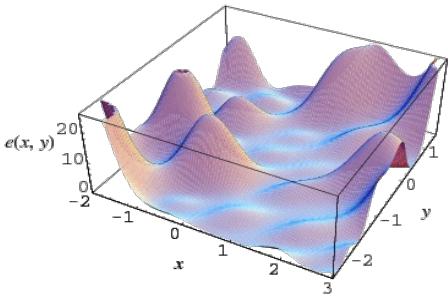
Exercises A. Optimization of ellipsoid shapes. For $P \in \mathbf{R}^{n \times n}$, with P symmetric and positive-definite, we define the ellipsoid $\mathbf{E}(P) := \{x : x^T P x \leq 1\}$.

A measure of the “size” of the ellipsoid is $\text{Tr} P^{-1}$, with Tr the trace (the sum of the diagonal elements of the matrix argument).

1. Motivate our choice of the size function. Hint: Figure out the semi-axis lengths of the ellipsoid as a function of P .
2. Show that $\mathbf{E}(P) = \{R^{-1}u : \|u\|_2 \leq 1\}$, where R is a factor of P (any matrix R such that $P = R^T R$; in matlab, R can be obtained via the command `chol()`.)
3. Show that for any $Q = Q^T$, Q positive-definite, the set $\mathbf{E}(Q)$ is contained in $\mathbf{E}(P)$ if and only if $Q - P$ is positive semi-definite.
4. For given $n \times n$ symmetric matrices P_i , $i = 1, 2$, both positive-definite, show how to compute an ellipsoid, centered at the origin, that contains both $\mathbf{E}(P_1), \mathbf{E}(P_2)$ and has minimum size.
5. Implement and plot your result with the data contained in [here](#). (The function $P \rightarrow \text{Tr} P^{-1}$ for P symmetric and positive definite, is implemented in CVX using `trace_inv()`.)

Non-Convex Problems

IN PROGRESS



Many practical problems of importance are non-convex, and most non-convex problems are hard (if not impossible) to solve exactly in a reasonable time. Hence the idea of using *heuristic* algorithms, which may or may not produce desired solutions.

In *alternate minimization* techniques, the optimization is carried out with some variables are held fixed in cyclical fashion; in *linearization* techniques, the objectives and constraints are linearized (or approximated by a convex function). Other techniques include search algorithms (such as genetic algorithms), which rely on simple solution update rules to progress. Another idea is to use *approximation* algorithms, which typically produce a bound on the problem (and sometimes, a feasible solution that achieves the bound).

An important class of non-convex problems deserves special treatment: Boolean problems, or more generally, problems with discrete (say, integer) variables. We only devote a brief space to these problems.

In contrast to convex problems, for which the landscape is more or less understood, there is no easy classification of non-convex problems; nor is there a short way to summarize the vast range of possibilities and heuristics. In this section we only provide a small sample of solution techniques and of applications.

Outline

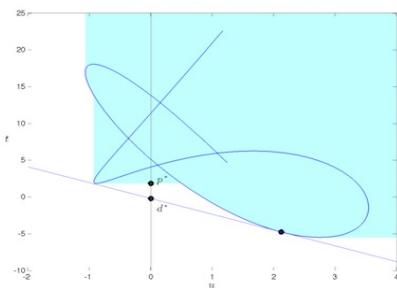
- Overview
- Coordinate descent methods
- Linearization
- Approximation via convexity
- Discrete optimization
- Applications
- Exercises

Approximation via convex models

4. Duality

- [Weak duality](#)
- [Strong duality](#)

Weak Duality



This chapter shows how the notion of weak duality allows to develop, in a systematic way, approximations of non-convex problems based on convex optimization. The above illustrates the idea, as detailed [here](#).

Starting with any given minimization problem, which we call “primal” for reference, we can form a “**dual**” problem. The dual problem is a concave maximization problem which provides a lower bound on the value of the original problem. This *weak duality* result allows to form bounds on non-convex problems using convex optimization.

When the original problem is convex, and with some mild extra conditions, the value of the “dual” problem is the same as that of the “primal”, and we say that *strong duality* holds. Strong duality is covered [here](#).

Geometric View of Weak Duality Two-dimensional view of the primal problem

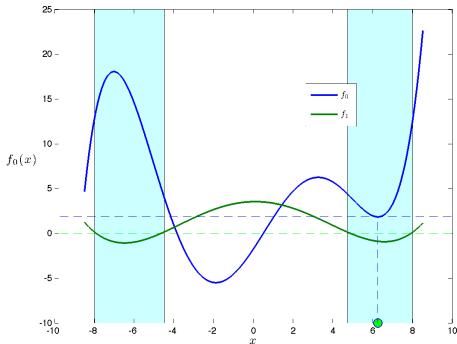
Let us assume for simplicity that $m = 1$, and consider the problem:

$$\min_x f_0(x) : f_1(x) \leq 0.$$

In fact, this can be done without loss of generality, using the pointwise maximum of the constraints as a single constraint.

To illustrate, we focus on the problem above, with f_0, f_1 defined as

$$f_0(x) := \begin{cases} 0.0025x^5 - 0.00175x^4 - 0.212625x^3 \\ + 0.3384375x^2 + 3.368x - 1.692 \\ +\infty \end{cases} \quad \begin{aligned} & -10 \leq x \leq 10, \\ & \text{otherwise,} \quad f_1(x) := 0.0025x^4 - 0.0005x^3 - 0.2129x^2 + 0.0320x + 3.5340. \end{aligned}$$



A one-dimensional problem: the picture shows the problem above, which is to minimize a fifth-order polynomial on the domain $[-10, 10]$, with one quadratic constraint. The constraint expresses the fact that x should belong to the *union* of two intervals (region in light blue). The optimal point is shown (in green on the x -axis).

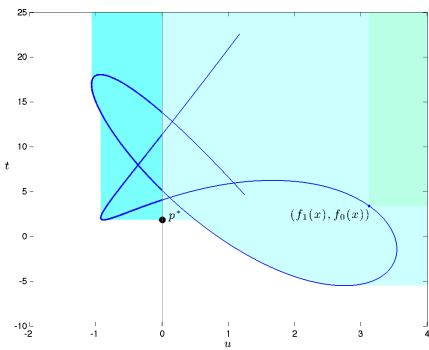
We can express the primal problem with two new scalar variables u, t , as follows:

$$\begin{aligned} p^* &= \min_{u,t,x} \{t : 0 \geq u \geq f_1(x), t \geq f_0(x)\} \\ &= \min_{u,t} t : (u, t) \in \mathbf{A}_{\text{feas}}, \end{aligned}$$

where

$$\begin{aligned} \mathbf{A} &:= \{(u, t) \in \mathbf{R}^2 : \exists x, u \geq f_1(x), t \geq f_0(x)\}, \\ \mathbf{A}_{\text{feas}} &:= \{(u, t) \in \mathbf{A} : u \leq 0\}. \end{aligned}$$

The set \mathbf{A} can be interpreted as the set of pairs (u, t) that can be upper bounds on constraint / objective function pairs, by some choice of the original decision variable x . The set \mathbf{A}_{feas} corresponds to pairs that are achievable by some x that is *feasible* for the original problem.



Dual function

The dual function is

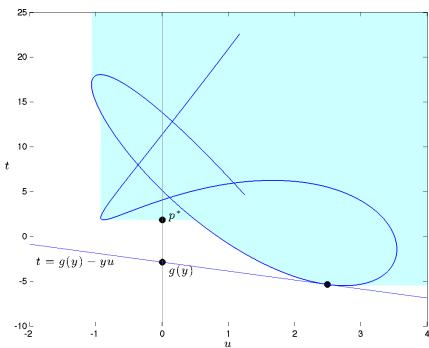
$$g(y) = \min_x f_0(x) + y f_1(x).$$

In the scalar problem above, the dual function is easy to compute, either by a direct (brute force) scan of values in $[-10, 10]$, or by taking derivatives of the above objective function, and roots of a 4-th order polynomial.

The geometric interpretation of $g(y)$ is as follows. We have

$$g(y) = \min_{u,t} t + yu : (u, t) \in \mathbf{A}.$$

Consider a non-vertical line in (u, t) plane, with non-positive slope given by $-y \leq 0$. This line can be expressed as $t + yu = \text{constant}$, where the value of the constant is the *intercept*, that is, the point on the line on the t -axis.

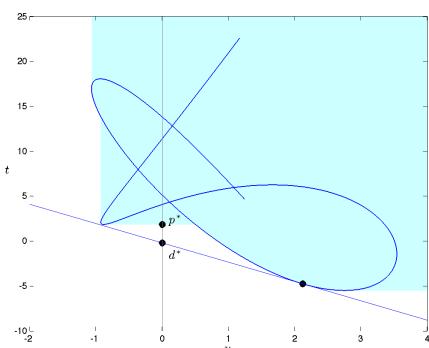


Dual problem

The dual problem:

$$d^* := \max_{y \geq 0} g(y),$$

amounts to finding the best line, that is, search for slopes $y \geq 0$ that achieves the highest intercept. It corresponds to replacing \mathbf{A} with its convex hull.



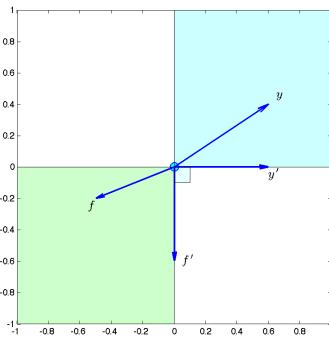
Two-dimensional representation of the problem above: The line in blue is the set of points of the form $(u, t) = (f_1(x), f_0(x))$ when x runs $[-10, 10]$. The set \mathbf{A} (blue and darker blue) is constructed as follows: for any point on the line, include all points that are to the upper right quadrant from that point (we show an example, in light green).

The part of \mathbf{A} in darker blue, which is \mathbf{A}_{feas} , corresponds to $u \leq 0$. The latter part of the blue line (which is inside \mathbf{A}) corresponds to the points x that are feasible for the original problem. The optimal value of the original problem, p^* , is the smallest value of t that can be achieved inside \mathbf{A}_{feas} .

Dual function for the problem above: We plot the line $t = g(y) - yu$, where $g(y) = \min_x f_0(x) + y f_1(x)$, for $y = 1$. The value of $g(1)$ corresponds to the line $t = g - u$ which has the smallest intercept g , and intersects the set \mathbf{A} . We check on the plot that $g(y)$ is a lower bound on the optimal value, irrespective of the value of the downward slope $-y \leq 0$.

Minimax Inequality/Primal problem as a minimax problem

The Lagrange function allows to rewrite the problem in an unconstrained fashion, as follows.



Right angle property: For any fixed $f \in \mathbf{R}^m$,

$$\max_{y \geq 0} y^T f = \begin{cases} 0 & \text{if } f \leq 0 \\ +\infty & \text{otherwise.} \end{cases}$$

That is, for any pair of vectors $f, -y \in \mathbf{R}_+^n$, the angle between f and $-y$ is always less or equal than 90° . This comes from the fact that the non-negative orthant \mathbf{R}_+^n is a convex cone with a right angle at the origin.

Thus, for fixed x , applying the above to $f = f(x) := (f_1(x), \dots, f_m(x))$,

$$\max_{y \geq 0} y^T f(x) = \begin{cases} 0 & \text{if } f \leq 0 \\ +\infty & \text{otherwise.} \end{cases}$$

The above function (of x) acts like a *barrier* for the feasible set of the primal problem: it is 0 whenever x is feasible, and $+\infty$ outside the feasible set. Thus:

$$p^* = \min_x \max_{y \geq 0} L(x, y).$$

Minimax inequality

The *minimax inequality* states that for any function of two vector variables $L : \mathbf{R}^n \times \mathbf{R}^m \rightarrow \mathbf{R}$, and two subsets $\mathbf{X} \subseteq \mathbf{R}^n$, $\mathbf{Y} \subseteq \mathbf{R}^m$, we have

$$\min_{x \in \mathbf{X}} \max_{y \in \mathbf{Y}} L(x, y) \geq \max_{y \in \mathbf{Y}} \min_{x \in \mathbf{X}} L(x, y).$$

The proof is very simple.

Applying this result to the expression of p^* as a minimax, we obtain

$$p^* \geq d^* = \max_{y \geq 0} \min_x L(x, y) = \max_{y \geq 0} g(y),$$

where g is the dual function.

Interpretation as a game

We can interpret the minimax inequality in the context of a game, with primal and dual variables acting as adversaries.

Consider a game with two players X and Y : X decides over the primal variables, and seeks to minimize the “cost”, $L(x, y)$; player Y plays the dual variables y , subject to the constraint $y \geq 0$, and seeks to maximize the “payoff”, $L(x, y)$. We assume that one of the players goes first, the game is played only once, and both players have full information on the other's choice, once their decision is made.

If X plays first, it is at a disadvantage: it does not know the decision of Y in advance, and must prepare for the worst outcome. In contrast, Y can adapt to whatever the decision of X is, and make a better choice than if it could not adapt. Therefore, we expect the cost (to X) to be higher in the case when X plays first than if it plays second.

Let us assume X plays first, and chooses the decision vector x . Since that decision is known to Y once it is made, the optimal strategy for Y is to set y to be optimal for the optimization problem (here x is fixed)

$$\max_{y \geq 0} L(x, y).$$

The best choice for X , knowing that Y will adapt this way to any of its decisions x , is to set x to be such that the above payoff (to Y) is the smallest. That is, the best decision for X (when it plays first) is to set x to solve the problem

$$p^* = \min_x \max_{y \geq 0} L(x, y).$$

The above represent the minimum cost that X is going to pay in the worst-case, that is, in the case when Y plays optimally as a second player.

Now assume that X plays second. This time, it is X who adapts to the decision of Y . For any given such decision y , the best decision (for X) is to solve

$$\min_x L(x, y).$$

This is nothing else than the dual function. That is, the dual function tells us the optimal strategy (for X) as a function of the dual variable y .

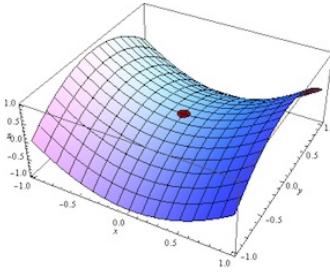
The best choice for Y , knowing that X will adapt this way to any of its decisions y , is to set y to be such that the above cost (to X) is the largest. That is, the best decision for Y (when it plays first) is to set y to solve the problem

$$d^* = \max_{y \geq 0} \min_x L(x, y).$$

The minimax inequality simply states that indeed X is at a disadvantage if it plays first.

Strong Duality

We examine the concept of duality in the context of a convex optimization problem.



For any minimization problem, weak duality allows us to form a dual problem which provides a lower bound on the original problem. The dual problem is always convex (it is a concave maximization problem).

We say that strong duality holds if the primal and dual optimal values coincide. In general, strong duality does not hold. However, if a problem is convex, and strictly feasible, then the value of the primal is the same as that of the dual, and the dual problem is attained. This is in essence Slater's theorem. If in addition the dual is strictly feasible, then both problems are attained. In that case, the Lagrangian has a saddle-point, as illustrated in the figure.

For convex problems where strong duality holds, and both primal and dual optimal values are attained, we can proceed to derive necessary and sufficient conditions for optimality, which are more amenable to algorithms than the supporting hyperplane conditions given [here](#). At optimum, the variables of the dual problem can be interpreted as *sensitivities* of the optimal value with respect to a class of perturbations. This is one important benefit of duality for engineering problems.

Strong duality for convex problems has a number of important applications, including for solving large-scale convex problems over a cluster of computers.

Outline

- [Slater's theorem for strong duality](#)
- [Optimality conditions](#)
- [Sensitivity](#)
- [Applications](#)
- [Exercises](#)

Slater Condition for Strong Duality Primal and Dual Problems

We consider a convex constrained optimization problem in [standard form](#):

$$p^* := \min_x f_0(x) : Ax = b, f_i(x) \leq 0, i = 1, \dots, m,$$

where $f_0, \dots, f_m : \mathbf{R}^n \rightarrow \mathbf{R}$ are convex functions, $A \in \mathbf{R}^{p \times n}, b \in \mathbf{R}^p$ define the affine inequality constraints.

To this problem we associate the Lagrangian, which is the function $L : \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R}^p \rightarrow \mathbf{R}$ with values

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \nu^T (Ax - b).$$

The corresponding dual function is the function $g : \mathbf{R}^m \times \mathbf{R}^p \rightarrow \mathbf{R}$ with values

$$g(\lambda, \nu) = \min_x f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \nu^T (Ax - b).$$

Recall that the function g is concave, and that it can assume $-\infty$ values. Its domain is

$$\text{dom } g := \{(\lambda, \nu) : g(\lambda, \nu) > -\infty\}.$$

Finally, the dual problem reads

$$d^* = \max_{\lambda \geq 0, \nu} g(\lambda, \nu).$$

Note that the sign constraints are imposed only on the dual variables corresponding to inequality constraints. Note also that there are (possibly) implicit constraints in the above problem, since we must have $(\lambda, \nu) \in \text{dom } g$.

Strong duality

The theory of weak duality seen here states that $p^* \geq d^*$. This is true always, even if the original problem is not convex. We say that *strong duality* holds if $p^* = d^*$.

Slater's sufficient condition for strong duality

Slater's theorem provides a *sufficient condition* for strong duality to hold. Namely, if

- The primal problem is convex;
- It is strictly feasible, that is, there exists $x_0 \in \mathbf{R}^n$ such that

$$Ax_0 = b, f_i(x_0) < 0, i = 1, \dots, m,$$

then, strong duality holds: $p^* = d^*$, and the dual problem is attained. (Proof)

Example:

- [Minimum distance to an affine subspace](#).
- [Dual of LP](#).
- [Dual of QP](#).

Geometry

The geometric interpretation of weak duality shows why strong duality holds for a convex, strictly feasible problem. For simplicity again, we consider the case with no equality constraints, and a single convex constraint.

Recall that we can express the primal problem with two new scalar variables u, t , as follows:

$$p^* = \min_{u,t} t : (u, t) \in \mathbf{A}, u \leq 0,$$

where

$$\mathbf{A} := \{(u, t) \in \mathbf{R}^2 : \exists x, u \geq f_1(x), t \geq f_0(x)\}.$$

Since the primal problem is convex, that is, f_0 and f_1 are convex functions, the above set is convex.

Strict primal feasibility means that the set \mathbf{A} cuts “inside” the right-half of the (u, t) -plane. If that property holds, then we can attain the optimal point $(0, p^*)$ by a tangent with a finite strictly negative slope. One implication is that $d^* = p^*$, that is, strong duality holds. This slope is precisely the optimal dual variable, λ^* ; thus the dual problem is attained.

Optimality Conditions Sufficient condition for dual optimum attainment

Slater condition, namely strict feasibility of the primal, ensures that the dual problem is attained.

Primal optimum attainment

Likewise, if in addition the dual problem is strictly feasible, that is if:

$$\exists \lambda > 0, \nu : g(\lambda, \nu) > -\infty,$$

then strong duality holds, and both problems are attained, that is: there exist (x, λ, ν) such that

- x is feasible for the primal problem;
- λ, ν are feasible for the dual problem: $\lambda \geq 0$, and $(\lambda, \nu) \in \text{dom } g$.

Example:

Complementary slackness

Assume that strong duality holds, and both primal and dual problems are attained, by x^* and (λ^*, ν^*) respectively. Then we have

$$f_0(x^*) = g(\lambda^*, \nu^*) \leq f_0(x^*) + \sum_{i=1}^m \lambda_i^* f_i(x^*) + \sum_{i=1}^p \nu_i^* h_i(x^*) \leq f_0(x^*),$$

where the first inequality is by definition of the dual function as a minimum over x , and the second from the fact that x^* is feasible. Hence the sum in the above is zero. Since every term in that sum is non-positive, each term is zero: $\lambda_i^* f_i(x^*) = 0, i = 1, \dots, m$. The above are called the *complementary slackness* conditions. They imply that if the i -th constraint is strictly satisfied, then the corresponding dual variable is zero. Conversely, if $\lambda_i > 0$ then we must have $f_i(x) = 0$.

Optimality conditions

Consider the following conditions:

- Primal feasibility:
- Dual feasibility: $\lambda \geq 0$
- Lagrangian stationarity: (in the case when every function involved is differentiable)

$$0 = \nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x) + \sum_{i=1}^p \nu_i \nabla h_i(x)$$

- Complementary slackness are called the *Karush-Kuhn-Tucker* (KKT) conditions.

If the problem is convex, and satisfies Slater's condition, then a primal point is optimal if and only if there exist (λ, ν) such that the KKT conditions are satisfied. Conversely, the above conditions guarantee that strong duality holds, and (x, λ, ν) are optimal.

Sensitivity Consider the problem:

$$p^* := \min_x f_0(x) \text{ s.t. } Ax = b, f_i(x) \leq 0, i = 1, \dots, m.$$

We are interested in analyzing how the value changes when we modify the problem's right-hand side. To this end, we consider a perturbed problem:

$$p^*(u, v) := \min_x f_0(x) : Ax = b + v, f_i(x) \leq u_i, i = 1, \dots, m,$$

where vectors u, v are given. In this setting, we have $p^*(0, 0) = p^*$, and the value $p^*(u, v)$ tells us how the optimal value changes when we change the right-hand sides (vector b and the upper bound on the f_i 's, which is nominally zero).

The dual of the problem is

$$d^* = \max_{\lambda \geq 0, \nu} g(\lambda, \nu).$$

The dual of perturbed problem is:

$$d^* = \max_{\lambda \geq 0, \nu} g(\lambda, \nu) - \lambda^T u - \nu^T v.$$

with the *same* dual function \mathcal{G} .

Applying weak duality to the perturbed problem:

$$p^*(u, v) \geq g(\lambda^*, \nu^*) - u^T \lambda^* - v^T \nu^*.$$

This provides a one-sided sensitivity result: for example, if $u \leq 0$, then (since $\lambda^* \geq 0$) the perturbed optimal value increases.

If $p^*(\cdot, \cdot)$ is differentiable, then (λ^*, ν^*) is the gradient with respect to (u, v) .

Applications of Strong Duality

- A simple problem: sum of the k largest components of a vector.
- [Decomposition methods for distributed convex optimization](#).
- Kernel methods in machine learning.
- Dual projected gradient method for QP.

Exercises

A. Consider the so-called "box-constrained" QP:
 $\min_x x^T Q x$ subject to: $l \leq x \leq u$,

where the vectors $u, l \in \mathbf{R}^n$ are given (with $l \leq u$) and Q is a positive-definite matrix. We further assume that Q has a "diagonal plus rank-one" structure, that is: $Q = D + qq^T$, where D is diagonal, positive-definite, and $q \in \mathbf{R}^n$ is given.

1. Assume first that $q = 0$. Show that the problem then has a very simple, closed-form solution. What is the complexity (in terms of number of flops) of the problem in that case?
2. Now assume $q \neq 0$. Derive a dual problem, after introducing a new variable z and the constraint $z = q^T x$. Express the dual problem as a convex optimization problem with one variable.
3. Discuss the benefits of solving the dual problem instead of the primal.
4. How can you recover an optimal primal solution based on that of the dual? Hint: use the result here.

B. Consider the sum of the k largest magnitude elements in a given vector $x \in \mathbf{R}^n$, with $k \leq n$:

$$s_k(x) := \sum_{i=1}^k |x|_{[i]},$$

where $|x|$ is the vector formed with the absolute values of the entries of x and, for a vector $z \in \mathbf{R}^n$, and $i \in \{1, \dots, n\}$, $z_{[i]}$ stands for the i -th largest component of z . We have

$$s_k(x) = \max_u u^T |x| : u \in \{0,1\}^n, \sum_{i=1}^n u_i = k.$$

1. We define a Lagrangian \mathcal{L} corresponding to the above problem, where we keep the Boolean constraints $u \in \{0,1\}^n$:

$$\mathcal{L}(u, \nu) := u^T |x| + \nu \left(k - \sum_{i=1}^n u_i \right).$$

Express the value $s_k(x)$ as the optimal value of a max-min problem. Hint: consider the function $u \rightarrow \min_\nu \mathcal{L}(u, \nu)$.

2. Show that the dual function reads

$$g(\nu) := \max_{u \in \{0,1\}^n} \mathcal{L}(u, \nu) = k\nu + \sum_{i=1}^n \max(0, |x|_i - \nu).$$

3. Explain why $s_k(x) \leq g(\nu)$ for any scalar ν . Hint: use the minimax inequality.

4. Find a value ν^* such that $g(\nu^*) = s_k(x)$.

5. Explain why $s_k(x) = d^*$, where d^* is the optimal value of the dual problem:

$$d^* := \min_\nu g(\nu).$$

6. Show that the problem

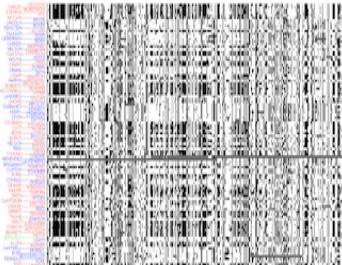
$$\min_x s_k(x) : Ax \leq b,$$

where $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$ are given, can be formulated as an LP in $2n+1$ variables and $3n+m$ constraints. Comment. Hint: Use the dual expression for $s_k(x)$, and introduce extra variables to represent maxima of linear functions.

5. Case Studies

- 5.1 [Senate voting](#)
- 5.2 [Antenna array design](#)
- 5.3 [Digital circuit design](#)
- 5.4 [Cash-flow management](#)

Senate Voting: Analysis and Visualization



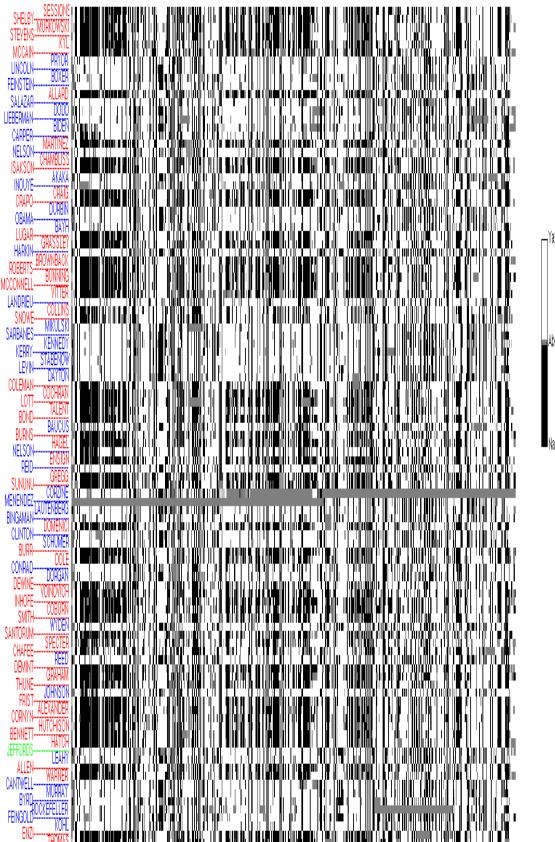
In this case study, we take data from the votes on bills in the US Senate (2004-2006), shown as a table above, and explore how we can visualize the data by projecting it, first on a line then on a plane. We investigate how we can choose the line or plane in a way that maximizes the variance in the result, via a principal component analysis method. Finally we examine how a variation on PCA that encourages sparsity of the projection directions allows to understand which bills are most responsible for the variance in the data.

Source: [VoteWorld](#).

Senate Voting: Data and Visualization ProblemData

The data consists of the votes of $n = 100$ Senators in the 2004-2006 US Senate (2004-2006), for a total of $m = 542$ bills. “Yay” (“Yes”) votes are represented as 1’s, “Nay” (“No”) as -1’s, and the other votes are recorded as 0. (A number of complexities are ignored here, such as the possibility of pairing the votes.)

This data can be represented here as a $m \times n$ “voting” matrix $X = [x_1, \dots, x_n]$, with elements taken from $\{-1, 0, 1\}$. Each column of the voting matrix $x_j, j = 1, \dots, n$ contains the votes of a single Senator for all the bills; each row contains the votes of all Senators on a particular bill.



Senate voting matrix: “Nay” votes are in black, “Yay” ones in white, and the others in grey. The transpose voting matrix is shown. The picture becomes very informative. Simply plotting the raw data matrix is often not very informative.

Source: [VoteWorld](#).

We can try to visualize the data set, by projecting each data point (each row or column of the matrix) on (say) a 1D-, 2D- or 3D-space. Each “view” corresponds to a particular projection, that is, a particular one-, two- or three-dimensional subspace on which we choose to project the data. The *visualization problem* consists of choosing an appropriate projection.

There are many ways to formulate the visualization problem, and none dominates the others. Here, we focus on the basics of that problem.

Senate Voting: Projecting Data onto a line

To simplify, let us first consider the simple problem of representing the high-dimensional data set on a simple line, using the method described [here](#).

Scoring Senators

Specifically we would like to assign a single number, or “score”, to each column of the matrix. We choose a direction $u \in \mathbf{R}^m$, and a scalar $v \in \mathbf{R}$. This corresponds to the affine “scoring” function $f : \mathbf{R}^m \rightarrow \mathbf{R}$, which, to a generic column $x \in \mathbf{R}^m$ of the data matrix, assigns the value

$$f(x) = u^T x + v.$$

We thus obtain a vector of values $f \in \mathbf{R}^n$, with $f_j = u^T x_j + v, j = 1, \dots, n$. It is often useful to center these values around zero. This can be done by choosing v such that

$$0 = \sum_{j=1}^n (u^T x_j + v) = u^T \left(\sum_{j=1}^n x_j \right) + n \cdot v,$$

that is: $v = -u^T \hat{x}$, where

$$\hat{x} := \frac{1}{n} \sum_{j=1}^n x_j \in \mathbf{R}^m$$

is the vector of [sample averages](#) across the columns of the matrix (that is, data points). The vector \hat{x} can be interpreted as the “average response” across experiments.

The values of our scoring function can now be expressed as

$$f(x) = u^T (x - \hat{x}).$$

In order to be able to compare the relative merits of different directions, we can assume, without loss of generality, that the vector u is normalized (so that $\|u\|_2 = 1$).

Centering data

It is convenient to work with the “centered” data matrix, which is

$$X_{\text{cent}} = (x_1 - \hat{x} \ \dots x_n - \hat{x}) = X - \hat{x} \mathbf{1}_n^T,$$

where $\mathbf{1}_n$ is the vector of ones in \mathbf{R}^n .

In matlab, we can compute the centered data matrix as follows.

Matlab syntax

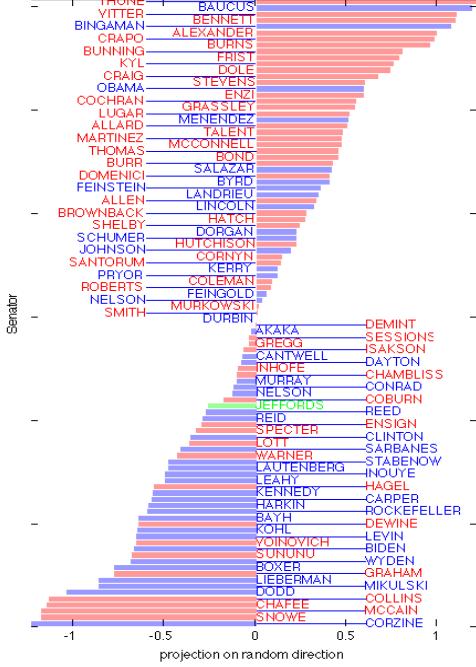
```
>> xhat = mean(X, 2);
>> [m, n] = size(X);
>> Xcent = X - xhat * ones(1, n);
```

We can compute the (row) vector scores using the simple [matrix-vector product](#):

$$f = u^T X_{\text{cent}} \in \mathbf{R}^{1 \times m}.$$

We can check that the average of the above row vector is zero:

$$f \mathbf{1}_n = u^T X_{\text{cent}} \mathbf{1}_n = u^T (X - \hat{x} \mathbf{1}_n^T) \mathbf{1}_n = u^T (X \mathbf{1}_n - n \cdot \hat{x}) = 0.$$



Scores obtained with random direction: This image shows the values of the projections of the Senator's votes $x_j - \hat{x}$ (that is, with average across Senators removed) on a (normalized) "random bill" direction. This projection shows no particular obvious structure. Note that the range of the data is much less than obtained with the average bill shown above.

Projection on a plane

We can also try to project the data on a plane, which involves assigning *two* scores to each data point.

Scoring map

This corresponds to the affine "scoring" map $f : \mathbf{R}^m \rightarrow \mathbf{R}$, which, to a generic column $x \in \mathbf{R}^m$ of the data matrix, assigns the two-dimensional value

$$f(x) = \begin{pmatrix} u_1^T x + v_1 \\ u_2^T x + v_2 \end{pmatrix} = U^T x + v,$$

where $u_1, u_2 \in \mathbf{R}^m$ are two vectors, and v_1, v_2 two scalars, while $U = [u_1, u_2] \in \mathbf{R}^{m \times 2}, v \in \mathbf{R}^2$.

The affine map f allows to generate n two-dimensional data points (instead of m -dimensional) $f_j = U^T x_j + v, j = 1, \dots, n$. As before, we can require that the f_j 's be centered:

$$0 = \sum_{j=1}^n f_j = \sum_{j=1}^n (U^T x_j + v),$$

by choosing the vector v to be such that $v = -U^T \hat{x}$, where $\hat{x} \in \mathbf{R}^m$ is the "average response" defined above. Our (centered) scoring map takes the form

$$f(x) = U^T(x - \hat{x}).$$

We can encapsulate the scores in the $2 \times n$ matrix $F = [f_1, \dots, f_n]$. The latter can be expressed as the matrix-matrix product

$$F = U^T X_{\text{cent}} = \begin{pmatrix} u_1^T X_{\text{cent}} \\ u_2^T X_{\text{cent}} \end{pmatrix},$$

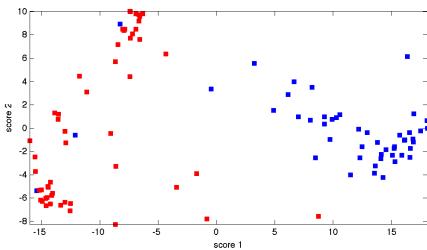
with X_{cent} the centered data matrix defined above.

Clearly, depending on which plan we choose to project on, we get a very different pictures. Some planes seem to be more "informative" than others. We return to this issue here.



Two-dimensional projection of the Senate voting matrix: This particular projection does not seem to be very informative. Notice in particular, the scale of the vertical axis. The data is all but crushed to a line, and even on the horizontal axis, the data does not show much variation.

Two-dimensional projection of the Senate voting matrix: This particular projection seems to allow to cluster the Senators along party line, and is therefore more informative. We explain how choose such a direction [here](#).



Senate Voting: Principal Component Analysis Direction of maximal variance Motivation

We have seen [here](#) how we can choose a direction in bill space, and then project the Senate voting data matrix on that direction, in order to visualize the data along a single line. Clearly, depending on how we choose the line, we will get very different pictures. Some show large variation in the data, others seems to offer a narrower range, even if we take care to normalize the directions.

What could be a good criterion to choose the direction we project the data on?

It can be argued that a direction that results in large variations of projected data is preferable to a one with small variations. A direction with high variation "explains" the data better, in the sense that it allows to distinguish between data points better. One criteria that we can use to quantify the variation in a collection of real numbers is the [sample variance](#), which is the sum of the squares of the differences between the numbers and their average.

Solving the maximal variance problem

Let us find a direction which maximizes the empirical variance. We seek a (normalized) direction u such that the empirical variance of the projected values $u^T a_j, j = 1, \dots, n$, is large. If \hat{a} is the vector of averages of the a_j 's, then the average of the projected values is $u^T \hat{a}$. Thus, the direction of maximal variance is one that solves the optimization problem

$$\max_{u : \|u\|_2=1} \frac{1}{n} \sum_{j=1}^n ((x_j - \hat{a})^T u)^2.$$

The above problem can be formulated as

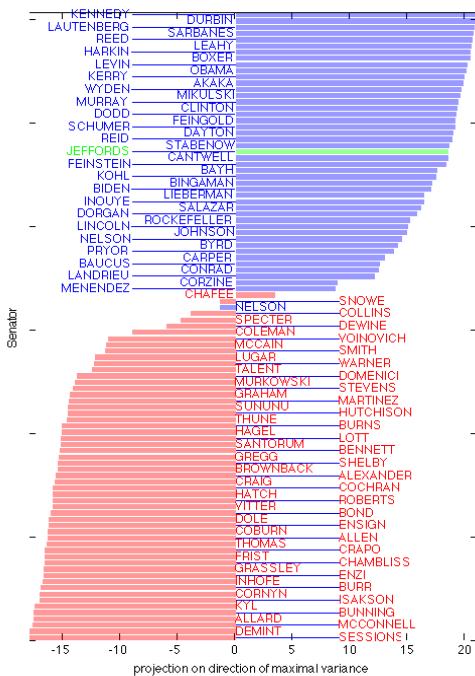
$$\max_{u : \|u\|_2=1} u^T \Sigma u,$$

where

$$\Sigma := \frac{1}{n} \sum_{j=1}^n (x_j - \hat{a})(x_j - \hat{a})^T$$

is the $m \times m$ [sample covariance matrix](#) of the data. The interpretation of the coefficient Σ_{kl} is that it provides the covariance between the votes of Senator k and those of Senator l .

We have seen the above problem before, under the name of the [Rayleigh quotient](#) of a symmetric matrix. Solving the problem entails simply finding an eigenvector of the covariance matrix Σ that corresponds to the largest eigenvalue.



This image shows the scores assigned to each Senator along the direction of maximal variance, $u_{\max}^T (x_j - \hat{x})$, $j = 1, \dots, n$, with u_{\max} a normalized eigenvector corresponding to the largest eigenvalue of the covariance matrix Σ . Republican Senators tend to score positively, while we find many Democrats on the negative score. Hence the direction could be interpreted as revealing the party affiliation.

Note that the largest absolute score (about 18) obtained in this plot is about three times bigger than that observed on the previous one. This is consistent with the fact that the current direction has maximal variance.

Principal component analysis Main idea

The main idea behind *principal components analysis* is to first find a direction that corresponds to maximal variance between the data points. The data is then projected on the hyperplane orthogonal to that direction. We obtain a new data set, and find a new direction of maximal variance. We may stop the process when we have collected enough directions (say, three if we want to visualize the data in 3D).

Mathematically, the process amounts to finding the eigenvalue decomposition of a positive semi-definite matrix: the *covariance matrix* of the data points. The directions of large variance correspond to the eigenvectors with the largest eigenvalues of that matrix. The projection to use to obtain, say, a two-dimensional view with the largest variance, is of the form $x \rightarrow Px$, where $P = [u_1, u_2]^T$ is a matrix that contains the eigenvectors corresponding to the first two eigenvalues.

Low rank approximations

In some cases, we are not specifically interested in visualizing the data, but simply to approximate the data matrix with a “simpler” one.

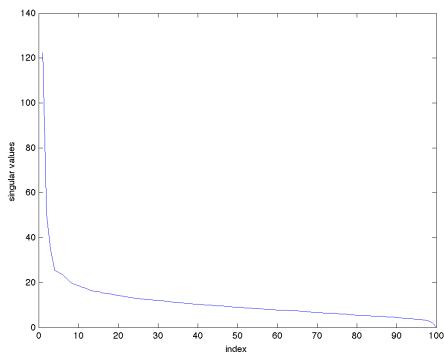
Assume we are given a (sample) covariance matrix of the data, Σ . Let us find the eigenvalue decomposition of Σ :

$$\Sigma = \sum_{i=1}^m \lambda_i u_i u_i^T = U \text{diag}(\lambda_1, \dots, \lambda_m) U^T,$$

where U is an $m \times m$ orthogonal matrix. Note that the trace of that matrix has an interpretation as the *total variance* in the data, which is the sum of all the variances of the votes of each Senator:

$$\text{Trace}(\Sigma) = \sum_{i=1}^m \lambda_i.$$

Now let us plot the values of λ_i 's in decreasing order.



This image shows the eigenvalues of the $m \times m$ covariance matrix of the Senate voting data, which contains the covariances between the votes of each pair of Senators.

Clearly, the eigenvalues decrease very fast. One is tempted to say that “most of the information” is contained in the first eigenvalue. To make this argument more rigorous, we can simply look at the ratio

$$\frac{\lambda_1}{\lambda_1 + \dots + \lambda_m}$$

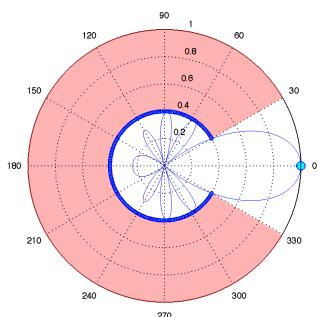
which is the ratio of the total variance in the data (as approximated by $\hat{\Sigma}$) to that of the whole matrix Σ .

In the Senate voting case, this ratio is of the order of 90%. It turns out that this is true of most voting patterns in democracies across history: the first eigenvalue “explains most of the variance”.

Senate Voting: Sparse Principal Component Analysis

In Progress

Antenna Array Design



In an antenna array the outputs of several emitting antenna elements are linearly combined to produce a composite array output. The array output has a directional pattern that depends on the relative weights or scale factors used in the combining process. The goal of *weight design* is to choose the weights to achieve a desired directional pattern.

The basic unit in a transmitting antenna is a isotropic harmonic oscillator, which emits an spherical, monochromatic wave at wavelength λ and frequency ω .

The oscillator generates an electromagnetic field with electrical component at a certain point P located at a distance d from the antenna in space is given by

$$\frac{1}{d} \mathbf{Re} \left[z \exp \left(j(\omega t - \frac{2\pi d}{\lambda}) \right) \right],$$

where $z \in \mathbf{C}$ is a design parameter that allows to scale and change the phase of the electrical field. We refer to this complex number as the *weight* of the antenna.

Array of oscillators

We now place n such oscillators at the locations $p_k \in \mathbf{R}^3, k = 1, \dots, n$. Each oscillator is associated with a complex weight $z_k, k = 1, \dots, n$. Then, the total electrical field received at a point $p \in \mathbf{R}^3$ is then the sum

$$E = \mathbf{Re} \left[\exp(j\omega t) \cdot \sum_{k=1}^n \frac{1}{d_k} z_k \exp(-\frac{2\pi j d_k}{\lambda}) \right],$$

where $d_k := \|p - p_k\|_2$ is the distance from p to $p_k, k = 1, \dots, n$.

Diagram of a linear array

To simplify, let us assume that the oscillators form a *linear array*: they are placed on an equidistant grid of points placed on the x -axis, that is, $p_k = k e_1, k = 1, \dots, n$, with $e_1 = (1, 0, 0)$ the first unit vector. Let us further assume that the point p under consideration is far away: $p = r u$, with $u \in \mathbf{R}^3$ a unit-norm vector that specifies the direction, and the distance to the origin, r , is large.

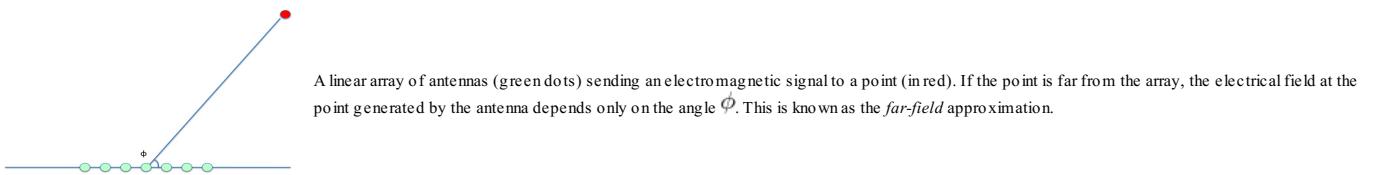
For a linear array, the electrical field E depends only on the angle between the array and the far away point under consideration, ϕ . In fact, a good approximation to E is of the form

$$E \approx \mathbf{Re} \left(\frac{1}{r} \exp(j\omega t - \frac{2\pi r}{\lambda}) \right) D_z(\phi)$$

where ϕ denotes the angle between the vector u and e_1 , and the function $D_z : [0, 2\pi] \rightarrow \mathbf{C}$, with complex values

$$D_z(\phi) := \sum_{k=1}^n z_k \exp\left(\frac{2\pi j k \cos(\phi)}{\lambda}\right)$$

is called the *diagram* of the antenna. We have used the subscript “ z ” to emphasize that the diagram depends on our choice of the complex weight vector, $z = (z_1, \dots, z_n)$.



[Diagram Shaping: Constraints and Trade-Offs](#) [Antenna Array Design > Physics](#) | [Diagram Shaping](#) | [Least-Squares Design](#) | [SOCP Design](#)

- Shaping the antenna diagram
- Normalization
- Sidelobe level constraint
- Thermal noise power constraint
- Examples of trade-offs

In this section we discuss basic constraints and trade-offs involved in the design of antenna arrays.

Shaping the antenna diagram

The squared modulus of the antenna’s diagram, $|D_z(\phi)|^2$, turns out to be proportional to the directional density of electromagnetic energy sent by the antenna. Hence, it is of interest to “shape” (by choice of the z ’s) the magnitude diagram $|D_z(\cdot)|$ in order to satisfy some directional requirements.

A typical requirement is that the antenna transmits well along a desired direction (on or near a given angle), and not for other angles. That way, the energy sent is concentrated around a given “target” direction, say $\phi_{\text{target}} = 0^\circ$, and small outside that band. Another type of requirement involves the thermal noise power generated by the antenna.

Normalization

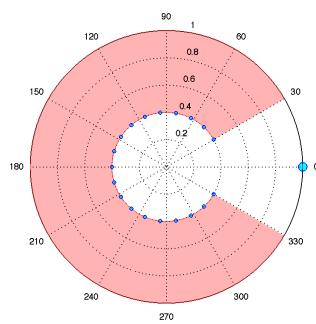
First, we normalize the energy sent along the target direction. When multiplying all weights by a common nonzero complex number, we do not vary the directional distribution of energy; therefore we lose nothing by normalizing the weights as follows: $\mathbf{Re}(D_z(0)) \geq 1$. These constraints are affine in the (real and imaginary parts of) the decision variable $z \in \mathbf{C}^n$.

Sidelobe level constraint

Now define a “pass-band” $[-\Phi, \Phi]$, where $\Phi > 0$ is given, inside which we wish the energy to be concentrated; the corresponding “stop-band” is the outside of that interval.

To enforce the concentration of energy requirement, we require $\forall \phi, |\phi| \geq \Phi : |D_z(\phi)| \leq \delta$, where δ is a desired attenuation level on the “stop-band” (this is sometimes referred to as the *sidelobe level*).

The sidelobe level constraint is actually an infinite number of constraints. We can simply discretize these constraints: $|D_z(\phi_i)| \leq \delta$, $i = 1, \dots, N$, where ϕ_1, \dots, ϕ_N are regularly spaced angles in the stop-band.



Antenna array design: sidelobe level constraints. The magnitude diagram must go through the blue point (on the right) at $\phi = 0^\circ$, and be contained in the white area otherwise. To simplify the design problem we can replace the sidelobe constraints by a finite number of constraints at given angles (in blue).

Thermal noise power constraint

It is often desirable to control the thermal noise power generated by the emitting antennas. It turns out that this power is proportional to the Euclidean norm of the (complex)

$$\text{Thermal noise power } \alpha \|z\|_2 = \sqrt{\sum_{i=1}^n |z_i|^2}.$$

vector z , that is:

Trade-off between sidelobe attenuation level and thermal noise power

A typical design problem would involve

- A normalization constraint which assigns a unit value to the magnitude diagram in a specific direction. The constraint is of the form $\mathbf{Re}(D_z(0)) \geq 1$.
- A constraint on the sidelobe attenuation level, which is of the form $|D_z(\phi_i)| \leq \delta$, $i = 1, \dots, N$.
- A constraint on the thermal noise power, $\|z\|_2 \leq \gamma$.

A typical trade-off curve would plot for example the best achievable thermal noise level γ for a given value of sidelobe attenuation level δ . Each point on the (δ, γ) curve is obtained by solving the optimization problem $\min_z \|z\|_2 : \mathbf{Re}(D_z(0)) \geq 1, |D_z(\phi_i)| \leq \delta, i = 1, \dots, N$.

Least-Squares Design [Antenna Array Design > Physics | Diagram Shaping](#) | Least-Squares Design | [SOCP Design](#)

- Example data
- Least-squares approach
- Data and example
- Trade-off curve

Example data

For our particular example, we have the following parameters:

- Number of antennas: $n = 16$.
- Wavelength: $\lambda = 8$.
- Pass-band size: $\Delta = \pi/6$. In what follows, we use the following matlab applet to define the data of our problem.

Matlab data

```
N = 10; % discretization parameter
n = 16; % number of antennas
lambda = 8; % wavelength
Phi = pi/6; % sidelobe parameter
```

Least-squares approach

We examine how least-squares can be used to strike a trade-off between the sidelobe level attenuation and thermal noise power. The initial problem reads $\min_z \|z\|_2 : \mathbf{Re}(D_z(0)) \geq 1, |D_z(\phi_i)| \leq \delta, i = 1, \dots, N$.

The basic idea is to penalize the constraints, as follows: we choose a “trade-off parameter” $\mu > 0$ and solve the problem

$$\min_z \|z\|_2^2 + \mu \sum_{i=1}^N |D_z(\phi_i)|^2 : \mathbf{Re}(D_z(0)) \geq 1. \quad D_z(\phi) := \sum_{k=1}^n z_k \exp\left(\frac{2\pi j k \cos(\phi)}{\lambda}\right)$$

[Remember](#) that the function D_z is linear in z : Hence, the above problem is a [linearly constrained least-squares problem](#).

CVX implementation

The above problem can be solved via linear algebra (SVD) methods. Here, we do not even bother to invoke SVD—CVX will do. A CVX implementation of this problem is given below. Note how we use a new vector variable to handle the objective function. Note also that CVX understands complex variables.

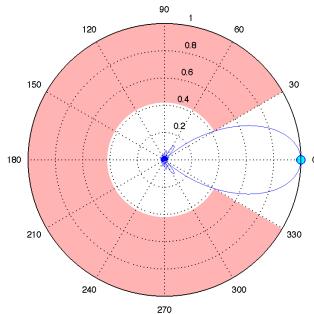
CVX implementation

```
Angles = linspace(Phi, pi, N); % angles in the stop band
a = 2*pi*sqrt(-1)/lambda; % intermediate parameter
```

```

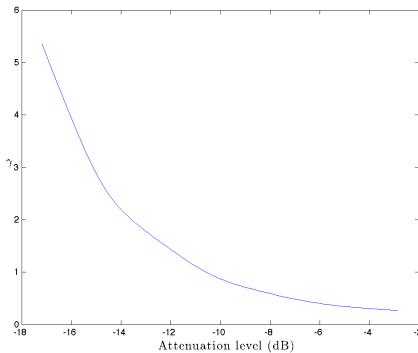
cvx_begin
    variable z(n,1) complex;
    variable r(N,1) complex;
    minimize( z'*z+mu*r'*r )
    subject to
        for i = 1:N,
            exp(a*cos(Angles(i))*(1:n))*z == r(i);
        end
        real( exp(a*(1:n))*z ) >= 1;
cvx_end

```



Antenna array design via least-squares: minimal combination of thermal noise and sidelobe level violations, computed at $N = 90$ points, with trade-off parameter $\mu = 0.5$. The corresponding sidelobe attenuation level is $\delta = -7.5386$ dB, and the thermal noise power is $\|z\|_2 = 0.5409$.

Note that the penalty approach adopted here will *not* enforce the desired constraints. We can only hope that, for μ large enough, *all* of the terms in the sum will go below the desired threshold δ . In practice, we expect that some terms might not. It seems to be hard to achieve the best trade-off this way. The following plot shows the trade-off curve that we can achieve.



Trade-off curve achieved via via least-squares.

Diagram Shaping via SOCP

minimum thermal noise power for given sidelobe level

We first seek to minimize the thermal noise power subject to a sidelobe level constraint. This problem can be cast as

$$\min_{z \in \mathbb{C}^n, \delta} \sum_{i=1}^n |z_i|^2 : \mathbf{Re}(D_z(0)) \geq 1, \quad |D_z(\phi_i)| \leq \delta, \quad i = 1, \dots, m.$$

The above N constraints are second-order cone constraints on the decision variables, since

they involve [magnitude constraints on a complex vector](#) that depends affinely on the decision variables. Hence the above problem is an SOCP.

A CVX implementation of this problem is given below. Note that CVX understands the magnitude of a complex variable and transforms the corresponding constraint into a second-order cone one internally.

CVX implementation

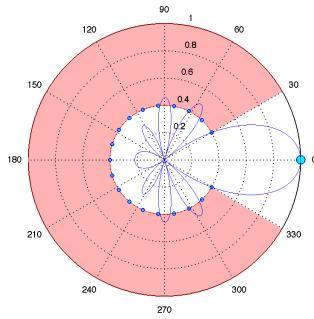
```

cvx_begin
    variable z(n,1) complex;
    minimize( norm(z,2) )
    subject to
        for i = 1:N,
            abs(exp(a*cos(Angles(i))*(1:n))*z) <= delta;
        end
        real( exp(a*(1:n))*z ) >= 1;
cvx_end

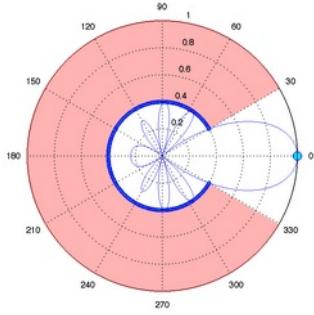
```

For a particular example, we take the same parameters as in the [least-squares design](#). We set the sidelobe level at $\delta = .4$. Measured in decibels (dB): $20 \log_{10}(\delta) = -7.9588$ dB.

Antenna array design: minimal thermal noise power given a sidelobe level constraint of -7.98 dB, enforced at $N = 10$ points. Due to our coarse discretization, the sidelobe constraints are only enforced at specific angles (in blue), but not everywhere satisfied.



The coarse discretization level of $N = 10$ may be an issue. This is readily solved by using a higher number, say $N = 90$.



Antenna array design: minimal thermal noise power given a sidelobe level constraint, enforced at $N = 90$ points. With a finer discretization, the sidelobe constraints are everywhere satisfied.

Minimum sidelobe level attenuation

Our goal is now to minimize the sidelobe attenuation level, δ , given the normalization requirement $\text{Re}(D_z(0)) \geq 1$.

$$\min_{z \in \mathbb{C}^n, \delta} \delta : \text{Re}(D_z(0)) \geq 1, |D_z(\phi_i)| \leq \delta, i = 1, \dots, m.$$

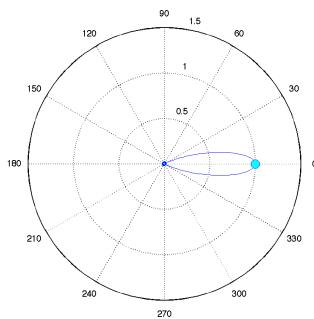
This can be cast as the SOCP

A CVX implementation is given below.

CVX implementation

```
cvx_begin
    variable z(n,1) complex;
    variable delta(1)
    minimize( delta )
    subject to
        for i = 1:N,
            abs(exp(a*cos(Angles(i))*(1:n))*z) <= delta;
        end
        real(exp(a*(1:n))*z) >= 1;
cvx_end
```

The result shows an optimal attenuation level in the stop band of $\delta^* =$, which, in decibels, is: $20 \log_{10}(\delta^*) = -46.4218$ dB.

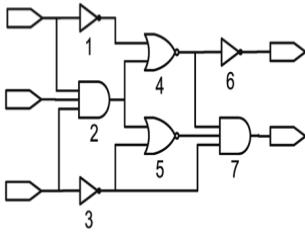


Antenna array design: the optimal magnitude diagram. The attenuation is excellent in the stop band, so that the stop-band magnitude is not distinguishable from zero in this plot.

Trade-off curve

Digital Circuit Design

We consider the problem of designing a kind of digital circuit known as a *combinational logic block*. In such a circuit, the basic building block is a *gate*: a circuit that implements some simple boolean function. A gate could be for example an *inverter*, which performs logical inversion,



or a more complicated function such as a “not-AND”, or NAND, which takes two boolean inputs A, B to produce the inversion of (A and B).

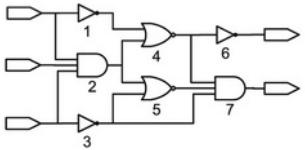
The basic idea is to design (size) the gates in such a way that the circuit operates fast, yet occupies a small area. There are other factors, such as power, involved in the design problem but we will not discuss this here.

The design variables are scale factors that determine the size of each gate, as well as its basic electrical parameters. Together with the (here, fixed) topology of the circuit, these parameters in turn influence the speed of the circuit.

This section uses [geometric programming models](#) to address the circuit design problem.

Digital Circuit Design: Problem Circuit Topology

The combinational circuit consists of n connected gates, with primary inputs and outputs. We assume that there are no loops in the corresponding graph. For each gate, we can define the *fan-in*, or set of predecessors of the gate in the circuit graph, and the *fan-out*, which is its set of successors.



Circuit topology: A digital circuit that consists of several “gates” (logical blocks with a few inputs and one output) connecting primary inputs, labeled 8, 9, 10, to primary outputs, labeled 11, 12. Each gate is represented by a symbol that specifies its type; for example, the gates labelled 1, 3 and 6 are inverters. For this circuit, the fan-in and fan-out of gate 2 is $\text{FI}(2) = \{8, 9, 10\}$, $\text{FO}(2) = \{4, 5\}$. By definition, the primary inputs have empty fan-in, and the primary outputs have empty fan-out.

Design Variables

The design variables in our models are the *scale factors*, which we denote by $x_i, i = 1, \dots, n$, which roughly determine the size of each gate. These scale factors satisfy $x_i \geq 1, i = 1, \dots, n$, where $x_i = 1$ corresponds to a minimum-sized gate, while a scale factor $x_i = 16$ corresponds to the case when all the devices in the gate have 16 times the widths of those in the minimum-sized gate.

The scale factors determine the size, and various electrical characteristics, such as resistance and conductance, of the gates. These relationship can be well approximated as follows:

- The *area* of gate i is proportional to the scale factor x_i : $A_i(x) = a_i x_i$, with $a_i > 0$. (So, you can simply think of the scale factors as the areas.)
- The *intrinsic capacitance* of gate i is of the form $C_i^{\text{intr}}(x) = C_i^{\text{intr}} x_i$, with $c_i > 0$ positive coefficients.
- The *load capacitance* of gate i is a linear function of the scale factors of the gates in the fan-out of gate i :

$$C_i(x) = \sum_{j \in \text{FO}(i)} C_j^{\text{in}} x_j,$$

where $C_j^{\text{in}} > 0$ are positive coefficients.

- Each gate has a resistance that is *inversely* proportional to the scale factor (the larger the gate, the more current can pass through it): $R_i(x) = r_i/x_i$, with $r_i > 0$ positive coefficients.
- The *gate delay* is a measure of how fast the gate implements the logical operation it is supposed to perform; this delay can be approximated as

$$D_i(x) \approx 0.7R_i(x)(C_i^{\text{intr}} + C_i(x)).$$

We observe that all the above parameters are actually *posynomials* in the (positive) design vector x .

Design Objective

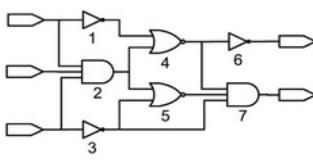
A possible design objective is to minimize the total delay D for the circuit. We can express the total delay as

$$D = \max_{1 \leq i \leq n} T_i,$$

where T_i represents the latest time at which the output of gate i can transition, assuming that the primary inputs signals transition at $t = 0$. (That is, T_i is the maximum delay over all paths that start at primary input and end at gate i .) We can express T_i via the recursion

$$T_i = \max_{j \in \text{FI}(i)} (T_j + D_i).$$

The operations involved in the computation of D involve only addition and point-wise maximum. Since each D_i is a posynomial in x , we can express the total delay as a [generalized posynomial](#) in x .



Recursion for the total delay: For the circuit on the left, the total delay D can be expressed as

$$\begin{aligned} T_i &= D_i, \quad i = 1, 2, 3, \\ T_4 &= \max(T_1, T_2) + D_4, \\ T_5 &= \max(T_2, T_3) + D_5, \\ T_6 &= T_4 + D_6, \\ T_7 &= \max(T_3, T_4, T_5) + D_7, \\ D &= \max(T_6, T_7). \end{aligned}$$

Digital Circuit Design: GP Model Generalized posynomial representation

We now consider the problem of choosing the scale factors to minimize the total delay, subject to an area constraint:

$$\min_x D(x) : A_i(x) \leq A_{\max}, \quad x_i \geq 1, \quad i = 1, \dots, n,$$

where A_{\max} is an upper bound on the area of each gate.

Since T_i is a maximum of function of D_i , itself a posynomial in \mathcal{X} , we can express the total delay as a posynomial in \mathcal{X} . Hence the above is a GP.

Explicit posynomial representation

Let us form the GP in a more explicit way, using the intermediate variables T_i , which we encountered in our definition of the delay. The basic idea is to replace the equality defining these intermediate variables, e.g. $T_4 = \max(T_1, T_2) + D_4$ by inequalities:

$$T_4 \geq T_1 + D_4, \quad T_4 \geq T_2 + D_4.$$

It turns out we “lose nothing” in this process; that is, at optimum, equality holds.

More generally, we replacing the constraint

$$T_i = \max_{j \in FI(i)} (T_j + D_i).$$

by a relaxed version:

$$T_i \geq \max_{j \in FI(i)} (T_j + D_i).$$

The above can be written:

$$T_i \geq (T_j + D_i), \quad j \in FI(i).$$

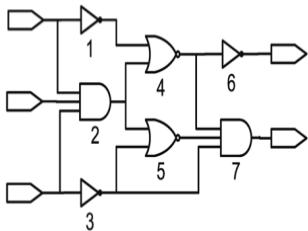
We obtain an explicit representation of the previous GP:

$$\begin{aligned} \min_{x, T > 0} D : \quad & A_i(x) \leq A_{\max}, \quad x_i \geq 1, \quad i = 1, \dots, n, \\ & D \geq T_i, \quad i = 1, \dots, n, \\ & T_i \geq (T_j + D_i(x)), \quad i = 1, \dots, n, \quad j \in FI(i). \end{aligned}$$

The above is also a GP. At the expense of adding n new variables and also adding constraints, we have obtained a sparser problem.

Digital Circuit Design: Example

We return to the example pictured before.



Design example: For the circuit, the gates have the following parameters:

Gate	C^{in}	C^{intr}	r	a
1, 3, 6	3	3	0.48	3
2, 7	4	6	0.48	8
4, 5	5	6	0.48	10

The design problem is

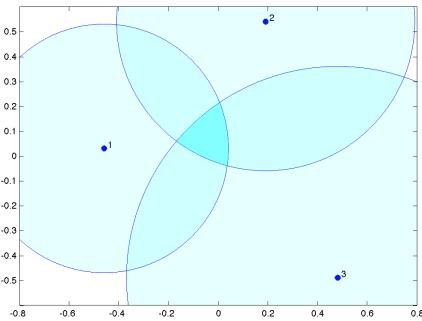
$$\begin{aligned} \min_{x, T > 0} D : \quad & A_i(x) \leq A_{\max}, \quad x_i \geq 1, \quad i = 1, \dots, n, \\ & D \geq T_i, \quad i = 1, \dots, n, \\ & T_i \geq (T_j + D_i(x)), \quad i = 1, \dots, n, \quad j \in FI(i), \end{aligned}$$

where $A_i(x) = a_i x_i$, and the delay functions D_i are defined [here](#).

Digital Circuit Design: Notes and References

GPs have a long history in circuit design. The presentation given here, as well as the example, is taken from the [paper](#) by S. Boyd, S.-J. Kim, D. Patil, and M. Horowitz; the paper contains many references.

Localization via Range Measurement

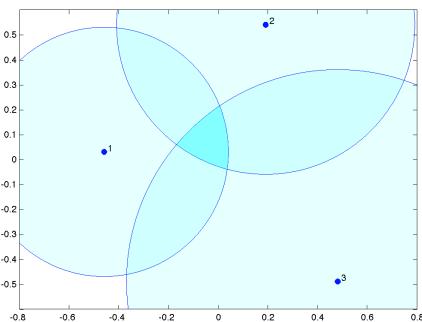


The objective of the localization problem is to infer, from a set of range measurements, the position of an object in three-dimensional space, as accurately as possible. The problem would arise in the context of trying to estimate the location of a cell phone user based on the measurements of strength of a signal emitted from the cell phone to a number of base stations (access points).

Problem Statement

The localization problem

We are given anchor positions $x_i \in \mathbf{R}^3$, and associated distances from these anchor points to an unknown object, $R_i, i = 1, \dots, m$. The problem is to estimate a position of the object, and associated measure of uncertainty around the estimated point. Geometrically, the measurements imply that the object is located at the intersection of the m spheres of centers x_i and radii $R_i, i = 1, \dots, m$. The main problem is to provide one point in the intersection located at some kind of “center”, and also a measure of the size of the intersection.



Localization problem with three range measurements in two dimensions, with data

$$X = (x_1, x_2, x_3) = \begin{pmatrix} -0.46 & 0.19 & 0.48 \\ 0.03 & 0.54 & -0.49 \end{pmatrix},$$

$$R^T = (R_1, R_2, R_3) = \begin{pmatrix} 0.5 & 0.6 & 0.85 \end{pmatrix}.$$

The goal is to find the “center” of the intersection area, where an object is located. Also we would like to quantify the size of the area.

Issues to be addressed

In general, the measurements may not be *consistent*, in the sense that the intersection of all the spheres may be empty. We will first address the problem of checking consistency, and adjust our measurements if necessary. Assuming then the measurements to be consistent, so that the spheres do intersect, in general the intersection may not be reduced to a unique point, which means that the measurement only allow to estimate the position with some level of uncertainty. We aim to provide both an estimate of the position, and a measure of the uncertainty.

There are two approaches to this problem. One consists in finding an *inner approximation* to the intersection, which is the *largest* sphere (or box) *contained* in it. This is in a sense an “optimistic” approach. Another approach is “pessimistic” and consists in finding the *smallest* sphere (or box) that *contains* the intersection. In both cases, the optimal radius of the sphere (or size of box) is a measure of uncertainty, and its center is the estimated position.

Consistency

The measurements may not be *consistent*, in the sense that the intersection of all the spheres may be empty. Here, we first address the problem of checking consistency, and adjust our measurements if necessary.

Checking consistency

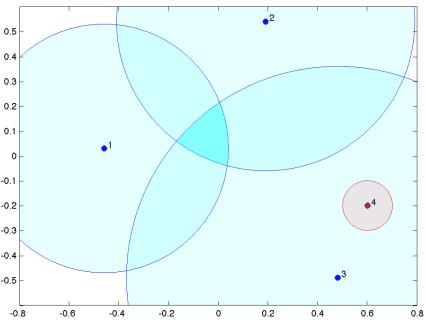
We begin by checking if our consistency condition holds. This can be formulated as an optimization problem, involving no (or, zero) objective:

$$\min_x 0 : \|x - x_i\|_2 \leq R_i, \quad i = 1, \dots, m.$$

The above is an SOCP. On exit, a solver such as CVX will provide either a feasible point x_f , or determine (unambiguously) that there is no feasible point.

CVXZ syntax: checking consistency

```
% input: 2xm data matrix X, m-vector of radii R
% output: feasible point xfeas, empty if intersection is empty
cvx_begin
    variable xfeas(2,1)
    minimize( 0 )
    subject to
        for i = 1:m,
        R(i) >= norm(xfeas-X(:,i),2);
    end
    cvx_end
    if ~isfinite(cvx_optval), xfeas = []; end
```



Localization problem with one range measurement added, with data

$$x_4 = \begin{pmatrix} 0.6 \\ -0.2 \end{pmatrix}, \quad R_4 = 0.1.$$

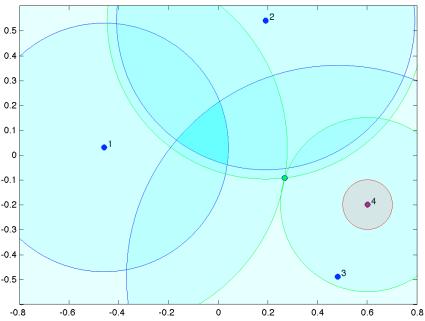
This last measurement (in red) corresponds to a faulty sensor, and makes the measurements inconsistent. Solving the above problem with CVX return an infeasibility message, and the optimal value is $+\infty$.

Adjusting inconsistent measurements

If there is no feasible point, we may want to modify our range measurements so that they become consistent. A variety of approaches are possible. Clearly, increasing the distances R_i will, at some point, make the problem feasible. Let us find the smallest increases that result in a non-empty intersection. We are led to the problem

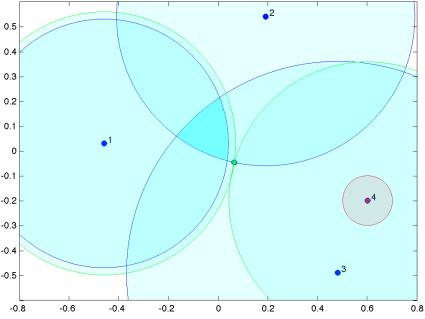
$$\min_{x, \delta} \|\delta\| : \delta \geq 0, \quad \|x - x_i\|_2 \leq R_i + \delta_i, \quad i = 1, \dots, m.$$

The above problem can be expressed as an SOCP, when the norm chosen is one of the three [popular norms](#) (l_1 , l_∞ , or Euclidean). The qualitative behavior of the solution depends on the choice of the norm.



If the norm chosen is the Euclidean one, we are minimizing the sum of the squares of the adjustments (increases) that are necessary to make our measurements consistent. This results in non-zero adjustments for all the measurements, and the new (unique) intersection point (in green) is far away from the initial intersection.

Choosing an l_1 norm will tend to make the smallest number of adjustment necessary. This would make sense if we believe that a few of our measurements are outliers, and due to faulty sensors.



Identifying a faulty sensor that resulted in inconsistent measurements: here we have solved the minimum l_1 -norm of adjustments necessary to make the measurements consistent. The approach identifies the offending measurement, since the optimal adjustment vector δ is almost zero except for its last component, which corresponds to the fourth sensor. We observe, however, that the identification is not perfect, as the measurement of sensor 1 is also adjusted, albeit only slightly. This is due to the fact that the l_1 -norm approach is a only a heuristic to solve cardinality minimization problems, as seen [here](#).

In the above problem, the sign constraint on δ implies that we assume that a fault in a sensor only results in under-estimating the radius (hence to a possible infeasible problem). We can remove the sign constraint if we want to address a problem of faulty sensors, where some of them over-estimate the radius measurement.

The above approach will in general make the smallest adjustments necessary. This means that in general at the outset of solving the problem, the intersection reduces to a single point (the optimal variable x delivered by the above procedure). We can choose that point to be our estimate. However, if indeed we suspect some faulty sensors, it may not make sense to proceed that way. Instead, we may use the above problem (with l_1 -norm objective) as a method to *detect* faulty sensors (those for which $\delta_i \neq 0$ at optimum). Then, we remove those faulty sensors from the problem, and proceed with a set of consistent measurements.

Assumption: consistent measurements

From now on, we assume that our measurements are consistent. Without loss of generality, we may translate the whole problem so that the feasible point that proves consistency, x_f , is zero. This means that the vector z with components $z_i = R_i^2 - x_i^T x_i$, $i = 1, \dots, m$, is non-negative componentwise ($z \geq 0$).

Inner Approximations

Inner spherical approximation

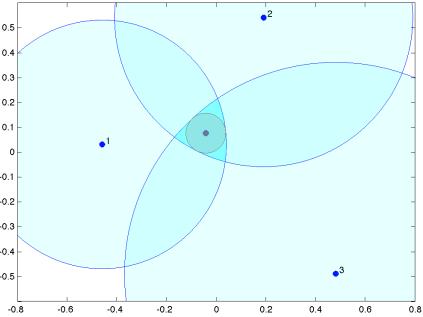
We first focus on the problem of finding the largest radius of a sphere contained in the intersection. It is easy to check that a sphere of center x_0 and radius R_0 is contained in a sphere of center x_i and radius R_i if and only if the differences in the radii exceed the distance between the centers:

$$R_i \geq R_0 + \|x_i - x_0\|_2.$$

Our inner approximation problem then becomes the SOCP

$$\max_{x_0, R_0} R_0 : R_i \geq R_0 + \|x_i - x_0\|_2, \quad i = 1, \dots, m.$$

We note that the measurements are inconsistent if and only if at optimum, $R_0^* < 0$. This is the same as saying that there is no point x_0 which satisfies the constraints $\|x_i - x_0\|_2 \leq R_i, i = 1, \dots, m$.



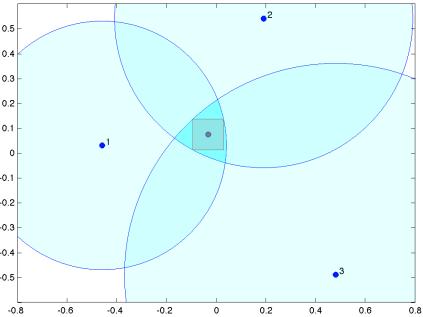
Inner spherical approximation to the intersection. This provides an estimated point (the center of the inner sphere), with an *optimistic* estimate of the uncertainty around it.

Inner box approximation

We can also consider the problem of finding the largest box inside the intersection. We simply ensure that the vertices of a box with size ρ are inside the intersection, and then maximize ρ . In 2D or 3D, this is easy, as there is a moderate number of vertices. The problem is written

$$\max_{\rho, x_0} \rho : \|x_0 + \rho v_k - x_i\|_2 \leq R_i, \quad i = 1, \dots, m, \quad k = 1, \dots, K.$$

In the above, K is the number of vertices of the box ($K = 4$ in 2D, $K = 8$ in 3D), and $v_k, k = 1, \dots, K$ are the vertices of the unit box, that is, the vectors with elements ± 1 .



Inner box approximation to the intersection. This provides an estimated point (the center of the inner box), with an *optimistic* estimate of the uncertainty around it. Here, the uncertainty is given as two intervals of confidence on each of the coordinates of the estimated point.

Outer Spherical Approximations

Let us consider a candidate sphere S_0 , of center x_0 and radius R . We formulate the condition that the sphere S_0 contains the intersection of all the spheres of center S_i , as follows:

$$(*) : \|x - x_0\|_2^2 \leq R_0^2 \text{ for every } x \text{ such that } \|x - x_i\|_2^2 \leq R_i^2, \quad i = 1, \dots, m.$$

This conditions appears to be hard to check.

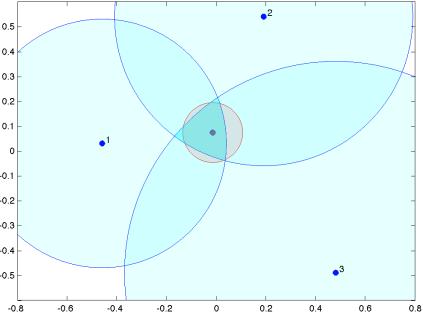
Outer spherical approximation via gridding

A first approach, which works well only in moderate dimensions (2D or 3D), simply entails gridding the boundary of the intersection. In 2D, we can parametrize the boundary explicitly, as a curve, using an angular parameter. For each angular direction $\theta \in [0, 2\pi]$, we can easily find the point that is located on the boundary of the intersection, in the direction given by θ : we simply maximize t such that the point $(t \cos \theta, t \sin \theta)$ is inside everyone of the spheres. (There is an explicit formula for the maximal value.)

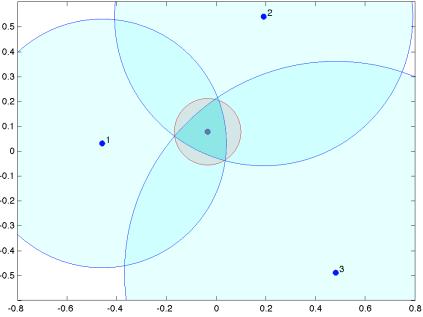
One we have computed N points on the boundary, $x^{(k)}, k = 1, \dots, N$, we simply solve the SOCP

$$\min_{x_0, R_0} R_0 : R_0 \geq \|x_0 - x^{(k)}\|_2, \quad k = 1, \dots, N.$$

The gridding approach suffers from the fact that we need to grid finely to be able to guarantee that the spherical approximation we are computed indeed contains all the points on the intersection. On the other hand, too fine a grid results in a large number of constraints, which in turn adversely impacts the time needed to solve the above problem.



Outer spherical approximation to the intersection via gridding. This provides an estimated point (the center of the outer sphere), with an estimate of the uncertainty around it. We have used only 13 points on the boundary to enforce the constraint that the outer sphere contains the intersection. As a result, our approximation is not really an outer approximation, as it does not contain all the intersection points.



Outer spherical approximation to the intersection via gridding. Here we have used 63 points on the boundary. The approximation can now be termed an outer approximation as it (almost!) does not leave out any point in the intersection.

Outer spherical approximation via Lagrange duality

We now develop an alternate approach which relies on [weak duality](#) concepts.

A sufficient condition

A sufficient condition for the above condition (*) to hold is that there exist a non-negative vector $y \in \mathbf{R}_+$ such that

$$\text{for every } x : \|x - x_0\|_2^2 \leq R_0^2 + \sum_{i=1}^m y_i (\|x - x_i\|_2^2 - R_i^2).$$

Indeed, it is easily checked that if a point x belongs to all the spheres, then indeed the sum above is less or equal than 0 when $y \geq 0$.

Minimizing the radius R_0 subject to the condition that the above holds for some $y \geq 0$ will result in a conservative (larger) estimate of the amount of uncertainty.

SOCOP formulation

Alternatively we can express the above condition as:

$$R_0^2 \geq F(x_0, y) := \sum_{i=1}^m y_i R_i^2 + \max_x \left(\|x - x_0\|_2^2 - \sum_{i=1}^m y_i \|x - x_i\|_2^2 \right).$$

The problem of minimizing the radius R_0 subject to the above condition can thus be written as

$$(R_0^2)^* := \min_{x_0, y \geq 0} F(x_0, y).$$

It turns out that we can express the function $F(x_0, y)$ in closed-form, as proven [here](#):

$$F(x_0, y) = \begin{cases} x_0^T x_0 + y^T z + \frac{\|Xy - x_0\|_2^2}{S-1} & \text{if } S := \sum_{i=1}^m y_i > 1, \\ x_0^T x_0 + y^T z & \text{if } \sum_{i=1}^m y_i = 1, \quad x_0 = Xy, \\ +\infty & \text{otherwise,} \end{cases}$$

where for notational convenience we use the matrix $X := (x_1, \dots, x_m)$ and the vector z with components $z_i := R_i^2 - x_i^T x_i, i = 1, \dots, m$. (Remember that our [measurement consistency condition](#) holds, so that $z \geq 0$.)

We are led to consider the sub-problem of minimizing $F(x_0, y)$ over x_0 , with y fixed. If $S := \sum_{i=1}^m y_i = 1$, then we must have $x_0 = Xy$. If $S > 1$, x_0 must solve

$$\min_{x_0} x_0^T x_0 + \frac{\|Xy - x_0\|_2^2}{S-1}.$$

Again, this a convex quadratic problem, with a (unique) solution obtained by taking derivatives. A minimizer is $x_0(y)$, where

$$x_0(y) := \frac{1}{\sum_{i=1}^m y_i} Xy,$$

that is, x_0 is the weighted average, with weights given by $y \geq 0$, of the points x_i . Note that the expression remains valid when $\sum_i y_i = 1$, since then we must have $x_0 = Xy$.

Plugging the above value of x_0 in the problem leads to a problem in variable y only:

$$(R_0^2)^* = \min_y y^T z + \frac{1}{\sum_{i=1}^m y_i} \|Xy\|_2^2 : y \geq 0, \sum_{i=1}^m y_i \geq 1.$$

The above problem can be solved as an SOCP with [rotated second-order cone constraints](#):

$$\min_{y, \alpha} y^T z + \alpha : \alpha(\sum_{i=1}^m y_i) \geq \|Xy\|_2^2, y \geq 0, \sum_{i=1}^m y_i \geq 1.$$

A simpler formulation (consistent measurements)

We can obtain a simpler formulation that involves QP only. We now use our measurement consistency assumption, which translates as $z \geq 0$. First we replace the variable y with two new variables S, p , and a new constraint. The new variables S, p are defined as

$$S := \sum_{i=1}^m y_i, \quad p_i = \frac{y_i}{S}, \quad i = 1, \dots, m,$$

and the new constraint is $\sum_{i=1}^m p_i = 1$. We obtain the new problem

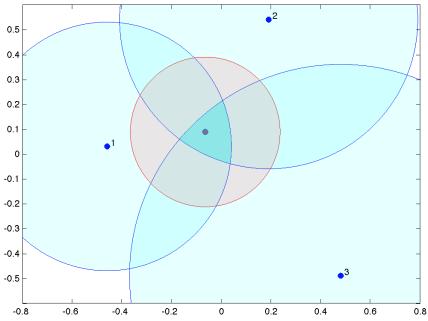
$$(R_0^2)^* = \min_{S, p} S(p^T z + \|Xp\|_2^2) : p \geq 0, \sum_{i=1}^m p_i = 1, S \geq 1.$$

Since $z \geq 0$, the term inside the parentheses is non-negative, and thus $S = 1$ is optimal. Hence the problem reduces to a QP:

$$(R_0^2)^* = \min_p p^T z + \|Xp\|_2^2 : p \geq 0, \sum_{i=1}^m p_i = 1.$$

This provides the optimal radius. The optimal point is

$$x_0^* = \frac{1}{S} X y = X p = \sum_{i=1}^m x_i p_i.$$



Outer spherical approximation via Lagrange duality. This provides an estimated point (the center of the outer sphere), with an estimate of the uncertainty around it. The approximation is very conservative.

An improved condition via affine duality

To improve our condition, we start from the equivalent condition for intersection containment:

$$t - 2x_0^T x + x_0^T x_0 \leq R_0^2 \text{ for every } x, t \text{ such that } t = x^T x, \quad t - 2x_i^T x + x_i^T x_i \leq R_i^2, \quad i = 1, \dots, m.$$

We now consider a relaxed of the form

$$\text{for every } x, t : t - 2x_0^T x + x_0^T x_0 \leq R_0^2 + \tau(t - x^T x) + \sum_{i=1}^m y_i(x, t) (t - 2x_i^T x + x_i^T x_i - R_i^2),$$

where $\tau \geq 0$, and the y_i 's are now non-negative *functions* of (x, t) . (Before, we chose them to have constant values.) The above is still hard to check in general, but becomes easy if we make the assumption that the functions $(x, t) \rightarrow y_i(x, t)$ are affine.

We proceed by taking the vector $y(x, t) := (y_i(x, t))_{i=1}^m$ to be of the form

$$y(x, t) = y + Y^T x + t\nu,$$

where $y \in \mathbf{R}^m$, $Y \in \mathbf{R}^{2 \times m}$ and $\nu \in \mathbf{R}^m$ will be variables. (The case before will be recovered upon setting $Y = 0, \nu = 0$.)

The condition above becomes a single quadratic condition on (x, t) :

$$\text{for every } x, t : t - 2x_0^T x + x_0^T x_0 \leq R_0^2 + \tau(t - x^T x) + \sum_{i=1}^m (y + Y^T x + t\nu)_i (t - 2x_i^T x + \|x_i\|_2^2 - R_i^2).$$

The above can be equivalently expressed as a positive semi-definiteness condition on a symmetric matrix that is affine in y, Y, ν :

$$\begin{pmatrix} \nu^T \mathbf{1} & -\nu^T X^T + \frac{1}{2} \mathbf{1}^T Y^T & \frac{1}{2} (\mathbf{1}^T y - \nu^T z - \tau - 1) \\ -X\nu - \frac{1}{2} Y \mathbf{1} & \tau I - Y X^T - X Y^T & x_0 - X y - \frac{1}{2} Y z \\ \frac{1}{2} (\mathbf{1}^T y - \nu^T z - \tau - 1) & (x_0 - X y - \frac{1}{2} Y z)^T & R_0^2 - y^T z - \eta \end{pmatrix} \succeq 0, \quad \eta \geq x_0^T x_0.$$

It remains to express the condition that the affine functions $y_i(x, t)$ should be non-negative on the intersection. Precisely, we seek to enforce that for every $i \in \{1, \dots, m\}$, the condition

$$y_i + Y_i^T x + \nu_i x^T x \geq 0 \text{ for every } x \text{ such that } x^T x - 2x_j^T x + x_j^T x_j \leq R_j^2, \quad j = 1, \dots, m$$

holds (here, Y_i stands for the i -th column of matrix Y). Now, we apply direct Lagrange duality. A sufficient condition for the above to hold is that there exist non-negative numbers N_{ij} , $j = 1, \dots, m$, such that

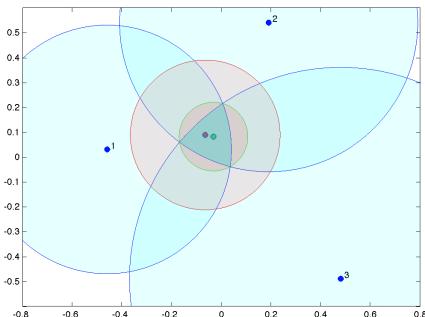
$$\text{for every } x : y_i + Y_i^T x + \nu_i x^T x \geq \sum_{j=1}^m N_{ij} (R_j^2 - x^T x + 2x_j^T x - x_j^T x_j).$$

Again, the above is equivalent to a single linear matrix inequality in the variables y, Y, ν and N :

$$\begin{pmatrix} \nu_i - \sum_{j=1}^m N_{ij} & \frac{1}{2} Y_i - \sum_{j=1}^m N_{ij} x_j \\ (\frac{1}{2} Y_i - \sum_{j=1}^m N_{ij} x_j)^T & y_i - \sum_{j=1}^m N_{ij} z_j \end{pmatrix} \succeq 0.$$

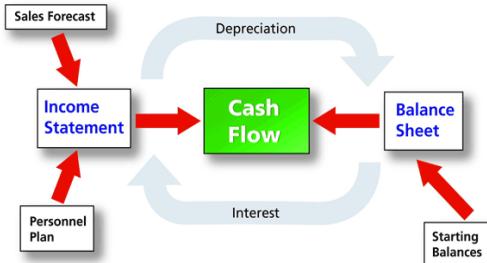
Putting this together, we obtain the SDP in variables $x_0, R_0^2, y, Y, \nu, \eta, N$:

$$\min R_0^2 : \begin{pmatrix} \nu^T \mathbf{1} & -\nu^T X^T + \frac{1}{2} \mathbf{1}^T Y^T & \frac{1}{2} (\mathbf{1}^T y - \nu^T z - \tau - 1) \\ -X\nu - \frac{1}{2} Y\mathbf{1} & \tau I - YX^T - XY^T & x_0 - Xy - \frac{1}{2} Yz \\ \frac{1}{2} (\mathbf{1}^T y - \nu^T z - \tau - 1) & (x_0 - Xy - \frac{1}{2} Yz)^T & R_0^2 - y^T z - \eta \\ \nu_i - \sum_{j=1}^m N_{ij} & \frac{1}{2} Y_i - \sum_{j=1}^m N_{ij} x_j & \\ (\frac{1}{2} Y_i - \sum_{j=1}^m N_{ij} x_j)^T & y_i - \sum_{j=1}^m N_{ij} z_j & \end{pmatrix} \succeq 0, \quad i = 1, \dots, m,$$



Improved outer spherical approximation via affine Lagrange duality. The refined approximation is now exact, in contrast with the previous approximation. The improvement is due to the fact that our relaxation involves Lagrange dual variables that are not constant anymore, but are affine in the primal variables.

Cash-Flow Management



In this problem, a company seeks to choose between three financial instruments to cover its liabilities over a six-months period into the future. We first establish an LP formulation for the case when the liability vector is fully known. Then we devise a robust counterpart to the problem to address the fact that the liability vector is not fully known in advance. Lastly, we examine an affine recourse approach, in order to take a better advantage of the dynamic nature of the problem, and obtain less conservative solutions.

This problem elaborates on an example in Cornuejols & Tutuncu, [Optimization Models in Finance](#), Cambridge University Press, 2007.

Cash-flow management problem

Early in January, a company faces the following net cash flow requirements, expressed in thousands of dollars:

January	February	March	April	May	June
150	100	-200	200	-50	-300

The table above says that the company expects to receive a net amount of \$150k at the end of January, and to have to pay \$200k at the end of March. To face these liabilities, the company has at its disposal three sources of funds:

- A line of credit of maximum amount \$100k, with interest rate 1% per month;
- In any of the first 3 months it can issue 90-day commercial paper (that is, take out a loan) bearing a total interest of 2% for the 3-month period;
- Excess funds (cash) can be invested at 0.3% per month.

The goal of the company is to maximize the *amount of cash at the end* of the last decision period (end of June).

An LP formulation of the cash-flow management problem

Variables. The variables in the problem are:

- The balance on the credit line x_i for months $i = 1, 2, 3, 4, 5$.
- The amount y_i of commercial paper issued ($i = 1, 2, 3$).
- The amount of excess funds z_i for each month $i = 1, 2, 3, 4, 5$.

Objective function. The goal of the company is to maximize z_6 . Abstractly, the problem reads

$$\text{maximize } z_6 \text{ subject to } \begin{cases} \text{Bounds on variables,} \\ \text{Cash-flow balance equations.} \end{cases}$$

In the above, the cash-flow balance equations express that we must meet the required liabilities.

Constraints. The bounds on the variables are

- Non-negativity: $x_i \geq 0, i = 1, 2, 3, 4, 5; z_i \geq 0, i = 1, 2, 3, 4, 5, 6; y_i \geq 0, i = 1, 2, 3$.
- Upper bounds on the credit line amount: $x_i \leq 100, i = 1, 2, 3, 4, 5; y_i \geq 0, i = 1, 2, 3$.

Let us now examine how these variables are coupled via the cash-flow balance equations. These express the simple reality that what is left in the bank is the amount the Company borrows minus what it pays.

January:

$$\underbrace{x_1}_{\text{from line of credit}} + \underbrace{y_1}_{\text{commercial paper}} - \underbrace{z_1}_{\text{excess funds}} = \underbrace{150}_{\text{cash flow requirement}}$$

February:

$$x_2 + y_2 - \underbrace{1.01x_1}_{\substack{\text{principal} \\ + \text{interest} \\ \text{on credit line}}} + \underbrace{1.003z_1}_{\substack{\text{interest} \\ \text{on excess} \\ \text{funds}}} - z_2 = 100.$$

March:

$$x_3 + y_3 - \underbrace{1.01x_2}_{\substack{\text{principal} \\ + \text{interest} \\ \text{on credit line}}} + \underbrace{1.003z_2}_{\substack{\text{interest} \\ \text{on excess} \\ \text{funds}}} - z_3 = -200.$$

April:

$$x_4 - \underbrace{1.02y_1}_{\substack{\text{payment} \\ \text{of commercial paper} \\ \text{issued in January}}} - 1.01x_3 + 1.003z_3 - z_4 = 200.$$

LP model. Putting all this together, we obtain an LP formulation:

$$\begin{aligned} & \max_{x, y, z} z_6 \\ \text{s.t.} \quad & x_1 + y_1 - z_1 = 150, \\ & x_2 + y_2 - 1.01x_1 + 1.003z_1 - z_2 = 100, \\ & x_3 + y_3 - 1.01x_2 + 1.003z_2 - z_3 = -200, \\ & x_4 - 1.02y_1 - 1.01x_3 + 1.003z_3 - z_4 = 200, \\ & x_5 - 1.02y_2 - 1.01x_4 + 1.003z_4 - z_5 = -50, \\ & -1.02y_3 - 1.01x_5 + 1.003z_5 - z_6 = -300, \\ & 100 \geq x_i \geq 0, \quad i = 1, \dots, 5, \\ & y_i \geq 0, \quad i = 1, 2, 3, \\ & z_i \geq 0, \quad i = 1, \dots, 6. \end{aligned}$$

We can write this in standard LP format:

$$\min_v c^T x : Av = b, 0 \leq v \leq \bar{v},$$

for appropriate matrix A and vectors b, c, \bar{v} .

CVX implementation and solution. The solution is obtained easily via CVX.

CVX implementation
cvx_begin

```

variable x(5,1);
variable y(3,1);
variable z(6,1);
maximize( z(6) );
A*[x; y; z] == b;
x <= 100; x >= 0;
y >= 0; z >= 0;
cvx_end

```

In the snippet above, A contains the coefficients of the balance equations. The schedule of the solution is given below.

month	x	y	z
1	0.0000	0	0.0000
2	22.5813	0	0.0000
3	0.0000	0	351.9442
4	0.0000	150.0000	0.0000
5	28.9671	77.4187	0.0000
6	0	174.7512	92.4969

Liability uncertainty and the robust counterpart

Uncertainty model

We now assume that the liability vector b is subject to uncertainty. Specifically, at each time t , the liability is

$$b(t) = \widehat{b}(t)(1 + 0.06u(t) + 0.02u(t-1))$$

for some values $u(t), u(t-1)$ that are only known to be in $[-1, 1]$. Here \widehat{b} contains the nominal liability values (as set [here](#)). The above model describes the fact that changes in the liability in one period has some effect in the next. In practice, such a model could be obtained from past observations.

The liability vector is now a function of uncertainty. In matrix form:

$$b(u) = \widehat{b} + Bu, \|u\|_\infty \leq 1,$$

where $u \in \mathbf{R}^6$, and

$$B = \text{diag}(\widehat{b}) \begin{pmatrix} 0.06 & 0 & 0 & 0 & 0 & 0 \\ 0.02 & 0.06 & 0 & 0 & 0 & 0 \\ 0 & 0.02 & 0.06 & 0 & 0 & 0 \\ 0 & 0 & 0.02 & 0.06 & 0 & 0 \\ 0 & 0 & 0 & 0.02 & 0.06 & 0 \\ 0 & 0 & 0 & 0 & 0.02 & 0.06 \end{pmatrix}.$$

Robust counterpart

Formal expression. It is impossible to find a solution such that the cash-flow constraints are satisfied, irrespective of the value of the liability vector $b(u)$. This is because these constraints are *equality* constraints. However we can first transform the original constraints of the problem into *inequality* constraints, and then enforce robustness. This is financially sound: we can always try to satisfy more liabilities than required, and optimize over that larger set of decisions.

We start with the nominal LP written with equality constraints (see [here](#)):

$$\max_{x,y,z} c^T v \text{ s.t. } Av \geq \widehat{b}, 0 \leq v \leq \bar{v}.$$

The robust counterpart expresses formally as

$$\max_{x,y,z} c^T v \text{ s.t. } \forall u, \|u\|_\infty \leq 1 : Av \geq \widehat{b} + Bu, 0 \leq v \leq \bar{v}.$$

A single constraint. Let us examine a particular constraint (at time $t = 1, \dots, 6$):

$$a(t)^T v \geq \widehat{b}(t) + B(t)^T u,$$

where $a(t)$ and $B(t)$ are the t -th row of A, B .

The robust counterpart obtains when insisting that the above remains true for every value of u permitted by the model (that is, $\|u\|_\infty \leq 1$). The constraint becomes

$$\forall u, \|u\|_\infty \leq 1 : a(t)^T v \geq \widehat{b}(t) + B(t)^T u.$$

As seen [here](#), the condition on $z \in \mathbf{R}^p, \beta \in \mathbf{R}$:

$$\forall u, \|u\|_\infty \leq 1 : z^T u \leq \beta,$$

is the same as:

$$\beta \geq \max_{u: \|u_i\|_1 \leq 1, i=1,\dots,p} z^T u = \|z\|_1,$$

Applying this result to our single constraints we obtain the robust counterpart to the cash-flow matching inequality at time t :

$$a(t)^T v \geq \hat{b}(t) + \|B(t)\|_1.$$

We can list of the above constraints for $t = 1, \dots, 6$ into a matrix form:

$$Av \geq \hat{b} + |B|1.$$

where $|B|$ is the matrix of absolute values in B , and 1 is the vector of ones. (Hence, $|B|1$ is a vector with the l_1 -norm of the rows of B in each entry.)

Robust counterpart.

After our derivations, we simply replace b by $\$b\$$ by $\hat{b} + |B|1$ in the original LP:

$$\max_{x,y,z} c^T v \text{ s.t. } Av \geq \hat{b} + |B|1, \quad 0 \leq v \leq \bar{v}.$$

The policy implied by the robust counterpart is quite simple in this problem: simply replace all the nominal values of the liability vector by their extreme ones, and solve the original problem with these new (worse) values.

Results and discussion.

(Terminal wealth in red)

Nominal solution:

month	x	y	z
1	0.0000	0	0.0000
2	22.0916	0	0.0000
3	0.0000	0	351.9442
4	0.0000	150.0000	0.0000
5	29.4665	77.9084	0.0000
6	0	174.2567	92.4969

The results delivered by the robust solution seem substantially worse (terminal wealth goes from 92.5k\\$ to 17.3k\\$). However the terminal wealth delivered by the robust solution has to be compared to the one delivered by the nominal policy when *uncertainty is present*.

With uncertainty, the nominal policy is actually a *complete failure*: the cash-flow requirements are simply not met! Can we do better?

Robust solution:

month	x	y	z
1	0.0000	0	0.0001
2	33.5287	0	0.0000
3	0.0000	0	377.0499
4	0.0000	159.0000	0.0000
5	33.9808	75.4714	0.0000
6	0	224.9128	17.2684

Affine recourse

Basic idea

We can do better than the simple robust counterpart by exploiting the fact that this problem involves *multiple decision periods*, and assuming that the uncertainty is revealed to us as time goes on. This approach develops an Affinely Adjustable Robust Counterpart.

For example, we may assume that at each time t the values of $(u(1), \dots, u(t-1))$ are available to the decision maker. This is a natural assumption: as the end of a period is reached, the actual realized liabilities become known.

This lets us make the decision vectors (strictly causal) functions of u . We assume that the decision variables are strictly causal *linear* functions:

$$x(u) = x + Xu, \quad y(u) = y + Yu, \quad z(u) = z + Zu,$$

where X, Y, Z are a strictly lower-triangular matrices. That way, $x(t), y(t), z(t)$ depend only on $(u(1), \dots, u(t-1))$. Here, the variables are x, y, z and X, Y, Z .

We can write the entire vector of variables $v = (x, y, z)$ as an affine function of u :

$$v(u) = v + Vu, \quad V = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

where v is a vector, and V is a matrix. Both are variables, and V is constrained by the triangular structure of X, Y, Z . Equipped with this form, we are ready to write the Affinely Adjustable Robust Counterpart to the problem.

Affinely Adjustable Robust Counterpart

Max-min form. Our problem with linear recourse is now

$$\begin{aligned} \max_{x,y,z} \quad & \min_{u: \|u\|_\infty \leq 1} c^T(v + Vu) \\ \text{s.t.} \quad & X, Y, Z \text{ strictly lower-triangular,} \\ & \forall u, \|u\|_\infty \leq 1 : A(v + Vu) \geq \hat{b} + Bu \\ & v + Vu \geq 0, \quad x + Xu \leq 100. \quad v = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad V = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \end{aligned}$$

Note that we consider the worst-case objective. We make sure that the constraints remain valid for every admissible u , including the sign constraints on the variables themselves. We can apply the same technique as in standard robust LP to get to a tractable solution.

Tractable form

$$\begin{array}{ll} \max_{v,V} & c^T v - \|V^T c\|_1 \\ \text{s.t.} & Av \geq \hat{b} + |B - AV|_1, \quad v \geq |V|_1, \quad x + |X|_1 \leq 100, \quad X, Y, Z \text{ strictly lower triangular,} \\ & v = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad V = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}. \end{array}$$

The problem is still an LP!

- The presence of V makes it easier to satisfy the cash-flow constraints.
- But it has a negative impact on the other constraints (bounds on variables), and the objective.
- With $V = 0$ we recover the robust solution seen before—information about the uncertainty can only help.

CVX syntax

CVXZ syntax: affine recourse model

```
cvx_begin
    variables x(5,1); y(3,1); z(6,1);
    variable X(5,6) lower_triangular;
    variable Y(3,6) lower_triangular;
    variable Z(6,6) lower_triangular;
    maximize( z(6) - c'*sum(abs(Z),2) );
    A*[x; y; z] >= b+sum(abs(A*[X; Y; Z]-B),2);
    x <= 100-sum(abs(X),2);
    x >= sum(abs(X),2);
    y >= sum(abs(Y),2);
    z >= sum(abs(Z),2);
    diag(X) == 0; diag(Y) == 0; diag(Z) == 0;
cvx_end
```

We use `lower_triangular` to specify lower-triangular variables. Further we use `diag(...)` == 0 constraints to specify the *strictly* lower-triangular structure.

Results

- Nominal solution: worst-case terminal wealth $-\infty$ (infeasible problem).
- Robust solution: worst-case terminal wealth 17.3.
- Linear recourse solution: worst-case terminal wealth 31.5.

Appendix

- [Examples](#).
- [Definitions](#).
- [Theorems and proofs](#).

Examples

- [Representing temperatures at different airports as a vector](#).
- [Bag-of-words representation of text](#).
- [Bag-of-words representation of text: measure of document similarity](#).

Representing temperatures at different airports as a vector

We record the temperatures at three different airports, and obtain the following table.

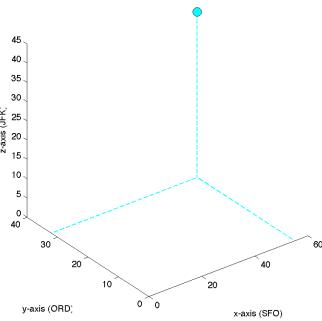
airport temperature ($^{\circ}\text{F}$)

SFO	55
ORD	32
JFK	43



We can represent the temperatures on a single temperature axis. This is known as the *dot* representation of a vector.

The dot representation could become very confusing if there were three temperatures to plot for each day in the year.



Alternatively, we can view the triple of temperatures as a *point* in a three-dimensional space. Each axis corresponds to temperatures at a specific location. The vector representation is still legible if we have more than one triplet of temperatures to represent. The vector representation cannot be viewed in more than three dimensions, that is, if we have more than three cities involved. However, the data can still be represented as a collection of vectors.

Bag-of-words representation of text

Consider the following text:

A (real) vector is just a collection of real numbers, referred to as the components (or, elements) of the vector; \mathbf{R}^n denotes the set of vectors with n elements. If $x \in \mathbf{R}^n$ denotes a vector, we use subscripts to denote elements, so that x_i is the i -th component of x . Vectors are arranged in a column, or a row. If x is a column vector, x^T denotes the corresponding row vector, and vice-versa.

The row vector $x = [5, 3, 4]$ contains the number of times each word in the list {vector, of, the} appear in the above paragraph. Vectors can be thus used to represent text documents. The representation, often referred to as the bag-of-words representation, is not faithful, as it ignores the respective order of appearance of the words. In addition, often, stop words (such as *the* or *of*) are also ignored.

See also:

- [Bag-of-words representation of text: measure of document similarity](#).
- [Gram matrix](#).

Bag-of-words representation of text: measure of document similarity

Returning to the [bag-of-words example](#), we can use the notion of angle to measure how two different documents are close to each other.

Given two documents, and a pre-defined list of words appearing in the documents (the dictionary), we can compute the vectors of frequencies x, y of the words as they appear in the documents. The angle between the two vectors is a widely used measure of closeness (similarity) between documents.

See also:

- [Bag-of-words representation of text](#).
- [Gram matrix](#).

Basis in \mathbf{R}^3

The set of three vectors in \mathbf{R}^3 :

$$x_1 := \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}, \quad x_3 = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix},$$

is not independent, since $x_1 - x_2 + x_3 = 0$, and its span has dimension 2. Since x_1, x_2 are independent (the equation $\lambda_1 x_1 + \lambda_2 x_2 = 0$ has $\lambda = 0$ as the unique solution), a basis for that span is, for example, $\{x_1, x_2\}$. In contrast, the collection $\{x_1, x_2, x_3 - e_1\}$ spans the whole space \mathbf{R}^3 , and thus forms a basis of that space.

Dimension of an affine subspace

The set \mathbf{L} in \mathbf{R}^3 defined by the linear equations

$$x_1 - 13x_2 + 4x_3 = 2, \quad 3x_2 - x_3 = 9$$

is an affine subspace of dimension 1. The corresponding linear subspace is defined by the linear equations obtained from the above by setting the constant terms to zero:

$$x_1 - 13x_2 + 4x_3 = 0, \quad 3x_2 - x_3 = 0$$

We can solve for x_3 and get $x_1 = x_2, x_3 = 3x_2$. We obtain a representation of the linear subspace as the set of vectors $x \in \mathbf{R}^3$ that have the form

$$x = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} t,$$

for some scalar $t = x_2$. Hence the linear subspace is the span of the vector $u := (1, 1, 3)$, and is of dimension 1.

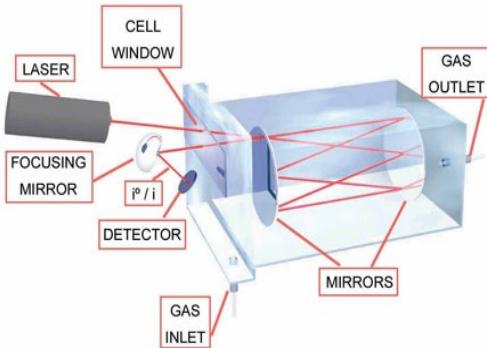
We obtain a representation of the original affine set by finding a particular solution x^0 , by setting say $x_2 = 0$ and solving for x_1, x_3 . We obtain

$$x^0 = \begin{pmatrix} 38 \\ 0 \\ -9 \end{pmatrix}.$$

The affine subspace \mathbf{L} is thus the line $x^0 + \text{span}(u)$, where x^0, u are defined above.

Beer-Lambert Law in Absorption Spectrometry

The *Beer-Lambert law* in optics is an empirical relationship that relates the absorption of light by a material, to the properties of the material through which the light is travelling. This is the basis of absorption spectrometry, which allows to measure the concentration of different gases in a chamber.



The principle of an absorption spectrometer, illustrated on the left, is as follows. Consider the following two experiments. First we do a control experiment, where we illuminate from one side a container containing some reference gas with light at a certain frequency. We measure the light intensity (say, I_0) at the other side of the container. Then, we add some other gas to the container, repeat the experiment, and measure the light intensity again (say, I). Depending on the absorption properties, as well as the concentration x , of the added gas, the light will be more or less absorbed with respect to the reference situation.

The Beer-Lambert law postulates that the log-ratio $\log(I/I_0)$ is linear in the concentration x . In other words, $y = \log(I/I_0) = ax$, where the constant a depends on the light frequency and on the gas.

If the container has a mixture of n “pure” gases in it, the law postulates that the logarithm of the ratio of the light intensities is a linear function of the concentrations of each gas in the mix. The log-ratio of intensities is thus of the form $y = a^T x$ for some vector $a \in \mathbf{R}^n$, where x is the vector of concentrations. The coefficients $a_j, j = 1, \dots, n$ correspond to the log-ratio of light intensities when $x = e_j$ (the j -th vector of the standard basis, which correspond to the j -th pure gas). The quantity a_j is called the coefficient of absorption of the j -th gas, and can be measured in the laboratory.

See also:[Absorption spectrometry: using measurements at different light frequencies](#).

Absorption spectrometry: using measurements at different light frequencies

Return to the absorption spectrometry setup described [here](#).

The Beer-Lambert law postulates that the logarithm of the ratio of the light intensities is a linear function of the concentrations of each gas in the mix. The log-ratio of intensities is thus of the form $y = a^T x$ for some vector $a \in \mathbf{R}^n$, where x is the vector of concentrations, and the vector $a \in \mathbf{R}^n$ contains the coefficients of absorption of each gas. This vector is actually also a function of the frequency of the light we illuminate the container with.

Now consider a container having a mixture of n “pure” gases in it. Denote by $x \in \mathbf{R}^n$ the vector of concentrations of the gases in the mixture. We illuminate the container at different frequencies $\lambda_1, \dots, \lambda_m$. For each experiment, we record the corresponding log-ratio $y_i, i = 1, \dots, m$, of the intensities. If the Beer-Lambert law is to be believed, then we must have

$$y_i = a_i^T x, \quad i = 1, \dots, m,$$

for some vectors $a_i \in \mathbf{R}^n$, which contain the coefficients of absorption of the gases at light frequency λ_i . More compactly:

$$y = Ax$$

where

$$A = \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix}.$$

Thus, A_{ij} is the coefficient of absorption of the j -th gas at frequency λ_i .

Since A_{ij} 's correspond to “pure” gases, they can be measured in the laboratory. We can then use the above model to infer the concentration of the gases in a mixture, given some observed light intensity log-ratio.

See also:[Absorption spectrometry: the Beer-Lambert law](#).

QR decomposition: examples [Matrices](#) > [QR decomposition](#) > Examples

Consider the 6×4 matrix

$$A = \begin{pmatrix} 0.488894 & 0.888396 & 0.325191 & 0.319207 \\ 1.03469 & -1.14707 & -0.754928 & 0.312859 \\ 0.726885 & -1.06887 & 1.3703 & -0.86488 \\ -0.303441 & -0.809499 & -1.71152 & -0.0300513 \\ 0.293871 & -2.94428 & -0.102242 & -0.164879 \\ -0.787283 & 1.43838 & -0.241447 & 0.627707 \end{pmatrix}.$$

This matrix is full column rank. Indeed, the matlab command $[Q, R] = qr(A, 0)$ yields a $6 \times 4Q$ and a $4 \times 4R$:

$$A = QR, \quad Q = \begin{pmatrix} 0.301109 & 0.460748 & -0.0940935 & 0.24499 \\ 0.637266 & 0.0433642 & -0.558601 & 0.251199 \\ 0.447688 & -0.0504968 & 0.519798 & -0.41105 \\ -0.186889 & -0.365412 & -0.617955 & -0.489793 \\ 0.180995 & -0.79363 & 0.163942 & 0.492111 \\ -0.484886 & 0.141088 & -0.0132838 & 0.475232 \end{pmatrix}, \quad R = \begin{pmatrix} 1.62364 & -2.02107 & 0.648726 & -0.420299 \\ 0 & 3.24897 & 0.720385 & 0.434711 \\ 0 & 0 & 2.14747 & -0.67116 \\ 0 & 0 & 0 & 0.744188 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

This shows that A is full column rank since R is invertible.

The command $[Q, R] = qr(A)$ actually produces the full QR decomposition, with Q now a 6×6 orthogonal matrix:

$$Q = \begin{pmatrix} 0.301109 & 0.460748 & -0.0940935 & 0.24499 & 0.692493 & -0.385519 \\ 0.637266 & 0.0433642 & -0.558601 & 0.251199 & -0.253083 & 0.390928 \\ 0.447688 & -0.0504968 & 0.519798 & -0.41105 & 0.332021 & 0.49763 \\ -0.186889 & -0.365412 & -0.617955 & -0.489793 & 0.452645 & 0.0699514 \\ 0.180995 & -0.79363 & 0.163942 & 0.492111 & 0.213643 & -0.15066 \\ -0.484886 & 0.141088 & -0.0132838 & 0.475232 & 0.309248 & 0.650633 \end{pmatrix},$$

$$R = \begin{pmatrix} 1.62364 & -2.02107 & 0.648726 & -0.420299 \\ 0 & 3.24897 & 0.720385 & 0.434711 \\ 0 & 0 & 2.14747 & -0.67116 \\ 0 & 0 & 0 & 0.744188 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

We can see what happens when the input is not full column rank: for example let's consider the matrix

$$A = \begin{pmatrix} 1.09327 & 1.10927 & -0.863653 & 1.32288 \\ -1.21412 & -1.1135 & -0.00684933 & -2.43508 \\ -0.769666 & 0.371379 & -0.225584 & -1.76492 \\ -1.08906 & 0.0325575 & 0.552527 & -1.6256 \\ 1.54421 & 0.0859311 & -1.49159 & 1.59683 \end{pmatrix}$$

(A is not full column rank, as it was constructed so that the last column is a combination of the first and the third.)

The (full) QR decomposition now yields:

$$R = \begin{pmatrix} 2.61388 & 0.909015 & -1.40302 & 3.82473 \\ 0 & 1.33807 & 0.0979073 & 0.0979073 \\ 0 & 0 & 1.16142 & 1.16142 \\ 0 & 0 & 0 & 0.00000 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

We observe that the last triangular element is virtually zero, and the last column is seen to be a linear combination of the first and the third. This shows that the rank of R (itself equal to the rank of A) is effectively 3.

Two orthogonal vectors

The two vectors in \mathbf{R}^3 :

$$x = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}, \quad y = \begin{pmatrix} 4 \\ -1 \\ -1 \end{pmatrix}$$

are orthogonal, since

$$x^T y = \underbrace{1 \times 4}_{x_1 \times y_1} + \underbrace{1 \times (-1)}_{x_2 \times y_2} + \underbrace{(-1) \times 3}_{x_3 \times y_3} = 0.$$

An hyperplane in 3D

Consider an affine set of dimension 2 in \mathbf{R}^3 , which we describe as the set of points $x \in \mathbf{R}^3$ such that there exists two parameters λ_1, λ_2 such that

$$x = \begin{pmatrix} 3\lambda_1 - 4\lambda_2 + 4 \\ \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix} + \lambda_1 \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix} + \lambda_2 \begin{pmatrix} -4 \\ 0 \\ 1 \end{pmatrix}.$$

The set \mathbf{H} can be represented as a translation of a linear subspace: $\mathbf{H} = x_0 + \mathbf{L}$, with

$$x_0 := \begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix},$$

and \mathbf{L} the span of the two *independent* vectors

$$u := \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}, \quad v := \begin{pmatrix} -4 \\ 0 \\ 1 \end{pmatrix}.$$

Thus, the set \mathbf{H} is of dimension 2 in \mathbf{R}^3 , hence it is an hyperplane. In \mathbf{R}^3 , hyperplanes are ordinary planes.

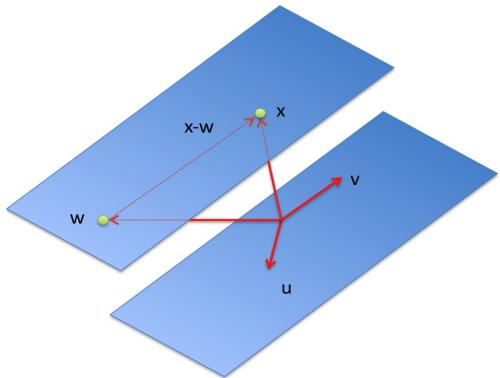
We can find a representation of the hyperplane in the standard form

$$\mathbf{H} = \{x : a^T(x - x_0) = 0\}.$$

We simply find a that is orthogonal to both u and v . That is, we solve the equations

$$0 = a^T u = 3a_1 + a_2 = 0, \quad 0 = a^T v = -4a_1 + a_3.$$

The above leads to $a = (a_1, -3a_1, 4a_1)$. Choosing for example $a_1 = 1$ leads to $a = (1, -3, 4)$.



The hyperplane \mathbf{H} can be expressed as $x_0 + \text{span}(u, v)$, where x_0 is a particular element, and u, v are two independent vectors. The set \mathbf{H} is represented in light blue; it is a translation of the corresponding span $\mathbf{L} = \text{span}(u, v)$. Any point $x \in \mathbf{H}$ is such that $x - x_0$ belongs to \mathbf{L} . Thus we can represent the hyperplane as the set of points such that $x - x_0$ is orthogonal to a , where a is any vector orthogonal to both u, v .

Power law model fitting

Returning to the example involving [power laws](#), we ask the question of finding the “best” model of the form

$$y = C x_1^{a_1} \dots x_n^{a_n},$$

given experiments with several input vectors $x^{(i)}$ and associated outputs $y_i, i = 1, \dots, m$. Here the variables of our problem are C , and the vector $a \in \mathbf{R}^n$. Taking logarithms, we obtain

$$\tilde{y}_i = a^T \tilde{x}^{(i)} + b, \quad i = 1, \dots, m.$$

We can write the above linear equations compactly as

$$\begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} \tilde{x}_1^T & 1 \\ \vdots & \vdots \\ \tilde{x}_m^T & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}.$$

In practice, the power law model is only an approximate model of the reality. Finding the best fit can be addressed via the optimization problem

$$\min_z \|X^T z - y\|_2,$$

where $z = (a, b) \in \mathbf{R}^{n+1}$, $X \in \mathbf{R}^{(n+1) \times m}$, with i -th column given by $(\tilde{x}_1, 1)$.

See also: [Power laws](#).

Gradient of a linear function

Consider the function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$f(x) = x_1 + 2x_2.$$

Its gradient is constant, with values

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

For a given $t \in \mathbf{R}$, the t -level set is the set of points such that $f(x) = t$:

$$\mathbf{L}_t(f) := \{(x_1, x_2) : x_1 + 2x_2 = t\}.$$

The level sets are hyperplanes, and are orthogonal to the gradient.

More generally, the gradient of the linear function with values $f(x) = a^T x$ is $\nabla f(x) = a$.

Linearization of a non-linear function

The [log-sum-exp](#) function

$$f(x) = \log(e^{x_1} + e^{x_2})$$

admits the gradient at the point x^0 given by

$$\nabla f(x_0) = \frac{1}{e^{x_1^0} + e^{x_2^0}} \begin{pmatrix} e^{x_1^0} \\ e^{x_2^0} \end{pmatrix}.$$

Hence f can be approximated near x^0 by the linear function

$$f(x) \approx \log(e^{x_1^0} + e^{x_2^0}) + \frac{1}{(e^{x_1^0} + e^{x_2^0})} ((x_1 - x_1^0)e^{x_1^0} + (x_2 - x_2^0)e^{x_2^0}).$$

Network flow [Matrices](#) > [Matrix products](#) > Example

We describe a *flow* (of goods, traffic, charge, information, etc) across the network as a vector $x \in \mathbf{R}^n$, which describes the amount flowing through any given arc. By convention, we use positive values when the flow is in the direction of the arc, and negative ones in the opposite case. The total flow leaving a given node i is then (remember our convention that the index j spans the arcs)

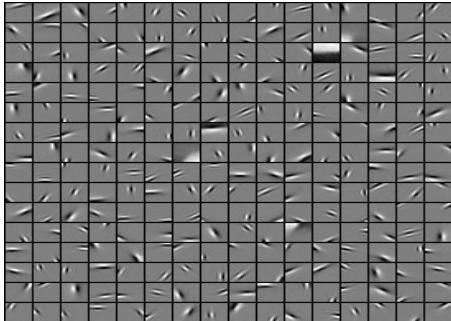
$$\sum_{j=1}^n A_{ij}x_j = (Ax)_i,$$

with $(Ax)_i$ our notation for the i -th component of vector Ax . Now we define the *external supply* as a vector $b \in \mathbf{R}^m$, with negative b_i representing an external demand at node i , and positive b_i a supply. We assume that the total supply equals the total demand, which means $\mathbf{1}^T b = 0$.

The *balance equations* for the supply vector b are $Ax = b$. These equations represent constraints the flow vector must satisfy in order to satisfy the external supply/demand represented by b .

See also: [Incidence matrix of a network](#).

Image Compression [Matrices](#) > [Matrix products](#) > Example



In image compression applications, we are given the pixel representation of a “target” image, as a vector in $y \in \mathbf{R}^m$. We would like to represent the image as a linear combination of “basic” images $a_j, j = 1, \dots, n$, where the matrix $A = [a_1, \dots, a_n] \in \mathbf{R}^{m \times n}$ is called the *dictionary*. The picture on the left shows a total of $n =$ “basic” images.

To represent the image in terms of the dictionary, we would like to find coefficients $x_j, j = 1, \dots, n$ such that

$$y = \sum_{j=1}^n x_j a_j,$$

or, more compactly, $y = Ax$.

If the representation of the image (that is, the vector x) has many zeros (we say: x is *sparse*), then we can represent the entire image with only a few values (the components of x that are not zero). We may then for example, send the image over a communication network at high speed. Provided the receiver has the dictionary handy, it can reconstruct perfectly the image.

In practice, it may be desirable to trade off the sparsity of the representation (via x) against the accuracy of the representation. Namely, we may prefer a representation x that achieves $Ax = y$ only approximately, but has way more zeros. The process of searching for a good sparsity/accuracy trade-off is called image compression.

A 2×2 orthogonal matrix [Matrices](#) > [Matrix products](#) > Example

The matrix

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

is orthogonal.

The vector $x = (2, 1)$ is transformed by the orthogonal matrix above into

$$Ux = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 3 \end{pmatrix}.$$

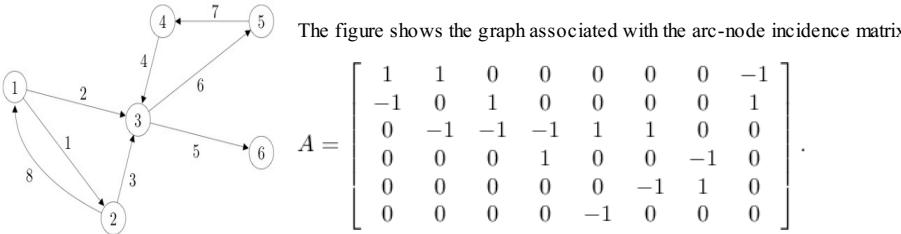
Thus, U corresponds to a rotation of angle 45° counter-clockwise.

Incidence matrix of a network[Matrices > Basics](#) > Example

Mathematically speaking, a *network* is a graph of m nodes connected by n directed arcs. Here, we assume that arcs are ordered pairs, with at most one arc joining any two nodes; we also assume that there are no self-loops (arcs from a node to itself). We do not assume that the edges of the graph are weighted— \square they are all similar.

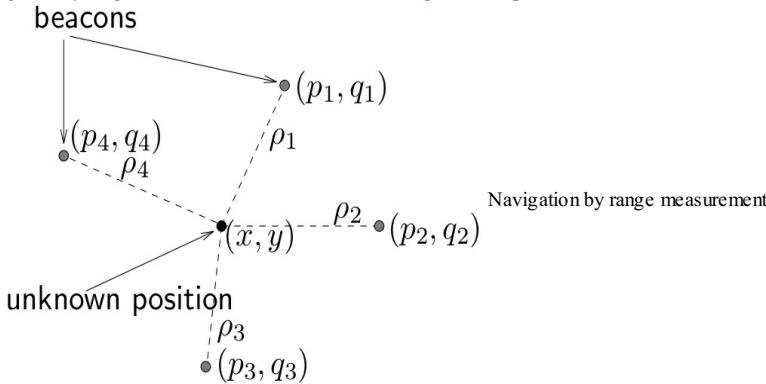
We can fully describe the network with the so-called *arc-node incidence matrix*, which is the $m \times n$ matrix defined as

$$A_{ij} = \begin{cases} 1 & \text{if arc } j \text{ starts at node } i \\ -1 & \text{if arc } j \text{ ends at node } i \\ 0 & \text{otherwise.} \end{cases}, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$



See also:[Network flow](#).

Navigation by range measurement[Matrices > Linear Maps](#) > Example



In the plane, we measure the distances ρ_i of an object located at an unknown position (x, y) from points with known coordinates (p_i, q_i) , $i = 1, \dots, 4$. The distance vector $\rho = (\rho_1, \dots, \rho_4)$ is a non-linear function of x , given by

$$\rho_i(x, y) = \sqrt{(x - p_i)^2 + (y - q_i)^2}, \quad i = 1, \dots, 4.$$

Now assume that we have obtained the position of the object (x_0, y_0) at a given time, and seek to predict the change in position δx that is consistent with observed small changes in the distance vector $\delta \rho$.

We can approximate the non-linear functions ρ_i via the first-order (linear) approximation. A linearized model around a given point (x_0, y_0) is $\delta \rho = A \delta x$, with A a 4×2 matrix with elements

$$a_{i1} = \frac{x_0 - p_i}{\sqrt{(x_0 - p_i)^2 + (y_0 - q_i)^2}}, \quad a_{i2} = \frac{y_0 - q_i}{\sqrt{(x_0 - p_i)^2 + (y_0 - q_i)^2}}, \quad i = 1, \dots, 4.$$

Singular value decomposition of a 4×5 matrix

Consider the matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{pmatrix}.$$

A singular value decomposition of this matrix is given by $A = U \tilde{S} V^T$, with

$$U = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \quad \tilde{S} = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad V^T = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sqrt{0.2} & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & \sqrt{0.2} \end{pmatrix}.$$

Notice above that \tilde{S} has non-zero values only in its diagonal, and can be written as

$$\tilde{S} = \text{diag}(S, 0, 0), \quad S := \text{diag}(\sigma_1, \sigma_2, \sigma_3),$$

with $\sigma_1 = 4, \sigma_2 = 3, \sigma_3 = \sqrt{5}$. The rank of A (which is the number of non-zero elements on the diagonal matrix \tilde{S}) is thus $r = 3 < \min(m, n)$. We can check that $V^T V = VV^T = I_5$, and $U U^T = U^T U = I_4$.

SVD: a 4×4 example

Consider a matrix $A \in \mathbf{R}^{4 \times 4}$, with SVD $A = U \tilde{S} V^T$ where $\tilde{S} = \text{diag}(10, 7, 0.1, 0.05)$.

From the SVD, we can understand the behavior of the mapping $x \rightarrow Ax$:

- input components along directions v_1 and v_2 are amplified (by about a factor 10) and come out mostly along plane spanned by u_1, u_2 .
- Input components along directions v_3 and v_4 are attenuated (by about a factor 10).
- The matrix A is nonsingular.
- For some applications you might say A is effectively rank 2.

A two-dimensional toy optimization problem

As a toy example of an optimization problem in two variables, consider the problem

$$\min_x 0.9x_1^2 - 0.4x_1x_2 - 0.6x_2^2 - 6.4x_1 - 0.8x_2 : -1 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq 3.$$

(Note that the term “subject to” has been replaced with the shorthand colon notation.)

The problem can be put in [standard form](#)

$$p^* := \min_x f_0(x) : f_i(x) \leq 0, \quad i = 1, \dots, m,$$

where:

- the decision variable is $(x_1, x_2) \in \mathbf{R}^2$;
- the objective function $f_0 : \mathbf{R}^2 \rightarrow \mathbf{R}$, takes values

$$f_0(x) := 0.9x_1^2 - 0.4x_1x_2 - 0.6x_2^2 - 6.4x_1 - 0.8x_2;$$

- the constraint functions $f_i : \mathbf{R}^n \rightarrow \mathbf{R}, i = 1, 2, 3, 4$ take values

$$\begin{aligned} f_1(x) &:= -x_1 - 1, \\ f_2(x) &:= x_1 - 2, \\ f_3(x) &:= -x_2, \\ f_4(x) &:= x_2 - 3. \end{aligned}$$

- p^* is the *optimal value*, which turns out to be $p^* = -10.2667$.

See also:

- [Geometric view of the toy problem](#)
- [Nomenclature for the toy problem](#).

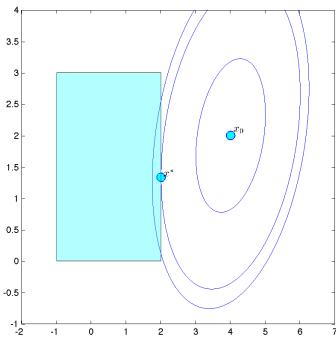
A toy 2D optimization problem: geometric view via the epigraph form

Consider the toy problem

$$\min_x 0.9x_1^2 - 0.4x_1x_2 - 0.6x_2^2 - 6.4x_1 - 0.8x_2 : -1 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq 3.$$

We can represent the problem in [epigraph form](#), as

$$\min_{x,t} t : t \geq 0.9x_1^2 - 0.4x_1x_2 - 0.6x_2^2 - 6.4x_1 - 0.8x_2, \quad -1 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq 3.$$



Geometric view of the toy optimization problem above. The level curves (curves of constant value) of the objective function are shown. The problem amounts to find the smallest value of t such that $t = f_0(x)$ for some feasible x . The plot shows the *unconstrained* minimum of the objective function, located at $\hat{x} = (4, 2)$.

See also:

- [An optimization problem in two variables](#).
- [Nomenclature for the toy problem](#).

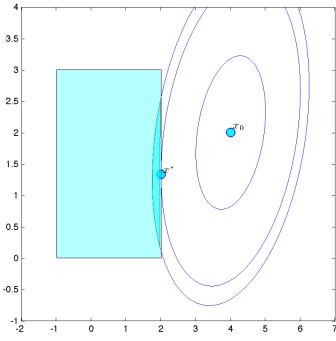
Nomenclature of a toy 2D optimization problem

Consider the [toy optimization problem in two variables](#):

$$\min_{\underline{x}} \underline{x}_1^2 - x_1 x_2 + 2x_2^2 - 3x_1 - 1.5x_2 : -1 \leq x_1 \leq 2, 0 \leq x_2 \leq 3.$$

For this problem:

- The optimal value is $p^* = -10.2667$.
 - The optimal set is the singleton $\mathbf{X}^{\text{opt}} = \{x^*\}$, with
- $$x^* = \begin{pmatrix} 2.00 \\ 1.33 \end{pmatrix}.$$
- Since the optimal set is not empty, the problem is attained.



An ϵ -sub-optimal set for the toy problem above is shown (in darker color), for $\epsilon = 0.9$. This corresponds to the set of feasible points that achieves an objective value less or equal than $p^* + \epsilon$.

See also:

- [Geometric view of the toy problem](#).

Eigenvalue Decomposition of a 2×2 Symmetric Matrix [Symmetric Matrices](#) > [Definitions](#) > Example

Let

$$A := \begin{pmatrix} 3/2 & -1/2 \\ -1/2 & 3/2 \end{pmatrix}.$$

We solve for the characteristic equation:

$$0 = \det(\lambda I - A) = (\lambda - (3/2))^2 - (1/4) = (\lambda - 1)(\lambda - 2).$$

Hence the eigenvalues are $\lambda_1 = 1, \lambda_2 = 2$. For each eigenvalue λ , we look for a unit-norm vector u such that $Au = \lambda u$. For $\lambda = \lambda_1$, we obtain the equation in $u = u_1$

$$0 = (A - \lambda_1)u_1 = \begin{pmatrix} 1/2 & -1/2 \\ -1/2 & 1/2 \end{pmatrix}u_1,$$

which leads to (after normalization) to an eigenvector $u_1 := (1/\sqrt{2})[1, 1]$. Similarly for λ_2 we obtain the eigenvector $u_2 := (1/\sqrt{2})[1, -1]$. Hence, A admits the SED

$$A = \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right)^T \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right).$$

See also: [Sums-of-squares for a quadratic form](#).

Quadratic functions in two variables [Symmetric Matrices](#) > [Definitions](#) > Example

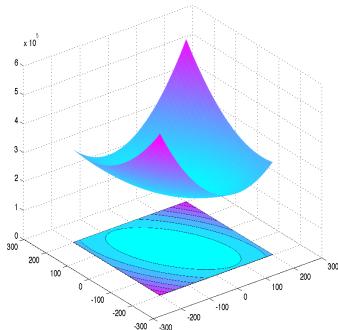
Two examples of quadratic function are $p, q : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$\begin{aligned} p(x) &= 4x_1^2 + 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 2 \times 10^5, \\ q(x) &= 4x_1^2 - 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 2 \times 10^5. \end{aligned}$$

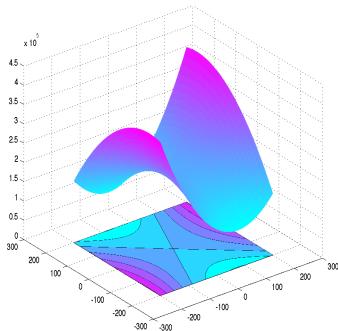
The function

$$r(x) = 4x_1^2 + 2x_2^2 + 3x_1x_2$$

is a form, since it has no linear or constant terms in it.



Level sets and graph of the quadratic function p . The epigraph is anything that extends above the graph in the z -axis direction. This function is “bowl-shaped”, or convex.

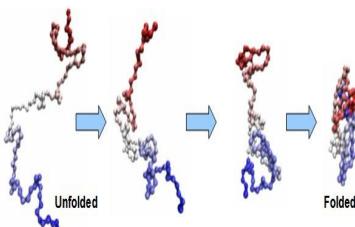


Level sets and graph of the quadratic function q . This quadratic function is not convex.

range of a 4×5 matrix via its SVD

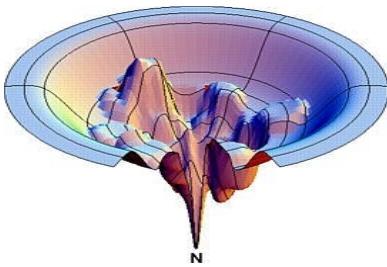
Returning to [this example](#), the Frobenius norm is the square root of the sum of the squares of the elements, and is equal to $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_3^2} = 5.4772$.

Protein folding



The *protein folding* problem is to predict the three-dimensional structure of a protein based on the sequence of the amino-acids that constitutes it. The amino-acids interact with each other (for example, they may be electrically charged). Such a problem is difficult to address experimentally, which calls for computer-aided methods.

In recent years, some authors have proposed to express the problem as an optimization problem, involving the minimization of a potential energy function, which is usually a sum of terms reflecting the interactions between pairs of amino-acids. The overall problem can be modeled as a non-linear optimization problem.



Unfortunately, protein folding problems remain challenging. One of the reasons is the size of the problem (number of variables and constraints). Another difficulty comes from the fact that the potential energy function (which the actual protein is minimizing) is not exactly known. Finally, the fact that the energy function is usually not linear, and not even convex, may lead algorithms to discover "spurious" (that is, wrong) molecular conformations. The picture on the left is a three-dimensional rendition of the level sets of the energy function.

Crew allocation in airline operations

Crew allocation problems in the airline industry involve assigning crews to airplanes. Such problems are modeled with boolean decision variables x_{ij} , with $x_{ij} = 1$ if the i -th crew is assigned to the j -th plane, and 0 otherwise. Other constraints may be added; for example, to express that exactly one crew is to be assigned to each plane, we write the constraint

$$\forall i : \sum_j x_{ij} = 1.$$

Theorems and Proofs

- [Cauchy-Schwartz inequality](#).
- [A theorem on non-negative quadratic forms and eigenvalues](#).

Cauchy-Schwartz inequality

Theorem: Cauchy-Schwartz inequality

For any two vectors $x, y \in \mathbf{R}^n$, we have

$$x^T y \leq \|x\|_2 \cdot \|y\|_2.$$

The above inequality is an equality if and only if x, y are collinear. In other words:

$$\max_{x : \|x\|_2 \leq 1} x^T y = \|y\|_2,$$

with optimal x given by $x^* = y/\|y\|_2$ if y is non-zero.

Proof: The inequality is trivial if either one of the vectors x, y is zero. Let us assume both are non-zero. Without loss of generality, we may re-scale x and assume it has unit Euclidean norm ($\|x\|_2 = 1$). Let us first prove that

$$x^T y \leq \|y\|_2.$$

We consider the polynomial

$$p(t) := \|tx - y\|_2^2 = t^2 - 2t(x^T y) + y^T y.$$

Since it is non-negative for every value of t , its discriminant $\Delta = (x^T y)^2 - y^T y$ is non-positive. The Cauchy-Schwartz inequality follows.

Let us now prove that equality holds if and only if x, y are collinear. The "only if" part is trivial. If the equality holds, then $\Delta = 0$, which implies that

$$p(t) = (t - x^T y)^2 + y^T y - (x^T y)^2 = (t - x^T y)^2 + \Delta = (t - x^T y)^2.$$

Then, the minimum value of $p(t)$ over t is zero. Hence $y = tx$ for some t , and x, y are collinear.

The last result is proven as follows. Let $v(P)$ be the optimal value of the problem. The Cauchy-Schwartz inequality implies that $v(P) \leq \|y\|_2$. To prove that the value is attained (it is equal to its upper bound), we observe that if $x = y/\|y\|_2$, then

$$x^T y = \frac{y^T y}{\|y\|_2} = \|y\|_2.$$

The vector $x = y/\|y\|_2$ is feasible for the optimization problem (P) . This establishes a lower bound on the value of $v(P)$.

$$\|y\|_2 \leq v(P) = \max_{x : \|x\|_2 \leq 1} x^T y.$$

Dimension of hyperplanes Theorem

A set H in \mathbf{R}^n of the form

$$H = \{x : a^T x = b\},$$

where $a \in \mathbf{R}^n$, $a \neq 0$, and $b \in \mathbf{R}$ are given, is an affine set of dimension $n - 1$.

Conversely, any affine set of dimension $n - 1$ can be represented by a single affine equation of the form $a^T x = b$, as in the above.

Proof:

- Consider a set \mathbf{H} described by a single affine equation:

$$a_1 x_1 + \dots + a_n x_n = b,$$

with $a \neq 0$. Let us assume for example that $a_1 \neq 0$. We can express x_1 as follows:

$$x_1 = b - \frac{a_2}{a_1} x_2 - \dots - \frac{a_n}{a_1} x_n.$$

This shows that the set is of the form $z_0 + \text{span}(z_1, \dots, z_{n-1})$, where

$$z_0 = \begin{pmatrix} b \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad z_1 = \begin{pmatrix} -\frac{a_2}{a_1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \dots, \quad z_{n-1} = \begin{pmatrix} -\frac{a_n}{a_1} \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

Since the vectors z_1, \dots, z_{n-1} are independent, the dimension of \mathbf{H} is $n - 1$. This proves that \mathbf{H} is indeed an affine set of dimension $n - 1$.

- The converse is also true. Any subspace \mathbf{L} of dimension $n - 1$ can be represented via an equation $a^T x = 0$ for some $a \neq 0$. A sketch of the proof is as follows. We use the fact that we can form a basis (z_1, \dots, z_{n-1}) for the subspace \mathbf{L} . We can then construct a vector a that is orthogonal to all of these basis vectors. By definition, \mathbf{L} is the set of vectors that are orthogonal to a .

Spectral theorem: eigenvalue decomposition for symmetric matrices

We can decompose any symmetric matrix $A \in \mathbf{S}^n$ with the *symmetric eigenvalue decomposition* (SED)

$$A = \sum_{i=1}^n \lambda_i u_i u_i^T = U \Lambda U^T, \quad \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n).$$

where the matrix of $U := [u_1, \dots, u_n]$ is orthogonal (that is, $U^T U = U U^T = I_n$), and contains the eigenvectors of A , while the diagonal matrix Λ contains the eigenvalues of A .

Proof: The proof is by induction on the size n of the matrix A . The result is trivial for $n = 1$. Now let $n > 1$ and assume the result is true for any matrix of size $n - 1$.

Consider the function of A , $t \rightarrow p(t) = \det(tI - A)$. From the basic properties of the [determinant](#), it is a polynomial of degree n , called the *characteristic polynomial* of A . By the [fundamental theorem of algebra](#), any polynomial of degree n has n (possibly not distinct) complex roots; these are called the eigenvalues of A . We denote these eigenvalues by $\lambda_1, \dots, \lambda_n$.

If λ is an eigenvalue of A , that is, $\det(\lambda I - A) = 0$, then $\lambda I - A$ must be non-invertible (see [here](#)). This means that there exist a non-zero real vector u such that $Au = \lambda u$. We can always normalize u so that $u^T u = 1$. Thus, $\lambda = u^T A u$ is real. That is, the eigenvalues of a symmetric matrix are always real.

Now consider the eigenvalue λ_1 and an associated eigenvector u_1 . Using the [Gram-Schmidt orthogonalization procedure](#), we can compute a $n \times (n - 1)$ matrix V_1 such that $[u_1, V_1]$ is orthogonal. By induction, we can write the $(n - 1) \times (n - 1)$ symmetric matrix $V_1^T A V_1$ as $Q_1 \Lambda_1 Q_1^T$, where Q_1 is a $(n - 1) \times (n - 1)$ matrix of eigenvectors, and $\Lambda_1 = \text{diag}(\lambda_2, \dots, \lambda_n)$ are the $n - 1$ eigenvalues of $V_1^T A V_1$. Finally, we define the $n \times (n - 1)$ matrix $U_1 := V_1 Q_1$. By construction the matrix $U := [u_1, U_1]$ is orthogonal.

We have

$$U^T A U = \begin{pmatrix} u_1^T \\ U_1^T \end{pmatrix} A \begin{pmatrix} u_1 & U_1 \end{pmatrix} = \begin{pmatrix} u_1^T A u_1 & u_1^T A U_1 \\ U_1^T A u_1 & U_1^T A U_1 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \Lambda_1 \end{pmatrix},$$

where we have exploited the fact that $U_1^T A u_1 = \lambda_1 U_1^T u_1 = 0$, and $U_1^T A U_1 = \Lambda_1$.

We have exhibited an orthogonal $n \times n$ matrix U such that $U^T A U$ is diagonal. This proves the theorem.

Rayleigh Quotients Theorem

Theorem. For a symmetric matrix A , we can express the smallest and largest eigenvalues, $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$, as

$$\begin{aligned} \lambda_{\min}(A) &= \min_x \{x^T A x : x^T x = 1\}, \\ \lambda_{\max}(A) &= \max_x \{x^T A x : x^T x = 1\}. \end{aligned}$$

Proof: The proof of the expression above derives from the SED of the matrix, and the invariance of the Euclidean norm constraint under orthogonal transformations. We show this only for the largest eigenvalue; the proof for the expression for the smallest eigenvalue follows similar lines. Indeed, with $A = U \Lambda U^T$, we have

$$\max_x \{x^T A x : x^T x = 1\} = \max_x \{x^T U \Lambda U^T x : x^T x = 1\}.$$

Now we can define the new variable $\tilde{x} = U^T x$, so that $x = U \tilde{x}$, and express the problem as

$$\begin{aligned} \max_x \{x^T A x : x^T x = 1\} &= \max_{\tilde{x}} \{\tilde{x}^T \Lambda \tilde{x} : \tilde{x}^T \tilde{x} = 1\} \\ &= \max_{\tilde{x}} \left\{ \sum_{i=1}^n \lambda_i \tilde{x}_i^2 : \sum_{i=1}^n \tilde{x}_i^2 = 1 \right\}. \end{aligned}$$

Clearly, the maximum is less than λ_{\max} . That upper bound is attained, with $\tilde{x}_i = 1$ for an index i such that $\lambda_i = \lambda_{\max}$, and $\tilde{x}_j = 0$ for $j \neq i$. This proves the result. This corresponds to setting $x = U\tilde{x} = u_i$, where u_i is the eigenvector corresponding to $\lambda_i = \lambda_{\max}$.

Optimal set of Least-Squares via SVD **Theorem: optimal set of ordinary least-squares**

The optimal set of the OLS problem

$$p^* := \min_x \|Ax - y\|_2$$

can be expressed as

$$\mathbf{X}^{\text{opt}} = A^\dagger y + \mathbf{N}(A).$$

where A^\dagger is the [pseudo-inverse](#) of A , and $A^\dagger y$ is the minimum-norm point in the optimal set. If A is full column rank, the solution is unique, and equal to $x^* = A^\dagger y = (A^T A)^{-1} A^T y$. In general, the particular solution $A^\dagger y$ is the minimum-norm solution to the least-squares problem.

Proof: The following proof relies on the [SVD](#) of A , and the [rotational invariance](#) of the Euclidean norm.

Optimal value of the problem

Using the SVD we can find the optimal set to the least-squares optimization problem $p^* := \min_x \|Ax - y\|_2^2$. Indeed, if (U, \tilde{S}, V) is an SVD of A , the problem can be written $p^* = \min_x \left\| U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T x - y \right\|_2^2 = \min_x \left\| \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T x - U^T y \right\|_2^2$, where we have exploited the fact that the Euclidean norm is invariant under the orthogonal transformation U^T . With $\tilde{x} := V^T x$, and $\tilde{y} := U^T y$, and changing the variable x to \tilde{x} , we express the above as $p^* = \min_{\tilde{x}} \left\| \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} \tilde{x} - \tilde{y} \right\|_2^2$.

Expanding the terms, and using the partitioned notations $\tilde{x} = (\tilde{x}_r, \tilde{x}_{n-r})$, $\tilde{y} = (\tilde{y}_r, \tilde{y}_{m-r})$, we obtain $p^* = \min_{\tilde{x}_r} \|S\tilde{x}_r - \tilde{y}_r\|_2^2 + \|\tilde{y}_{m-r}\|_2^2$. Since S is invertible, we can reduce the first term in the objective to zero with the choice $\tilde{x}_r = S^{-1}\tilde{y}_r$. Hence the optimal value is $p^* = \|\tilde{y}_{m-r}\|_2^2$. We observe that the optimal value is zero if and only if $y \in \mathbf{R}(A)$, which is exactly the same as $\tilde{y}_{m-r} = 0$.

Optimal set

Let us detail the optimal set for the problem. The variable \tilde{x} is partly determined, via its first r components: $\tilde{x}_r^* = S^{-1}\tilde{y}_r$. The remaining $n - r$ variables contained in \tilde{x}_{n-r} are free, as \tilde{x}_{n-r} does not appear in the objective function of the above problem.

Thus, optimal points are of the form $x = V\tilde{x}$, with $\tilde{x} = (\tilde{x}_r^*, \tilde{x}_{n-r})$, $\tilde{x}_r^* = S^{-1}\tilde{y}_r$, and \tilde{x}_{n-r} free.

To express this in terms of the original SVD of A , we observe that $x = V\tilde{x} = V(\tilde{x}_r, \tilde{x}_{n-r})$ means that $x = V_r\tilde{x}_r + V_{n-r}\tilde{x}_{n-r} = V_r S^{-1}\tilde{y}_r + V_{n-r}\tilde{x}_{n-r}$ where V is partitioned as $V = (V_r, V_{n-r})$, with $V_r \in \mathbf{R}^{n \times r}$ and $V_{n-r} \in \mathbf{R}^{n \times (n-r)}$. Similarly, the vector \tilde{y}_r can be expressed as $\tilde{y}_r = U_r^T y$, with U_r formed with the first r columns of U . Thus, any element x^* in the optimal set is of the form $x^* = x_{MN} + V_{n-r}z : z \in \mathbf{R}^{n-r}$, where $x_{MN} := V_r S^{-1} U_r^T y$. (We will soon explain the acronym appearing in the subscript.) The free components \tilde{x}_{n-r} correspond to the degrees of freedom allowed to by the nullspace of A .

Minimum-norm optimal point

The particular solution to the problem, x_{MN} , is the [minimum-norm solution](#), in the sense that it is the element of \mathbf{X}^{opt} that has the smallest Euclidean norm. This is best understood in the space of \tilde{x} -variables.

Indeed, the particular choice $\tilde{x} = (\tilde{x}_r^*, 0)$ corresponds to the element in the optimal set that has the smallest Euclidean norm. Indeed, the norm of \tilde{x} is the same as that of its rotated version, \tilde{x} : the first r elements in $\tilde{x}\tilde{x}^*$ are fixed, and since $\|\tilde{x}\|_2^2 = \|\tilde{x}_{n-r}\|_2^2 + \|\tilde{x}_{n-r}\|_2^2$, we see that the minimal norm is obtained with $\tilde{x}_{n-r} = 0$.

Optimal set via the pseudo-inverse

The matrix $V_r S^{-1} U_r^T$, which appears in the expression of the particular solution x_{MN} mentioned above, is nothing else than the [pseudo-inverse](#) of A , which is denoted A^\dagger . Indeed, we can express the pseudo-inverse in terms of the SVD as $A^\dagger = V \begin{pmatrix} S^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T$. With this convention, the minimum-norm optimal point is $A^\dagger y$. Recall that the last $n - r$ columns of V form a basis for the nullspace of A . Hence the optimal set of the problem is $\mathbf{X}^{\text{opt}} = A^\dagger y + \mathbf{N}(A)$. When A is full column rank ($r = n \leq m$, and $A^T A \succ 0$), the optimal set reduces to a singleton, as the nullspace is $\{0\}$. The unique optimal point expresses as $x_{\text{LS}} = A^\dagger y = (A^T A)^{-1} A^T y$.

Definitions

- [Vector norms](#)
- [Linear and affine maps](#)
- [Square-to-linear function](#)
- [Log-Sum-Exp \(LSE\) function](#).

Sample and weighted mean, expected value

The [sample mean](#) (or, average) of given numbers x_1, \dots, x_n , is defined as

$$\hat{x} := \frac{1}{n}(x_1 + \dots + x_n).$$

The sample average can be interpreted as a scalar product:

$$\hat{x} = p^T x,$$

where $x = (x_1, \dots, x_n)$ is the vector containing the samples, and $p = (1/n)\mathbf{1}$, with $\mathbf{1}$ the vector of ones.

More generally, for any vector $p \in \mathbf{R}^n$, with $p_i \geq 0$ for every i , and $p_1 + \dots + p_n = 1$, we can define the corresponding *weighted average* as $p^T x$. The interpretation of p is in terms of a discrete probability distribution of a random variable X , which takes the value x_i with probability p_i , $i = 1, \dots, n$. The weighted average is then simply the *expected value* (or, mean) of X under the probability distribution p . The expected value is often denoted $\mathbf{E}_p(X)$, or $\mathbf{E}(X)$ if the distribution p is clear from context.

Sample variance and standard deviation

The *sample variance* of given numbers x_1, \dots, x_n , is defined as

$$\sigma^2 := \frac{1}{n} ((x_1 - \hat{x})^2 + \dots + (x_n - \hat{x})^2),$$

where \hat{x} is the [sample average](#) of x_1, \dots, x_n . The sample variance is a measure of the deviations of the numbers x_i with respect to the average value \hat{x} .

The *sample standard deviation* is the square root of the sample variance, σ^2 . It can be expressed in terms of the Euclidean norm of the vector $x = (x_1, \dots, x_n)$, as

$$\sigma = \frac{1}{\sqrt{n}} \|x - \hat{x}\mathbf{1}\|_2,$$

where $\|\cdot\|_2$ denotes the Euclidean norm.

More generally, for any vector $p \in \mathbf{R}^n$, with $p_i \geq 0$ for every i , and $p_1 + \dots + p_n = 1$, we can define the corresponding *weighted variance* as

$$\sum_{i=1}^n p_i(x_i - \hat{x})^2.$$

The interpretation of p is in terms of a discrete probability distribution of a random variable X , which takes the value x_i with probability p_i , $i = 1, \dots, n$. The weighted variance is then simply the *expected value* of the squared deviation of X from its mean $\mathbf{E}X$, under the probability distribution p .

See also: [sample and weighted average](#).

Gram matrix

Consider $m n$ -vectors x_1, \dots, x_m . The *Gram matrix* of the collection is the $m \times m$ matrix G with elements $G_{ij} = x_i^T x_j$. The matrix can be expressed compactly in terms of the matrix $X = [x_1, \dots, x_m]$ as

$$G = X^T X = \begin{bmatrix} x_1^T \\ \vdots \\ x_m^T \end{bmatrix} \begin{bmatrix} x_1 & \dots & x_m \end{bmatrix}.$$

Assume that each vector x_i is normalized: $\|x_i\|_2 = 1$. Then the coefficient G_{ij} can be expressed as $G_{ij} = \cos \theta_{ij}$, where θ_{ij} is the angle between the vectors x_i and x_j . Thus G_{ij} is a measure of how similar x_i and x_j are.

The matrix G arises for example in text document classification, with G_{ij} a measure of similarity between the i -th and j -th document, and x_i, x_j their respective bag-of-words representation (normalized to have Euclidean norm 1). It also arises in the name of the Kernel matrix in the [Kernel least-squares](#) approach.

See also:

- [Bag-of-words representation of text](#).
- [Bag-of-words representation of text: measure of document similarity](#).

Rate of return of a financial portfolio Rate of return of a single asset

The rate of return r (or, return) of a financial asset over a given period (say, a year, or a day) is the interest obtained at the end of the period by investing in it. In other words, if, at the beginning of the period, we invest a sum S in the asset, we will earn $S_{\text{end}} := (1+r)S$ at the end. That is:

$$r = \frac{S_{\text{end}} - S}{S} \text{ Log-returns}$$

Often, the rates of return are approximated, especially if the period length is small. If $r \ll 1$, then

$$r = \frac{S_{\text{end}}}{S} - 1 \approx y := \log \frac{S_{\text{end}}}{S},$$

with the latter quantity known as *log-return*.

Rate of return of a portfolio

For n assets, we can define the vector $r \in \mathbf{R}^n$, with r_i the rate of return of the i -th asset.

Assume that at the beginning of the period, we invest a sum S in all the assets, allocating a fraction x_i (in %) in the i -th asset. Here $x \in \mathbf{R}^n$ is a non-negative vector which sums to one. Then the portfolio we constituted this way will earn

$$S_{\text{end}} := \sum_{i=1}^n (1 + r_i) x_i S.$$

The rate of return of the portfolio is the relative increase in wealth:

$$\frac{S_{\text{end}} - S}{S} = \sum_{i=1}^n (1 + r_i) x_i - 1 = \sum_{i=1}^n x_i - 1 + \sum_{i=1}^n r_i x_i = r^T x.$$

The rate of return is thus the scalar product between the vector of individual returns r and of the portfolio allocation weights x .

Note that, in practice, rates of return are never known in advance, and they can be negative (although, by construction, they are never less than -1).

Solving triangular systems of equations: backwards substitution example

$$Rx = \begin{pmatrix} 3 & 8 & 3 \\ 0 & 6 & -1 \\ 0 & 0 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ -3 \end{pmatrix}$$

Consider the triangular system we solve for the last variable x_3 first, obtaining (from the last equation) $x_3 = 1$. We plug this value of x_3 into the first and second equation, obtaining a new triangular system in two variables x_1, x_2 : $\begin{pmatrix} 3 & 8 \\ 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 - 3x_3 \\ 2 + x_3 \end{pmatrix} = \begin{pmatrix} -2 \\ 3 \end{pmatrix}$.

We proceed by solving for the last variable x_2 . The last equation yields $x_2 = 3/6 = 1/2$. Plugging this value into the first equation gives $3x_1 = -2 - 8x_2 = -6$.

$$Rx^{(1)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad Rx^{(2)} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad Rx^{(3)} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \quad \text{The matrix}$$

We can apply the idea to find the inverse of the square upper triangular matrix R , by solving

$$R^{-1} = (x^{(1)} \mid x^{(2)} \mid x^{(3)}) = \left(\begin{array}{ccc|cc} 1/3 & -4/9 & 13/27 \\ 0 & 1/6 & -1/18 \\ 0 & 0 & -1/3 \end{array} \right).$$

$[x^{(1)}, x^{(2)}, x^{(3)}]$ is then the inverse of R . We find As illustrated above, the inverse of a triangular matrix is triangular.

Linear and Affine Maps

Here are some equivalent definitions of linear or affine functions and maps.

A map $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ is *linear* if and only if either one of the following conditions hold.

- f preserves scaling and addition of its arguments:

- for every $x \in \mathbf{R}^n$, and $\alpha \in \mathbf{R}$, $f(\alpha x) = \alpha f(x)$; and
- for every $x_1, x_2 \in \mathbf{R}^n$, $f(x_1 + x_2) = f(x_1) + f(x_2)$.

- f vanishes at the origin: $f(0) = 0$, and transforms any line segment in \mathbf{R}^n into another segment in \mathbf{R}^m :

$$\forall x, y \in \mathbf{R}^n, \quad \forall \lambda \in [0, 1] : f(\lambda x + (1 - \lambda)y) = \lambda f(x) + (1 - \lambda)f(y).$$

- f is differentiable, vanishes at the origin, and the matrix of its derivatives is constant.

- There exist $A \in \mathbf{R}^{m \times n}$ such that

$$\forall x \in \mathbf{R}^n : f(x) = Ax.$$

A map $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ is *affine* if and only if the map $g : \mathbf{R}^n \rightarrow \mathbf{R}^m$ with values $g(x) = f(x) - f(0)$ is linear.

Example: the function $f : \mathbf{R}^3 \rightarrow \mathbf{R}$, with values

$$f(x) = -2.1x_1 + 3.5x_2 + 4.2x_3 + \alpha$$

is linear when $\alpha = 0$, and affine otherwise. In contrast, the function with values

$$f(x) = -2.1x_1 + 3.5x_2x_3 + 4.2x_3 + \alpha$$

is not linear nor affine, whatever the value of α .

Definition: vector norm

Informally, a (vector) norm is a function which assigns a length to vectors.

Any sensible measure of length should satisfy the following basic properties: it should be a convex function of its argument (that is, the length of an average of two vectors should be always less than the average of their lengths); it should be positive-definite (always non-negative, and zero only when the argument is the zero vector), and preserve positive scaling (so that multiplying a vector by a positive number scales its norm accordingly).

Formally, a *vector norm* is a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ which satisfies the following properties.

Definition of a vector norm

1. *Positive homogeneity*: for every $x \in \mathbf{R}^n$, $\alpha \geq 0$, we have $f(\alpha x) = \alpha f(x)$.

2. *Triangle inequality*: for every $x, y \in \mathbf{R}^n$, we have

$$f(x+y) \leq f(x) + f(y)$$

1. *Definiteness*: for every $x \in \mathbf{R}^n$, $f(x) = 0$ implies $x = 0$.

A consequence of the first two conditions is that a norm only assumes non-negative values, and that it is convex.

Popular norms include the so-called l_p -norms, where $p = 1, 2$ or $p = \infty$:

$$\|x\|_p := \left(\sum_{i=1}^n |x_i|^p \right)^{1/p},$$

with the convention that when $p = \infty$, $\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$

Dual norm and generalized Cauchy-Schwartz inequality

For a given norm $\|\cdot\|$ on \mathbf{R}^n , the *dual norm*, denoted $\|\cdot\|_*$, is the function from \mathbf{R}^n to \mathbf{R} with values

$$\|y\|_* = \max_x x^T y : \|x\| \leq 1.$$

The above definition indeed corresponds to a norm: it is convex, as it is the pointwise maximum of convex (in fact, linear) functions $x \rightarrow y^T x$; it is homogeneous of degree 1, that is, $\|\alpha x\|_* = \alpha \|x\|_*$ for every $x \in \mathbf{R}^n$ and $\alpha \geq 0$.

By definition of the dual norm,

$$x^T y \leq \|x\| \cdot \|y\|_*.$$

This can be seen as a generalized version of the [Cauchy-Schwartz inequality](#), which corresponds to the Euclidean norm.

Examples:

- The norm dual to the Euclidean norm is itself. This comes directly from the [Cauchy-Schwartz inequality](#).
- The norm dual to the l_∞ -norm is the l_1 -norm. This is because the inequality

$$x^T y \leq \|x\|_\infty \cdot \|y\|_1$$

holds trivially, and is attained for $x = \text{sign}(y)$.

- The dual to the dual norm above is the original norm we started with. (The proof of this general result is more involved.)

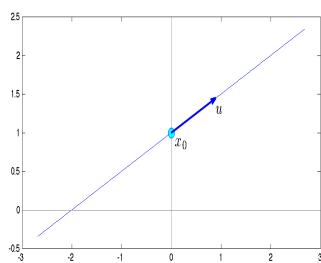
Lines in \mathbf{R}^n

The *line* in \mathbf{R}^n passing through $x_0 \in \mathbf{R}^n$ and with direction $u \in \mathbf{R}^n$, $u \neq 0$, is the set of vectors x such that $x - x_0$ is parallel to u :

$$\{x_0 + tu : t \in \mathbf{R}\}.$$

We can always assume without loss of generality that the direction u is normalized, that is $\|u\|_2 = 1$.

Lines are affine sets of dimension one (since they are translations of the span of one vector).



A line in \mathbf{R}^2 passing through the point $x_0 = (2, 0)$, with (normalized) direction $u = (0.8944, 0.4472)$.

Euclidean projection on a set

An *Euclidean projection* of a point $x_0 \in \mathbf{R}^n$ on a set $S \subseteq \mathbf{R}^n$ is a point that achieves the smallest Euclidean distance from x_0 to the set. That is, it is any solution to the optimization problem

$$\min_x \|x - x_0\|_2 : x \in S.$$

When the set \mathbf{S} is [convex](#), there is a unique solution to the above problem. In particular, the projection on an affine subspace is unique.

Example: assume that \mathbf{S} is the [hyperplane](#).

$$\mathbf{S} = \{x \in \mathbf{R}^3 : 2x_1 + x_2 - x_3 = 1\}.$$

The projection problem reads as a [linearly constrained least-squares problem](#), of particularly simple form:

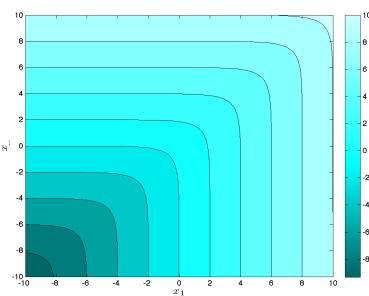
$$\min_x \|x\|_2 : 2x_1 + x_2 - x_3 = 1.$$

The projection of $x_0 = 0$ on \mathbf{S} turns out to be aligned with the coefficient vector $a = (2, 1, -1)$. Indeed, components of x orthogonal to a don't appear in the constraint, and only increase the objective value. Setting $x = ta$ in the equation defining the hyperplane and solving for the scalar t we obtain $t = 1/(a^T a) = 1/6$, so that the projection is $x^* = a/(a^T a) = (1/3, 1/6, -1/6)$.

Definition: Log-Sum-Exp (LSE) function

The *log-sum-exp* (LSE) function in \mathbf{R}^n is the function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, with domain the whole space \mathbf{R}^n , and value at a point $x \in \mathbf{R}^n$ given by

$$f(x) = \log \left(\sum_{i=1}^n e^{x_i} \right).$$



The log-sum-exp function in \mathbf{R}^2 . For large positive values, the function is a smooth approximation to the maximum function $(x_1, x_2) \rightarrow \max(x_1, x_2)$.

The log-sum-exp function is increasing with respect to each argument, and convex.

Proof: The monotonicity of the log-sum-exp function is obvious. The convexity is obtained as follows. As seen [here](#), the Hessian of the log-sum-exp function is

$$\nabla^2 f(x) = \frac{1}{S(x)^2} (S(x)\text{diag}(s(x)) - s(x)s(x)^T),$$

where $s(x) = (e^{x_1}, \dots, e^{x_n})$, and $S(x) = \sum_{i=1}^n s_i(x)$.

We need to check that for every $z \in \mathbf{R}^n$, we have $z^T \nabla^2 f(x) z \geq 0$. Let us fix a vector $z \in \mathbf{R}^n$. We have

$$\begin{aligned} S(x)^2 \cdot z^T \nabla^2 f(x) z &= z^T (S(x)\text{diag}(s(x)) - s(x)s(x)^T) z \\ &= (\sum_{i=1}^n s_i(x)z_i^2) (\sum_{i=1}^n s_i(x)) - (\sum_{i=1}^n s_i(x)z_i)^2 \geq 0, \end{aligned}$$

due to the Cauchy-Schwartz inequality.

Power laws

Consider a physical process which has inputs x_j , $j = 1, \dots, n$, and a scalar output y . Inputs and output are physical, positive quantities, such as volume, height, or temperature. In many cases, we can (at least empirically) describe such physical processes by *power laws*, which are non-linear models of the form

$$y = \alpha x_1^{a_1} \dots x_n^{a_n},$$

where $\alpha > 0$, and the coefficients a_j , $j = 1, \dots, n$ are real numbers. For example, the relationship between area, volume and size of basic geometric objects; the [Coulomb law](#) in electrostatics; birth and survival rates of (say) bacteria as functions of concentrations of chemicals; heat flows and losses in pipes, as functions of the pipe geometry; analog circuit properties as functions of circuit parameters; etc.

The relationship $x \rightarrow y$ is not linear nor affine, but if we introduce the new variables $\tilde{y} := \log y$, $\tilde{x}_j := \log x_j$, $j = 1, \dots, n$, then the above equation becomes an affine one:

$$\tilde{y} = \log \alpha + \sum_{j=1}^n a_j \log x_j = a^T \tilde{x} + b,$$

where $b := \log \alpha$.

See also: [Fitting power laws to data](#), [Beer-Lambert law](#) in spectroscopy.

Definition: Linear and Affine Maps

Here are some equivalent definitions of linear or affine functions and maps.

A map $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ is *linear* if and only if either one of the following conditions hold.

- f preserves scaling and addition of its arguments:

- for every $x \in \mathbf{R}^n$, and $\alpha \in \mathbf{R}$, $f(\alpha x) = \alpha f(x)$; and
- for every $x_1, x_2 \in \mathbf{R}^n$, $f(x_1 + x_2) = f(x_1) + f(x_2)$.

- f vanishes at the origin: $f(0) = 0$, and transforms any line segment in \mathbf{R}^n into another segment in \mathbf{R}^m :

$$\forall x, y \in \mathbf{R}^n, \forall \lambda \in [0, 1] : f(\lambda x + (1 - \lambda)y) = \lambda f(x) + (1 - \lambda)f(y).$$

- f is differentiable, vanishes at the origin, and the matrix of its derivatives is constant.

- There exist $A \in \mathbf{R}^{m \times n}$ such that

$$\forall x \in \mathbf{R}^n : f(x) = Ax.$$

A map $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ is *affine* if and only if the map $g : \mathbf{R}^n \rightarrow \mathbf{R}^m$ with values $g(x) = f(x) - f(0)$ is linear.

Example: the function $f : \mathbf{R}^3 \rightarrow \mathbf{R}$, with values

$$f(x) = -2.1x_1 + 3.5x_2 + 4.2x_3 + \alpha$$

is linear when $\alpha = 0$, and affine otherwise. In contrast, the function with values

$$f(x) = -2.1x_1 + 3.5x_2x_3 + 4.2x_3 + \alpha$$

is not linear nor affine, whatever the value of α .

Gradient of a function

The *gradient* of a differentiable function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ contains the first derivatives of the function with respect to each variable. As seen [here](#), the gradient is useful to find the linear approximation of the function near a point.

Definition

The gradient of f at x_0 , denoted $\nabla f(x_0)$, is the vector in \mathbf{R}^n given by

$$\nabla f(x_0) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x_0) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_0) \end{pmatrix}.$$

Examples:

- *Distance function:* The distance function from a point $p \in \mathbf{R}^2$ to another point $x \in \mathbf{R}^2$ is defined as

$$\rho(x) = \|x - p\|_2 = \sqrt{(x_1 - p_1)^2 + (x_2 - p_2)^2}.$$

The function is differentiable, provided $(x, y) \neq (p, q)$, which we assume. Then

$$\nabla \rho(x) = \frac{1}{\sqrt{(x_1 - p_1)^2 + (x_2 - p_2)^2}} \begin{pmatrix} x_1 - p_1 \\ x_2 - p_2 \end{pmatrix}.$$

- *Log-sum-exp function:* Consider the “log-sum-exp” function $\text{lse} : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$\text{lse}(x) := \log(e^{x_1} + e^{x_2}).$$

The gradient of L at x is

$$\nabla \text{lse}(x) = \frac{1}{z_1 + z_2} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix},$$

where $z_i := e^{x_i}$, $i = 1, 2$. More generally, the gradient of the function $\text{lse} : \mathbf{R}^n \rightarrow \mathbf{R}$ with values

$$\text{lse}(x) = \log \left(\sum_{i=1}^n e^{x_i} \right)$$

is given by

$$\nabla f(x) = \frac{1}{\sum_{i=1}^n e^{x_i}} \begin{pmatrix} e^{x_1} \\ \dots \\ e^{x_n} \end{pmatrix} = \frac{1}{Z} z,$$

where $z = (e^{x_1}, \dots, e^{x_n})$, and $Z = \sum_{i=1}^n z_i$.

Composition rule with an affine function

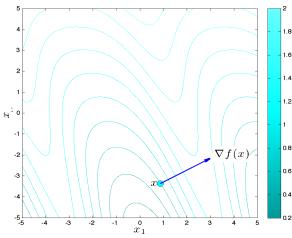
If $A \in \mathbf{R}^{m \times n}$ is a matrix, and $b \in \mathbf{R}^m$ is a vector, the function $g : \mathbf{R}^m \rightarrow \mathbf{R}$ with values

$$g(x) = f(Ax + b)$$

is called the *composition* of the affine map $x \rightarrow Ax + b$ with f . Its gradient is given by (see [here](#) for a proof)

$$\nabla g(x) = A^T \nabla f(Ax + b).$$

Geometrically, the gradient can be read on the plot of the level set of the function. Specifically, at any point x , the gradient is perpendicular to the level set, and points outwards from the sub-level set (that is, it points towards *higher* values of the function).



Level and sub-level sets of the function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ with values

$$f(x) = \text{lse}(\sin(x_1 + 0.3x_2), 0.2x_2).$$

The gradient at a point (shown in red) is perpendicular to the level set, and points outside the corresponding sub-level set. The length of the gradient determines how fast the function changes locally (The length of the gradient has been scaled up by a factor of 5).

Hessian of a FunctionDefinition

The *Hessian* of a twice-differentiable function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ at a point $x \in \text{dom } f$ is the matrix containing the second derivatives of the function at that point. That is, the Hessian is the matrix with elements given by

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}(x), \quad 1 \leq i, j \leq n.$$

The Hessian of f at x is often denoted $\nabla^2 f(x)$.

The second-derivative is independent of the order in which derivatives are taken. Hence, $H_{ij} = H_{ji}$ for every pair (i, j) . Thus, the Hessian is a [symmetric matrix](#).

ExamplesHessian of a quadratic function

Consider the quadratic function

$$q(x) = x_1^2 + 2x_1x_2 + 3x_2^2 + 4x_1 + 5x_2 + 6.$$

The Hessian of q at x is given by

$$\frac{\partial^2 q}{\partial x_i \partial x_j}(x) = \begin{pmatrix} \frac{\partial^2 q}{\partial x_1^2}(x) & \frac{\partial^2 q}{\partial x_1 \partial x_2}(x) \\ \frac{\partial^2 q}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 q}{\partial x_2^2}(x) \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 2 & 6 \end{pmatrix}.$$

For quadratic functions, the Hessian is a constant matrix, that is, it does not depend on the point at which it is evaluated.

Hessian of the log-sum-exp function

Consider the “log-sum-exp” function $\text{lse} : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$\text{lse}(x) := \log(e^{x_1} + e^{x_2}).$$

The gradient of lse at x is

$$\nabla \text{lse}(x) = \frac{1}{z_1 + z_2} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix},$$

where $z_i := e^{x_i}$, $i = 1, 2$. The Hessian is given by

$$\nabla^2 \text{lse}(x) = \frac{z_1 z_2}{(z_1 + z_2)^2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

More generally, the Hessian of the function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ with values

$$\text{lse}(x) = \log \left(\sum_{i=1}^n e^{x_i} \right)$$

is as follows.

- First the gradient at a point x is (see [here](#)):

$$\nabla \text{lse}(x) = \frac{1}{\sum_{i=1}^n e^{x_i}} \begin{pmatrix} e^{x_1} \\ \dots \\ e^{x_n} \end{pmatrix} = \frac{1}{Z} z,$$

where $z = (e^{x_1}, \dots, e^{x_n})$, and $Z = \sum_{i=1}^n z_i$.

- Now the Hessian at a point x is obtained by taking derivatives of each component of the gradient. If $g_i(x)$ is the i -th component, that is,

$$g_i(x) = \frac{e^{x_i}}{\sum_{i=1}^n e^{x_i}} = \frac{z_i}{Z},$$

then

$$\frac{\partial g_i(x)}{\partial x_i} = \frac{z_i}{Z} - \frac{z_i^2}{Z^2},$$

and, for $j \neq i$:

$$\frac{\partial g_i(x)}{\partial x_j} = -\frac{z_i z_j}{Z^2}.$$

More compactly:

$$\nabla^2 \text{lse}(x) = \frac{1}{Z^2} (Z \text{diag}(z) - zz^T).$$

Sample covariance matrix definition

For a vector $z \in \mathbf{R}^m$, the [sample variance](#) σ^2 measures the average deviation of its coefficients around the [sample average](#) \hat{x} :

$$\hat{z} := \frac{1}{m}(z_1 + \dots + z_m), \quad \sigma^2 := \frac{1}{m}((z_1 - \hat{z})^2 + \dots + (\hat{z} - z_m)^2)$$

Now consider a matrix $X = [x_1, \dots, x_m] \in \mathbf{R}^{n \times m}$, where each column x_i represents a data point in \mathbf{R}^n . We are interested in describing the amount of variance in this data set. To this end, we look at the numbers we obtain by [projecting the data along a line](#) defined by the direction $u \in \mathbf{R}^n$. This corresponds to the (row) vector in \mathbf{R}^m $z = (u^T x_1, \dots, u^T x_m) = u^T X \in \mathbf{R}^m$. The corresponding sample mean and variance are

$$\hat{z} = u^T \hat{x}, \quad \sigma^2(u) := \frac{1}{m} \sum_{k=1}^m (u^T x_k - u^T \hat{x})^2,$$

where $\hat{x} := (1/m)(x_1 + \dots + x_m) \in \mathbf{R}^n$ is the sample mean of the vectors x_1, \dots, x_m .

The sample variance along direction u can be expressed as a quadratic form in u :

$$\sigma^2(u) = \frac{1}{n} \sum_{k=1}^n [u^T (x_k - \hat{x})]^2 = u^T \Sigma u,$$

where Σ is a $n \times n$ symmetric matrix, called the [sample covariance matrix](#) of the data points:

$$\Sigma := \frac{1}{m} \sum_{k=1}^m (x_k - \hat{x})(x_k - \hat{x})^T.$$

Properties

The covariance matrix satisfies the following properties.

- The sample covariance matrix allows to find the variance along any direction in data space.
- The diagonal elements of Σ give the variances of each vector in the data.
- The trace of Σ gives the sum of all the variances.
- The matrix Σ is positive semi-definite, since the associated quadratic form $u \rightarrow u^T \Sigma u$ is non-negative everywhere.

Matlab syntax

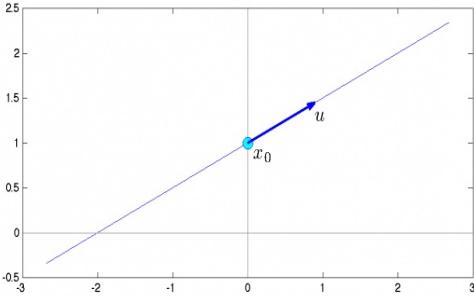
The following matlab syntax assumes that the m data points in \mathbf{R}^n are collected in a $n \times m$ matrix $X: X = [x_1, \dots, x_m]$

Matlab syntax

```
xhat = mean(X, 2); % mean of columns of matrix X
Xc = X - xhat * ones(1, m); % centered data matrix
Sigma = (1/m) * Xc' * Xc; % covariance matrix
Sigma = cov(X', 1); % built-in command produces the same thing
```

Projection of a vector on a line A line

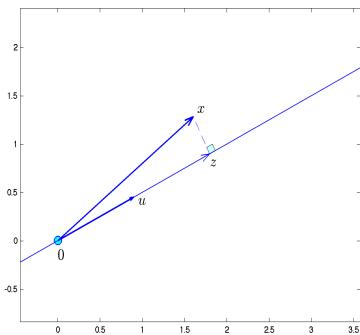
The line in \mathbf{R}^n passing through $x_0 \in \mathbf{R}^n$ and with direction $u \in \mathbf{R}^n, u \neq 0$, is the set of vectors x such that $x - x_0$ is parallel to $u: \{x_0 + tu : t \in \mathbf{R}\}$. We can always assume without loss of generality that the direction u is normalized, that is $\|u\|_2 = 1$.



A line in \mathbf{R}^2 passing through the point $x_0 = (2, 0)$, with (normalized) direction $u = (0.8944, 0.4472)$.

Definition

The *projection* of a given point \mathcal{X} on the line is a vector \mathcal{Z} located on the line, that is closest to \mathcal{X} (in Euclidean norm). This corresponds to a simple optimization problem: $\min_t \|x - x_0 - tu\|_2$. (This particular problem is part of a general class of optimization problems known as least-squares.)



Projection of the vector $x = (1.6, 2.28)$ on a line passing through the origin ($x_0 = 0$) and with (normalized) direction $u = (0.8944, 0.4472)$. At optimality the “residual” vector $x - z$ is orthogonal to the line, hence $z = tu$, with $t = x^T u = 2.0035$. Any other point on the line is farther away from the point x .

Closed-form expression

Assuming that u is normalized, the optimal solution to the above problem is $t^* = u^T(x - x_0)$, and the expression for the projected vector is $z^* = x_0 + t^*u = x_0 + u^T(x - x_0)u$. In

$$z^* = x_0 + \frac{u^T(x - x_0)}{u^T u} u.$$

the case when u is not normalized, the expression is

Proof: Let us first assume that u is normalized. We express the square of the objective function as $\|x - x_0 - tu\|_2^2 = t^2 - 2\alpha t + \beta^2 = (t - \alpha)^2 + \text{constant}$, where $\alpha := u^T(x - x_0)$, $\beta := \|x - x_0\|_2$. The minimum is clearly attained when the first term is zero, which yields $t^* = u^T(x - x_0)$, as claimed. ♣

State-space models of linear dynamical systems

Many discrete-time dynamical systems can be modeled via linear state-space equations, of the form

$$x(t+1) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t), \quad t = 0, 1, 2, \dots$$

where $x(t) \in \mathbf{R}^n$ is the state, which encapsulates the state of the system at time t , $u(t) \in \mathbf{R}^p$ contains control variables, $y(t) \in \mathbf{R}^k$ contains specific outputs of interest, and A, B, C, D are matrices of appropriate size.

In effect, a linear dynamical model postulates that the state at the next instant is a linear function of the state at past instants, and possibly other “exogenous” inputs; and that the output is a linear function of the state and input vectors.

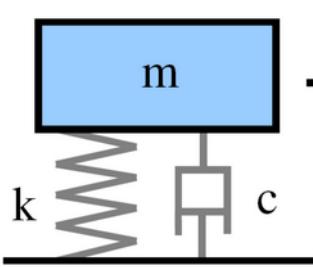
A continuous-time model would take the form of a differential equation

$$\frac{d}{dt}x(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t), \quad t \geq 0.$$

Finally, the so-called *time-varying* models involve time-varying matrices A, B, C, D (see an example below).

Motivation

The main motivation for state-space models is to be able to model high-order derivatives in dynamical equations, using only first-order derivatives, but involving vectors instead of scalar quantities.



Consider for example the second-order differential equation

$m\ddot{y}(t) + c\dot{y}(t) + ky(t) = u(t),$
which describes the evolution of a damped mass-spring system, with u the external force acting on the mass, and y the vertical position.
(Here \dot{y} and \ddot{y} are the first and second derivatives of y , respectively.)

The above involves second-order derivatives of a scalar function $y(\cdot)$. We can express it in an equivalent form involving only first-order derivatives, by defining the state vector to be

$$x(t) := \begin{pmatrix} y(t) \\ \dot{y}(t) \end{pmatrix}.$$

The price we pay is that now we deal with a *vector* equation instead of a scalar equation:

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ -\frac{c}{m} & -\frac{k}{m} \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t).$$

The position $y(t)$ is a linear function of the state:

$$y(t) = Cx(t),$$

with $C = (1, 0)$.

A nonlinear system

In the case of non-linear systems, we can also use state-space representations. In the case of autonomous systems (no external input) for example, these come in the form

$$\dot{x}(t) = f(x(t))$$

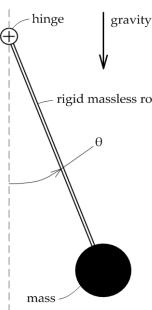
where $f : \mathbf{R}^{n+p} \rightarrow \mathbf{R}^n$ is now a non-linear map. Now assume we want to model the behavior of the system near an equilibrium point x_0 (such that $f(x_0) = 0$). Let us assume for simplicity that $x_0 = 0$.

Using the [first-order approximation](#) of the map f , we can write a linear approximation to the above model:

$$\frac{d}{dt}x(t) = Ax(t), \quad t \geq 0,$$

where

$$A = \frac{\partial f}{\partial x}(0).$$



The motion of a pendulum can be described by the dimensionless nonlinear equation

$$\ddot{\theta} = -\sin \theta$$

The linearization around the equilibrium point $\theta = 0$ yields $\ddot{\theta} = -\theta$. The linearization around the equilibrium point $\theta = \pi$ yields $\ddot{\theta} = 1 - \theta$.

Functions and maps

In this livebook, we define *functions* as objects which take an argument in \mathbf{R}^n , and return a value in \mathbf{R} . We use the notation

$$f : \mathbf{R}^n \rightarrow \mathbf{R},$$

to refer to a function with “input” space \mathbf{R}^n . The “output” space for functions is \mathbf{R} .

Example: The function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$ with values

$$f(x) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

gives the distance from the point (x_1, x_2) to (y_1, y_2) .

We allow for functions to take infinity values. The *domain* of a function f , denoted $\text{dom } f$, is defined as the set of points where the function is finite.

Example:

- Define the logarithm function as the function $f : \mathbf{R} \rightarrow \mathbf{R}$, with values $f(x) = \log x$ if $x > 0$, and $-\infty$ otherwise. The domain of the function is thus \mathbf{R}_{++} (the set of positive reals).

Maps

We reserve the term *map* to refer to functions which return more than a single value, and use the notation

$$f : \mathbf{R}^n \rightarrow \mathbf{R}^m,$$

to refer to a map with input space \mathbf{R}^n and output space \mathbf{R}^m . The *components* of the map f are the (scalar-valued) functions $f_i, i = 1, \dots, m$.

Example: [a map](#).

Determinant of a square matrix

- Definition
- An important result
- Geometry
- Formula
- Determinant and inverse
- Some properties

We review a few important facts about determinants here. For a self-contained exposition that includes proofs, see [this text](#) by Carl de Boor.

Definition

The *determinant* of a square, $n \times n$ matrix A , denoted $\det A$, is defined by an algebraic formula of the coefficients of A . The following formula for the determinant, known as *Laplace's expansion formula*, allows to compute the determinant recursively:

$$\det A = \sum_{i=1}^n (-1)^{i+1} A_{i,1} \det(C_i),$$

where $C_{i,1}$ is the $(n-1) \times (n-1)$ matrix obtained from A by removing the i -th row and first column. (The first column does not play a special role here: the determinant remains the same if we use any other column.)

The determinant is the *unique* function of the entries of A such that

1. $\det I = 1$.
2. $A \rightarrow \det A$ is a linear function of any column (when the others are fixed).
3. $\det A$ changes sign when two columns are permuted.

There are other expressions of the determinant, including *Leibnitz formula* (proven [here](#)):

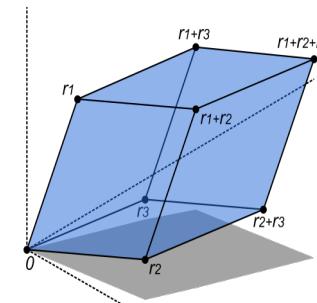
$$\det A = \sum_{\sigma \in S_n} \text{sign}(\sigma) A_{\sigma(i),i}$$

where S_n denotes the set of permutations σ of the integers $1, 2, \dots, n$. Here, $\text{sign}(\sigma)$ denotes the *sign* of the permutation σ , which is the number of pairwise exchanges required to transform $\sigma(1), \sigma(2), \dots, \sigma(n)$ into $1, 2, \dots, n$.

Important result

An important result is that a square matrix is invertible if and only if its determinant is not zero. We use this key result when introducing [eigenvalues of symmetric matrices](#).

Geometry



The determinant of a 3×3 matrix A with columns r_1, r_2, r_3 is the volume of the parallelepiped defined by the vectors r_1, r_2, r_3 . (Source: [wikipedia](#)). Hence the determinant is a measure of scale that quantifies how the linear map associated with A , $x \rightarrow Ax$, changes volumes.

In general, the absolute value of the determinant of a $n \times n$ matrix is the volume of the parallelepiped

$$\{Ax : 0 \leq x_i \leq 1, i = 1, \dots, n\}.$$

This is consistent with the fact that when A is not invertible, its columns define a parallelepiped of zero volume.

Determinant and inverse

The determinant can be used to compute the *inverse* of a square, full rank (that is, invertible) matrix A : the inverse $B = A^{-1}$ has elements given by

$$B_{ij} = \frac{(-1)^{i+j}}{\det A} \det(\tilde{A}_{ij}), \text{ where } \tilde{A}_{ij} \text{ is a matrix obtained from } A \text{ by removing its } i\text{-th row and } j\text{-th column. For example, the determinant of a } 2 \times 2 \text{ matrix } A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

is given by

$$\det A = ad - bc.$$

It is indeed the volume of the area of a parallelepiped defined with the columns of A , (a, c) , (b, d) . The inverse is given by

$$A^{-1} = \frac{1}{(ad - bc)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Some properties of determinants of triangular matrices. If a matrix is square, triangular, then its determinant is simply the product of its diagonal coefficients. This comes right from Laplace's expansion formula above. Determinant of transpose. The determinant of a square matrix and that of its transpose are equal. Determinant of a product of matrices. For two invertible square matrices, we have $\det AB = \det A \cdot \det B$.

In particular:

$$\det A^{-1} = \frac{1}{\det A}.$$

This also implies that for an orthogonal matrix U , that is, a $n \times n$ matrix with $U^T U = I$, we have

$$1 = \det U^T U = (\det U^T) \det U = (\det U)^2$$

Determinant of block matrices. As a generalization of the above result, we have for three compatible blocks A, C, D :

$$\det \left(\begin{array}{c|c} A & 0 \\ \hline C & D \end{array} \right) = \det D \cdot \det A.$$

A more general formula is

$$\det \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right) = \det D \cdot \det(A - BD^{-1}C).$$

Permutation matrices

A $n \times n$ matrix P is a permutation matrix if it is obtained by permuting the rows or columns of an $n \times n$ identity matrix according to some permutation of the numbers 1 to n . Permutation matrices are [orthogonal](#) (hence, their inverse is their transpose: $P^{-1} = P^T$) and satisfy $P^2 = P$.

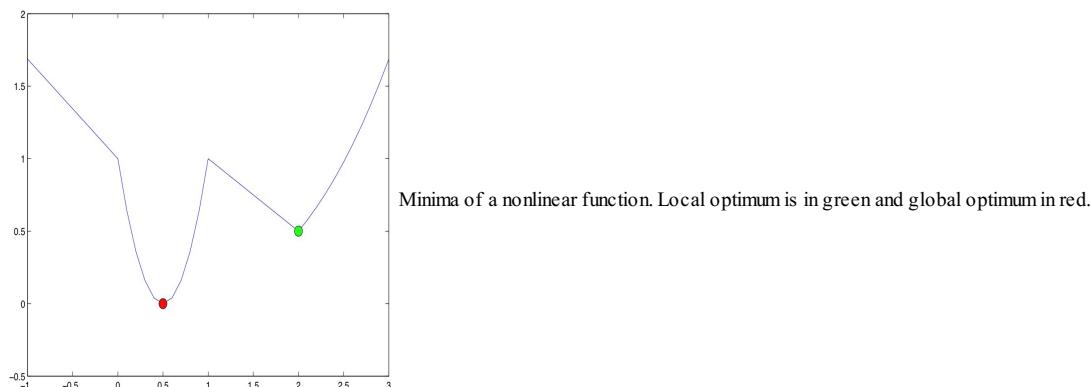
For example, the matrix

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

is obtained by exchanging the columns 2 and 3, and 4 and 5, of the 6×6 identity matrix.

A permutation matrix allows to exchange rows or columns of another via the matrix-matrix product. For example, if we take any 5×6 matrix A , then AP (with P defined above) is the matrix A with columns 2, 3 and 4, 5 exchanged.

Global vs. local minima



Backwards substitution for solving triangular linear systems

Consider a triangular system of the form $Rx = y$, where the vector $y \in \mathbf{R}^m$ is given, and R is upper-triangular.

Let us first consider the case when $m = n$, and R is invertible. Thus, R has the form

$$R = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ 0 & r_{22} & & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & & 0 & r_{nn} \end{pmatrix}$$

with each r_{ii} , $i = 1, \dots, n$ non-zero.

The backwards substitution first solves for the last component of x using the last equation:

$$x_n = \frac{1}{r_{nn}} y_n,$$

and then proceeds with the following recursion, for $j = n - 1, \dots, 1$:

$$x_j = \frac{1}{r_{jj}} \left(y_j - \sum_{k=j+1}^n r_{jk} x_k \right).$$

Example: [Solving a \$3 \times 3\$ triangular system by backwards substitution](#)

Laplacian matrix of a graph [Symmetric Matrices](#) > [Definitions](#) > Example

Another important symmetric matrix associated with a graph is the *Laplacian matrix*. This is the matrix $L = A^T A$, with A the arc-node incidence matrix. It can be shown that the (i, j) element of the Laplacian matrix is given by

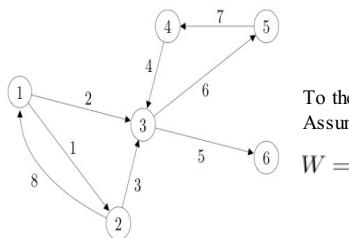
$$L_{ij} = \begin{cases} \# \text{ arcs incident to node } i & \text{if } i = j, \\ -1 & \text{if there is an arc joining node } i \text{ to node } j, \\ 0 & \text{otherwise.} \end{cases}$$

See also:

- [Arc-node incidence matrix of a graph](#).
- [Edge-weight matrix of a graph](#).

Edge weight matrix of a graph [Symmetric Matrices](#) > [Definitions](#) > Example

A symmetric matrix is a way to describe a weighted, undirected graph: each edge in the graph is assigned a weight W_{ij} . Since the graph is undirected, the edge weight is independent of the direction (from i to j or vice-versa). Hence, W is symmetric.



To the graph in the figure, we can associate the corresponding undirected graph, obtained by ignoring the direction of the arrows. Assuming that all the edges have the same weight, the undirected graph has the edge weight matrix given by

$$W =$$

See also:

- [Arc-node incidence matrix of a graph](#).

Sample average of vectors

The *sample average* of given vectors $x_1, \dots, x_n \in \mathbf{R}^m$, is defined as the vector $\hat{x} \in \mathbf{R}^m$, with

$$\hat{x} := \frac{1}{n}(x_1 + \dots + x_n).$$

Homeworks

- Homework 1.
- [Homework 2](#).
- Homework 3.

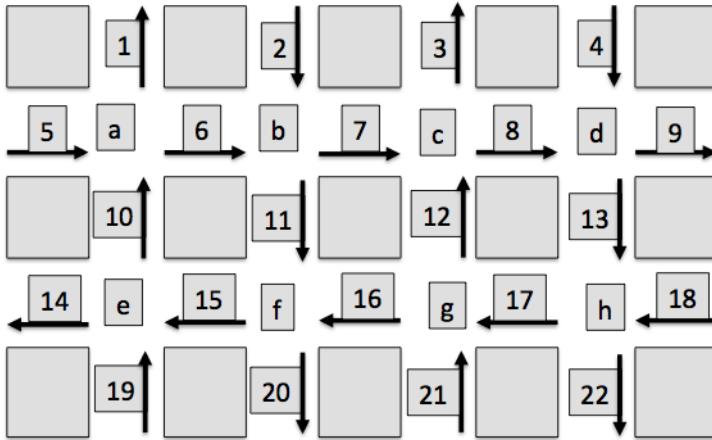
Homework 1

Homework 2

Problem 1 (Least norm estimation on traffic flow networks)

You want to estimate the traffic (in San Francisco for example, but we'll start with a smaller example). You know the road network as well as the historical average of flows on each road segment.

1. We call q_i the flow of vehicles on each road segment $i \in I$. Write down the linear equation that corresponds to the conservation of vehicles at each intersection $j \in J$. Hint: think about how you might represent the road network in terms of matrices, vectors, etc.
2. The goal of the estimation is to estimate the traffic flow on each of the road segment. The flow estimates should satisfy the conservation of vehicles exactly at each intersection. Among the solutions that satisfy this constraint, we are searching for the estimate that is the closest to the historical average, q_i , in the ℓ_2 -norm sense. The vector q has size $|I|$ and the i -th element represent the average for the road segment i . Pose the optimization problem.
3. Explain how to solve this problem mathematically. Detail your answer (do not only give a formula but explain where it comes from).



4.

Figure 1: Example of traffic estimation problem. The intersections are labeled a to h. The road segments are labeled 1 to 22. The arrows indicate the direction of traffic.

Formulate the problem for the small example of Figure 1 and solve it using the historical average given in Table 1. What is the flow that you estimate on road segments 1, 3, 6, 15 and 22?

segment	average	measured
1	2047.6	2028
2	2046.0	2008
3	2002.6	2035
4	2036.9	
5	2013.5	2019
6	2021.1	
7	2027.4	
8	2047.1	
9	2020.9	2044
10	2049.2	
11	2015.1	
12	2035.1	
13	2033.3	
14	2027.0	2043
15	2034.9	
16	2033.3	
17	2008.9	
18	2006.4	
19	2050.0	2030
20	2008.6	2025
21	2001.6	
22	2028.1	2045

Table 1. Table of flows: historical averages q (center column), and some measured flows (right column).

5. Now, assume that besides the historical averages, you are also given some flow measurements on some of the road segments of the network. You assume that these flow measurements are correct and want your estimate of the flow to match these measurements perfectly (besides matching the conservation of vehicles of course). The right column of Table 1 lists the road segments for which we have such flow measurements. Do you estimate a different flow on some of the links? Give the difference in flow you estimate for road segments 1, 3, 6, 15 and 22. Also check that your estimate gives you the measured flow on the road segments for which you have measured the flow.

Exercise 2 (Exploiting structure in linear equations)

Consider the linear equation in $x \in \mathbb{R}^n$

$$Ax = y$$

where $A \in \mathbf{R}^{m \times n}$, $y \in \mathbf{R}^m$. Answer the following questions to the best of your knowledge.

1. The time required to solve the general system depends on the sizes m , n and the entries of A . Provide a rough estimate of that time as a function of m , n only. You may assume that m , n are of the same order.
2. Assume now that $m = n$, $A = D + uv^T$, where D is diagonal, invertible, and $u, v \in \mathbf{R}^n$. How would you exploit this structure to solve the above linear system, and what is a rough estimate of the complexity of your algorithm?
3. What if A is upper-triangular?

Exercise 3 (Formulating problems as LPs or QPs)

Formulate the problem

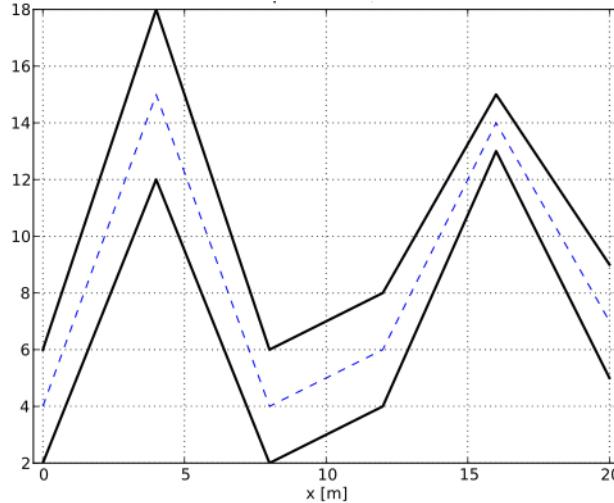
$$p_j^* = \min_x f_j(x)$$

for different functions f_j , $j = 1, \dots, 5$, with values given in Table 2, as QPs or LPs, or, if you cannot, explain why. In our formulations, we always use $x \in \mathbf{R}^n$ as the variable, and assume that $A \in \mathbf{R}^{m \times n}$, $y \in \mathbf{R}^m$, and $k \subseteq \{1, \dots, m\}$ are given. If you obtain an LP or QP formulation, make sure to put the problem in standard form, stating precisely what the variables, objective and constraints are.

$$\begin{aligned} f_1(x) &= \|Ax - y\|_\infty + \|x\|_1 \\ f_2(x) &= \|Ax - y\|_2^2 + \|x\|_1 \\ f_3(x) &= \|Ax - y\|_2^2 - \|x\|_1 \\ f_4(x) &= \|Ax - y\|_2^2 + \|x\|_1^2 \\ f_5(x) &= \sum_{i=1}^k |Ax - y|_{[i]} + \|x\|_2^2 \end{aligned}$$

Table 2: Table of the values of different functions f . $|z|_{[i]}$ denotes the element in a vector z that has the i -th largest magnitude.

Exercise 4 (A slalom problem)



A two-dimensional skier must slalom down a slope, by going through n parallel gates of known position (x_i, y_i) , and of width C_i , $i = 1, \dots, n$. The initial position (x_0, y_0) is given, as well as the final one, (x_n+1, y_n+1) . Here, the x-axis represents the direction down the slope, from left to right.

i	x_i	y_i	c_i
0	0	4	N/A
1	4	5	3
2	8	4	2
3	12	6	2
4	16	5	1
5	20	7	2
6	24	4	N/A

Table 3: parameters for the slalom problem.

1. Find the path that minimizes the total length of the path. Your answer should come in the form of an optimization problem.
2. Try solving the problem numerically, with the data given in Table 3.

Exercise 5 (LS with uncertain A matrix)

Consider a linear least-squares problem where the matrix involved is random. Precisely, the residual vector is of the form $A(\delta)x - b$ where the $m \times n$ A matrix is affected by stochastic uncertainty. In particular, assume that

$$A(\delta) = A_0 + \sum_{i=1}^p A_i \delta_i,$$

where the random variables δ_i , $i = 1, \dots, p$ are independent, with zero mean and variance σ_i^2 , $i = 1, \dots, p$. The standard least-squares objective function

$$\delta \rightarrow \|A(\delta)x - b\|_2^2$$

is now random, since it depends on δ . We seek to determine x such that the expected value (with respect to the random variable δ) is minimized. Is such a problem convex? If yes, to which class does it belong to (LP, LS, QP, etc.)?

Standard Forms of SOCP [SOCOP > Second-order cone](#) | Standard forms | [Group sparsity](#) | [Applications](#)

- Second-order cone programs: standard forms
- Quadratically constrained quadratic programming

Second-order cone programs: standard forms inequality form. A *second-order cone program* (or SOCP, for short) is an optimization problem of the form

$$\min_x c^T x : \|A_i x + b_i\|_2 \leq c_i^T x + d_i, \quad i = 1, \dots, m,$$

where $A_i \in \mathbf{R}^{p_i \times n}$ are given matrices, $b_i \in \mathbf{R}^{p_i}$, $c_i \in \mathbf{R}^n$ vectors, and d_i 's scalars.

The problem is convex, since the constraint functions of the corresponding [standard form](#)

$$f_i(x) = \|A_i x + b_i\|_2 - (c_i^T x + d_i), \quad i = 1, \dots, m,$$

are.

Examples:

- [Linear program as SOCP](#).
- [Facility location](#).
- [Robust least-squares](#).
- [Separation of ellipsoids](#).

$$\min_x c^T x : (A_i x + b_i, c_i^T x + d_i) \in \mathbf{K}_{p_i}, \quad i = 1, \dots, m.$$

Conic form. We can put the above problem in the so-called “conic” format

Since the cones \mathbf{K}_{p_i} are convex, and the mappings $x \rightarrow (A_i x + b_i, c_i^T x + d_i)$ are affine, the feasible set is convex.

Rotated second-order cone constraints. Since the rotated second-order cone can be expressed as some linear transformation of an ordinary second-order cone, we can include rotated second-order cone constraints, as well as ordinary linear inequalities or equalities, in the formulation. This allows to formulate LPs and QPs as special cases of SOCP.

Examples:

- [Quadratic program as SOCP](#).
- Logarithmic Chebyschev approximation

Quadratically constrained quadratic programming Definition. A quadratically constrained quadratic programming (QCQP for short) is a problem of the form

$$\min_x a_0^T x + x^T Q_0 x : x^T Q_i x + a_i^T x \leq b_i, \quad i = 1, \dots, m,$$

where $Q_i \succeq 0$, $i = 1, \dots, m$. This condition ensures that the problem is convex. QCQPs contain LPs and QPs as special case.

Examples:

SOCOP formulation. We can formulate QCQPs as SOCPs, by introducing new variables and affine equality constraints. ([Proof](#)).

Group sparsity and the l_1/l_2 -norm trick [SOCOP > Second-order cone](#) | Standard forms | Group sparsity | Applications

- Group sparsity problem
- The l_1/l_2 -norm heuristic

Group sparsity problem

In many applications, such as [image annotation](#), we would like to encourage the occurrences of whole blocks of zeros in the decision vector. Assume for example that the decision vector is partitioned into two disjoint groups $x = (x_1, \dots, x_k)$, with $x_i \in \mathbf{R}^{n_i}$, $i = 1, \dots, k$, with $n = n_1 + \dots + n_k$. We seek to minimize the number of non-zero blocks x_i , while satisfying some constraint, say $x \in \mathbf{P}$, with \mathbf{P} a given polytope.

The l_1/l_2 -norm heuristic

We can use the [\$l_1\$ -norm trick](#) on the vector of Euclidean norms $(\|x_1\|_2, \dots, \|x_k\|_2)$. (This will encourage many of the blocks x_i to be zero; those blocks that are not zero will not, in general, be sparse.)

This leads to the problem

$$\min_x \sum_{i=1}^k \|x_i\|_2 : x \in \mathbf{P}.$$

The above is a second-order cone program. This is easily seen after introducing a new vector variable

$$\min_{x,y} \sum_{i=1}^k y_i : x \in \mathbf{P}, y_i \geq \|x_i\|_2, i = 1, \dots, p.$$

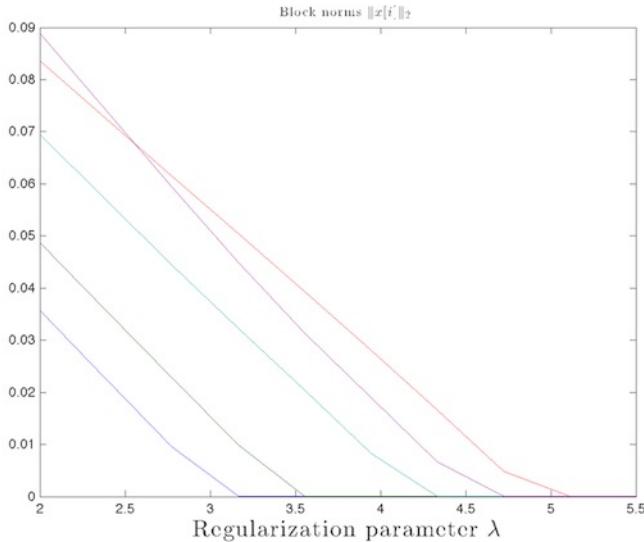
The following CVX snippets actually uses the above representation for the least-squares problem with a block-norm penalty:

$$\min_x \|Ax - b\|_2 + \sum_{i=1}^k \|x_i\|_2.$$

In our implementation we assume that each block x_i is of the same length k .

CVX syntax: least-squares with group sparsity

```
cvx_begin
    variable x(n);
    variable y(k);
    minimize( norm(A*x-b,2) + lambda*sum( y ) )
    subject to
        for i=1:k,
            y(i) >= norm(x((i-1)*p+1:p),2);
        end
    cvx_end
```



In this example, we solve the problem for a random instance of the problem (with $n = 100, m = 1000, k = 5$), and various values of the parameter λ . As λ grows, we observe that the block norms go to zero one by one. Thus the parameter allows to control the sparsity of the block norms.

Applications of SOCP [SOCP > Second-order cone | Standard forms | Group sparsity | Applications](#)

- [Minimum surface area](#).
- [Total variation image restoration](#).
- [Antenna array design](#).
- [Image annotation via group sparsity](#).
- [Sensor network localization](#).

Exercises [SOCP > Exercises](#) Second-order cones A. A single second-order cone constraint. Draw the set of points $x \in \mathbf{R}^2$ such that

$$\rho\|x\|_2 \leq 2x_1 + x_2 - 3,$$

where $\rho = 0.3, 1, 3, 5$. In general, for which values of ρ , if any, is this set an ellipsoid? When is it not empty?

B. Squaring and second-order cones. When considering a second-order cone constraint, a temptation might be to square it in order to obtain a classical convex quadratic constraint. This might not always work. Consider the constraint

$$x_1 + 2x_2 \geq \|x\|_2,$$

and its squared counterpart:

$$(x_1 + 2x_2)^2 \geq \|x\|_2^2.$$

Is the set defined by the second inequality convex? Discuss.

C. A robust constraint. Let $\hat{u} \in \mathbf{R}^n$ and $v \in \mathbf{R}$, $\rho \geq 0$ be given. Consider the following condition on a given $x \in \mathbf{R}^n$:

$u^T x \leq v$ for every $u \in \mathbf{R}^n$ such that $\|u - \hat{u}\|_2 \leq \rho$.

1. Show that this condition is equivalent to a second-order cone condition on \mathcal{X} .
2. What is the geometrical interpretation of this condition?
3. Plot the set of $x \in \mathbf{R}^2$ that satisfy the condition with $\hat{u} = (2, 1)$, $v = 3$, and $\rho = 2$.

Standard forms **D. A complicated function.** We would like to minimize the function $f : \mathbf{R}^3 \rightarrow \mathbf{R}$, with values:

$$f(x) = \max \left(x_1 + x_2 - \min(\min(x_1 + 2, x_2 + 2x_1 - 5), x_3 - 6), \frac{(x_1 - x_3)^2 + 2x_2^2}{1 - x_1} \right),$$

with the constraint $x_1 < 1$.

1. Explain precisely how to formulate the problem as an SOCP in standard form. Hint: Remember that $\forall z \in \mathbf{R}^n$, for all $a, b \in \mathbf{R}$ non negative, the constraint $\|z\|_2^2 \leq ab$, $a \geq 0$, $b \geq 0$ is equivalent to a second order cone constraint.
2. Write a CVX code that solves the problem. Hint: use the function `quad_over_lin.m`.

E. SOCP and QCQP. Quadratically constrained quadratic programs are minimization problems involving convex quadratic objectives and constraints. [This section](#) shows that QCQP's can be expressed as SOCP's. Is the converse true, in other words, can we always write an SOCP as a QCQP? Discuss.

F. Robust least-squares. For a given matrix $A \in \mathbf{R}^{m \times n}$ and a vector $b \in \mathbf{R}^m$, we consider the robust least-squares problem

$$\min_x \max_{\Delta : \|\Delta\| \leq \rho} \|(A + \Delta)x - b\|_2,$$

where $\|\Delta\|$ denotes the largest singular value of the matrix Δ , and $\rho \geq 0$ is given. This problem arises when we seek to solve a least-squares problem in which the matrix A is subject to an additive perturbation, and we would like to find a solution that minimizes the worst-case residual error.

1. Prove that for any given vectors $x, z \in \mathbf{R}^n$, we have $\|\Delta x + z\|_2 \leq \rho\|x\|_2 + \|z\|_2$ whenever $\|\Delta\| \leq \rho$.
2. Show that the bound you found in part (a) is attained, by exhibiting a matrix Δ that achieves the bound. Hint: use a rank-one matrix constructed with x, z .
3. Formulate the robust least-squares problem as an SOCP in standard form.
4. Solve the problem in the case when the bound on the perturbation Δ is a component-wise bound, of the form $\max_{i,j} |\Delta_{ij}| \leq \rho$.

Applications

G. k -Ellipses. Consider k points x_1, \dots, x_k in \mathbf{R}^2 . For a given positive number d , we define the k -ellipse with radius d as the set of points $x \in \mathbf{R}^2$ such that the sum of the distances from x to the points x_i is equal to d .

1. How do k -ellipses look like when $k = 1$ or $k = 2$?
2. Express the problem of computing the *geometric median*, which is the point that minimizes the sum of the distances to the points x_i , $i = 1, \dots, k$, as an SOCP in standard form.
3. Using CVX, write a code with input $X = (x_1, \dots, x_k) \in \mathbf{R}^{2 \times k}$ that plots the corresponding k -ellipse.

H. Sparse Classification for word imaging. The image of a given query word in a given corpus of text news can be defined as a short list of other words with which this query is strongly associated. To be easily understandable, the list should be extremely short with respect to the number of terms present in the corpus.

One way to obtain a word image is to use l_1 -norm penalization in a classification algorithm, where indicator of the query word's appearance in each headline is used as that headline's label/response ($y \in \{1, -1\}^m$), and the indicators for all other words are used as predictors/features ($X = [x_1, \dots, x_m] \in \mathbf{R}^{n \times m}$). The few features (words) that are then predictive of the appearance of the query term correspond to that query's image in the corpus.

As explained [here](#), we consider the following optimization problem, also called l_1 -norm Support Vector Machine (L1-SVM):

$$\min_{w \in \mathbf{R}^n, b \in \mathbf{R}} \sum_{i=1}^m [1 - y_i(w^T x_i + b)]_+ + \lambda \|w\|_1$$

where $z_+ := \max(0, z)$. By imposing a sparsity constraint on the weight vector, we can single out the few words that are most able to predict the presence or absence of a query word in any document. These selected words are then considered the list of words comprising the query word's image.

We look at the Word Imaging problem in a small-scale setting. Our original data is the headlines of New York Times between Jan 1 2006 and Dec 31 2006: there are 84612 headlines and 160,624 distinct words in total. To make the problem accessible to you, we preprocessed the text data and down-sampled (with special care) both the number of headlines and that of distinct words to 997 and 1045 respectively. We try to obtain the image of the query word "Microsoft". The label y , the predictor X and the dictionary of terms dict are defined in sparseSVM.mat. Note that y_j indicates whether "Microsoft" shows up in the j -th headline, X_{ij} indicates whether j -th headline contains i -th word, and $dict_i$ is the i -th word.

1. Show how to formulate L1-SVM as a linear program.
2. Using CVX, write a MATLAB program to solve the problem of minimizing the approximate loss for the given data. What are the top 20 words (i.e. top 20 features with highest coefficients) that predict the presence of the query word "Microsoft"? Does the result make sense to you? Experiment with $\lambda \in \{0.1, 0.5, 1, 5, 10\}$ and see how the list of top words changes.
3. Consider now the Support Vector Machine setup where L2 regularization is used (call it the L2-SVM)

$$\min_{w \in \mathbf{R}^n, b \in \mathbf{R}} \sum_{i=1}^m [1 - y_i(w^T x_i + b)]_+ + \lambda \|w\|_2$$

Show how to formulate the above as an SOCP.

5. Follow the procedure from part 2 to solve this problem using the same lambdas. Discuss your results.
6. Compare the top 20 features extracted by the L1-SVM and the L2-SVM for each lambda. How do they compare across the lambdas? Which formulation makes more sense to you? Discuss.

- The uncertainty problem
- Robust optimization approach

$$\min_x c^T x : a_i^T x \leq b_i, \quad i = 1, \dots, m$$

The uncertainty problem Data uncertainty. Consider the [linear program](#) in inequality form:

In practice, the data of the linear program (contained in the vectors c, b , and $a_i, i = 1, \dots, m$) are often not well-known. Solving the LP without taking into account the uncertainty in the problem's data might make the supposedly "optimal" solution very sub-optimal, or even infeasible.

Example: [Uncertainty in the drug production problem](#).

Implementation errors. Uncertainty can originate also from *implementation errors*. Often, the optimal variable x^* corresponds to a certain action or implementation process, which may be fraught with errors.

In an [antenna array design](#) problem for example, the optimal antenna weights are in reality characteristics of certain physical devices and as such cannot be implemented exactly as they are computed via the optimization model. Or, in a manufacturing process, the planned production amounts are never exactly implemented due to, say, production plant failures.

Implementation errors can result in catastrophic behavior, in the sense that when the optimal variable x is impacted by such errors, the constraints might become inactive, or the cost function become much worse (higher) than thought.

Example: [Antenna array design with relative implementation error](#).

Robust optimization approach Main idea. In robust optimization, we assume that the data in the LP is not exactly known. We postulate that a *model of uncertainty* is known.

In its simplest version, we assume that the a_i 's are known to belong to a given set $\mathbf{U}_i \subseteq \mathbf{R}^n$. We can think of those sets as *sets of confidence* for the coefficients of the linear program above.

$$\min_x c^T x : \forall a_i \in \mathbf{U}_i, \quad a_i^T x \leq b_i, \quad i = 1, \dots, m.$$

Robust counterpart. The *robust counterpart* to the original LP above is defined as follows.

The interpretation of the above problem is that it attempts to find a solution that is feasible, *independent* of the particular choice of the coefficient vectors a_i within their respective sets of confidence.

The robust counterpart is always convex, independent of the shape of the sets of confidence \mathbf{U}_i . (Indeed, the feasible set is still the intersection of half-spaces; in contrast with the original LP, their number is can be infinite, unless \mathbf{U}_i are finite.)

For some classes of uncertainty sets \mathbf{U}_i , the robust counterpart is computationally tractable. We detail [here](#) three such cases, which are important in applications.

Robust half-space constraints. The feasible set of the robust counterpart is the intersection of single constraints called *robust half-space constraints*.

Given a subset \mathbf{U} of \mathbf{R}^n , and $b \in \mathbf{R}$, the condition on x :

$$\forall a \in \mathbf{U}, \quad a^T x \leq b,$$

is called a *robust half-space constraint*. The condition is always convex, irrespective of the choice of the set \mathbf{U} .

To analyze a robust half-space constraint, the main idea is to formulate it in terms of an optimization problem, as

$$b \geq \max_{a \in \mathbf{U}} a^T x.$$

For some choices of \mathbf{U} , we can solve the maximization problem efficiently. We explore this [next](#).

Exercises [Robust LP](#) > Exercises

A. We consider the binary classification problem discussed [here](#). When the data is not separable, the following optimization problem is solved:

$$\min_{a,b} \sum_{i=1}^m (1 - y_i(a^T x_i - b))_+,$$

where the notation $\alpha_+ := \max(\alpha, 0)$ denotes the positive part of a real number α . The above can be formulated as an LP:

$$\min_{a,b,v} \sum_{i=1}^m v_i : v_i \geq 0, \quad y_i(a^T x_i - b) \geq 1 - v_i, \quad i = 1, \dots, m.$$

Now consider a robust version of the first problem:

$$\min_{a,b} \max_{X \in \mathcal{X}} \sum_{i=1}^m (1 - y_i(a^T x_i - b))_+,$$

where the data matrix $X = [x_1, \dots, x_m]$ is now uncertain, and only known to belong to the set

$$\mathcal{X} := \{[x_1, \dots, x_m] : \|x_i - \bar{x}_i\|_2 \leq \rho, \quad i = 1, \dots, m\}.$$

In the above, $\rho \geq 0$ is a given measure of spherical uncertainty on the columns, with $\bar{x}_i, i = 1, \dots, m$ the nominal values of the data points.

1. Formulate the robust counterpart as an SOCP.
2. Is the problem you have obtained the same as the robust counterpart to the LP formulation above? Discuss.

B. Apply the AARC approach for the [drug production problem](#), with the uncertainty model given [here](#). The motivation is that the uncertainty on the active agent in the raw material is assumed to be revealed after having decided on the production levels. We seek to avoid resolving the LP each time we get a new value of those parameters.

Compare the following approaches:

1. The nominal problem.
2. The robust counterpart.
3. A simple adjustment policy, as described at the end of this [section](#).
4. The full affinely adjustable robust counterpart model.

Posynomials [GP](#) > Posynomials | [Standard Forms](#) | [Applications](#)

- Monomials
- Posynomials and generalized posynomials
- Convex representation via the log-sum-exp function

Monomials Definition. A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is a *monomial* if its domain is \mathbf{R}_{++} (the set of vectors with positive components) and its values take the form $f(x) = cx_1^{a_1} \dots x_n^{a_n}$,

where $c > 0$ and $a \in \mathbf{R}^n$. We can write in short-hand notation:

$$f(x) = cx^a,$$

where we follow the *power law notation*: by convention, for two vectors $x, a \in \mathbf{R}^n$, x^a is the product $x_1^{a_1} \dots x_n^{a_n}$.

Example. The function with domain $\{x \in \mathbf{R}^2 : x_1 > 0, x_2 > 0\}$ and values $f(x) = 3x_1^{1.2}x_2^{-0.003}$ is a monomial, while $-f$ is not.

Log-linearity and power laws. Monomials are closely related to linear or affine functions: indeed, if f is a monomial in variable x , then $\log f$ is affine in the vector $\log x := (\log x_1, \dots, \log x_n)$. Hence monomial functions could be called ‘log-linear’, although the term is not widely used.

Just as linear models are important in (approximate) models between general variables, monomials play an ubiquitous role for modeling relationships between *positive* variables, such as prices, concentrations, energy, or geometric data such as length, area and volume, etc. Like their linear counterpart, power laws can be [easily fitted](#) to experimental data, via least-squares methods.

Examples:

- The *Stephan law* of physics states that the power radiated by a body evolves as AT^4 , where A is the surface area of the body, and T the temperature; this is a monomial in variables A, T .
- In industrial engineering, the *experience curve* relates the cost C_1 of producing a unit for the first time, to the cost of producing the unit for the n -th time, as $C_n = C_1 n^{-a}$, where n is the cumulative number of unit produced, and exponent a depends on the industry. Thus C is a monomial in C_1 and n .

Posynomials

A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is a *posynomial* if its domain is \mathbf{R}_{++} (the set of vectors with positive components) and its values take the form of a *non-negative sum* of monomials:

$$f(x) = \sum_{k=1}^K c_k f_k(x),$$

where $c_k \geq 0, k = 1, \dots, K$, and each f_k is a monomial.

The values of a posynomial can be always written as

$$f(x) = \sum_{k=1}^K c_k x_1^{a_{k1}} \dots x_n^{a_{kn}},$$

for some $c \in \mathbf{R}_+^K$, and $A = (a_{kj}) \in \mathbf{R}^{K \times n}$. If we denote by $a_k, k = 1, \dots, K$ the k -th row of A , then we can write in short-hand notation

$$f(x) = \sum_{k=1}^K c_k x^{a_k},$$

where we follow the above power law notation to define what x^a means when x, a are two vectors of the same size.

Examples:

- [Construction and operating costs for a water tank](#).
- [Signal-to-noise ratios in wireless communications](#).

Generalized Posynomials. A *generalized posynomial* function is any function obtained from posynomials using addition, multiplication, pointwise maximum, and raising to constant positive power. For example, the function $f : \mathbf{R}_{++}^3 \rightarrow \mathbf{R}$ with values $\max(2x_1^{2.3}x_2^7 + 4x_2, x_1x_2x_3^{3.14}, \sqrt{x_1 + x_3^3})$

is a generalized posynomial.

Convex Representation

Monomials and (generalized) posynomials are not convex. However, with a simple transformation of the variables, we can transform them into convex ones.

Convex representation of posynomials. Consider a posynomial function f . Instead of the original (positive) variables, we use the new variable $y_i = \log x_i, i = 1, \dots, n$. We then take the

logarithm of the function f . Let us look at the effect of such transformations on monomials and posynomials.

- For a monomial f with values $f(x) = cx_1^{a_1} \dots x_n^{a_n}$, where $a \in \mathbf{R}^n$ and $c > 0$, we have

$$\log f(x) = a^T y + b,$$

where $y_i = \log x_i, i = 1, \dots, n$, and $b = \log c$. Our transformation yields an affine function.

- For a posynomial f with values

$$f(x) = \sum_{k=1}^K c_k x^{a_k},$$

where $c > 0$, we have

$$\log f(x) = \log \left(\sum_{k=1}^K e^{a_k^T y + b_k} \right),$$

where $b_k := \log c_k, k = 1, \dots, K$. The above can be written

$$\log f(x) = \text{lse}(Ay + b),$$

where A is the $K \times n$ matrix with rows $a_1, \dots, a_K, b \in \mathbf{R}^K$, and lse is the [log-sum-exp function](#). Recall that this function is [convex](#).

Thus, we can view a posynomial as the log-sum-exp function of an affine combination of the logarithm of the original variables. Since the lse function is convex, this transformation will allow us to use convex optimization to optimize posynomials.

Remark: [Why do we take the log?](#)

Convex representation of generalized posynomials. Adding variables, and with the logarithmic change of variables seen above, we can also transform generalized posynomial inequalities into convex ones.

For example, consider the posynomial $f : \mathbf{R}_{++}^n \rightarrow \mathbf{R}$ with values

$$f(x) = \max(f_1(x), f_2(x)),$$

where f_1, f_2 are two posynomials. For $t > 0$, the constraint

$$f(x) \leq t$$

can be expressed as *two* posynomial constraints in (x, t) .

Likewise, for $t > 0$, consider the power constraint

$$f_1(x)^\alpha \leq t,$$

with f an ordinary posynomial and $\alpha > 0$. Since $\alpha > 0$, the above is equivalent to

$$f(x) \leq t^{1/\alpha},$$

which in turn is equivalent to the posynomial constraint

$$g(x, t) := t^{-1/\alpha} f(x) \leq 1.$$

Hence, by adding as many variables as necessary, we can express a generalized posynomial constraint as a set of ordinary ones.

Standard Forms of GP [GP](#) > [Posynomials](#) | Standard Forms | [Applications](#)

- Geometric Programs
- Convex Form
- CVX Syntax

Geometric Programs

A *geometric program* (GP for short) is a problem with generalized posynomial objective and inequality constraints, and (possibly) monomial equality constraints.

In standard form, a GP can be written as

$$\min_x f_0(x) : f_i(x) \leq 1, \quad i = 1, \dots, m, \quad h_j(x) = 1, \quad j = 1, \dots, p,$$

where f_0, \dots, f_m are generalized posynomials and h_1, \dots, h_p are monomials.

Assuming for simplicity that the generalized posynomials involved are ordinary posynomials, we can express a GP explicitly, in the so-called standard form:

$$\min_x \sum_{k=1}^{K_0} c_{k0} x^{a_0} : \quad \begin{aligned} & \sum_{k=1}^{K_i} c_{ki} x^{a_{ki}} \leq 1, \quad i = 1, \dots, m, \\ & g_j x^{f_j} = 1, \quad j = 1, \dots, p. \end{aligned}$$

where a_0, \dots, a_m and c_1, \dots, c_m are vectors in \mathbf{R}^n , and $c_i > 0, k = 1, \dots, m, g > 0$ are vectors with positive components. In the above, we follow the [power law notation](#).

Converting the problem into the standard form when the original formulation involves generalized posynomials entails adding new variables and constraints (see [here](#)).

Example: Optimization of a water tank.

Convex Form GPs in standard form are not convex, but a simple transformation allows to put them into an equivalent, convex form. The idea, already seen in the context of [posynomial functions](#), is to take logarithms of the original variables. Again we assume that the posynomials involved are not generalized posynomials, but ordinary ones.

Consider a posynomial constraint of the form

$$f(x) := \sum_{k=1}^K c_k x^{a_k} \leq 1.$$

where the vector c has positive components: $c > 0$. We can express this constraint in terms of the new variable $y \in \mathbf{R}^n$, defined as $y_i = \log x_i, i = 1, \dots, n$.

$$\log f(x) = \text{lse}(Ay + b) := \log \left(\sum_{k=1}^K e^{a_k^T y + b_k} \right) \leq 0,$$

where $b_k := \log c_k, k = 1, \dots, K$, A is the $K \times n$ matrix with rows a_1, \dots, a_K , and **lse** is the [log-sum-exp function](#).

Note that if the posynomial was actually a monomial, there would be only one term in the **lse** function, and the constraint would reduce to an ordinary affine inequality.

Since the log-sum-exp function is [convex](#), the above constraint is actually convex in the new variable y .

By this argument, the GP

$$\min_x \sum_{k=1}^{K_0} c_{k0} x^{a_{k0}} : \begin{aligned} \sum_{k=1}^{K_i} c_{ki} x^{a_{ki}} &\leq 1, \quad i = 1, \dots, m, \\ g_j x^{f_j} &= 1, \quad j = 1, \dots, p. \end{aligned}$$

can be expressed as a *convex* problem

$$\min_x \text{lse}(A_0 y + b_0) : \begin{aligned} \text{lse}(A_i y + b_i) &\leq 0, \quad i = 1, \dots, m, \\ Fy + g &= 0, \end{aligned}$$

for appropriate matrices A_0, \dots, A_m, C and vectors b_0, \dots, b_m, d . Precisely, we set $b_i = \log c_i$, and A_i to be the matrices with rows $a_{ki}, k = 1, \dots, K_i, i = 0, \dots, m$; the matrix F contains the vectors f_j as rows, $j = 1, \dots, p$.

Example: Convex form of the water tank optimization problem.

CVX Syntax

With the above convex representation of a GP, We can directly invoke CVX with the `logsumexp_sdp` function. CVX knows it is convex. The extension “`sdp`” at the end of the function refers to the fact that the function is actually approximated via a technique based on [semidefinite programming](#) (SDP).

In CVX there is actually no need to transform the problem into the standard convex format. All you need to do is let CVX know that the problem is a GP. This is done by replacing the command `cvx_begin` with `cvx_begin gp`. This is illustrated [here](#).

Some Applications of GP [GP](#) > [Posynomials](#) | [Standard Forms](#) | Applications

- [Digital circuit design](#).

Exercises [Geometric Programming](#) > Exercises

A. We seek to optimize the shape of a box-shaped structure with height h , width w , and depth d . Due to cost concerns, we have an upper limit A_{wall} on the total wall area $2(hw + hd)$, another upper limit A_{floor} on the floor area wd . For structural reasons, we have lower and upper bounds α, β on the aspect ratio h/w , and similar bounds γ, δ on the aspect ratio w/d . Subject to these constraints, we wish to maximize the volume of the structure, hwd .

1. Express the problem in the standard form for an optimization problem.
2. Express this problem as a geometric program (GP) in standard GP form. (*Note:* you are not asked to put it in the standard convex form for GPs.)

From LP to Conic Optimization [SDP](#) > Conic Problems | [LMIs](#) | [Standard Forms](#) | Applications

In the late 1980s, researchers were trying to generalize linear programming. At that time, LP was known to be solvable efficiently. Theory even showed that LPs could be solved in time roughly cubic in the number of variables or constraints. The new interior-point methods for LP had just become available, and their excellent practical performance matched the theoretical complexity bounds. It seemed however that, beyond linear problems, one encountered a wall. Except for a few special problem classes, such as QP, it appeared that as soon as a problem contained non-linear terms, one would no longer hope for the nice practical and theoretical efficiency allowed to LP.

In previous decades, it had been noted that convex problems could be efficiently solved in theory (under some mild conditions), however the known methods were extremely slow in practice. It seemed however that to harness the power of interior-point methods and apply it to problems other than LP, one had to look closely at convex optimization.

A breakthrough occurred by rethinking the role of the set of non-negative vectors, which is the basic object in LP. In a standard form, an LP can be written as

$$\min_x c^T x : Ax = b, \quad x \in \mathbf{R}_+^n,$$

where \mathbf{R}_+^n is the set of non-negative vectors. Researchers asked: “what are the basic characteristics of \mathbf{R}_+^n that make interior-point methods work so well? In other words, are there other sets that could be used in place of \mathbf{R}_+^n , and still allow efficient methods”? It turns out that it is always possible to restrict attention to cones, that is, sets that are invariant by positive scaling of their elements.

At first glance, one can replace the set \mathbf{R}_+^n by any convex cone, say \mathbf{K} ; the new problem

$$\min_x c^T x : Ax = b, \quad x \in \mathbf{K}$$

is convex. But, as mentioned above, convexity alone is not sufficient to guarantee the existence of a practically efficient algorithm to solve it. It turns out that there are a few convex cones \mathbf{K} that are amenable to an efficient extension of interior-point methods.

One of the second-order cone, or any combination of second-order cones (arising when, say some variables are in a cone, others in another, and all are coupled by affine equalities). The SOCP class was the first beneficiary of the new class of algorithms.

Another class of problems involve the cone of positive semi-definite matrices, which leads to the class of semi-definite programs (SDP). SDPs involve a linear objective, matrix variables, and affine equality constraints, and positive semi-definiteness constraints on the (matrix) variables.

For both SOCP and SDP, the interior-point methods that work so well for LP can be extended. This comes at a price: the size of SOCPs that can be solved in reasonable time is smaller than for LP, and the complexity of SDPs is even higher. What we gain in the process is a much wider expressive power of the model.

Linear Matrix Inequalities [SDP > Conic problems](#) | LMIs | Standard Forms | Applications

- Definition
- LMIs and convex sets
- Special cases

Definition Positive Semi-Definite Matrices. Recall from [here](#) that a $n \times n$ symmetric matrix F is *positive semi-definite* (PSD) if and only if every one of its eigenvalues is non-negative. We use the notation $F \succeq 0$ to mean that F is PSD.

An alternative condition for F to be PSD is that the associated quadratic form is non-negative:

$$\forall z \in \mathbf{R}^n : z^T F z \geq 0.$$

The set of PSD matrices is convex, since the conditions above represent (an infinite number of) ordinary linear inequalities on the elements of the matrix F .

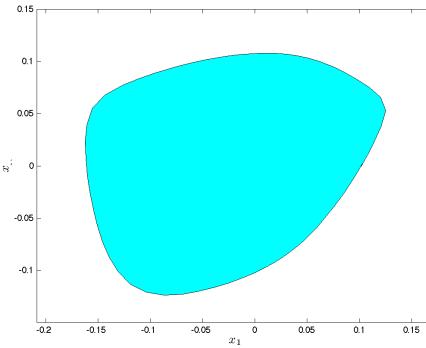
Examples:

- [A simple \$3 \times 3\$ example](#).
- For any vector $v \in \mathbf{R}^n$, the dyad $F = vv^T$ is PSD, since the associated quadratic form is
$$q(x) = x^T(vv^T)x = (v^T x)^2 \geq 0.$$
- More generally, for any rectangular matrix A , the “square” matrix $F = A^T A$ is PSD.
- The converse is true: any PSD matrix can be factored as $A^T A$ for some appropriate matrix A .

$$F_0 + \sum_{i=1}^m x_i F_i \succeq 0,$$

Standard form. A *linear matrix inequality* is a constraint of the form

where the $n \times n$ matrices F_0, \dots, F_m are symmetric.



An LMI in two variables:

$$F(x) := x_1 F_1 + x_2 F_2 \preceq I,$$

where F_1, F_2 are the two 5×5 symmetric matrices

$$F_1 = \begin{pmatrix} -1.3 & -4.2 & -0.1 & 2.1 & -1 \\ -4.2 & -0.1 & -1.7 & -4.5 & 0.9 \\ -0.1 & -1.7 & 2.3 & -4.4 & -0.4 \\ 2.1 & -4.5 & -4.4 & 3.3 & -1.7 \\ -1. & 00.9 & -0.4 & -1.7 & 4.7 \end{pmatrix},$$

$$F_2 = \begin{pmatrix} 1.6 & 3.9 & 1.6 & -5.3 & -4. \\ 3.9 & -1.8 & -4.7 & 1 & 02.9 \\ 1.6 & -4.7 & -1.3 & 1.6 & -2.6 \\ -5.3 & 1 & 01.6 & 2.7 & 2.6 \\ -4. & 02.9 & -2.6 & 2.6 & -3.4 \end{pmatrix}.$$

The matrices F_0, \dots, F_m are referred to as the coefficient matrices. Sometimes, these matrices are not explicitly defined. That is, if $F : \mathbf{R}^m \rightarrow \mathbf{S}^n$ is an affine map that takes its values in the set of symmetric matrices of order n , then $F(x) \succeq 0$ is an LMI.

An alternate form for LMIs is as the intersection of the positive semi-definite cone with an affine set:

$$X \in \mathcal{A}, \quad X \succeq 0,$$

where \mathcal{A} is affine. The form we have seen before, and the above one, are equivalent, in the sense that we can always transform one into the other (at the expense possibly of adding new variables and constraints).

LMIs and Convex Sets

Let us denote by \mathbf{X} the set of points $x \in \mathbf{R}^m$ that satisfy the above LMI.

$$\mathbf{X} := \left\{ x : F_0 + \sum_{i=1}^m x_i F_i \succeq 0 \right\}.$$

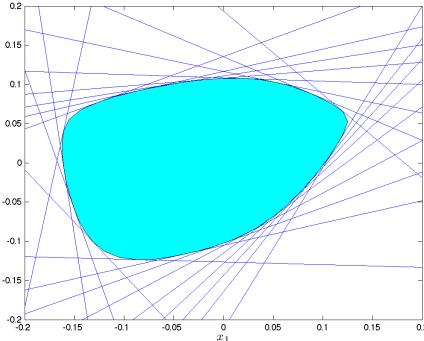
The set \mathbf{X} is convex. Indeed, we have $F(x) \succeq 0$ if and only if

$$\forall z \in \mathbf{R}^n : z^T F(x) z \geq 0.$$

Since

$$z^T F(x) z = f_0(z) + \sum_{i=1}^m x_i f_i(z)$$

with $f_i(z) := z^T F_i z$, $i = 0, \dots, m$, we obtain that the condition on x : $F(x) \succeq 0$ can be represented as an intersection of (an infinite number of) half-space conditions.



Supporting hyperplanes: The picture shows how the set seen above can be interpreted as the intersection of a number of half-spaces. Each half-space corresponds to an affine inequality of the form $z^T(I - F(x))z \geq 0$, for a specific vector z . The vectors z are chosen so that the line touches the boundary of the LMI set, so that it defines a [supporting hyperplane](#).

Multiple LMIs. We can combine multiple LMIs into one. Consider two affine maps from \mathbf{R} to a space of symmetric matrices of order n_1, n_2 , $F_1 : \mathbf{R}^m \rightarrow \mathbf{S}^{n_1}, F_2 : \mathbf{R}^m \rightarrow \mathbf{S}^{n_2}$. Then the two LMIs $F_1(x) \succeq 0, F_2(x) \succeq 0$

are equivalent to *one* LMI, involving a larger matrix of size $(n_1 + n_2) \times (n_1 + n_2)$:

$$\begin{pmatrix} F_1(x) & 0 \\ 0 & F_2(x) \end{pmatrix} \succeq 0.$$

This corresponds to intersecting the two LMI sets.

Special Cases

LMIs include as special cases the following.

Ordinary affine inequalities. Consider single affine inequality in $x \in \mathbf{R}^n$: $a^T x \leq b$,

where $a \in \mathbf{R}^n, b \in \mathbf{R}$. (The above set describes a half-space.) The above is a trivial special case of LMI, where the coefficient matrices are scalar: $F_0 = b, F_i = -a_i, i = 1, \dots, n$.

Using the result above on multiple LMIs, we obtain that the set of ordinary affine inequalities

$$a_i^T x \leq b_i, \quad i = 1, \dots, m$$

can be cast as a single LMI $F(x) \succeq 0$, where

$$F(x) = \begin{pmatrix} b_1 - a_1^T x & & \\ & \ddots & \\ & & b_m - a_m^T x \end{pmatrix}.$$

Second-order cone inequalities. [Second-order cone \(SOC\) inequalities](#) can be represented as LMIs. To see this, let us start with the “basic”

$$\begin{pmatrix} t & y^T \\ y & tI_n \end{pmatrix} = \begin{pmatrix} t & y_1 & \dots & y_n \\ y_1 & t & & \\ \vdots & & \ddots & \\ y_n & & & t \end{pmatrix} \succeq 0.$$

SOC $\|y\|_2 \leq t$, with $y \in \mathbf{R}^n$ and $t \in \mathbf{R}$. The SOC can be represented as

Indeed, we check that for every $z \in \mathbf{R}^n, \alpha \in \mathbf{R}$, we have

$$\begin{pmatrix} \alpha \\ z \end{pmatrix}^T \begin{pmatrix} t & y^T \\ y & tI_n \end{pmatrix} \begin{pmatrix} \alpha \\ z \end{pmatrix} = \|tz + \alpha y\|_2^2 + \alpha^2(t - \|y\|_2^2) \geq 0$$

if and only if $\|y\|_2 \leq t$.

More generally, a second-order cone inequality of the form

$$\|Ax + b\|_2 \leq c^T x + d,$$

with $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^m, c \in \mathbf{R}^n, d \in \mathbf{R}^n$, can be written as the LMI

$$\begin{pmatrix} c^T x + d & (Ax + b)^T \\ (Ax + b) & (c^T x + d)I_n \end{pmatrix} \succeq 0.$$

The proof relies on the [Schur complement lemma](#).

- Standard inequality form
- Standard conic form
- CVX syntax

Standard Inequality Form

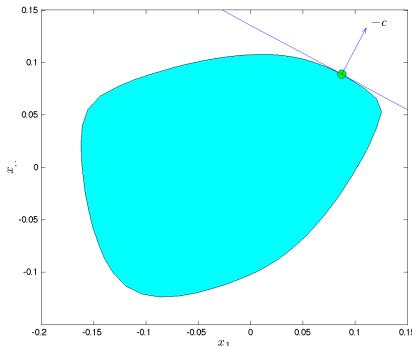
A *semidefinite program* (SDP) is a problem of minimizing a linear function over an LMI constraint. In standard inequality form, an SDP is written as

$$\min_x c^T x : F(x) := F_0 + x_1 F_1 + \dots + x_m F_m \succeq 0,$$

where F_0, \dots, F_m are given symmetric matrix, $c \in \mathbf{R}^n$, and $x \in \mathbf{R}^m$ is a vector variable.

The above problem generalizes the [LP in inequality form](#):

$$\min c^T x : a_i^T x \leq b_i, \quad i = 1, \dots, m.$$



An SDP in two variables: $\min_x c^T x$ subject to: $F(x) := x_1 F_1 + x_2 F_2 \preceq I$, where $c = ()$, and F_1, F_2 are the two 5×5 symmetric matrices

$$F_1 = \begin{pmatrix} -1.3 & -4.2 & -0.1 & 2.1 & -1 \\ -4.2 & -0.1 & -1.7 & -4.5 & 0.9 \\ -0.1 & -1.7 & 2.3 & -4.4 & -0.4 \\ 2.1 & -4.5 & -4.4 & 3.3 & -1.7 \\ -1. & 00.9 & -0.4 & -1.7 & 4.7 \end{pmatrix},$$

$$F_2 = \begin{pmatrix} 1.6 & 3.9 & 1.6 & -5.3 & -4 \\ 3.9 & -1.8 & -4.7 & 1. & 02.9 \\ 1.6 & -4.7 & -1.3 & 1.6 & -2.6 \\ -5.3 & 1. & 01.6 & 2.7 & 2.6 \\ -4. & 02.9 & -2.6 & 2.6 & -3.4 \end{pmatrix}.$$

Standard Conic Form

[Recall](#) that we can define the scalar product between two matrices $X, Y \in \mathbf{R}^{m \times n}$ as

$$\langle X, Y \rangle = \text{Tr}(X^T Y).$$

If X, Y are both square, and symmetric, then the scalar product is simply the trace of the product.

In standard conic form, an SDP is written as

$$\min_X \langle C, X \rangle : X \succeq 0, \quad \langle A_i, X \rangle = b_i, \quad i = 1, \dots, m,$$

where A_1, \dots, A_m and C are given symmetric $n \times n$ matrices, and $X \in \mathbf{S}^n$ is the matrix variable.

The above generalizes the standard conic form for LP, which can be written as

$$\min_x c^T x : Ax = b, \quad x \geq 0.$$

Just as in LP, the standard inequality and standard conic forms are equivalent, in the sense that we can always convert one into the other, possibly at the expense of introducing new variables and constraints.

CVX syntax

In CVX, a constraint $X \succeq 0$, when X is a symmetric $n \times n$ matrix variable, is encoded with the `semidefinite` assignment. It is important to let CVX know X is symmetric, when declaring it as a variable.

CVX syntax

```
cvx_begin
variables X(n,n) symmetric;
minimize( trace(C*X) )
subject to
for i = 1:m,
trace(A{i}*X) == b(i);
end
X == semidefinite(n);
cvx_end
```

Some Applications of SDP [SDP > Conic problems | LMIs | Standard Forms](#) | Applications

- [Bounding portfolio risk with incomplete covariance information](#)
- [Boolean quadratic programming](#)
- [Matrix completion](#)
- [Robust stability of linear dynamical systems](#)

Exercises [Semi-Definite Programming](#) > Exercises A. Optimization of ellipsoid shapes. For $P \in \mathbf{R}^{n \times n}$, with P symmetric and positive-definite, we define the ellipsoid

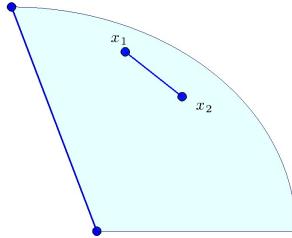
$$\mathbf{E}(P) := \{x : x^T P x \leq 1\}.$$

A measure of the “size” of the ellipsoid is $\mathbf{Tr} P^{-1}$, with \mathbf{Tr} the trace (the sum of the diagonal elements of the matrix argument).

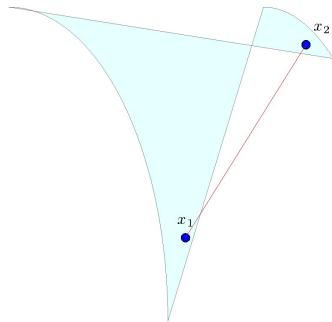
1. Motivate our choice of the size function. Hint: Figure out the semi-axis lengths of the ellipsoid as a function of P .
2. Show that $\mathbf{E}(P) = \{R^{-1}u : \|u\|_2 \leq 1\}$, where R is a factor of P (any matrix R such that $P = R^T R$; in matlab, R can be obtained via the command `chol`.)
3. Show that for any $Q = Q^T$, Q positive-definite, the set $\mathcal{E}(Q)$ is contained in $\mathcal{E}(P)$ if and only if $Q - P$ is positive semi-definite.
4. For given $n \times n$ symmetric matrices P_i , $i = 1, 2$, both positive-definite, show how to compute an ellipsoid, centered at the origin, that contains both $\mathcal{E}(P_1), \mathcal{E}(P_2)$ and has minimum size.
5. Implement and plot your result with the data contained in [here](#). (The function $P \rightarrow \mathbf{Tr} P^{-1}$ for P symmetric and positive definite, is implemented in CVX using `trace_inv`.)

A convex and a non-convex set

The intuitive idea of a convex set is best given by a picture.



A convex set in \mathbf{R}^2 . For any two pair of points, such as x_1 and x_2 , the line segment joining the two points is entirely in the set. Note that if the two points are chosen to be those indicated on the boundary, the line segment itself is at the boundary in this case — but that is OK, provided the set is closed (it contains its boundary).



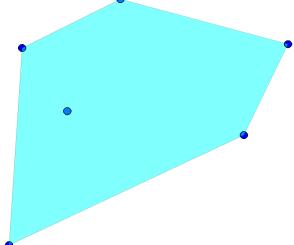
Convex and conic hull

Convex hull of a finite set of points

The *convex hull* of a set of points $\{x_1, \dots, x_m\}$ is defined as the set

$$\mathbf{Co}(x_1, \dots, x_m) := \left\{ \sum_{i=1}^m \lambda_i x_i : \lambda_i \geq 0, i = 1, \dots, m, \sum_{i=1}^m \lambda_i = 1 \right\}.$$

By definition, this set is convex. Note the analogy with the notion of span of a collection of vectors. Here also, we consider combinations of vectors $\sum_{i=1}^m \lambda_i x_i$, but we restrict the weights λ to be non-negative and sum to one.



Example: Convex hull generated by six points in \mathbf{R}^2 . Note that one of the points is in the interior of the convex hull, so that the same convex hull is generated with the remaining five points.

Matlab syntax to plot the convex hull (for $n = 2$)

```
>> inds = convhull(x,y);
>> plot(x,y);
```

Example: The probability simplex.

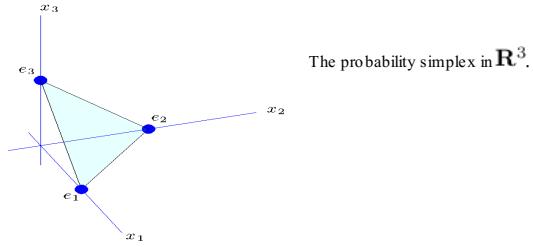
The figure below shows the convex hull of the unit basis vectors e_1, e_2, e_3 in \mathbf{R}^3 . This is the set

$$\text{Co}(e_1, e_2, e_3) = \{\lambda_1 e_1 + \lambda_2 e_2 + \lambda_3 e_3 : \lambda_i \geq 0, i = 1, 2, 3, \lambda_1 + \lambda_2 + \lambda_3 = 1\}.$$

By definition, the vector $\lambda_1 e_1 + \lambda_2 e_2 + \lambda_3 e_3$ is the vector in \mathbf{R}^3 with components $(\lambda_1, \lambda_2, \lambda_3)$. Hence, the set above is the set of vectors in \mathbf{R}^3 with non-negative components which sum to one:

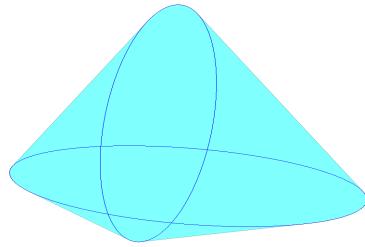
$$\text{Co}(e_1, e_2, e_3) = \{\lambda \in \mathbf{R}^3 : \lambda \geq 0, \mathbf{1}^T \lambda = 1\},$$

where the component-wise inequality notation $\lambda \geq 0$ expresses the fact that the vector λ has non-negative components, and $\mathbf{1}$ stands for the vector of ones. This set is called the *probability simplex*.



Convex hull of a set

More generally, for any given set \mathbf{C} in \mathbf{R}^n , we can define its *convex hull* as the set of convex combinations of any finite collection of points contained in it.



Example: The convex hull of the union of two ellipses.

Conic hull

The conic hull of a set of points $\{x_1, \dots, x_m\}$ is defined as

$$\left\{ \sum_{i=1}^m \lambda_i x_i : \lambda \in \mathbf{R}_+^m \right\}.$$

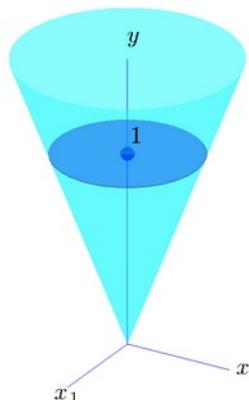
Example: The conic hull of the union of the three-dimensional simplex above and the singleton $\{\mathbf{0}\}$ is the whole set \mathbf{R}_+^3 , which is the set of real vectors that have non-negative components. The figure shows that indeed any vector $v \in \mathbf{R}_+^3$ can be obtained by a positive scaling of a vector v_n in the three-dimensional simplex: $v_n = \alpha v$, with $\alpha := v_1 + v_2 + v_3 \geq 0$.

Second-order cone

The set in \mathbf{R}^{n+1}

$$\mathbf{K}_n := \{(x, y) \in \mathbf{R}^{n+1} : y \geq \|x\|_2\}$$

is a convex cone, called the *second-order cone*.



Example: The second-order cone is sometimes called “ice-cream cone”. In \mathbf{R}^3 , it is the set of triples (x_1, x_2, y) with

$$y^2 \geq x_1^2 + x_2^2, \quad y \geq 0.$$

The blue circle corresponds to the set

$$\left\{ (x_1, x_2, y) : y = 1 \geq \sqrt{x_1^2 + x_2^2} \right\}.$$

Proof of convexity

The fact that \mathbf{K}_n is convex can be proven directly from the basic definition of a convex set. Alternatively, we may express \mathbf{K}_n as an intersection of half-spaces, as follows.

From the Cauchy-Schwartz inequality, we have

$$y \geq \|x\|_2 \iff \forall u, \|u\|_2 \leq 1 : y \geq u^T x,$$

we have

$$\mathbf{K}_n = \bigcap_{u : \|u\|_2 \leq 1} \{(x, y) \in \mathbf{R}^{n+1} : y \geq u^T x\}.$$

Each one of the sets involved in the intersection is a half-space.

Semidefinite cone Positive semidefinite matrices

A symmetric matrix $P = P^T \in \mathbf{R}^{n \times n}$ is said to be *positive semi-definite* if the associated quadratic form is non-negative, that is:

$$\forall x \in \mathbf{R}^n : x^T P x \geq 0.$$

An alternate condition is that every eigenvalue of P is non-negative. We use the acronym PSD to refer to the term “positive semidefinite”, and the notation $P \succeq 0$ to express that P is PSD.

A matrix is said to be *positive definite* if the above condition is satisfied strictly for non-zero x :

$$\forall x \in \mathbf{R}^n, x \neq 0 : x^T P x > 0.$$

An alternate condition is that every eigenvalue of P is positive. We use the acronym PD to refer to the term “positive definite”, and the notation $P \succ 0$ to express that P is PD.

Semidefinite cone

The set of PSD matrices in $\mathbf{R}^{n \times n}$ is denoted \mathbf{S}_+ . That of PD matrices, \mathbf{S}_{++} .

The set \mathbf{S}_+ is a convex cone, called the *semidefinite cone*. The fact that it is convex derives from its expression as the intersection of half-spaces in the subspace \mathbf{S}^n of symmetric matrices. Indeed, we have

$$\mathbf{S}_+ = \bigcap_{x \in \mathbf{R}^n} \{P \in \mathbf{S}^n : x^T P x \geq 0\}.$$

Rank-one PSD matrices

PSD matrices with rank one can be expressed as $P = vv^T$ for some $v \in \mathbf{R}^n$. The associated quadratic form is a squared linear form:

$$x^T P x = x^T (v v^T) x = (x^T v)(v^T x) = (v^T x)^2.$$

Link with covariance matrices

[Covariance matrices](#) are PSD matrices, since they can be expressed as an expected value of a squared linear form: if X is a random variable, the covariance matrix of X is defined as

$$\mathbf{E}(X - \mathbf{E}(X))(X - \mathbf{E}(X))^T,$$

where \mathbf{E} denotes the expectation operator. Conversely, any PSD matrix can be interpreted as a covariance matrix, for some distribution. Hence, the PSD cone is exactly the set of covariance matrices.

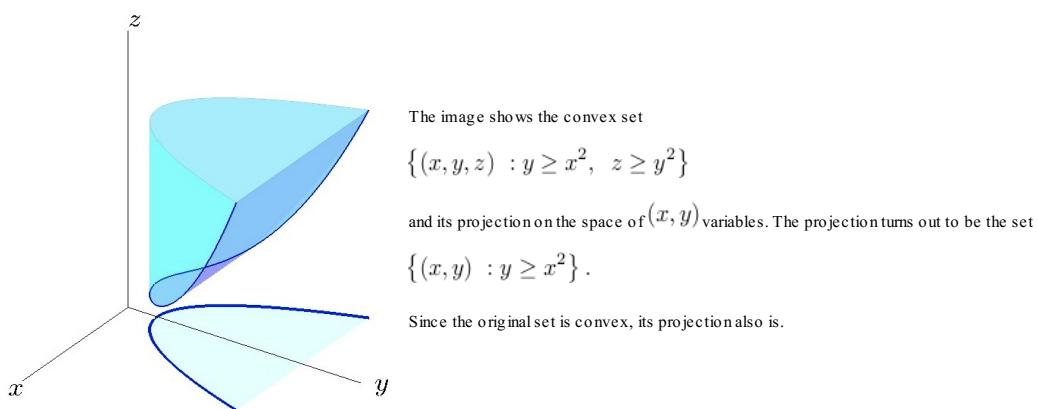
Projection of a convex set on a subspace

The projection of a point on a set is the point that is closest (in the sense of Euclidean norm) to the set. Mathematically the projection of a point x on a convex set \mathcal{C} is the (unique) solution to the optimization problem

$$\pi(x) := \arg \min_y \|x - y\|_2 : y \in \mathcal{C}.$$

The projection of a whole set on another set is simply the set of projections $\pi(x)$, when x runs \mathcal{C} .

The projection of a convex set on any subspace or affine set is convex.



Log-Determinant Function and Properties

The *log-determinant* function is a function from the set of symmetric matrices in $\mathbf{R}^{n \times n}$, with domain the set of positive definite matrices, and with values

$$f(X) = \begin{cases} \log \det X & \text{if } X \succ 0, \\ +\infty & \text{otherwise.} \end{cases}$$

The function can be expressed in terms of the (real, positive) eigenvalues of the argument matrix X ; it does not depend on its eigenvectors.

This function provides a measure of the volume of an ellipsoid. Precisely, the volume of the ellipsoid

$$\mathbf{E} = \{x : x^T X^{-1} x \leq 1\}$$

is given by $\text{vol}(\mathbf{E}) = C_n \prod_{i=1}^n \sqrt{\lambda_i(X)}$, where C_n is a constant (given by the volume of the unit sphere in \mathbf{R}^n). Thus, $\log \text{vol}(\mathbf{E}) = \frac{1}{2}f(X) + \text{constant}$.

This means that the volume of the ellipsoid is a function of the product of the eigenvalues of the matrix X .

Proof of the concavity of the log-determinant function: We use the fact that a function is convex if and only if its restriction to a line is.

Convexity of quadratic functions in two variables

We return to the example described [here](#). We consider the two quadratic functions $p, q : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$\begin{aligned} p(x) &= 4x_1^2 + 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 2 \times 10^5, \\ q(x) &= 4x_1^2 - 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 2 \times 10^5. \end{aligned}$$

The Hessian of p is independent of x , and given by the constant matrix:

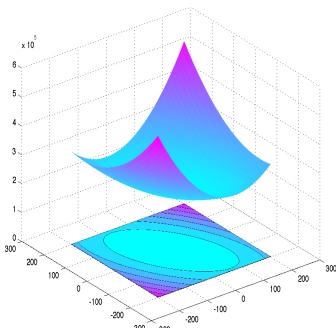
$$\nabla^2 p = \begin{pmatrix} 8 & 3 \\ 3 & 4 \end{pmatrix}.$$

We check that the eigenvalues of p are positive, since the determinant, as well as the trace, of the above matrix are. Therefore, p is convex.

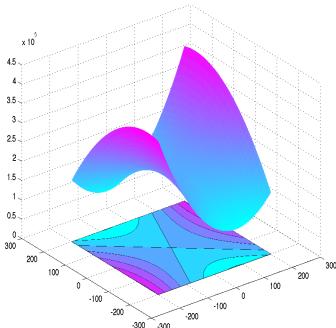
Likewise, the Hessian of q is

$$\nabla^2 q = \begin{pmatrix} 8 & 3 \\ 3 & -4 \end{pmatrix}.$$

This time, the Hessian has a negative eigenvalue, so q is not convex.



Level sets and graph of the quadratic function p . The epigraph is anything that extends above the graph in the z -axis direction. This function is ‘‘bowl-shaped’’, or convex.



Level sets and graph of the quadratic function q .

Definition: Square-to-Linear Function and Properties

The *square-to-linear* function in \mathbf{R}^n is the function $f : \mathbf{R}^n \times \mathbf{R} \rightarrow \mathbf{R}$, with domain the set

$$\text{dom } f = \{(x, y) \in \mathbf{R}^n \times \mathbf{R} : y > 0\}$$

and values given by

$$f(x) = \begin{cases} \frac{x^T x}{y} & \text{if } y > 0, \\ +\infty & \text{otherwise.} \end{cases}$$

This function is convex, since its domain is, and inside the interior of the domain, the Hessian is given by

$$\nabla^2 f(x) = \frac{2}{y^3} \begin{pmatrix} y^2 I & -yx \\ -yx^T & x^T x \end{pmatrix} = .$$

We check that the Hessian is positive semi-definite: for any $w = (z, t) \in \mathbf{R}^n \times \mathbf{R}$, we have

$$\frac{y^3}{2} w^T \nabla^2 f(x) w = \begin{pmatrix} z \\ t \end{pmatrix}^T \begin{pmatrix} y^2 I & -yx \\ -yx^T & x^T x \end{pmatrix} \begin{pmatrix} z \\ t \end{pmatrix} = \|yz - tx\|_2^2 \geq 0.$$

Negative Entropy Function and Properties

The *negative entropy* function in \mathbf{R}^n is the function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, with domain the set of vectors with *positive* components, and values on the domain given by

$$f(x) = \sum_{i=1}^n x_i \log x_i.$$

This function is convex.

Proof: Since the function is a sum of functions, each of which depends on one variable not appearing in the others, we just need to check the convexity of the function of *one* variable

$$f(\xi) = \begin{cases} \xi \log \xi & \text{if } \xi > 0, \\ +\infty & \text{otherwise.} \end{cases}$$

The convexity of the latter derives directly from the second-order condition:

$$\forall \xi > 0 : \frac{d^2}{d\xi^2} f(\xi) = \frac{1}{\xi} > 0.$$

Schur Complement Lemma

Lemma: Schur Complement

Let S be a symmetric matrix partitioned into blocks:

$$S = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix},$$

where both A, C are symmetric and square. Assume that C is [positive definite](#). Then the following properties are equivalent:

- S is positive semi-definite.
- The *Schur complement* of C in S , defined as the matrix $A - BC^{-1}B^T$, is positive semi-definite.

Proof: Recall that the matrix S is positive semi-definite if and only if $x^T S x \geq 0$ for any vector x . Partitioning the vector x similarly to S , as $x = (y, z)$, we obtain that S is positive semi-definite if and only if

$$\forall (z, y) : g(y, z) := \begin{pmatrix} y \\ z \end{pmatrix}^T \begin{pmatrix} A & B \\ B^T & C \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} \geq 0.$$

This is equivalent to: for every y ,

$$0 \geq f(y) := \min_z g(y, z).$$

Since S is positive semi-definite, the corresponding quadratic function g is convex, jointly in its two arguments. Due to the [partial minimization](#) result, we obtain that the partial minimum $f(y)$ is convex as well.

It is easy to obtain a closed-form expression for f . We simply have to minimize the convex quadratic function g with respect to its second argument. Since the problem of minimizing g is not constrained, we just set the gradient of g with respect to z to zero (see [here](#)):

$$\nabla_z g(y, z) = 2(Cz + B^T y) = 0,$$

which leads to the (unique) optimizer $z^*(y) := -C^{-1}B^T y$. Plugging this value we obtain:

$$f(y) = g(y, z^*(y)) = y^T (A - BC^{-1}B^T) y.$$

Since f is convex, its Hessian must be positive semi-definite. Hence $A - BC^{-1}B^T \succeq 0$, as claimed.

Proving convexity via monotonicity

Consider the function $f : \mathbf{R} \rightarrow \mathbf{R}$, with values $f(x) = x^4$.

We can express the function as the composition of the function

$$g(x) = x^2$$

with the function h with values

$$h(y) = \begin{cases} y^2 & \text{if } y \geq 0 \\ +\infty & \text{otherwise.} \end{cases}$$

That is, $f(x) = h(g(x))$. Since $g(x)$ belongs to the domain of h for every $x \in \mathbf{R}$, the domain of f is indeed the whole real line.

The functions g, h are both convex, and h is monotone increasing. Hence, by the [monotonicity property](#), the composition $f = h \circ g$ is also convex. Note that the domain of h is chosen to ensure monotonicity; the function $y \rightarrow y^2$ with domain the whole real line is not monotonic, as it decreases for $y \leq 0$ then increases for $y \geq 0$.

Local and Global Optima in Convex Optimization

Theorem: Global vs. local optima in convex optimization

For convex problems, any locally optimal point is globally optimal. In addition, the optimal set is convex.

Proof: Let x^* be a local minimizer of f_0 on the set \mathbf{X} , and let $y \in \mathbf{X}$. By definition, $x^* \in \text{dom } f_0$. We need to prove that $f_0(y) \geq f_0(x^*) = p^*$. There is nothing to prove if $f_0(y) = +\infty$, so let us assume that $y \in \text{dom } f_0$. By convexity of f_0 and \mathbf{X} , we have $x_\theta := \theta y + (1 - \theta)x^* \in \mathbf{X}$, and:

$$f_0(x_\theta) \leq \theta(f_0(y) - f_0(x^*)).$$

Since x^* is a local minimizer, the left-hand side in this inequality is nonnegative for all small enough values of $\theta > 0$. We conclude that the right hand side is nonnegative, i.e., $f_0(y) \geq f_0(x^*)$, as claimed.

Also, the optimal set is convex, since it can be written

$$\mathbf{X}^{\text{opt}} = \{x \in \mathbf{R}^n : f_0(x) \leq p^*, x \in \mathbf{X}\}.$$

This ends our proof.

Minimization of a convex quadratic function

Here we consider the problem of minimizing a convex quadratic function without any constraints. Specifically, consider the problem

$$p^* := \min_x q(x) := \frac{1}{2}x^T Ax + b^T x$$

where $A = A^T \in \mathbf{R}^{n \times n}$, and $b \in \mathbf{R}^n$. We assume that q is convex, meaning that its Hessian A is [positive semi-definite](#).

The optimality condition for an unconstrained problem is $\nabla q(x) = 0$, which here reduces to

$$\nabla q(x) = Ax + b = 0 \quad \text{Case when } A \text{ is positive-definite}$$

When A is positive-definite, that is, it has only positive (non-zero) eigenvalues, then there is a *unique* minimizer. The above optimality condition provides it: $x^* = -A^{-1}b$.

Degenerate case: a simple example

Consider the case with

$$q(x) = \frac{1}{2}x_1^2 + b_1x_1 + b_2x_2.$$

The optimality condition is

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

These equations are feasible if and only $b_2 = 0$ (that is, b is in the range of A). In that case, the set of optimal points are of the form $x_1 = b_1, x_2$ arbitrary.

Otherwise, when $b_2 \neq 0$, the optimality conditions are not feasible, and the minimum value is actually $p^* = -\infty$. It is only attained in the limit, for example with x_1 fixed arbitrarily, $x_2 \rightarrow \infty$, with sign of x_2 equal to that of $-b_2$.

General case

If A is not invertible, but b is in the range of A , then there exist $x_0 \in \mathbf{R}^n$ such that $b = -Ax_0$. Then any point x such that $x - x_0$ is in the nullspace of A is optimal, since

$$q(x) = \frac{1}{2}x^T Ax - x_0^T Ax = \frac{1}{2}(x - x_0)^T A(x - x_0) - \frac{1}{2}x_0^T Ax_0.$$

In particular, x_0 is optimal, and the corresponding value of the problem is $p^* = -\frac{1}{2}x_0^T Ax_0$.

If b is not in the range space of A , then there are no minimizing points. This means that the minimum is not attained. We can in fact show that $p^* = -\infty$, and construct a sequence of points that achieves this value in the limit. We simply invoke the [fundamental theorem of linear algebra](#). For symmetric matrices, the theorem specializes to the fact that range and nullspace are orthogonal. Since $b \notin \mathbf{R}(A)$, b can be written as $b = Ar + c$, with $c \neq 0$ and $Ac = 0$. Set $x(t) := -tc$, with $t \in \mathbf{R}$; since $Ax(t) = 0$, we obtain $q(x(t)) = -b^T tc = -tc^T c$. Letting $t \rightarrow +\infty$ achieves the desired result.

Proof via the eigenvalue decomposition

A constructive proof involves the eigenvalue decomposition of the symmetric matrix A . We have $A = U\Lambda U^T$, with Λ a diagonal matrix containing the (non-negative) eigenvalues of A , and U an orthogonal matrix of eigenvectors. The original problem can be expressed in terms of the new variable $\bar{x} = U^T x$, as follows:

$$\min_{\bar{x}} \frac{1}{2} \bar{x}^T \Lambda \bar{x} + \bar{b}^T \bar{x},$$

with $\bar{b} := U^T b$. Now decompose Λ as $\Lambda = \text{diag}(\Lambda_+, 0)$ with Λ_+ containing the *positive* eigenvalues. We decompose the variable \bar{x} as $\bar{x} = (\bar{x}_+, \bar{x}_0)$, with vector \bar{x}_+ (resp. \bar{x}_0) the variables corresponding to the positive (resp. zero) eigenvalues. Decompose \bar{b} similarly, as $\bar{b} = (\bar{b}_+, \bar{b}_0)$. The problem writes

$$\min_{\bar{x}_+, \bar{x}_0} \frac{1}{2} \bar{x}_+^T \Lambda_+ \bar{x}_+ + \bar{b}_+^T \bar{x}_+ + \bar{b}_0^T \bar{x}_0.$$

The problem looks exactly like the simple 2D example above.

The optimality conditions then write

$$\bar{x}_+ = \Lambda_+^{-1} \bar{b}_+, \quad 0 = \bar{b}_0.$$

If $\bar{b}_0 = 0$, the solutions are of the form $x = U\bar{x} = U(\Lambda_+^{-1} \bar{b}_+, \bar{x}_0)$, with \bar{x}_0 free. (We recover a unique solution when there are no zero eigenvalues, that is, A is invertible.)

Otherwise, there are no optimal points. Choosing $\bar{x} = (0, -t\bar{b}_0)$, with $t \rightarrow +\infty$, achieves the optimal value $p^* = -\infty$.

Optimality condition for a convex, unconstrained problem

Consider the unconstrained optimization problem

$$\min_x f_0(x),$$

where $f_0 : \mathbf{R}^n \rightarrow \mathbf{R}$ is convex and differentiable.

The [optimality condition](#)

$$\forall y \in \mathbf{X} : \nabla f_0(x)^T (y - x) \geq 0.$$

where the feasible set \mathbf{X} is now the whole space \mathbf{R}^n , reduces to

$$\nabla f_0(x) = 0.$$

Example:

Optimality condition for a convex, linearly constrained problem

Consider the linearly constrained optimization problem

$$\min_x f_0(x) : Ax = b,$$

where $f_0 : \mathbf{R}^n \rightarrow \mathbf{R}$ is convex and differentiable, and $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$ define the equality constraints.

A point x is optimal for the above problem if and only if it satisfies

$$Ax = b, \quad \nabla f_0(x) = A^T \lambda,$$

for some vector $\lambda \in \mathbf{R}^m$.

Proof: Let us re-formulate the [optimality condition](#) is

$$x \in \mathbf{X} \text{ and } \forall y \in \mathbf{X} : \nabla f_0(x)^T (y - x) \geq 0.$$

where the feasible set \mathbf{X} is now the affine set

$$\{x : Ax = b\}.$$

We can write the above as: $Ax = b$ and

$$\forall z, \quad Az = 0 : \nabla f_0(x)^T z \geq 0.$$

Since we can always flip the sign of vectors z with $Az = 0$, we can express the above as

$$\forall z, \quad Az = 0 : \nabla f_0(x)^T z = 0.$$

This means that $\nabla f_0(x)$ should be orthogonal to the nullspace of A .

From the [fundamental theorem of linear algebra](#), this in turn says that $\nabla f_0(x)$ should belong to the range of the transpose matrix, $\mathbf{R}(A^T)$. In other words, the above condition is equivalent to $\exists \lambda \in \mathbf{R}^m : \nabla f_0(x) = A^T \lambda$.

We conclude that the optimality conditions for x to be optimal are that there exists a vector λ such that

$$Ax = b, \quad \nabla f_0(x) = A^T \lambda.$$

This ends our proof.

Example: assume that $f(x) = (1/2)\|x\|_2^2$, the previous conditions reduces to

$$Ax = b, \quad x = A^T \lambda,$$

which further reduces to $AA^T\lambda = b$. When A is full row rank (none of the constraints involve redundant row vectors), we obtain the formula $x = A^T(AA^T)^{-1}b$ for the minimum-norm solution to a linear equation, as seen [here](#).

A half-space in \mathbf{R}^2

Consider the set in \mathbf{R}^2 defined by a single affine inequality:

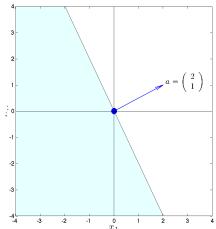
$$\mathbf{H} = \{x : 2x_1 + x_2 \leq 3\},$$

This set is a half-space, that is, a set of the form

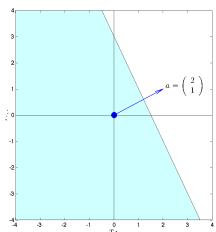
$$\mathbf{H} = \{x : a^T x \leq b\},$$

with

$$a = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \quad b = 3.$$



The half-space $\{x : 2x_1 + x_2 \leq 0\}$. The vector $a = (2, 1)$ points outwards the set, at a 90° angle. The set is comprised of points forming an obtuse angle with a . The boundary is the hyperplane defined by the single linear equation $2x_1 + x_2 = 0$. This is the subspace of vectors orthogonal to a .



The half-space $\{x : 2x_1 + x_2 \leq 3\}$. The vector $a = (2, 1)$ points outwards the set, at a 90° angle. The boundary is the hyperplane defined by the single linear equation $2x_1 + x_2 = 3$.

Graph, epigraph, level and sub-level sets of a function

Consider a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$. We can define four sets relevant to f : the *graph* and the *epigraph*, both are subsets of \mathbf{R}^{n+1} ; *level sets* and *sub-level sets* are subsets of \mathbf{R}^n .

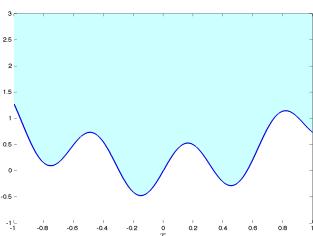
Graph

The *graph* of f is the set of input-output pairs that f can attain, that is:

$$G(f) := \{(x, f(x)) \in \mathbf{R}^{n+1} : x \in \mathbf{R}^n\}$$

The *epigraph*, denoted $\text{epi } f$, describes the set of input-output pairs that f can achieve, as well as “anything above” (*epi* in Greek means “above”):

$$\text{epi } f := \{(x, t) \in \mathbf{R}^{n+1} : x \in \mathbf{R}^n, t \geq f(x)\}.$$



The function $f : \mathbf{R} \rightarrow \mathbf{R}$, with domain $(-1, 1)$ and value inside the domain $f(x) = x^2 + (1/2)\sin(10x)$. The graph corresponds to the points in blue, and the epigraph is in light blue. The epigraph extends to infinity above the graph. Points outside the domain are not shown.

Level sets

Level sets are sets of points that achieve exactly a certain value for f . Precisely, the t -level set of f is defined by

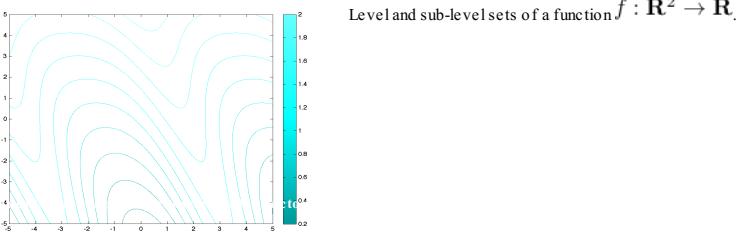
$$L_t(f) := \{x \in \mathbf{R}^n : x \in \mathbf{R}^n, t = f(x)\}.$$

In two-dimensions ($n = 2$), the level sets are referred to as *level curves*.

Sub-level sets

A related notion is that of *sub-level set*. This is now the set of points that achieve at most a certain value for f , or below. Precisely, the t -sub-level set of f is defined by

$$S_t(f) := \{x \in \mathbf{R}^n : x \in \mathbf{R}^n, t \geq f(x)\}.$$



Component-wise inequality convention for vectors

If u, v are two vectors in \mathbf{R}^n , the notation $u \leq v$ corresponds to the component-wise inequality: that is,

$$u \leq v \iff u_i \leq v_i, \quad i = 1, \dots, n.$$

With our convention, we can write

$$u := \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix} \leq \begin{pmatrix} 3 \\ 0 \\ 4 \end{pmatrix} := v.$$

A similar convention applies to strict inequalities. With the vectors u, v defined above, we can write the stronger statement $u < v$.

Not every pair of vectors can be compared in a component-wise fashion. For example, the vectors

$$u := \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix}, \quad v := \begin{pmatrix} 3 \\ 0 \\ -4 \end{pmatrix}.$$

do not satisfy $u \leq v$, but we cannot write $u > v$ either.

Thus, the component-wise inequality convention induces only a *partial order* on vectors. This is contrast with real numbers, for which the ordinary inequality induces a total order, in the sense that any pair of real numbers can be compared.

A polyhedron in \mathbf{R}^2

Consider the set in \mathbf{R}^2 defined by a three affine inequalities:

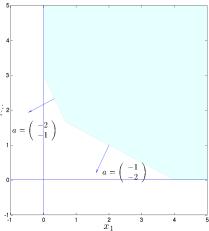
$$\mathbf{P} = \{x : 2x_1 + x_2 \geq 3, \quad x_1 + 2x_2 \geq 4, \quad 0 \leq x_1 \leq 5, \quad 0 \leq x_2 \leq 5\}.$$

This set is a polyhedron, that is, a set of the form

$$\mathbf{P} = \{x : Ax \leq b\},$$

with the component-wise inequality convention, and

$$A = \begin{pmatrix} -2 & -1 \\ -1 & -2 \\ -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} -3 \\ -4 \\ 0 \\ 0 \\ 5 \\ 5 \end{pmatrix}.$$



The polyhedron \mathbf{P} defined above. Each row of the matrix A corresponds to a vector pointing outwards one of the facet of the polyhedron.

Probability simplex in \mathbf{R}^n

The *probability simplex* in \mathbf{R}^n is the polyhedron

$$\mathbf{P} = \left\{ p \in \mathbf{R}^n : p \geq 0, \quad \sum_{i=1}^n p_i = 1 \right\}.$$

In the above, we use the [component-wise notation](#) to specify that every element of $p \in \mathbf{P}$ is non-negative.

The above is indeed a polyhedron, as it is defined by affine equalities and inequalities (precisely, one affine equality and n affine inequalities).

The reason for the name given to that set is that it describes all the possible probability distributions of a random variable that can take a finite (here, n) possible values. With this interpretation, for $p \in \mathbf{P}$, p_i stands for the probability that the random variable takes the i -th value.

A Drug Production Problem

Problem description
A company produces two kinds of drugs, DrugI and DrugII, containing a specific active agent A, which is extracted from raw materials purchased on the market.

There are two kinds of raw materials, RawI and RawII, which can be used as sources of the active agent. The related production, cost and resource data are given in the tables below. The goal is to find the production plan which maximizes the profit of the company.

Problem data

Drug production data:

Parameter	DrugI	DrugII
Selling price, \$ per 1000 packs	5,500	6,100
Content of agent A, g per 1000 packs	0.500	0.600
Manpower required, hours per 1000 packs	90.0	100.0
Equipment required, hours per 1000 packs	40.0	50.0
Operational costs, \$ per 1000 packs	700	800

Contents of raw materials:

Raw material	Purchasing price, \$ per kg	Content of agent A, g per kg
RawI	100.00	0.01
RawII	199.90	0.02

Resources:

Budget, \$	Manpower, hours	Equipment, hours	Capacity of raw materials storage, kg
100,000	2,000	800	1,000

Linear programming formulation Variables

Let us denote by x_{DrugI} , x_{DrugII} the amounts (in 1000 of packs) of Drug I and II produced, while x_{RawI} , x_{RawII} denote the amounts (in kg) of raw materials to be purchased.

Objective function

According to the problem data, the objective to be minimized in this problem has the form $f_0(x) = f_{\text{costs}}(x) - f_{\text{income}}(x)$, where

$$f_{\text{costs}}(x) = 100 \cdot x_{\text{RawI}} + 199.90 \cdot x_{\text{RawII}} + 700 \cdot x_{\text{DrugI}} + 800 \cdot x_{\text{DrugII}}$$

represents the purchasing and operational costs, and

$$f_{\text{income}}(x) = 5,500 \cdot x_{\text{DrugI}} + 6100 \cdot x_{\text{DrugII}}$$

represents the income from selling the drugs.

Constraints

We have total of five inequality constraints, and additional sign constraints on the variables.

Balance of active agent:

$$0.01 \cdot x_{\text{RawI}} + 0.02 \cdot x_{\text{RawII}} - 0.05 \cdot x_{\text{DrugI}} - 0.600 \cdot x_{\text{DrugII}} \geq 0.$$

This constraint says that the amount of raw material must be enough to produce the drugs.

Storage constraint:

$$x_{\text{RawI}} + x_{\text{RawII}} \leq 1000.$$

This constraint says that the capacity of storage for the raw materials is limited.

Manpower constraint:

$$90.0 \cdot x_{\text{DrugI}} + 100.0 \cdot x_{\text{DrugII}} \leq 2000,$$

which expresses the fact that the resources in manpower are limited, we cannot allocate more than 2,000 hours to the project.

Equipment constraint:

$$40.0 \cdot x_{\text{DrugI}} + 50.0 \cdot x_{\text{DrugII}} \leq 800.$$

This says that the resources in equipment are limited.

Budget constraint:

$$100.0 \cdot x_{\text{RawI}} + 199.90 \cdot x_{\text{RawII}} + 700 \cdot x_{\text{DrugI}} + 800 \cdot x_{\text{DrugII}} \leq 100,000.$$

This limits the total budget.

Sign constraints:

$$x_{\text{RawI}} \geq 0, \quad x_{\text{RawII}} \geq 0, \quad x_{\text{DrugI}} \geq 0, \quad x_{\text{DrugII}} \geq 0.$$

A CVX implementation of the problem is as follows.

CVX implementation

```
% min c'*x : A*x <= b, x >= 0
% x = (RawI, RawII, DrugI, DrugII)
% data:
% objective function: we minimize cost minus profit
c = [100 199.9 -5500 -6100]';
% constraints
A = [-0.01 -0.02 0.500 0.600; % balance
      1 1 0 0; % storage of raw material
      0 0 90.0 100.0; % manpower
      0 0 40.0 50.0; % equipment
      100.0 199.9 700 800; % budget
];
% right-hand side
b = [0; 1000; 2000; 800; 100000];
% solve problem via CVX
cvx_begin
    variable x(4,1)
    minimize( c'*x )
    subject to
        A*x <= b;
        x >= 0;
cvx_end
pstar = cvx_optval; % optimal value of the problem
```

Solving this problem, we obtain the following optimal value and a corresponding optimal point x^* :

$$p^* = -8819.658, \quad x_{\text{RawI}}^* = 0, \quad x_{\text{RawII}}^* = 438.789, \quad x_{\text{DrugI}}^* = 17.552, \quad x_{\text{DrugII}}^* = 0.$$

Note that both the budget and the balance constraints are active (that is, the production process utilizes the entire budget and the full amount of active agent contained in the raw materials). The solution promises the company a modest, but quite respectable profit of 8.8%.

Projection on a polyhedron

Recall that a polyhedron is an intersection of a finite number of half-spaces. A polyhedron \mathbf{P} can be written as

$$\mathbf{P} = \{x : Ax \leq b\},$$

where $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$, and the symbol \leq refers to the [component-wise inequality](#) between vectors.

The *Euclidean projection* (or projection for short) of the origin on the polyhedron \mathbf{P} is the (unique) solution to the optimization problem

$$\min_{x \in \mathbf{P}} \|x\|_2.$$

Without loss of generality, we can square the objective and solve the problem

$$\min_x x^T x : Ax \leq b.$$

The above is a QP.

A linear program in 2D

Consider the optimization problem

$$\max_x 3x_1 + 1.5x_2 : -1 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq 3.$$

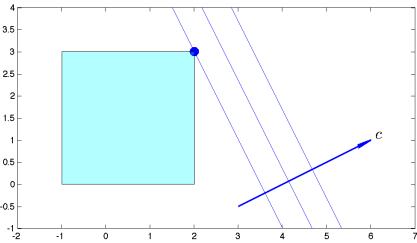
The problem can be put in standard form:

$$\min_x -3x_1 - 1.5x_2 : x_1 \leq 2, \quad -x_1 \leq 1, \quad x_2 \leq 3, \quad -x_2 \leq 0.$$

This is an LP, since the objective and constraint functions are all affine.

Geometric view of the toy linear program above. The level curves (curves of constant value) of the objective function are shown: these are straight lines orthogonal to the objective vector, $c = (3, 1.5)$. The

problem amounts to find the largest value of t such that $t = c^T x$ for some feasible x . The optimal point is $x^* = (2, 3)$.



A Quadratic Program with Two Variables

Consider the problem

$$\min_x x_1^2 - x_1 x_2 + 2x_2^2 - 3x_1 - 1.5x_2 : -1 \leq x_1 \leq 2, 0 \leq x_2 \leq 3.$$

The problem is a [quadratic program](#):

- the constraints are affine inequalities in the decision vector x ;
- the objective function can be expressed as

$$f_0(x) = c^T x + x^T Q x$$

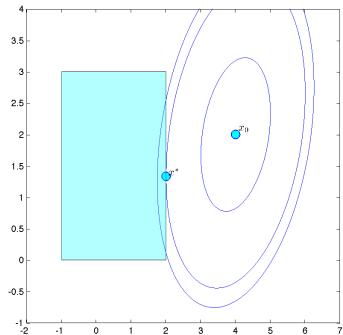
where

$$c = \begin{pmatrix} -3 \\ -1.5 \end{pmatrix}, \quad Q = \begin{pmatrix} 1 & -1/2 \\ -1/2 & 2 \end{pmatrix}.$$

We check that Q is positive semi-definite by computing its eigenvalue decomposition:

$$Q = U \Lambda U^T, \quad U = \begin{pmatrix} -0.3827 & 0.9239 \\ 0.9239 & 0.3827 \end{pmatrix}, \quad \Lambda = \begin{pmatrix} 2.2071 & 0 \\ 0 & 0.7929 \end{pmatrix},$$

and checking that the eigenvalues (appearing on the diagonal of Λ) are non-negative.



Geometric view of the quadratic program above. The problem is a QP since the objective function is quadratic convex, and the constraints are affine inequalities.

LP and QP in conic form

The LP

$$\min_x f^T x : Fx \leq g$$

can always be represented in the conic form

$$\min_z c^T z : Az = b, \quad z \geq 0,$$

for appropriate matrix A and vector c .

To prove this, we introduce three new sets of non-negative variables: $s = g - Fx$, which represents the constraints, and x_+, x_- , which contain the positive and negative parts of vector x . We have $x = x_+ - x_-$, with $x_+ := \max(x, 0) \geq 0$ and $x_- = \max(-x, 0) \geq 0$. The constraint $Fx \leq g$ then reads $s \geq 0$.

Let us define the new variable $z = (x_+, x_-, s)$. The relationship between s, x_+, x_- can then be expressed as $Az = b$, with

$$A = (F, -F, I), \quad b = g.$$

The objective of the original problem can be written as $f^T x = c^T z$, with

$$c := \begin{pmatrix} f \\ -f \\ 0 \end{pmatrix}.$$

Putting this together, we express the original LP as claimed.

LP Formulation of l_1 - and l_∞ -norm Regression

The l_∞ -norm regression problem:

$$\min_x \|Ax - y\|_\infty$$

where $A \in \mathbf{R}^{m \times n}$, $y \in \mathbf{R}^m$ are given, can be written as the LP

$$\min_{t,x} t : t \geq a_i^T x - y_i \geq -t, \quad i = 1, \dots, m,$$

where a_i stands for the i -th row of A .

Similarly, the l_1 -norm regression problem:

$$\min_x \|Ax - y\|_1$$

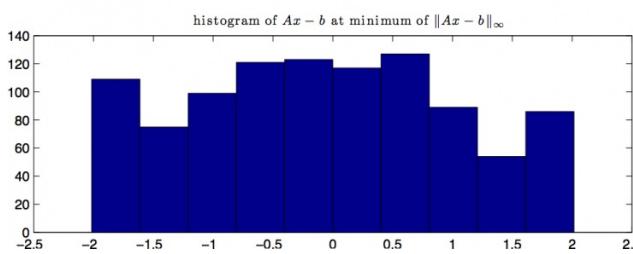
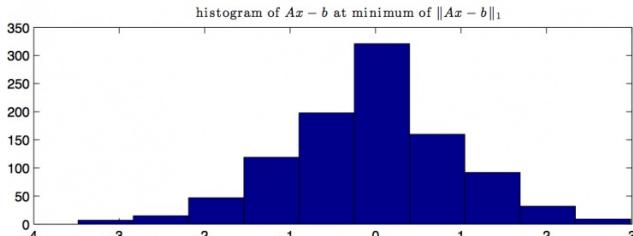
can be written as the LP

$$\min_{t,x} \sum_{i=1}^m t_i : t_i \geq a_i^T x - y_i \geq -t_i, \quad i = 1, \dots, m$$

CVX does not require the user to transform the problem into LP format. The following syntax will work for any $p \in [1, +\infty]$:

CVX implementation

```
cvx_begin
    variable x(n,1)
    minimize( norm(A*x-y, p) )
cvx_end
```



A regression problem with random data $A \in \mathbf{R}^{1000 \times 100}$, $b \in \mathbf{R}^{1000}$. The l_1 -norm tends to encourage sparsity of the residual vector: the top panel indeed shows a lot of the residual vector elements are zero. In contrast, the l_∞ norm (bottom panel) encourages a lot of elements to have similar magnitude.

Cardinality of a vector

The cardinality of a vector $x \in \mathbf{R}^n$ is the number of non-zero elements in it. It is sometimes called the l_0 -norm of x , although the cardinality function is *not* a norm.

The cardinality is denoted $\text{Card}(x)$, or $\|x\|_0$.

For example, the vector

$$x = (1, -2, 0, 0, 4, -5)$$

has cardinality $\text{Card}(x) = 4$.

The cardinality function is useful in many problems, however it is difficult to optimize. In [cardinality minimization](#) problems, the l_1 -norm is often used as a surrogate.

- Binary classification
- Linear binary classification
- Feature selection: encouraging sparsity
- Robustness

Binary classification problems Where do they arise? *Binary classification* problems arise when we seek to separate two sets of data points in \mathbf{R}^n , each corresponding to a given class. We seek to separate the two data sets using simple “boundaries” in \mathbf{R}^n , typically hyperplanes. Once the boundary is found, we can use it to predict the class a new point belongs to, by simply checking on which side of the boundary it falls.

Such problems arise in many situations:

- In *recommendation systems*, say for movies, we have information about a given user's movie preferences. Hence we may separate the whole movie database in two different sets (movies that the user's like, and the others). The goal of classification is to determine whether a given new movie will be liked by the user.
- In *spam filtering*, we may have a set of emails which we know are spam, and another set of emails that are known to be legitimate.
- In *fault detection*, we have a set of past signals, some of which are known to correspond to a fault, and others known to correspond to a situation free of fault. The goal of classification is to predict whether a new measured signal corresponds to a fault.
- In *document classification* we may be interested in determining whether a given news article is about a given topic of interest, or not.
- In *time-series prediction*, we may try to determine whether a future value (such as a stock price) will increase or decrease with respect to its current value.

Features. Each data point in the classification problem is a vector that contains a numerical representation the *features* that we use to make a class prediction. Features may include:

- frequencies (or some other numerical score) of given words in a dictionary (see the [bag-of-words representation of text](#));
- Boolean variables that determine the presence or absence of a specific feature (such as whether a specific actor played in the movie that the data point represents);
- Numerical values such as blood pressure, temperature, prices, etc.

Linear classification

Assume we are given a collection of data points, $x_i \in \mathbf{R}^n$, which comes with a label $y_i \in \{-1, 1\}$ that determines which class it belongs to.

The linear binary classification problems involves a “linear boundary”, that is a hyperplane. An hyperplane can be described via the equation

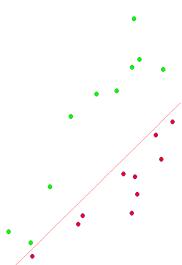
$$a^T x = b,$$

for some $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$.

Such a line is said to correctly classify these two sets if all data points with $y_i = +1$ fall on one side (hence $a^T x \geq b$) and all the others on the other side (hence $a^T x \leq b$). Hence, the affine inequalities on (a, b) :

$$y_i(a^T x_i - b) \geq 0, \quad i = 1, \dots, m,$$

guarantee correct classification. The above constitute linear (in fact, affine) inequalities on the decision variable, $(a, b) \in \mathbf{R}^{n+1}$. This fact is the basis on which we can build a linear programming solution to a classification problem.



The image shows a linear classifier which separates two sets of points in \mathbf{R}^2 . In two dimensions, a hyperplane corresponds to a simple line. In this example, the data sets are linearly separable.

Classification rule. Once a classifier (a, b) is found, we can classify a new point $x \in \mathbf{R}^n$ by assigning to it the label $\hat{y} := \text{sign}(a^T x - b)$.

The above constitutes the *classification rule*.

Strict separability. The data is strictly separable (meaning that the separating hyperplane does not contain any data points) if and only if the above inequalities can be made strict for some choice of (a, b) . If that is the case, then we can always scale the variable (a, b) and obtain the inequalities $y_i(a^T x_i - b) \geq 1, \quad i = 1, \dots, m$. Case of non-separable data sets. The previous constraints are feasible if and only if the data is strictly separable. If the data is not strictly separable, we can allow for errors in the inequalities, and minimize the sum of these errors.

We obtain the problem

$$\min_{a, b, v} \sum_{i=1}^m v_i : v_i \geq 0, \quad y_i(a^T x_i - b) \geq 1 - v_i, \quad i = 1, \dots, m.$$

The geometric interpretation of the above is that we are minimizing the sum of the distances from the wrongly classified points, to the separating hyperplane.

In the above problem, we can eliminate the variable v and obtain a formulation involving the minimization of a [polyhedral function](#):

$$\min_{a,b} \sum_{i=1}^m (1 - y_i(a^T x_i - b))_+$$

where the notation $\alpha_+ := \max(\alpha, 0)$ denotes the positive part of a real number α .

Feature selection Motivation. In many cases, a *sparse* classifier, that is, a vector a with many zeros, is desirable.

Indeed, the classification rule $\hat{y} := \text{sign}(a^T x - b)$ involves the scalar product between the classifier vector a and a feature vector x . If $a_i = 0$ for some i , then the rule ignores the value of the i -th feature to make a prediction about the label. In this sense, the i -th feature is “not important” in the classification task. Thus, the classifier allows not only to make a prediction, but also to *single out* important features in the data. This is referred to as *feature selection*.

Feature selection via l_1 -norm. Assume that the data is separable. We can try to minimize the cardinality of (number of non-zero elements in) the classifier vector a , under the constraint that the classifier (a, b) has no errors on the data set. This is a hard problem.

Instead, we can use the [l₁-norm heuristic](#), which consists of replacing the cardinality objective by the l_1 -norm of a . In the separable case, we end up with the constrained polyhedral minimization problem

$$\min_{a,b} \|a\|_1 : y_i(a^T x_i - b) \geq 1, \quad i = 1, \dots, m.$$

The above can be formulated as an LP.

If the data is not separable, we need to trade-off our concern for sparsity, against the error function we used above. This is done via the polyhedral minimization problem:

$$\sum_{i=1}^m (1 - y_i(a^T x_i - b))_+ + \lambda \|a\|_1,$$

where $\lambda > 0$ is a penalty parameter that allows to choose our trade-off.

Robustness

See the related discussion [here](#) on the interplay between robustness and penalization here.

Against box uncertainty. In some cases, the data points are not known exactly. A typical uncertainty model is to assume that each data point x_i is only known to belong to a “box” around a given point \hat{x}_i : $\|x_i - \hat{x}_i\|_\infty \leq \rho$,

where $\rho > 0$ is a measure of the size of the uncertainty, and \hat{x}_i 's represent the (known) “nominal” values of the data points.

Let us assume that the nominal data points are strictly separable. A *robust classifier* corresponds to a hyperplane that not only separates the known points \hat{x}_i , but the boxes around them. That is, a robust hyperplane satisfies, for every $i = 1, \dots, m$:

$$\forall x_i : \|x_i - \hat{x}_i\|_\infty \leq \rho : y_i(a^T x_i - b) \geq 0$$

Let $x_i = \hat{x}_i + \rho \delta_i$ with $\|\delta_i\|_\infty \leq 1$. The above condition writes

$$\forall \delta_i, \|\delta_i\|_\infty \leq 1 : y_i(a^T \hat{x}_i - b) \geq -\rho y_i a^T \delta_i,$$

which in turn can be written

$$y_i(a^T \hat{x}_i - b) \geq \max_{\delta: \|\delta\|_\infty \leq 1} (-\rho y_i a^T \delta) = \rho \|a\|_1,$$

where we have used the [generalized Cauchy-Schwartz inequality](#) property, valid for any vector a :

$$\max_{\delta: \|\delta\|_\infty \leq 1} a^T \delta = \|a\|_1$$

The above condition is equivalent to

$$y_i(a^T x_i - b) \geq \rho \|a\|_1, \quad i = 1, \dots, m.$$

The above condition guarantees that the data points remain separated even if we perturb the original ones (staying in the boxes). This approach requires the knowledge of the “uncertainty size”, ρ .

If ρ is not known, we can simply find a classifier which maximizes the allowable perturbation level ρ . Using homogeneity, we can always enforce $\rho \|a\|_1 = 1$. Then, maximizing ρ is equivalent to minimizing the l_1 -norm, that is:

$$\min_{a,b} \|a\|_1 : y_i(a^T x_i - b) \geq 1, \quad i = 1, \dots, m.$$

This model is the same as the one we saw for feature selection based on the l_1 -norm heuristic.

Against spherical uncertainty. We can consider another data uncertainty model, in which the data points are only known to belong to hyper-spheres: $\|x_i - \hat{x}_i\|_2 \leq \rho$,

where $\rho > 0$ is a measure of the size of the uncertainty.

Assuming again the nominal data points to be strictly separable, leads to the problem

$$\min_{a,b} \|a\|_2 : y_i(a^T x_i - b) \geq 1, \quad i = 1, \dots, m.$$

The above is a QP.

If the data is not separable, we need to trade-off our concern for robustness, against the error function we used above. This is done via the QP problem:

$$\min_{a,b} \sum_{i=1}^m (1 - y_i(a^T x_i - b))_+ + \lambda \|a\|_2^2,$$

where $\lambda > 0$ is a penalty parameter that allows to choose our trade-off.

Piece-wise constant fitting

We observe a noisy time-series (encoded in a finite-dimensional vector y) which is almost piece-wise constant. The goal in piece-wise constant fitting is to find what the constant levels are. In biological or medical applications, such levels might have interpretations as "states" of the system measured.

Unfortunately, the signal is not exactly piece-wise constant, but a noisy version of such a signal. Thus, we seek to obtain an estimate of the signal, say \hat{x} , such that \hat{x} has as few changes in it as possible. We model the latter by minimizing the cardinality of the "difference" vector Dx , where D is the *difference* matrix

$$D = \begin{bmatrix} -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & \dots & 0 \\ & & \ddots & & \\ \dots & & 0 & -1 & 1 \end{bmatrix}.$$

Matrix D is constructed so that $Dx = (x_2 - x_1, x_3 - x_2, \dots, x_n - x_{n-1})$.

We are led to the problem

$$\min_x \|x - y\|_2^2 : \text{Card}(Dx) \leq K$$

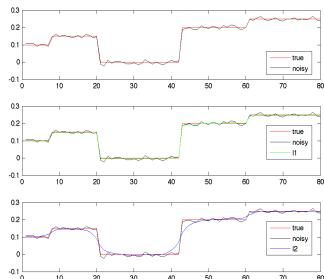
where K is an estimate on the number of jumps in the signal. Here, objective function in the problem is a measure of the error between the noisy measurement and the estimate \hat{x} .

The l_1 -norm heuristic consists in replacing the top (hard) problem by the QP:

$$\min_x \|x - y\|_2^2 : \|Dx\|_1 \leq K.$$

CVX implementation:

```
cvx_begin
    variable x(n)
    minimize( sum_square(x-y) )
    subject to
        norm(D*x,1) <= K;
cvx_end
```

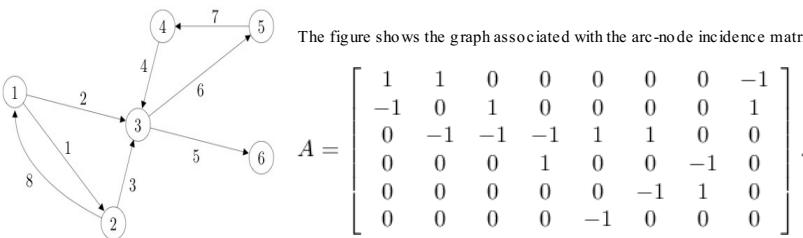


Piece-wise constant fitting. Top panel: the true signal and its noisy version. Middle: l_1 -constrained fitting, showing a good fit with the true (unknown) signal. Bottom: l_2 -constrained fitting.

Network Flows [LP and QP](#) > [Applications](#) > [Back](#) | Networks | [Next](#)

- Network models
- Examples

Network models What is a network? We return to the network model described [here](#). We consider a network (directed graph) having m nodes connected by n directed arcs (ordered pairs (i, j)). We assume there is at most one arc from node i to node j , and no self-loops. We define the *arc-node incidence matrix* $A \in \mathbf{R}^{m \times n}$ to be the matrix with coefficients $A_{ij} = 1$ if arc j starts at node i , -1 if it ends there, and 0 otherwise. Note that the column sums of A are zero: $\mathbf{1}^T A = 0$.



Flows. A flow (of traffic, information, charge) is represented by a vector $x \in \mathbf{R}^n$, and the {em total flow leaving node i } is then $(Ax)_i = \sum_{j=1}^n A_{ij}x_j$. Examples Minimum cost network flow. The *minimum cost network flow* problem has the LP form

where c_i is the cost of flow through arc i , l, u provide upper and lower bounds on x and $b \in \mathbf{R}^m$ is an *external supply* vector. This vector may have positive or negative components, as it represents supply and demand. We assume that $\mathbf{1}^T b = 0$, so that the total supply equals the total demand. The constraint $Ax = b$ represents the balance equations of the network.

Maximum flow. A more specific example is the *max flow* problem, where we seek to maximize the flow between node 1 (the source) and node m (the sink). It bears the form

$$\min_{x,t} t : Ax = te, \quad l \leq x \leq u,$$

with $e^T = (1, 0, \dots, 0, 1)$.

LP Relaxations of Boolean Problems [LP and QP > Applications > Back](#) | Boolean problems | [Next](#)

- Boolean problems
- LP relaxation
- When is the LP relaxation exact?

Boolean problems

A Boolean optimization problem is one where the variables are constrained to be Boolean. An example of Boolean problem is the so-called Boolean LP

$$p^* = \min_x c^T x : Ax \leq b, \quad x \in \{0, 1\}^n.$$

Such problems are usually very hard to solve exactly: this would require a complete enumeration of all the exponential number of possible points in $\{0, 1\}^n$.

LP relaxation

The LP relaxation takes the form

$$p_{LP}^* := \min_x c^T x : Ax \leq b, \quad 0 \leq x \leq 1.$$

The relaxation provides a lower bound on the original problem: $p_{LP}^* \leq p^*$. Hence, its optimal points may not be feasible (not Boolean). Even though a solution of the LP relaxation may not necessarily be Boolean, we can often interpret it as a (non fractional) solution to the original problem. For example, in a graph coloring problem, the LP relaxation colors the nodes of the graph not with a single color, but with many.

Exact solutions

Boolean problems are not always hard to solve. Indeed, in some cases, one can show that the LP relaxation provides an exact solution to the Boolean problem, as optimal points turn out to be Boolean. A few examples in this category:

Weighted bipartite matching. The *weighted bipartite matching* problem is to match N people to N tasks, in a one-to-one fashion. The cost of matching person i to task j is A_{ij} . The problem reads

$$\begin{aligned} \min_x \sum_{i,j=1}^N A_{ij} x_{ij} : \quad & x_{ij} \in \{0, 1\} \\ & \forall j, \sum_{i=1}^N x_{ij} = 1 \quad (\text{one person for each task}) \\ & \forall i, \sum_{j=1}^N x_{ij} = 1 \quad (\text{one task for each person}) \end{aligned}$$

Shortest path. The *shortest path* problem has the form

$$\min_x \mathbf{1}^T x : Ax = (1, 0, \dots, 0, -1), \quad x \in \{0, 1\}^n,$$

where A stands for the [incidence matrix](#) of the network, and arcs with $x_i = 1$ form a shortest forward path between nodes 1 and m . As before the LP relaxation in this case is exact, in the sense that its solution is Boolean. The shortest path problem can be solved very efficiently with specialized algorithms based on the LP relaxation to the above formulation.

Mean-Variance Trade-Off in Portfolio Optimization [LP and QP > Applications > Back](#) | Portfolio Optimization | [Next](#) | Portfolio optimization with sign constraints

We return to the [portfolio optimization problem](#). This time, we assume that shorting is not allowed. In other words, we can only invest in any given asset, or decide not to invest; but we can't borrow and hold any negative position. The problem we have introduced now takes the form

$$\min_x x^T \Sigma x : \hat{r}^T x = \mu, \quad x \geq 0,$$

where μ is our target for the nominal return, and Σ is a positive semi-definite matrix that contains information about the uncertainty on the assets' returns (see [here](#)). The above is a QP.

CVX implementation

```
cvx_begin
    variable x(n,1);
    minimize( x'*S*x );
    subject to
        rhat'*x == mu;
        x >= 0;
cvx_end
```

Other constraints

We can add other linear or affine inequality constraints to the problem.

Sector bounds

For example, we can put upper bounds on some elements of x , to limit the amount of investment in any single asset. We can also require that the total sum invested in a given sector (corresponding to say, the first k assets) does not exceed a fraction α of the total sum invested. This can be modeled by

$$\min_x x^T \Sigma x : \hat{r}^T x = \mu, \quad x \geq 0, \quad \sum_{i=1}^k x_i \leq \alpha \left(\sum_{i=1}^n x_i \right).$$

Again, the above is a QP, with just one more affine inequality.

Diversification

We can also impose some diversification constraints. For example, we may impose that no group of k (with $k < n$) assets contain more than 80% of the total invested. We write this as

$$s_k(x) := \sum_{i=1}^k x_{[i]} \leq 0.8 \sum_{i=1}^n x_i,$$

where $x_{[i]}$ is the i -th largest component of x , so that $s_k(x)$ is the sum of the k largest elements in x . The function $s_k(x)$ is convex, since it is the [point-wise maximum](#) of linear functions (hence it is [polyhedral](#)):

$$s_k(x) = \max_{(i_1, \dots, i_k) \in \{1, \dots, n\}^k} x_{i_1} + \dots + x_{i_k}.$$

The function s_k is implemented in CVX by invoking `sum_largest`.

CVX implementation

```
cvx_begin
    variable x(n,1);
    minimize( x'*S*x );
    subject to
        rhat'*x == mu;
        x >= 0;
        sum_largest(x,10) <= 0.8*sum(x);
cvx_end
```

The above expression for s_k simply states that the function s_k is the pointwise maximum of the $C_{n,k}$ (n choose k) linear functions obtained by choosing any subset of k indices in $\{1, \dots, n\}$. For example, with $k = 2$ and $n = 3$:

$$s_2(x) = \max(x_1 + x_2, x_2 + x_3, x_3 + x_1).$$

Since s_k is polyhedral, we can definitely represent the constraint $s_k(x) \leq 0.8 \sum_i x_i$ as a finite set of affine inequality constraints, that is, a polyhedron. However the list of constraints is huge: for $n = 100, k = 10$, we have to list more than 10^{13} constraints!

A more efficient way is based in the following expression, proven [here](#):

$$s_k(x) = \min_t kt + \sum_{i=1}^n \max(0, x_i - t).$$

In this form, the diversification constraint can be expressed as: there exist a scalar t and a n -vector u such that

$$kt + \sum_{i=1}^n u_i \leq 0.8 \sum_{i=1}^n x_i, \quad u \geq 0, \quad u \geq x - t\mathbf{1}.$$

In the above, $\mathbf{1}$ is the vector of ones in \mathbf{R}^n . Here, t, u are additional variables that allow a QP representation with a moderate number of variables and constraints:

$$\begin{aligned} \min_{u,t,x} x^T \Sigma x : \quad & \hat{r}^T x = \mu, \quad x \geq 0, \\ & kt + \sum_{i=1}^n u_i \leq 0.8 \sum_{i=1}^n x_i, \\ & u \geq 0, \quad u \geq x - t\mathbf{1}. \end{aligned}$$

The lesson here is that a polyhedron in a n -dimensional space, with an exponential number of facets, can be represented as a polyhedron in a higher dimensional space (in our case, with dimension $2n + 1$), with a moderate number of facets (precisely $2n + 1$ facets). By adding just a few dimensions to the problem we are able to deal (implicitly) with a very high number of constraints.

Filter Design Finite Impulse Response filters. A single-input, single-output filter is a dynamical system, with input a signal with values in $u(t) \in \mathbf{R}$, and output another signal with values $y(t) \in \mathbf{R}$. Here t stands for the (discrete) time variable. Finite impulse response filter. A finite-impulse response (FIR) filter is a particular type of filter which has the form

$$y(t) = \sum_{i=0}^{n-1} h_i u(t-i), \quad t \in \mathbb{Z}.$$

The vector $h = (h_0, \dots, h_{n-1}) \in \mathbf{R}^n$ is called the *impulse response* of the filter. Its name derives from the fact that the response of the filter above to the so-called impulse

$$u(t) = \begin{cases} 1 & \text{if } t = 0, \\ 0 & \text{otherwise,} \end{cases}$$

is precisely the *finitely supported* signal

$$y(t) = \begin{cases} h_t & \text{if } 0 \leq t \leq n-1, \\ 0 & \text{otherwise.} \end{cases}$$

Frequency response. The Fourier transform of the impulse response is a complex-valued function $H : \mathbf{R} \rightarrow \mathbf{C}$ with values

$$H(\omega) = \sum_{t=0}^{n-1} h_t e^{-j\omega t}.$$

This function is of importance, since it dictates how the filter responds to periodic signals. Precisely, if the input is a complex exponential $u(t) = e^{j\omega t}$, then the output will be the scaled complex exponential $y(t) = H(\omega)u(t)$.

Example: An example of a FIR filter is a moving average filter. The moving average filter of length 2 is of the form

$$y(t) = \frac{1}{2}(u(t) + u(t-1)), \quad t \in \mathbf{R}.$$

Linear-phase FIR filters. Linear phase FIR filters turns out to be simpler to design. Such filters involves an odd number of taps: $n = 2N + 1$, with an impulse response that is symmetric around the midpoint: $h_t = h_{n-1-t}$; $t = 0, \dots, n-1$.

The qualifier *linear phase* comes from the fact that, for such filters, the frequency response bears the form

$$H(\omega) = e^{-j\omega N} \tilde{H}(\omega),$$

where \tilde{H} is the *em real-valued* function with values

$$\tilde{H}(\omega) := 2h_0 \cos(N\omega) + 2h_1 \cos((N-1)\omega) + \dots + h_N.$$

Hence, the phase of the complex number $H(\omega)$ is a linear function of ω .

A design problem

A design problem involving FIR filters typically involves the FIR filter's impulse response as a variable. The goal is to adjust the response of the filter under a variety of inputs.

For example, we would like to ensure that the filter rejects high-frequency signals, but amplifies low-frequency ones to a certain degree. These two requirements involve the magnitude of the filter's frequency response.

- *Stop-band* constraint: the first constraint may look like

$$|H(\omega)| \leq \delta_2, \quad \omega \geq \Omega_s,$$

where Ω_s is a "stop-band" frequency bound and δ_2 corresponds to an attenuation level we seek to achieve at high frequencies.

- *Pass-band* constraint: The second constraint may look like

$$\delta_1 \leq |H(\omega)|, \quad \omega \leq \Omega_p,$$

where Ω_p is a "pass-band" frequency bound and δ_1 corresponds to a lower bound on the amplification level that we seek to achieve at low frequencies.

The first constraint is convex in the design variable h . The second is not. In addition, both constraints actually involve an {em infinite} number of constraints, one for each frequency ω .

LP formulation issue of non-convexity. To address the issue of non-convexity, we restrict our search for filters, to filters that are linear-phase. In that case, the magnitude of the frequency response can be written as $|H(\omega)| = |\tilde{H}(\omega)|$,

where now $\tilde{H}(\omega)$ is a real number. Without loss of generality, we can choose the impulse response vector h to be such that $\tilde{H}(0) > 0$. Then the pass-band constraint

$$\delta_1 \leq |\tilde{H}(\omega)|, \quad 0 \leq \omega \leq \Omega_p,$$

becomes

$$\delta_1 \leq \tilde{H}(\omega), \quad 0 \leq \omega \leq \Omega_p,$$

which is now convex in the design variable h . In fact, it is an infinite number of affine inequalities, of the form

$$2h_0 \cos(N\omega) + 2h_1 \cos((N-1)\omega) + \dots + h_N \geq \delta_1, \quad 0 \leq \omega \leq \Omega_p.$$

The stop-band constraint is also an infinite number of affine inequalities:

$$2h_0 \cos(N\omega) + 2h_1 \cos((N-1)\omega) + \dots + h_N \leq \delta_2, \quad \omega \geq \Omega_s.$$

Frequency discretization. To address the issue of an infinite number of constraints, we simply *discretize* the frequency domain. Instead of enforcing the constraints for every frequency in an interval, we simply enforce them on a finite set inside the interval.

Let us choose a finite set of frequencies $\omega_i, i = 1, \dots, N_s$ that belong to the high-frequency region $[\Omega_s, +\infty]$. The stop-band constraint is now approximated by the {em finite} number of affine inequalities in h :

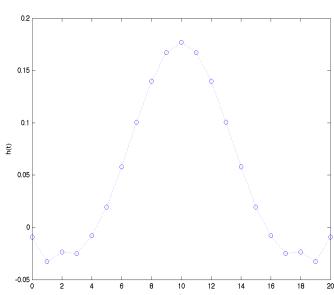
$$2h_0 \cos(N\omega_i) + 2h_1 \cos((N-1)\omega_i) + \dots + h_N \leq \delta_2, \quad i = 1, \dots, N_s.$$

Likewise we choose another set of frequencies $\omega_i, i = N_s + 1, \dots, N_s + N_p$ that belong to the low-frequency region $[0, \Omega_p]$. The pass-band constraint is also approximated by a finite number of affine inequalities in h :

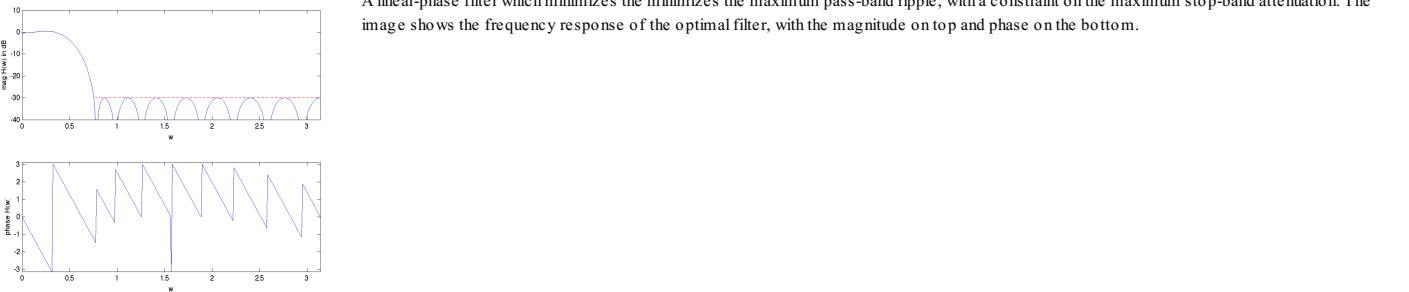
$$2h_0 \cos(N\omega_i) + 2h_1 \cos((N-1)\omega_i) + \dots + h_N \geq \delta_1, \quad i = N_s + 1, \dots, N_s + N_p.$$

Together, the constraints can be written as $h \in \mathcal{P}$, where $\mathcal{P} \subseteq \mathbf{R}^n$ is a polytope.

Variant: pass-band ripple minimization. A variant for the above is to consider both a lower and upper bound on the low-frequency response. Often it is desirable to have a low pass-band *ripple*, which is a measure on the "flatness" of the frequency response's magnitude at low frequencies. A ripple level of δ_1 is enforced via the constraint $\delta_1 \geq 2h_0 \cos(N\omega_i) + 2h_1 \cos((N-1)\omega_i) + \dots + h_N \geq 1/\delta_1, \quad i = N_s + 1, \dots, N_s + N_p$.



A linear-phase filter which minimizes the maximum pass-band ripple, with a constraint on the maximum stop-band attenuation. The image shows the impulse response of the optimal filter.



Trade-off curves. Based on these constraints, we may plot the *trade-off* curve between high-frequency attenuation against low-frequency amplification.

Magnitude constraints on affine complex vectors

Many design problems involve complex variables and magnitude constraints. Such constraints can be often handled via SOCP.

The basic idea is that the magnitude of a complex number $z = z_R + jz_I$, with z_R, z_I the real and imaginary parts, can be expressed as the Euclidean norm of the 2-vector (z_R, z_I) :

$$|z| = \sqrt{z_R^2 + z_I^2} = \left\| \begin{pmatrix} z_R \\ z_I \end{pmatrix} \right\|_2.$$

For example, consider a problem involving a magnitude constraint on a complex number $f(x)$, where $x \in \mathbf{R}^n$ is a design variable, and the complex-valued function $f : \mathbf{R}^n \rightarrow \mathbf{C}$ is affine. The values of such a function can be written as

$$f(x) = (a_R^T x + b_R) + j(a_I^T x + b_I),$$

where $a_R, a_I \in \mathbf{R}^n, b_R, b_I \in \mathbf{R}$.

For $t \in \mathbf{R}$, the magnitude constraint

$$|f(x)| \leq t$$

can be written as

$$\left\| \begin{pmatrix} a_R^T x + b_R \\ a_I^T x + b_I \end{pmatrix} \right\|_2 \leq t,$$

which is a second-order cone constraint on (x, t) .

SOCPs include LPs as Special Case

The linear program (LP)

$$\min_x c^T x : a_i^T x \leq b_i, \quad i = 1, \dots, m,$$

can be cast as an SOCP

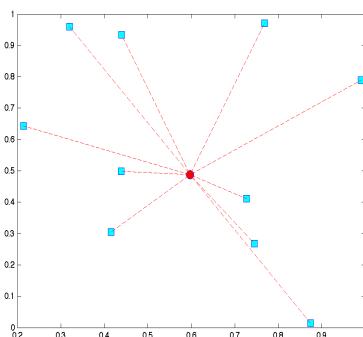
$$\min_x c^T x : \|C_i x + d_i\|_2 \leq b_i - a_i^T x, \quad i = 1, \dots, m,$$

with $C_i = 0, d_i = 0, i = 1, \dots, m$.

Facility Location

Consider the problem of locating a warehouse to serve a number of service locations. The design variable is the location of the warehouse, $x \in \mathbf{R}^2$, while the service locations are given by the vector $x_i \in \mathbf{R}^2, i = 1, \dots, N$.

Minimizing the maximum distance



One possible objective function for this problem involves the *maximum distance* from the warehouse to any location. This is the problem

$$\min_x \max_{1 \leq i \leq N} \|x - x_i\|_2,$$

which can be cast as the SOCP

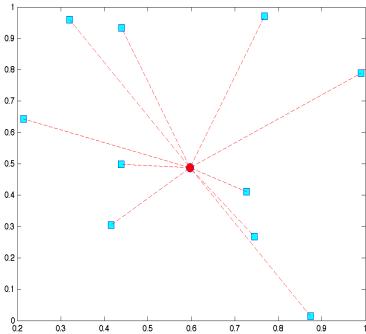
$$\min_{x, t} t : t \geq \|x - x_i\|_2, \quad i = 1, \dots, m.$$

This can be coded up with the CVX applet:

CVX syntax

```
% pts is a 2 x n matrix of n points
cvx_begin
    variable x(2,1);
    variable t(n,1);
    minimize( max(t) )
    subject to
        for i = 1:n,
            t(i) >= norm(x-pts(:,i));
        end
cvx_end
```

Minimizing the average distance



Often, the *average distance* is a good measure of the transportation costs involved. This leads to the problem

$$\min_x \frac{1}{N} \sum_{i=1}^N \|x - x_i\|_2,$$

which can be cast as the SOCP

$$\min_{x,y} \frac{1}{N} \sum_{i=1}^N y_i : y_i \geq \|x - x_i\|_2, \quad i = 1, \dots, m.$$

This can be coded up with the CVX applet:

CVX syntax

```
% pts is a 2 x n matrix of n points
cvx_begin
    variable x(2,1);
    variable t(n,1);
    minimize( max(t) )
    subject to
        for i = 1:n,
            t(i) >= norm(x-pts(:,i));
        end
cvx_end
```

Robust Least-Squares

We start from the least-squares problem:

$$\min_x \|Ax - y\|_2,$$

where $A \in \mathbf{R}^{m \times n}$, $y \in \mathbf{R}^m$.

Now assume that the matrix A is imperfectly known. A simple model is to assume that the matrix is only known to be within a certain “distance” (in matrix space) to a given “nominal” matrix \hat{A} . Precisely, let us assume that assume $\|\hat{A} - A\| \leq \rho$, where $\|\cdot\|$ denotes the [largest singular value norm](#), and $\rho \geq 0$ measures the size of the uncertainty. We now address the *robust least-squares problem*:

$$\min_x \max_{\|\Delta A\| \leq \rho} \|(\hat{A} + \Delta A)x - y\|_2.$$

The interpretation of this problem is that it is trying to minimize the *worst-case* value of the residual norm.

For fixed x , and using the fact that the Euclidean norm is convex, we have

$$\|(\hat{A} + \Delta A)x - y\|_2 \leq \|\hat{A}x - y\|_2 + \|(\Delta A)x\|_2.$$

By definition of the [largest singular value norm](#), and given our bound on the size of the uncertainty, we have

$$\|(\Delta A)x\|_2 \leq \|\Delta A\| \|x\|_2 \leq \rho \|x\|_2.$$

The first inequality comes from the very definition of the largest singular value norm, which implies that for any matrix Δ , the Euclidean norm of Δx is bounded by the product of the largest singular value of Δ and the Euclidean norm of x .

Thus, we have a bound on the objective value of the robust problem:

$$\max_{\|\Delta A\| \leq \rho} \|(\hat{A} + \Delta A)x - y\|_2 \leq \|Ax - y\|_2 + \rho \|x\|_2.$$

It turns out that the upper bound is attained by some choice of the matrix ΔA , specifically:

$$\Delta A = \frac{\rho}{\|\hat{A}x - y\|_2 \cdot \|x\|_2} (\hat{A}x - y)x^T.$$

Hence the robust least-squares is equivalent to the problem

$$\min_x \|\hat{A}x - y\|_2 + \rho \|x\|_2$$

The above becomes an SOCP after introducing variables to make the objective linear:

$$\min_{x,u,v} u + \rho v : u \geq \|\widehat{A}x - y\|_2, v \geq \|x\|_2.$$

As given, this SOCP can be solved using [SVD](#) methods. However, problems involving constraints (such as sign constraints on x) cannot be solved by SVD, while they still can be solved with SOCP.

Separation of Ellipsoids

We consider the following geometrical problem: find a hyperplane that separates two ellipsoids in \mathbf{R}^n , or determine there is none.

Putting a sphere in a half-space

We first seek a condition that determines whether the sphere of radius 1 centered at $\mathbf{0}$ is entirely contained in the half-space

$$\mathbf{H} := \{x : a^T x \leq b\},$$

where $a \in \mathbf{R}^n$, $b \in \mathbf{R}$ are given.

The condition is equivalent to the fact that

$$b \geq \max_{x : \|x\|_2 \leq 1} a^T x.$$

In view of the Cauchy-Schwartz inequality, we have

$$\forall x, \|x\|_2 \leq 1 : a^T x \leq \|a\|_2 \cdot \|x\|_2 \leq \|a\|_2.$$

Clearly, the right-hand side is attained by some vector on the unit circle, namely $x = a/\|a\|_2$. Thus, the condition we seek is

$$b \geq \|a\|_2.$$

Putting an ellipsoid in a half-space

Now consider the case with an ellipsoid instead of a sphere. An ellipsoid \mathbf{E} can be described as an affine transformation of the unit sphere:

$$\mathbf{E} = \{\hat{x} + Ru : \|u\|_2 \leq 1\},$$

where $\hat{x} \in \mathbf{R}^n$ is the “center”, and R is a matrix that determines the “shape” of the ellipsoid.

The containment condition $\mathbf{H} \supseteq \mathbf{E}$ is equivalent to

$$b \geq \max_{u : \|u\|_2 \leq 1} a^T(\hat{x} + Ru).$$

Using the same argument as before, we obtain the condition

$$b \geq a^T \hat{x} + \|R^T a\|_2.$$

Note that the condition that \mathbf{E} should be on the *other* side of the hyperplane is readily obtained by changing a, b in $-a, -b$, that is:

$$b \leq a^T \hat{x} - \|R^T a\|_2.$$

Ellipsoid separation

$$\mathbf{E}_i = \{\hat{x}_i + R_i u : \|u\|_2 \leq 1\}, \quad i = 1, 2,$$

where $\hat{x}_i \in \mathbf{R}^n$ are the centers, and R_i the shape matrices, $i = 1, 2$.

The hyperplane $\{x : a^T x = b\}$ separates the two ellipsoids if and only if

$$b_1 := a^T \hat{x}_1 + \|R_1^T a\|_2 \leq b \leq a^T \hat{x}_2 - \|R_2^T a\|_2 := b_2.$$

Thus, the existence of such a separating hyperplane is equivalent to the existence of $a \in \mathbf{R}^n$ such that

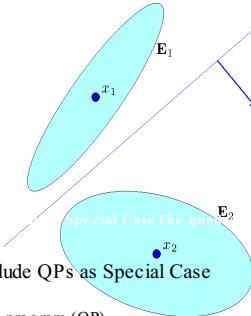
$$a^T(\hat{x}_2 - \hat{x}_1) \geq \|R_1^T a\|_2 + \|R_2^T a\|_2.$$

Exploiting the homogeneity with respect to a , we can always normalize the condition so that $a^T \hat{x}_2 - a^T \hat{x}_1 = 1$. The SOCP

$$p^* = \min_a \|R_1^T a\|_2 + \|R_2^T a\|_2 : a^T \hat{x}_2 - a^T \hat{x}_1 = 1.$$

allows to determine if the ellipsoids are separable: they are if and only if $p^* \leq 1$. In this case, an appropriate value of b is

$$b = \frac{b_1 + b_2}{2}.$$



SOCPs include QPs as Special Case

The quadratic program (QP)

$$\min_x c^T x + x^T Q x : a_i^T x \leq b_i, \quad i = 1, \dots, m,$$

where $Q = Q^T \succeq 0$, can be cast as an SOCP, as follows.

First we introduce a new variable and a new linear equality. With $w := Q^{1/2}x$, we express the QP as

$$\min_{x,w} c^T x + w^T w : w := Q^{1/2}x, \quad a_i^T x \leq b_i, \quad i = 1, \dots, m.$$

Then, we add two new scalar variables, and a rotated cone constraint, to handle the second term in the objective. The QP bears the form

$$\min_{x,y,z,w} c^T x + y : yz \geq w^T w, \quad z = 1, \quad w = Q^{1/2}x, \quad a_i^T x \leq b_i, \quad i = 1, \dots, m$$

The above is indeed an SOCP:

- The first constraint is a rotated cone constraint;
- The second and the third are affine equalities;
- The last m constraints are ordinary affine inequalities.

SOCPs include QCQPs as a Special Case The quadratically constrained quadratic programming problem

$$\min_x a_0^T x + x^T Q_0 x : x^T Q_i x + a_i^T x \leq b_i, \quad i = 1, \dots, m,$$

where $Q_i \in \mathbf{R}^{n \times n}$, $Q_i \succeq 0$, $i = 1, \dots, m$, can be expressed as an SOCP with rotated second-order cone constraints:

$$\min_{x,t,w_0, \dots, w_m} a_0^T x + t : (w_0, t, 1) \in \mathbf{K}_n, \quad ; w_0 = Q_0^{1/2}x, \\ (w_i, b_i - a_i^T x, 1) \in \mathbf{K}_n, \quad w_i = Q_i^{1/2}x, \quad i = 1, \dots, m.$$

In the above, the notation $Q^{1/2}$ denotes the square-root of a PSD matrix Q .

Proof: We first represent the problem in epigraph form:

$$\min_{x,t} a_0^T x + t : x^T Q_0 x \leq t, \quad x^T Q_i x + a_i^T x \leq b_i, \quad i = 1, \dots, m.$$

Now, a constraint of the form

$$a^T x + x^T Q x \leq b$$

is equivalent to the existence of w, y, z such that

$$w^T w \leq yz, \quad z = 1, \quad w = Q^{1/2}x, \quad y = b - a^T x,$$

where $Q^{1/2}$ is the square-root of the PSD matrix Q .

Applying this to the constraints of the above formulation, we obtain an equivalent representation of the original QCQP:

$$\min_{x,t} a_0^T x + t : w_0^T w_0 \leq t, \quad w_0 = Q_0^{1/2}x, \quad w_i^T w_i \leq b_i - a_i^T x, \quad w_i = Q_i^{1/2}x, \quad i = 1, \dots, m.$$

The above can be written as given in the theorem, as claimed.

Minimum Surface Area [SOCP](#) > [Applications](#) > Minimum Surface Area | [Next](#)

- The minimum surface area problem
- Discretization
- SOCP formulation
- Examples

The minimum surface area problem Consider a surface in \mathbf{R}^3 that is described by a function from the square $C := [0, 1] \times [0, 1]$ to \mathbf{R} . The corresponding surface area is

$$A(f) := \int_C \sqrt{1 + \|\nabla f(x, y)\|_2^2} dx dy.$$

The *minimum surface area* problem is to find the function f which minimizes the area $A(f)$, subject to boundary values. To be specific, we will assume that we are given values of f on the left and right side of the square, that is

$$f(x, 0) = l(x), \quad f(x, 1) = r(x), \quad x \in [0, 1],$$

where $l : \mathbf{R} \rightarrow \mathbf{R}$ and $r : \mathbf{R} \rightarrow \mathbf{R}$ are two given functions.

The above is an infinite-dimensional problem, in the sense that the variable is a function, not a finite-dimensional vector.

Discretization

We can discretize the square with a square grid, with points (ih, jh) , $0 \leq i, j \leq K$, where K is an integer, and where $h = 1/K$ the (uniform) spacing of the grid. We represent the variable of our problem, f , as a matrix $F \in \mathbf{R}^{(K+1) \times (K+1)}$, with elements $F_{ij} = f(ih, jh)$. Similarly, we represent the boundary conditions as vectors of length L, R ,

To approximate the gradient, we start from the first-order expansion of a function of two variables, valid for some small increment h :

$$\frac{\partial F}{\partial x}(x, y) \approx \frac{1}{h}(f(x+h, y) - f(x, y)).$$

We obtain that the gradient of F at a grid point can be approximated as

$$\nabla F(ih, jh) \approx \begin{pmatrix} K(F_{i+1,j} - F_{i,j}) \\ K(F_{i,j+1} - F_{i,j}) \\ 1 \end{pmatrix}, \quad 0 \leq i, j \leq K-1. \quad \text{SOCP formulation}$$

The discretized version of our problem is thus

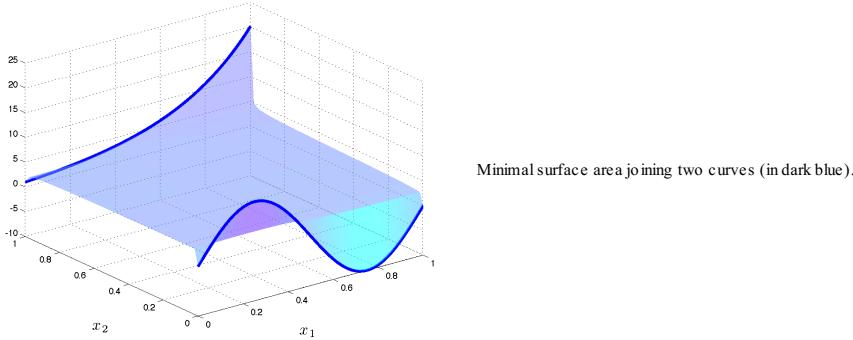
$$\min_F \frac{1}{K^2} \sum_{0 \leq i, j \leq K-1} \left\| \begin{pmatrix} K(F_{i+1,j} - F_{i,j}) \\ K(F_{i,j+1} - F_{i,j}) \\ 1 \end{pmatrix} \right\|_2 : F(i, 0) = l(ih), \quad F(i, 1) = r(ih), \quad 0 \leq i \leq K.$$

The CVX syntax for this problem can be as follows.

CVX syntax

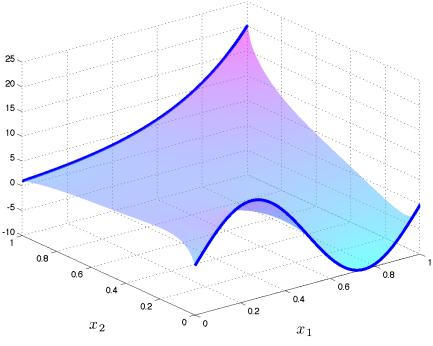
```
>> % input: left_vals and right_vals, two row vectors of length K+1
>> h = 1/K;
cvx_begin
    variables F(K+1,K+1)
    variables T(K,K)
    minimize( sum(T(:)) )
    subject to
        for j = 1:K, for i = 1:K,
            norm([K*(F(i+1,j)-F(i,j)); K*(F(i,j+1)-F(i,j)); 1],2) <= T(i,j);
        end, end
        F(1,:) == left_vals;
        F(K+1,:) == right_vals;
cvx_end
```

Examples



It is interesting to compare the minimal surface area with one that is obtained by squaring the norms. This corresponds to the QP

$$\min_F \frac{1}{K^2} \sum_{0 \leq i, j \leq K-1} \left\| \begin{pmatrix} K(F_{i+1,j} - F_{i,j}) \\ K(F_{i,j+1} - F_{i,j}) \\ 1 \end{pmatrix} \right\|_2^2 : F(i, 0) = l(ih), \quad F(i, 1) = r(ih), \quad 0 \leq i \leq K.$$



A similar problem where surface area is replaced with a least-squares criterion. We observe that, in contrast to the solution above, this new surface has much less “constant” values. In effect the minimal surface area formulation tends to assign zero values to the gradient more aggressively.

Total Variation Image Restoration [SOCP > Applications > Back](#) | Total Variation Image Restoration | [Next](#)

- The image restoration problem
- Discretization
- SOCP formulation

The image restoration problem

Digital images always contain noise. In image restoration, the problem is to filter out the noise. Early methods involved least-squares but the solutions exhibited the ‘‘ringing’’ phenomenon, with spurious oscillations near edges in the restored image. To address this phenomenon, one may add to the objective of the least-squares problem a term which penalizes the variations in the image.

We may represent a given (noisy) image as function from the square $C := [0, 1] \times [0, 1]$ to \mathbf{R} . We define the *image restoration* problem as minimizing, over functions $\hat{f} : C \rightarrow \mathbf{R}$, the objective

$$\int_C \|\nabla \hat{f}(x)\|_2 dx + \lambda \int_C (\hat{f}(x) - f(x))^2 dx,$$

where the function \hat{f} is our estimate. The first term penalizes functions which exhibit large variations, while the second term accounts for the distance from the estimate to the noisy image, f .

The above is an infinite-dimensional problem, in the sense that the variable is a function, not a finite-dimensional vector.

Discretization

We can discretize the square with a square grid, as follows:

$$x_{ij} = \begin{pmatrix} \frac{i}{K} \\ \frac{j}{K} \end{pmatrix}, \quad 0 \leq i, j \leq K.$$

We represent the data of our problem, f , as a matrix $F \in \mathbf{R}^{(K+1) \times (K+1)}$, with elements $F_{ij} = f(x_{ij})$. Similarly, we represent the variable \hat{f} of our problem with a $(K+1) \times (K+1)$ matrix \hat{F} , which contains the values of \hat{f} at the grid points x_{ij} .

To approximate the gradient, we start from the first-order expansion of a function of two variables, valid for some small increment h :

$$\frac{\partial \hat{f}}{\partial x}(x, y) \approx \frac{1}{h}(f(x+h, y) - f(x, y))$$

Applying this to a grid point, with the small increment set to $h = 1/K$, we obtain that the gradient of \hat{f} at a grid point can be approximated as

$$\nabla \hat{f}(x_{ij}) \approx G_{ij} := \begin{pmatrix} K(\hat{F}_{i+1,j} - \hat{F}_{i,j}) \\ K(\hat{F}_{i,j+1} - \hat{F}_{i,j}) \end{pmatrix}, \quad 1 \leq i, j \leq K$$

with the convention that the terms involved are zero on the boundary (that is, if either i or j is n).

SOCP formulation

The discretized version of our problem is thus

$$\min_{\hat{F}} \frac{1}{K^2} \sum_{0 \leq i, j \leq K} \left(\left\| \begin{pmatrix} K(\hat{F}_{i+1,j} - \hat{F}_{i,j}) \\ K(\hat{F}_{i,j+1} - \hat{F}_{i,j}) \end{pmatrix} \right\|_2 + \lambda(\hat{F}_{ij} - F_{ij})^2 \right).$$

Sensor Network Localization [SOCP > Applications > Back](#) | Sensor network localization

In progress; see here.

Uncertainty in the Drug Production Problem

Return to the drug production problem described [here](#).

Box uncertainty model

We now assume a very small variation in some data in the problem. Specifically, we assume that the content of the active agent in the raw materials are subject to variation, with a margin of relative error of 0.5% (raw material I) and 2% (raw material II).

The possible values of the coefficients are shown as intervals, in the following table:

Contents of raw materials:

Raw material	Content of agent A, g per kg
RawI	[0.00995, 0.01005]
RawII	[0.0196, 0.0204]

Impact on solution

Recall the solution to the nominal problem (when uncertainty is ignored):

$$p^* = -8819.658, \quad x_{\text{RawI}} = 0, \quad x_{\text{RawII}} = 438.789, \quad x_{\text{DrugI}} = 17.552, \quad x_{\text{DrugII}} = 0.$$

The uncertainty affects the constraint on the balance of the active agent. In the nominal problem, this constraint was

$$0.01 \cdot x_{\text{RawI}} + 0.02 \cdot x_{\text{RawII}} - 0.05 \cdot x_{\text{DrugI}} - 0.600 \cdot x_{\text{DrugII}} \geq 0.$$

At optimum, this constraint is active. Therefore, even with a tiny error in the first and second coefficient, the constraint becomes invalid.

An adjustment policy

To remedy the problem, there is a simple solution: adjust the levels of production of the drugs, so as to satisfy the balance constraint. Let us adjust the production of Drug I, since that of Drug II is zero according to the original plan.

Clearly, if the actual content of active ingredient increases, the balance constraint will remain valid. In such a case, there is nothing to adjust, and the original production plan is still valid, and optimal. The balance constraint does become invalid only if “nature is against us”, that is when the level of active agent is less than originally thought.

Since the original optimal production plan recommends to purchase only the raw material II, a change in the corresponding coefficient (nominally set at 0.02) to the lesser value 0.0196 results, if we are to adopt the above simple “adjustment policy”, in a variation in the amount of production of Drug I from 17552 packs (the nominal value) to the (2% less) value of 17201 packs. Accordingly, the cost function will decrease from the nominal value of 8,820 to the 21% (!) less value 6,929.

This shows that for this problem, even a tiny variation in a single coefficient can result in a substantial decrease in the profit predicted by the model.

If we are to believe that the uncertain coefficients are actually random, and take their extreme values with $1/2$ probability each, then the expected value of the cost (still with the above adjustment policy) will be also random, with expected value $(8,820 + 6,929)/2 = 7,8745$. Thus, the expected loss due to random uncertainty is still high: 11%.

Antenna Array Design [SOCP](#) > [SOC inequalities](#) | [Standard Forms](#) | [Applications](#) > Antenna Array Design | [Next](#)

In an antenna array the outputs of several emitting antenna elements are linearly combined to produce a composite array output. The array output has a directional pattern that depends on the relative weights or scale factors used in the combining process. The goal of *weight design* is to choose the weights to achieve a desired directional pattern.

- [Physics of antenna arrays](#)
- [Diagram shaping](#)
- [Example](#)

Implementation Errors in an Antenna Array Design Problem

Return to the antenna array design problem described [here](#). Let us specifically consider the problem of minimizing the sidelobe level subject to a normalization constraint, described [here](#). This problem takes the form of an SOCP:

$$\min_{z \in \mathbb{C}^n, \delta} \delta : \operatorname{Re}(D_z(0)) \geq 1, \quad |D_z(\phi_i)| \leq \delta, \quad i = 1, \dots, m. \quad \text{Uncertainty model}$$

We assume that the antenna weights (contained in the complex vector $z^* \in \mathbb{C}^n$) are subject to implementation errors. It is natural to model these errors as *multiplicative perturbations* on the antenna weights, of the form

$$z_k^* \rightarrow (1 + \xi_k) z_k^*$$

where the relative errors $\xi_k \in \mathbf{C}$ are bounded: $|\xi_k| \leq \rho, k = 1, \dots, n$. Here, ρ is a measure of the maximum amount of implementation errors.

Note that such uncertainty errors amount to uncertainties in the coefficients of the original SOCP.

Impact on solution

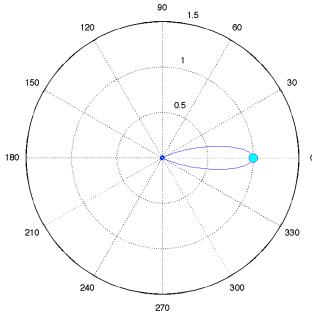
Assume that we have solved the above SOCP, and found a solution z^* .

Now let us perturb the solution according to the uncertainty model above. We generate random samples of the relative error vector $\xi \in \mathbf{C}^n$, inside the box $\{\xi : \|\xi\|_\infty \leq \rho\}$.

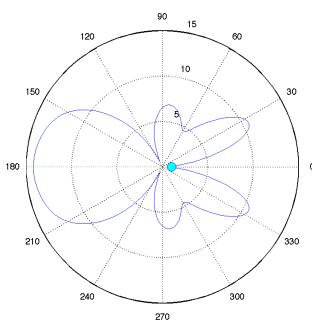
When we change the antenna weights, the normalization requirement $\operatorname{Re}(D_z(0)) \geq 1$ is not necessarily met. To compare diagrams meaningfully, we scale the sample diagrams to make $|D(0)|$ equal to 1, and look at the resulting diagrams.

When solving the problem with nominal data and an equidistant 90-point angular grid, we end up with the nice nominal solution depicted [here](#). For this solution and no data perturbations, the optimal sidelobe level is as low as $\delta^* = 0.0048$.

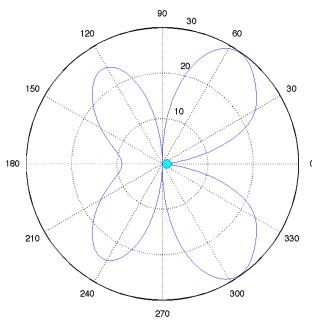
Unfortunately, these nice results are nothing but a dream. In reality, with random actuation errors of level as low as $\rho = 0.01\%$, our supposedly optimal design becomes a *complete disaster*. With 1000 samples, the sidelobe level for the nominal design jumps, on the average, from its nominal value 0.0048 to 1.2342, even attaining a maximum of 29.7889! In fact, in some of these random realizations of the diagram, most of the energy is sent in the opposite direction, as seen below.



The dream: the optimal magnitude diagram. The attenuation is excellent in the stop band, so that the stop-band magnitude is not distinguishable from zero in this plot.



The reality: a particular magnitude diagram obtained after perturbing the optimal weights of the above design with a relative error of 0.01%. This design is a complete disaster: most of the energy is sent in the *opposite* direction from the target.



The reality: another particular magnitude diagram obtained after perturbing the optimal weights of the above design with a relative error of 0.01%. This design is also a complete disaster: the sidelobe level is has jumped from 0.0048 to almost 30, an increase by four orders of magnitude.

What is going on?

What happens is that the optimal antenna weights are quite high in magnitude, to the order $10^3\text{-}10^4$. Thus, even a small relative error in the weights can have consequences. The physics of our setup does not help: the distances between consecutive oscillators is equal to $1/8$ of the wavelength, and the spatial angle of interest — the one where we would like to send as much energy as possible — is comprised of directions whose angular distances from the axial direction are at most 30° .

This is a problem similar to what happens when trying to find a solution to a least-squares problem where the data matrix has low rank: the solution is highly sensitive to changes in the data.

Robust LP Solution to the Drug Production Problem

Return to the drug production problem described [here](#), and the corresponding uncertainty issue discussed [here](#).

Robust LP model

The only constraint affected by uncertainty was the active agent balance inequality. In the nominal problem, this constraint was

$$a_1 \cdot x_{\text{RawI}} + a_2 \cdot x_{\text{RawII}} - 0.05 \cdot x_{\text{DrugI}} - 0.600 \cdot x_{\text{DrugII}} \geq 0,$$

where the nominal values of the coefficients were set to $a_1 = 0.01$ and $a_2 = 0.02$.

The uncertainty affects the coefficients a_1, a_2 , which are assumed to be only known within intervals: $a_1 \in [0.00995, 0.01005]$ and $a_2 \in [0.0196, 0.0204]$.

To obtain a robust solution, we can solve a new problem where we insist that the above constraint holds for every possible value of the coefficients within their respective intervals.

In this problem, since the variables are non-negative, the worst-case value of the coefficients corresponds to a particular scenario where both coefficients are set to their lowest possible value: we set $a_1 = 0.00995$ and $a_2 = 0.0196$ and solve the corresponding LP.

Robust LP solution

The new (robust) solution is then

$$p^* = -8,294.566, \quad x_{\text{RawI}} = 877.7319, \quad x_{\text{RawII}} = 0, \quad x_{\text{DrugI}} = 17.4669, \quad x_{\text{DrugII}} = 0.$$

This is to be compared with the nominal solution, which was:

$$p^* = -8819.658, \quad x_{\text{RawI}} = 0, \quad x_{\text{RawII}} = 438.789, \quad x_{\text{DrugI}} = 17.552, \quad x_{\text{DrugII}} = 0.$$

The profit goes down, by a relatively small amount of 6%. This is to be compared with the expected loss of 11% (and worst-case loss of 22% if we had stuck to our original solution, and tried to adjust it to uncertainty (see [here](#)).

Note also that the qualitative behavior of the robust solution is very different from that of the nominal one. In the latter, we only purchase the raw material II, and produce only Drug I. The robust solution, in contrast, recommends to purchase only raw material II and produce Drug I. This makes sense, as the uncertainty in the active ingredient amount in the raw material II is higher.

Scalar Product and Norms [Vectors, Matrices > Vectors](#) | Scalar products | [Matrices](#) | [Linear functions](#) | [Applications](#)

- Scalar product
- Norms
- Cauchy-schwartz inequality and angles
- Orthogonality
- Hyperplanes and half-spaces

Scalar product

$$x^T y = \sum_{i=1}^n x_i y_i.$$

The scalar product (or, dot product) between two vectors $x, y \in \mathbf{R}^n$ is the scalar denoted $x^T y$, and defined as

The scalar product is sometimes denoted $\langle x, y \rangle$. The motivation for our notation above will come [later](#), when we define the matrix-vector product.

We say that the vectors are *orthogonal* if $x^T y = 0$.

Matlab syntax

```
>> x = [1; 2; 3]; y = [4; 5; 6];
>> scal_prod = x'*y;
```

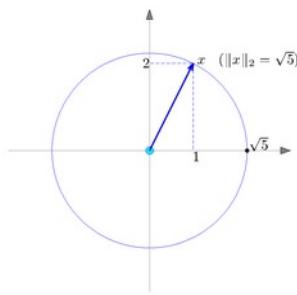
Examples:

- [Two orthogonal vectors in \$\mathbf{R}^3\$](#) .
- [Rate of return of a financial portfolio](#).
- [Sample and weighted average](#).
- [Beer-Lambert law in absorption spectroscopy](#).

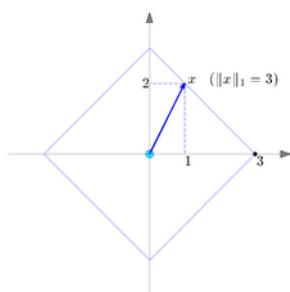
Vector norms

Measuring the size of a scalar value is unambiguous — we just take the magnitude (absolute value) of the number. However, when we deal with higher dimensions, and try to define the notion of size, or length, of a vector, we are faced with many possible choices.

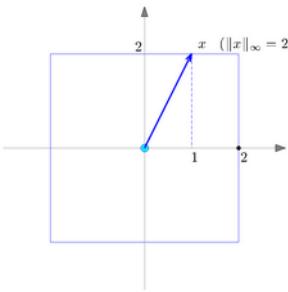
Norms are real-valued functions that satisfy a basic set of rules that a sensible notion of size should involve. You can consult the formal definition of a norm [here](#). In this course, we focus on the following three popular norms for a vector $x \in \mathbf{R}^n$:



The *Euclidean norm*: $\|x\|_2 := \sqrt{\sum_{i=1}^n x_i^2} = \sqrt{x^T x}$, corresponds to the usual notion of distance in two or three dimensions. The set of points with equal l_2 -norm is a circle (in 2D), a sphere (in 3D), or a *hyper-sphere* in higher dimensions.



The *l_1 -norm*: $\|x\|_1 = \sum_{i=1}^n |x_i|$, corresponds to the distance travelled on a rectangular grid to go from one point to another.



The l_{∞} -norm: $\|x\|_{\infty} := \max_{1 \leq i \leq n} |x_i|$, is useful in measuring peak values.

Matlab syntax

```
>> x = [1; 2; -3];
>> r2 = norm(x,2); % 2-norm
>> r1 = norm(x,1); % 1-norm
>> rinf = norm(x,inf); % infinity norm
```

Examples:

- A given vector will in general have different “lengths” under different norms. For example, the vector $x = [1, -2, 3]^T$ yields $\|x\|_2 = 3.7417$, $\|x\|_1 = 6$, and $\|x\|_{\infty} = 3$.
- [Sample standard deviation](#).

Cauchy-Schwartz inequality, angles

The Cauchy-Schwartz inequality allows to bound the scalar product of two vectors in terms of their Euclidean norm.

Cauchy-Schwartz inequality: For any two vectors $x, y \in \mathbf{R}^n$, we have $x^T y \leq \|x\|_2 \cdot \|y\|_2$, with equality if and only if x, y are collinear.

$$\cos \theta = \frac{x^T y}{\|x\|_2 \|y\|_2}.$$

When none of the vectors x, y involved is zero, we can define the corresponding angle as θ such that $\cos \theta = \frac{x^T y}{\|x\|_2 \|y\|_2}$. The notion above generalizes the usual notion of angle between two directions in two dimensions, and is useful in measuring the similarity (or, closeness) between two vectors. When the two vectors are orthogonal, that is, $x^T y = 0$, we obtain that their angle is $\theta = 90^\circ$.

The Cauchy-Schwartz inequality can be generalized to other norms, using the concept of [dual norm](#).

Example:

- [Projection of a vector on a line](#).
- [Similarity of two documents](#).

Orthonormal basis

A basis $(u_i)_{i=1}^n$ is said to be *orthogonal* if $u_i^T u_j = 0$ if $i \neq j$. If in addition, $\|u_i\|_2 = 1$, we say that the basis is *orthonormal*.

Example: An orthonormal basis in \mathbf{R}^3 . The collection of vectors $\{u_1, u_2\}$, with $u_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $u_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, forms an orthonormal basis of \mathbf{R}^2 .

Hyperplanes and half-spaces

A *hyperplane* is a set described by a single affine equality. Precisely, an hyperplane in \mathbf{R}^n is a set of the form $\mathbf{H} = \{x : a^T x = b\}$, where $a \in \mathbf{R}^n$, $a \neq 0$, and $b \in \mathbf{R}$ are given. When $b = 0$, the hyperplane is simply the set of points that are orthogonal to a ; when $b \neq 0$, the hyperplane is a translation, along direction a , of that set.

Hyperplanes are affine sets, of dimension $n - 1$ (see the proof [here](#)). Thus, they generalize the usual notion of a plane in \mathbf{R}^3 . Hyperplanes are very useful because they allow to separate the whole space in two regions. The notion of half-space formalizes this.

Example:

- [A hyperplane in \$\mathbf{R}^3\$](#) .

Half-spaces

A half-space is a subset of \mathbf{R}^n defined by a single affine inequality. Precisely, an half-space in \mathbf{R}^n is a set of the form $\mathbf{H} = \{x : a^T x \leq b\}$, where $a \in \mathbf{R}^n$, $a \neq 0$, and $b \in \mathbf{R}$ are given.

Based on the notion of angle between two vectors, we can understand the meaning of an inequality of the form $a^T x \leq b$, where $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$ are given.

Let us examine the case when $b = 0$. The condition $a^T x > 0$ means that the angle between x and a is acute, while $a^T x > 0$ means the angle is obtuse. The set $\{x : a^T x \leq 0\}$ defines a halfspace with boundary passing through 0, and outward vector a . When $b \neq 0$, the half-space $\{x : a^T x \leq b\}$ is a translated version of the

Strong Duality for QPPrimal Problem

Consider the QP

$$\min_x \frac{1}{2} x^T Q x + c^T x : Ax \leq b,$$

where $c \in \mathbf{R}^n$, $Q \in \mathbf{R}^{n \times n}$, $Q = Q^T \succeq 0$, $A \in \mathbf{R}^{m \times n}$, and $b \in \mathbf{R}^m$. We assume for simplicity that $Q \succ 0$ (that is, Q is positive definite).

Dual problem

Let us form the dual of this problem. The Lagrangian is the function $L : \mathbf{R}^n \times \mathbf{R}^m \rightarrow \mathbf{R}$ with values

$$L(x, \lambda) = \frac{1}{2}x^T Qx + c^T x + \lambda^T (Ax - b),$$

and the dual function is $g : \mathbf{R}^m \times \mathbf{R}^p \rightarrow \mathbf{R}$, with values

$$g(\lambda) = \min_x L(x, \lambda) = \min_x \frac{1}{2}x^T Qx + c^T x + \lambda^T (Ax - b).$$

The above problem is an unconstrained convex optimization problem, so optimal points are characterized by the condition that the gradient of the objective is zero:

$$0 = \nabla_x L(x, \lambda) = Qx + c + A^T \lambda.$$

Since Q is invertible, we obtain the minimizer as $x = -Q^{-1}(A^T \lambda + c)$, and the dual function as

$$g(\lambda) = -\lambda^T b - \frac{1}{2}(A^T \lambda + c)^T A Q^{-1}(A^T \lambda + c).$$

The dual problem reads

$$d^* = \max_{\lambda \geq 0} g(\lambda) = \max_{\lambda \geq 0} -\lambda^T b - \frac{1}{2}(A^T \lambda + c)^T A Q^{-1}(A^T \lambda + c).$$

The dual, just like the primal, is a QP.

We note that the dual feasible set has a particularly simple form, in contrast to that of the primal, which is a generic polytope.

Strong Duality Result

We can apply Slater's theorem to this QP, and obtain that a sufficient condition for strong duality to hold is that the QP is strictly feasible, that is, there exist x_0 such that $Ax_0 < b$.

However, if $Q \succ 0$, it can be shown that *strong duality always holds*.

Strong Duality for LP

Consider the LP

$$\min_x c^T x : Ax \leq b,$$

where $c \in \mathbf{R}^n$, $A \in \mathbf{R}^{m \times n}$, and $b \in \mathbf{R}^m$.

Dual

Let us form the dual of this problem. The Lagrangian is the function $L : \mathbf{R}^n \times \mathbf{R}^m \rightarrow \mathbf{R}$ with values

$$L(x, \lambda) = c^T x + \lambda^T (Ax - b),$$

and the dual function is $g : \mathbf{R}^m \rightarrow \mathbf{R}$, with values

$$g(\lambda) = \min_x L(x, \lambda) = \min_x c^T x + \lambda^T (Ax - b).$$

The above problem involves the minimization of an affine function without any constraints. The optimal value is thus

$$g(\lambda) = \begin{cases} -\lambda^T b & \text{if } A^T \lambda + c = 0 \\ -\infty & \text{otherwise.} \end{cases}$$

The dual problem reads

$$d^* = \max_{\lambda \geq 0} g(\lambda) = \max_{\lambda \geq 0} -\lambda^T b \quad \lambda \geq 0, \quad A^T \lambda + c = 0.$$

The dual, just like the primal, is an LP. We note that the feasible set has a particularly simple form, in contrast to the primal, the feasible set of which is a generic polytope.

Strong duality result

It can be shown that strong duality always holds for LPs, provided either the primal or the dual is feasible. In contrast with [Slater's condition for generic convex problems](#), strict feasibility is not required.

Minimum Euclidean distance to a subspace: strong duality

Consider the problem of finding the minimum Euclidean distance to an affine subspace. The problem can be written as the convex problem

$$p^* = \min_x \frac{1}{2}\|x\|_2^2 : Ax = b,$$

for appropriate matrix $A \in \mathbf{R}^{m \times n}$ and vector $b \in \mathbf{R}^n$. This problem does not have any inequality constraints. We assume that A is full row rank (hence, $AA^T \succ 0$), which guarantees that the problem is feasible for any choice of the vector b .

The dual function is

$$g(\nu) = \min_x \frac{1}{2} \|x\|_2^2 + \nu^T(b - Ax).$$

Weak duality tells us that for any given ν , $g(\nu)$ is a lower bound on the optimal value of the primal problem: $p^* \geq g(\nu)$.

For a given ν , the corresponding value of the dual function, $g(\nu)$, can be computed explicitly, since it involves the unconstrained minimization of a convex (quadratic) function. In fact, there is a unique solution to the above problem:

$$x(\nu) := A^T \nu.$$

Hence

$$g(\nu) = \frac{1}{2} \|x(\nu)\|_2^2 + \nu^T(b - Ax(\nu)) = b^T \nu - \frac{1}{2} \nu^T (AA^T) \nu.$$

The dual problem can also be solved explicitly, since it involves the unconstrained minimization of a convex (quadratic) function again. In fact, when A is full row rank ($AA^T \succ 0$), the optimal ν is unique, and given by $\nu^* = (AA^T)^{-1}b$. Hence, the optimal value of the dual is

$$d^* = \max_{\nu} g(\nu) = \frac{1}{2} b^T (AA^T)^{-1} b.$$

We can check that strong duality holds for this problem. Indeed, Slater's condition holds (the problem is convex, has only affine equality constraints, and these are feasible). We can check this directly, and come up with an optimal point for the primal problem. To see this, we set

$$x^* := x(\nu^*) = A^T (AA^T)^{-1} b.$$

We then observe that x^* is feasible, since $Ax^* = b$, thus:

$$d^* \leq p^* \leq \frac{1}{2} \|x^*\|_2^2.$$

Since the corresponding value of the objective equals the lower bound d^* , that is:

$$\frac{1}{2} \|x^*\|_2^2 = \frac{1}{2} b^T (AA^T)^{-1} b = d^*,$$

we conclude that $p^* \leq \frac{1}{2} \|x^*\|_2^2$, that is, x^* is indeed optimal for the primal problem.

Projection of a vector on a line

The line in \mathbf{R}^n passing through $x_0 \in \mathbf{R}^n$ and with direction $u \in \mathbf{R}^n$, $u \neq 0$, is the set of vectors x such that $x - x_0$ is parallel to u : $\{x_0 + tu : t \in \mathbf{R}\}$. We can always assume without loss of generality that the direction u is normalized, that is $\|u\|_2 = 1$.

A line in \mathbf{R}^2 passing through the point $x_0 = (2, 0)$, with (normalized) direction $u = (0.8944, 0.4472)$.

Definition

The projection of a given point x on the line is a vector z located on the line, that is closest to x (in Euclidean norm). This corresponds to a simple optimization problem: $\min_t \|x - x_0 - tu\|_2^2$. (This particular problem is part of a general class of optimization problems known as [least-squares](#).)

Projection of the vector $x = (1.6, 2.28)$ on a line passing through the origin ($x_0 = 0$) and with (normalized) direction $u = (0.8944, 0.4472)$. At optimality the “residual” vector $x - z$ is orthogonal to the line, hence $z = tu$, with $t = x^T u = 2.0035$. Any other point on the line is farther away from the point x .

Closed-form expression

Assuming that u is normalized, the optimal solution to the above problem is $t^* = u^T(x - x_0)$, and the expression for the projected vector is $z^* = x_0 + t^* u$. In the case when u is not normalized, the expression is $z^* = x_0 + \frac{u^T(x - x_0)}{u^T u} u$.

Proof: Let us first assume that u is normalized. We express the square of the objective function as $t^2 - 2\alpha t + \beta^2 = (t - \alpha)^2 + \text{constant}$, where $\alpha = u^T(x - x_0)$, $\beta = \|p - p_0\|_2^2$. The minimum is clearly attained when the first term is zero, which yields $t^* = u^T(x - x_0)$, as claimed. ♦

A simple 3×2 matrix [Matrices](#) > [Basics](#) > Example

Consider the 3×2 matrix

$$A = \begin{pmatrix} 3 & 4.5 \\ 2 & 1.2 \\ -0.1 & 8.2 \end{pmatrix}.$$

The matrix can be interpreted as the collection of two column vectors: $A = (a_1, a_2)$, where a_j 's contain the columns of A :

$$a_1 = \begin{pmatrix} 3 \\ 2 \\ -0.1 \end{pmatrix}, \quad a_2 = \begin{pmatrix} 4.5 \\ 1.2 \\ 8.2 \end{pmatrix},$$

Geometrically, A represents 2 points in a 3-dimensional space.

$$A = \begin{pmatrix} b_1^T \\ b_2^T \\ b_3^T \end{pmatrix},$$

Alternatively, we can interpret A as a collection of 3 row vectors in \mathbf{R}^2 :

where b_i , $i = 1, 2, 3$ contain the rows of A :

$$b_1 = \begin{pmatrix} 3 \\ 4.5 \end{pmatrix}, \quad b_2 = \begin{pmatrix} 2 \\ 1.2 \end{pmatrix}, \quad b_3 = \begin{pmatrix} -0.1 \\ 8.2 \end{pmatrix}.$$

Geometrically, A represents 3 points in a 2-dimensional space.

A two-dimensional toy optimization problem

As a toy example of an optimization problem in two variables, consider the problem

$$\min_x 0.9x_1^2 - 0.4x_1x_2 - 0.6x_2^2 - 6.4x_1 - 0.8x_2 : -1 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq 3.$$

(Note that the term “subject to” has been replaced with the shorthand colon notation.)

The problem can be put in [standard form](#)

$$p^* := \min_x f_0(x) : f_i(x) \leq 0, \quad i = 1, \dots, m,$$

where:

- the decision variable is $(x_1, x_2) \in \mathbf{R}^2$;
- the objective function $f_0 : \mathbf{R}^2 \rightarrow \mathbf{R}$, takes values

$$f_0(x) := 0.9x_1^2 - 0.4x_1x_2 - 0.6x_2^2 - 6.4x_1 - 0.8x_2;$$

- the constraint functions $f_i : \mathbf{R}^n \rightarrow \mathbf{R}$, $i = 1, 2, 3, 4$ take values

$$\begin{aligned} f_1(x) &:= -x_1 - 1, \\ f_2(x) &:= x_1 - 2, \\ f_3(x) &:= -x_2, \\ f_4(x) &:= x_2 - 3. \end{aligned}$$

- p^* is the *optimal value*, which turns out to be $p^* = -10.2667$.

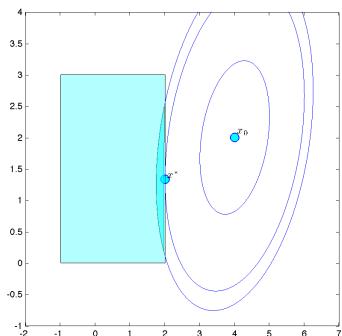
- The optimal set is the singleton $\mathbf{X}^{\text{opt}} = \{x^*\}$, with

$$x^* = \begin{pmatrix} 2.00 \\ 1.33 \end{pmatrix}.$$

Since the optimal set is not empty, the problem is attained.

We can represent the problem in [epigraph form](#), as

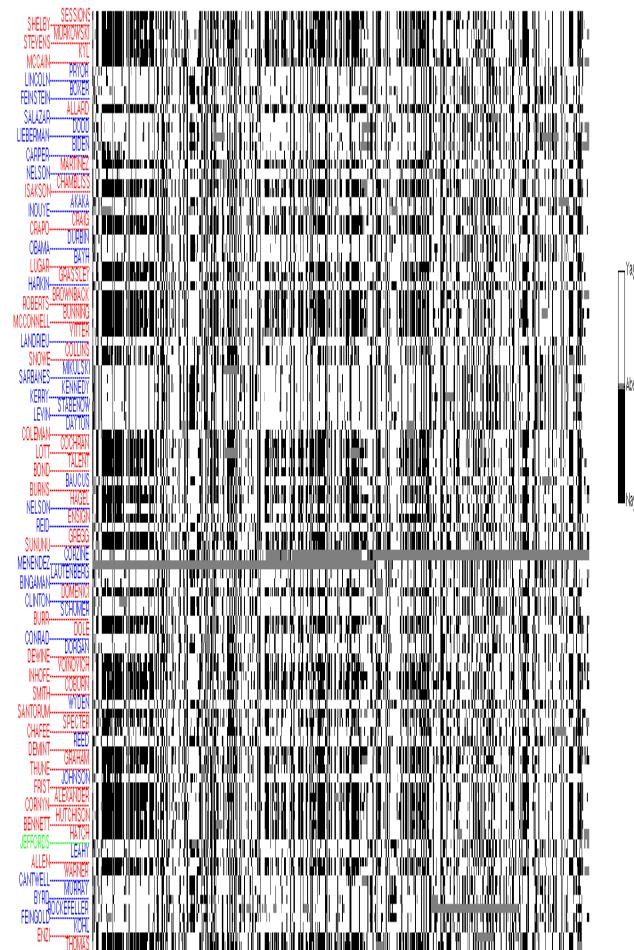
$$\min_{x,t} t : t \geq 0.9x_1^2 - 0.4x_1x_2 - 0.6x_2^2 - 6.4x_1 - 0.8x_2, \quad -1 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq 3.$$



Geometric view of the toy optimization problem above. The level curves (curves of constant value) of the objective function are shown. The problem amounts to find the smallest value of t such that $t = f_0(x)$ for some feasible x . The plot also shows the *unconstrained* minimum of the objective function, located at $\hat{x} = (4, 2)$. An ϵ -[sub-optimal set](#) for the toy problem above is shown (in darker color), for $\epsilon = 0.9$. This corresponds to the set of feasible points that achieves an objective value less or equal than $p^* + \epsilon$.

The data consists of the votes of $n = 100$ Senators in the 2004-2006 US Senate (2004-2006), for a total of $m = 542$ bills. “Yay” (“Yes”) votes are represented as 1’s, “Nay” (“No”) as -1’s, and the other votes are recorded as 0. (A number of complexities are ignored here, such as the possibility of pairing the votes.)

This data can be represented here as a $m \times n$ “voting” matrix $X = [x_1, \dots, x_n]$, with elements taken from $\{-1, 0, 1\}$. Each column of the voting matrix x_j , $j = 1, \dots, n$ contains the votes of a single Senator for all the bills; each row contains the votes of all Senators on a particular bill.



Senate voting matrix: “Nay” votes are in black, “Yay” ones in white, and the others in grey. The transpose voting matrix is shown. The picture becomes very noisy over time, as some Senators are replaced over time. Simply plotting the raw data matrix is often not very informative.

Source: [VoteWorld](#).

Gram matrix

Consider mn -vectors x_1, \dots, x_m . The *Gram matrix* of the collection is the $m \times m$ matrix G with elements $G_{ij} = x_i^T x_j$. The matrix can be expressed compactly in terms of the matrix $X = [x_1, \dots, x_m]$, as

$$G = X^T X = \begin{pmatrix} x_1^T \\ \vdots \\ x_m^T \end{pmatrix} \begin{pmatrix} x_1 & \dots & x_m \end{pmatrix}.$$

By construction, a Gram matrix is always [symmetric](#), meaning that $G_{ij} = G_{ji}$ for every pair (i, j) . It is also [positive semi-definite](#), meaning that $u^T Gu \geq 0$ for every vector $u \in \mathbf{R}^n$ (this comes from the identity $u^T Gu = \|Xu\|_2^2$).

Assume that each vector x_i is normalized: $\|x_i\|_2 = 1$. Then the coefficient G_{ij} can be expressed as

$$G_{ij} = \cos \theta_{ij},$$

where θ_{ij} is the angle between the vectors x_i and x_j . Thus G_{ij} is a measure of how similar x_i and x_j are.

The matrix G arises for example in text document classification, with G_{ij} a measure of similarity between the i -th and j -th document, and x_i, x_j their respective bag-of-words representation (normalized to have Euclidean norm 1).

See also:

- [Bag-of-words representation of text](#).
- [Bag-of-words representation of text: measure of document similarity](#).

Single factor model of financial price data

Consider a $m \times T$ data matrix which contains the [log-returns](#) of m assets over T time periods (say, days).

A *single-factor model* for this data is one based on the assumption that the matrix is a dyad:

$$A = uv^T,$$

where $v \in \mathbf{R}^T$, and $u \in \mathbf{R}^m$. In practice, no component of u and v is zero (if that is not the case, then a whole row or column of A is zero, and can be ignored in the analysis).

According to the single factor model, the entire market behaves as follows. At any time t ($1 \leq t \leq T$), the log-return of asset i ($1 \leq i \leq m$) is of the form

$$A_{it} = u_i v_t.$$

The vectors u and v has the following interpretation.

- For any asset, the rate of change in log-returns between two time instants $t_1 \leq t_2$ is given by the ratio v_{t_2}/v_{t_1} , *independent* of the asset. Hence, v gives the *time profile* for *all* the assets: every asset shows the same time profile, up to a scaling given by u .
- Likewise, for any time t , the ratio between the log-returns of two assets i and j at time t is given by u_i/u_j , *independent* of t . Hence u gives the *asset profile* for *all* the time periods. Each time shows the same asset profile, up to a scaling given by v .

While single-factor models may seem crude, they often offer a reasonable amount of information. It turns out that with many financial market data, a good single factor model involves a time profile v equal to the log-returns of the *average* of all the assets, or some weighted average (such as the SP 500 index). With this model, all assets follow the profile of the entire market.

The QR decomposition of a matrix

- Basic idea
- Case when the matrix has linearly independent columns
- General case
- Full QR decomposition

Basic idea

The basic goal of the QR decomposition is to *factor* a matrix as a product of two matrices (traditionally called Q, R , hence the name of this factorization). Each matrix has a simple structure which can be further exploited in dealing with, say, [linear equations](#).

The QR decomposition is nothing else than the [Gram-Schmidt procedure](#) applied to the columns of the matrix, and with the result expressed in matrix form. Consider a $m \times n$ matrix $A = (a_1, \dots, a_n)$, with each $a_i \in \mathbf{R}^m$ a column of A .

Case when A is full column rank

Assume first that the a_i 's (the columns of A) are linearly independent. Each step of the G-S procedure can be written as

$a_i = (a_i^T q_1)q_1 + \dots + (a_i^T q_{i-1})q_{i-1} + \|\tilde{q}_i\|_2 q_i$, $i = 1, \dots, n$. We write this as $a_i = r_{i1}q_1 + \dots + r_{i,i-1}q_{i-1} + r_{ii}q_i$, $i = 1, \dots, n$, where $r_{ij} = (a_i^T q_j)$ ($1 \leq j \leq i - 1$) and $r_{ii} = \|\tilde{q}_i\|_2$.

Since the q_i 's are unit-length and normalized, the matrix $Q = (q_1, \dots, q_n)$ satisfies $Q^T Q = I_n$. The QR decomposition of a $m \times n$ matrix A thus allows to write the

$$A = QR, \quad Q = \begin{pmatrix} q_1 & \dots & q_n \end{pmatrix}, \quad R = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ 0 & r_{22} & & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & & 0 & r_{nn} \end{pmatrix} \quad \text{matrix in } \text{factored form:}$$

,upper-triangular, where Q is a $m \times n$ matrix with $Q^T Q = I_n$, and R is $n \times n$

Matlab syntax

```
>> [Q,R] = qr(A,0); % A is a mxn matrix, Q is mxn orthogonal, R is nxn upper triangular
```

Example:[QR decomposition of a 4x6 matrix](#).

Case when the columns are not independent

When the columns of A are not independent, at some step of the G-S procedure we encounter a zero vector \tilde{q}_j , which means a_j is a linear combination of a_{j-1}, \dots, a_1 . The [modified Gram-Schmidt procedure](#) then simply skips to the next vector and continues.

$$R = \begin{pmatrix} * & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{pmatrix}.$$

In matrix form, we obtain $A = QR$, with $Q \in \mathbf{R}^{m \times r}$, $r = \text{Rank}(A)$, and R has an upper staircase form, for example: (This is simply an upper triangular matrix with some rows deleted. It is still upper triangular.)

$$R = \begin{pmatrix} R_1 & R_2 \end{pmatrix} P^T, \quad \left(\begin{array}{c|c} R_1 & R_2 \end{array} \right) := \begin{pmatrix} * & * & * & * & * & * \\ 0 & * & 0 & * & * & * \\ 0 & 0 & * & 0 & 0 & * \end{pmatrix},$$

We can permute the columns of R to bring forward the first non-zero elements in each row:

where P is a [permutation matrix](#) (that is, its columns are the unit vectors in some order), whose effect is to permute columns. (Since P is orthogonal, $P^{-1} = P^T$.) Now, R_1 is square, upper triangular, and *invertible*, since none of its diagonal elements is zero.

The QR decomposition can be written $AP = Q \begin{pmatrix} R_1 & R_2 \end{pmatrix}$, where

1. $Q \in \mathbf{R}^{m \times r}$, $Q^T Q = I_r$;
2. r is the rank of A ;
3. R_1 is $r \times r$ upper triangular, invertible matrix;
4. R_2 is a $r \times (n - r)$ matrix;
5. P is a $m \times m$ permutation matrix.

Matlab syntax

```
>> [Q,R,inds] = qr(A,0); % here inds is a permutation vector such that A(:,inds) = Q*R
```

Full QR decomposition

The full QR decomposition allows to write $A = QR$ where $Q \in \mathbf{R}^{m \times m}$ is square and orthogonal ($Q^T Q = QQ^T = I_m$). In other words, the columns of Q are an orthonormal basis for the whole output space \mathbf{R}^m , not just for the range of A .

We obtain the full decomposition by appending an $m \times m$ identity matrix to the columns of A : $A \rightarrow [A, I_m]$. The QR decomposition of the augmented matrix allows to write $AP = QR = (Q_1 \ Q_2) \begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix}$, where the columns of the $m \times m$ matrix $Q = [Q_1, Q_2]$ are orthogonal, and R_1 is upper triangular and invertible. (As before, P is a permutation matrix.) In the G-S procedure, the columns of Q_1 are obtained from those of A , while the columns of Q_2 come from the extra columns added to A .

The full QR decomposition reveals the rank of A : we simply look at the elements on the diagonal of R that are not zero, that is, the size of R_1 .

Matlab syntax

```
>> [Q,R] = qr(A); % A is mxn matrix, Q is mxm orthogonal, R is mxn upper triangular
```

Example: [QR decomposition of a 4x6 matrix](#).

Full Rank Matrices Theorem

A matrix $A \in \mathbf{R}^{m \times n}$ is

- full column rank if and only if $A^T A$ is invertible.
- full row rank if and only if AA^T is invertible.

Proof: The matrix is full column rank if and only if its nullspace is reduced to the singleton $\{0\}$, that is, $Ax = 0 \implies x = 0$. If $A^T A$ is invertible, then indeed the condition $Ax = 0$ implies $A^T Ax = 0$, which in turn implies $x = 0$.

Conversely, assume that the matrix is full column rank, and let x be such that $A^T Ax = 0$. We then have $x^T A^T Ax = \|Ax\|_2^2 = 0$, which means $Ax = 0$. Since A is full column rank, we obtain $x = 0$, as desired.

The proof for the other property follows similar lines.

Rank-nullity theorem

The nullity (dimension of the nullspace) and the rank (dimension of the range) of a $m \times n$ matrix A add up to the column dimension of A , n .

Proof: Let p be the dimension of the nullspace $N(A)$ ($p \leq n$). Let U_1 be a $n \times p$ matrix such that its columns form an orthonormal basis of $N(A)$. In particular, we have $AU_1 = 0$. Using the QR decomposition of the matrix $[U_1, I_n]$ we obtain a $n \times (n - p)$ matrix U_2 such that the matrix $[U_1, U_2]$ is orthogonal. Now define the $m \times (n - p)$ matrix $V := AU_2$.

We proceed to show that the columns of V form a basis for the range of A . To do this, we first prove that the columns of V span the range of A . Then we will show that these $n - p$ columns are independent. This will show that the dimension of the range (that is, the rank) is indeed equal to $n - p$.

Since U is an orthonormal matrix, for any $x \in \mathbf{R}^n$, there exist two vectors x_1, x_2 such that $x = U_1 x_1 + U_2 x_2$. If $x \in \mathbf{R}(A)$, then $Ax = AU_2 x_2 = V x_2 \in \mathbf{R}(V)$. This proves that the columns of V span the range of A : $\mathbf{R}(A) = \mathbf{R}(V)$.

Now let us show that the columns of V are independent. Assume a vector z satisfies $Vz = 0$, and let us show $z = 0$. We have $Vz = AU_2 z = 0$, which implies that $U_2 z$ is in the nullspace of A . Hence there exist another vector y such that $U_2 z = U_1 y$. This is contradicted by the fact that $[U_1, U_2]$ is an orthogonal matrix: pre-multiplying the last equation by U_2^T , and exploiting the fact that $U_2^T U_2 = I_{n-p}$, $U_2^T U_1 = 0$, we obtain $z = U_2^T U_2 z = U_2^T U_1 y = 0$.

Orthogonal complement of a subspace

Let S be a subspace of \mathbf{R}^n . The orthogonal complement of S , denoted S^\perp , is the subspace of \mathbf{R}^n that contains the vectors orthogonal to all the vectors in S .

If the subspace is described as the range of a matrix: $S = \{Ax : x \in \mathbf{R}^n\}$, then the orthogonal complement is the set of vectors orthogonal to the rows of A , which is the nullspace of A^T .

Example: consider the line in \mathbf{R}^3 passing through the origin and generated by the vector $u = (1, 2, 3)$. This is a subspace of dimension 1:

$$S = \{tu : t \in \mathbf{R}\} = \left\{ \begin{pmatrix} t \\ 2t \\ 3t \end{pmatrix} : t \in \mathbf{R} \right\}.$$

To find the orthogonal complement we find the set of vectors that are orthogonal to any vector of the form tu , with arbitrary $t \in \mathbf{R}$. This is the same set as the set of vectors orthogonal to u itself. So we solve for $u^T x = 0$ with $x \in \mathbf{R}^3$: $x_1 + 2x_2 + 3x_3 = 0$. This is equivalent to

$x_1 = -2x_2 - 3x_3$. This equation characterizes the elements of the orthogonal complement S^\perp , in the sense that any $x \in S^\perp$ can be written as $x = \begin{pmatrix} -2\alpha - 3\beta \\ \alpha \\ \beta \end{pmatrix} = \alpha u + \beta v$, for some scalars α, β , where $u = \begin{pmatrix} -2 \\ 1 \\ 0 \end{pmatrix}$, $v = \begin{pmatrix} -3 \\ 0 \\ 1 \end{pmatrix}$. The orthogonal complement is thus the span of the vectors u, v : $S^\perp = \text{span}(u, v)$.

Fundamental theorem of linear algebra

Let $A \in \mathbf{R}^{m \times n}$. The sets $\mathbf{N}(A)$ and $\mathbf{R}(A^T)$ form an *orthogonal decomposition* of \mathbf{R}^n , in the sense that any vector $x \in \mathbf{R}^n$ can be written as $x = y + z$, $y \in \mathbf{N}(A)$, $z \in \mathbf{R}(A^T)$, $y^T z = 0$. In particular, we obtain that the condition on a vector x to be orthogonal to any vector in the nullspace of A implies that it must be in the range of its transpose: $x^T y = 0$ whenever $Ay = 0 \iff \exists \lambda \in \mathbf{R}^m : x = A^T \lambda$.

Proof: The theorem relies on the fact that if a SVD of a matrix A is $A = U \tilde{S} V^T$, $\tilde{S} = \text{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0)$ then an SVD of its transpose is simply obtained by transposing the three-term matrix product involved: $A^T = (U \tilde{S} V^T)^T = V \tilde{S} U^T$. Thus, the left singular vectors of A are the right singular vectors of A^T .

From this we conclude in particular that the range of A^T is spanned by the first r columns of V . Since the nullspace of A is spanned by the last $n - r$ columns of V , we observe that the nullspace of A and the range of A^T are two orthogonal subspaces, whose dimension sum to that of the whole space. Precisely, we can express any given vector x in terms of a linear combination of the columns of V ; the first r columns correspond to the vector $z \in \mathbf{R}(A^T)$ and the last $n - r$ to the vector $y \in \mathbf{N}(A)$:

$$x = V(V^T x) = \underbrace{\sum_{i=1}^r \tilde{x}_i v_i}_{=z} + \underbrace{\sum_{i=r+1}^n \tilde{x}_i v_i}_{=y}$$

This proves the first result in the theorem.

The last statement is then an obvious consequence of this first result: if x is orthogonal to the nullspace, then the vector y in the theorem above must be zero, so that $x \in \mathbf{R}(A^T)$.

Image Compression via Least-Squares

We can use least-squares to represent an image in terms of a linear combination of “basic” images, at least approximately.

An image can be represented, via its pixel values or some other mechanism, as a (usually long) vector $y \in \mathbf{R}^m$. Now assume that we have a library of n “basic” images, also in pixel form, that are represented as vectors a_1, \dots, a_n . Each vector a_j could contain the pixel representation of a unit vector in some basis, such as a two-dimensional Fourier basis; however, we do not necessarily assume here that the a_j 's form a basis.

Let us try to find the best coefficients $x_j, j = 1, \dots, n$, which allow to approximate the given image (given by $y \in \mathbf{R}^m$) as a linear combination of the a_j 's with coefficients x_j . Such a combination can be expressed as the matrix-vector product Ax , where $A = [a_1, \dots, a_n]$ is the $m \times n$ matrix that contains the basic images. The best fit can be found via the least-squares problem

$$\min_x \|Ax - y\|_2.$$

Once the representation is found, and if the optimal value of the problem above is small, we can safely represent the given image via the vector x . If the vector x is sparse, in the sense that it has many zeros, such a representation can yield a substantial saving in memory, over the initial pixel representation y .

The sparsity of the solution of the problem above for a range of possible images y is highly dependent on the images' characteristics, as well as on the collection of basic images contained in A . In practice, it is desirable to trade-off the accuracy measure above, against some measure of sparsity of the optimal vector x .

Set of solutions to the least-squares problem via QR decomposition

The set \mathbf{S} of solutions to the least-squares problem

$$\min_x \|Ax - y\|_2^2,$$

where $A \in \mathbf{R}^{m \times n}$, and $y \in \mathbf{R}^m$ are given, can be expressed in terms of the full QR decomposition of A :

$$AP = QR, \quad R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix},$$

where $R_{11} \in \mathbf{R}^{r \times r}$ is upper triangular and invertible, $R_{12} \in \mathbf{R}^{r \times (n-r)}$, P is a permutation matrix, and Q is $m \times m$ and orthogonal.

Precisely we have $\mathbf{S} = \{x_0 + Nz : z \in \mathbf{R}^{n-r}\}$, with N a matrix whose columns span the nullspace of A :

$$x_0 = P \begin{pmatrix} R_{11}^{-1} y_1 \\ 0 \end{pmatrix}, \quad N = \begin{pmatrix} R_{11}^{-1} R_{12} \\ I \end{pmatrix} \in \mathbf{R}^{n \times (n-r)}.$$

Proof: Since Q and P are orthogonal, we have, with $\bar{x} := P^T x$, $\bar{y} := Q^T y$:

$$Ax - y = APP^T x - y = QR\bar{x} - y = Q(R\bar{x} - Q^T y) = Q(R\bar{x} - \bar{y})$$

Exploiting the fact that Q leaves Euclidean norms invariant, we express the original least-squares problem in the equivalent form:

$$\min_{\bar{x}} \|R\bar{x} - \bar{y}\|_2.$$

Once the above is solved, and \bar{x} is found, we recover the original variable x with $x = P\bar{x}$.

Now let us decompose \bar{x} and \bar{y} in a manner consistent with the block structure of R : $\bar{x} = (x_1, x_2)$, $\bar{y} = (y_1, y_2)$, with x_1, y_1 two r -vectors. Then

$$R\bar{x} - \bar{y} = \begin{pmatrix} R_{11}x_1 + R_{12}x_2 - y_1 \\ -y_2 \end{pmatrix},$$

which leads to the following expression for the objective function:

$$\|R\bar{x} - \bar{y}\|_2^2 = \|R_{11}x_1 + R_{12}x_2 - y_1\|_2^2 + \|y_2\|_2^2$$

The optimal choice for the variables x_1, x_2 is to make the first term zero, which is achievable with

$$x_1 = R_{11}^{-1}(y_1 - R_{12}x_2),$$

where x_2 is free, and describes the ambiguity in the solution. The optimal residual is $\|y_2\|_2^2$.

We are essentially done: with $x_2 = z$, we can write

$$P^T x = \bar{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \begin{pmatrix} R_{11}^{-1}y_1 \\ 0 \end{pmatrix} + \begin{pmatrix} R_{11}^{-1}R_{12} \\ I \end{pmatrix} z,$$

that is: $x = P\bar{x} = x_0 + Nz$, with

$$x_0 = P \begin{pmatrix} R_{11}^{-1}y_1 \\ 0 \end{pmatrix}, \quad N = P \begin{pmatrix} R_{11}^{-1}R_{12} \\ I \end{pmatrix} \in \mathbf{R}^{n \times (n-r)}.$$

Auto-Regressive (AR) models for time-series prediction

A popular model for the prediction of time series is based on the so-called auto-regressive model

$$y_t = \theta_1 y_{t-1} + \dots + \theta_m y_{t-m}, \quad t = 1, \dots, m,$$

where θ_i 's are constant coefficients, and m is the “memory length” of the model. The interpretation of the model is that the next output is a linear function of the past. Elaborate variants of auto-regressive models are widely used for prediction of time series arising in finance and economics.

To find the coefficient vector theta in \mathbf{R}^m , we collect observations $(y_t)_{0 \leq t \leq T}$ (with $T \geq m$) of the time series, and try to minimize the total squared error in the above equation:

$$\min_{\theta} : \sum_{t=m}^T (y_t - \theta_1 y_{t-1} - \dots - \theta_m y_{t-m})^2.$$

This can be expressed as a linear least-squares problem, with appropriate data A, y .

See also: [Linear regression via Least-Squares](#).

Portfolio Optimization via Linearly Constrained Least-Squares

We consider a universe of n financial assets, in which we seek to invest over one time period. We denote by $r \in \mathbf{R}^n$ the vector containing the [rates of return](#) of each asset. A portfolio corresponds to a vector $x \in \mathbf{R}^n$, where x_i is the amount invested in asset i . In our simple model, we assume that “shorting” (borrowing) is allowed, that is, there are no sign restrictions on x .

As explained [here](#), the return of the portfolio is the scalar product $R(x) := r^T x$. We do not know the return vector r in advance. We assume that we know a reasonable prediction \hat{r} of r . Of course, we cannot rely only on the vector \hat{r} only to make a decision, since the actual values in r could fluctuate around \hat{r} . We can consider two simple ways to model the uncertainty on r , which result in similar optimization problems.

Mean-variance trade-off

A first approach assumes that r is a random variable, with known [mean](#) \hat{r} and [covariance matrix](#) Σ . If past values r_1, \dots, r_N of the returns are known, we can use the following estimates

$$\hat{r} = \frac{1}{N} \sum_{i=1}^N r_i, \quad \Sigma = \frac{1}{N} \sum_{i=1}^N (r_i - \hat{r})(r_i - \hat{r})^T.$$

Note that, in practice, the above estimates for the mean \hat{r} and covariance matrix Σ are very unreliable, and more sophisticated estimates should be used.

Then the mean value of the portfolio's return $R(x)$ takes the form $\hat{R}(x) = \hat{r}^T x$, and its [variance](#) is

$$\sigma(x)^2 := \frac{1}{N} \sum_{i=1}^N (r_i^T x - \hat{r}^T x)^2 = x^T \Sigma x.$$

We can strike a trade-off between the “performance” of the portfolio, measured by the mean return, against the “risk”, measured by the variance, via the optimization problem

$$\min_x x^T \Sigma x : \hat{r}^T x = \mu,$$

where μ is our target for the nominal return. Since Σ is positive semi-definite, that is, it can be written as $\Sigma = A^T A$ with $A = (r_1 - \hat{r}, \dots, r_N - \hat{r})$, the above problem is a [linearly constrained least-squares](#).

An ellipsoidal model

To model the uncertainty in r , we can use the following deterministic model. We assume that the true vector r lies in a given ellipsoid E , but is otherwise unknown. We describe E by its center \hat{r} and a “shape matrix” determined by some invertible matrix L :

$$\mathbf{E} := \{r = \hat{r} + Lu : \|u\|_2 \leq 1\}.$$

We observe that if $r \in \mathbf{E}$, then $r^T x$ will be in an interval $[\alpha_{\min}, \alpha_{\max}]$, with

$$\alpha_{\min} = \min_{r \in \mathbf{E}} r^T x, \quad \alpha_{\max} = \max_{r \in \mathbf{E}} r^T x.$$

Using the Cauchy-Schwartz inequality, as well as the form of \mathbf{E} given above, we obtain that

$$\alpha_{\max} = \hat{r}^T x + \max_{u: \|u\|_2 \leq 1} u^T (L^T x) = \hat{r}^T x + \|L^T x\|_2.$$

Likewise,

$$\alpha_{\min} = \hat{r}^T x - \|L^T x\|_2.$$

For a given portfolio vector x , the true return $r^T x$ will lie in an interval $[\hat{r}^T x - \sigma(x), \hat{r}^T x + \sigma(x)]$, where $\hat{r}^T x$ is our “nominal” return, and $\sigma(x)$ is a measure of the “risk” in the nominal return:

$$\sigma(x) = \|L^T x\|_2.$$

We can formulate the problem of minimizing the risk subject to a constraint on the nominal return:

$$\min_x x^T \Sigma x : \hat{r}^T x = \mu,$$

where μ is our target for the nominal return, and $\Sigma := LL^T$. This is again a [linearly constrained least-squares](#). Note that we obtain a problem that has exactly the same form as the stochastic model seen before.

Portfolio Optimization via Linearly Constrained Least-Squares

We consider a universe of n financial assets, in which we seek to invest over one time period. We denote by $r \in \mathbf{R}^n$ the vector containing the [rates of return](#) of each asset. A portfolio corresponds to a vector $x \in \mathbf{R}^n$, where x_i is the amount invested in asset i . In our simple model, we assume that “shorting” (borrowing) is allowed, that is, there are no sign restrictions on x .

As explained [here](#), the return of the portfolio is the scalar product $R(x) := r^T x$. We do not know the return vector r in advance. We assume that we know a reasonable prediction \hat{r} of r . Of course, we cannot rely only on the vector \hat{r} only to make a decision, since the actual values in r could fluctuate around \hat{r} . We can consider two simple ways to model the uncertainty on r , which result in similar optimization problems.

Mean-variance trade-off

A first approach assumes that r is a random variable, with known [mean](#) \hat{r} and [covariance matrix](#) Σ . If past values r_1, \dots, r_N of the returns are known, we can use the following estimates

$$\hat{r} = \frac{1}{N} \sum_{i=1}^N r_i, \quad \Sigma = \frac{1}{N} \sum_{i=1}^N (r_i - \hat{r})(r_i - \hat{r})^T.$$

Note that, in practice, the above estimates for the mean \hat{r} and covariance matrix Σ are very unreliable, and more sophisticated estimates should be used.

Then the mean value of the portfolio's return $R(x)$ takes the form $\hat{R}(x) = \hat{r}^T x$, and its [variance](#) is

$$\sigma(x)^2 := \frac{1}{N} \sum_{i=1}^N (r_i^T x - \hat{r}^T x)^2 = x^T \Sigma x.$$

We can strike a trade-off between the “performance” of the portfolio, measured by the mean return, against the “risk”, measured by the variance, via the optimization problem

$$\min_x x^T \Sigma x : \hat{r}^T x = \mu,$$

where μ is our target for the nominal return. Since Σ is positive semi-definite, that is, it can be written as $\Sigma = A^T A$ with $A = (r_1 - \hat{r}, \dots, r_N - \hat{r})$, the above problem is a [linearly constrained least-squares](#).

An ellipsoidal model

To model the uncertainty in r , we can use the following deterministic model. We assume that the true vector r lies in a given ellipsoid \mathbf{E} , but is otherwise unknown. We describe \mathbf{E} by its center \hat{r} and a “shape matrix” determined by some invertible matrix L :

$$\mathbf{E} := \{r = \hat{r} + Lu : \|u\|_2 \leq 1\}.$$

We observe that if $r \in \mathbf{E}$, then $r^T x$ will be in an interval $[\alpha_{\min}, \alpha_{\max}]$, with

$$\alpha_{\min} = \min_{r \in \mathbf{E}} r^T x, \quad \alpha_{\max} = \max_{r \in \mathbf{E}} r^T x.$$

Using the Cauchy-Schwartz inequality, as well as the form of \mathbf{E} given above, we obtain that

$$\alpha_{\max} = \hat{r}^T x + \max_{u: \|u\|_2 \leq 1} u^T (L^T x) = \hat{r}^T x + \|L^T x\|_2.$$

Likewise,

$$\alpha_{\min} = \hat{r}^T x - \|L^T x\|_2.$$

For a given portfolio vector x , the true return $r^T x$ will lie in an interval $[\hat{r}^T x - \sigma(x), \hat{r}^T x + \sigma(x)]$, where $\hat{r}^T x$ is our “nominal” return, and $\sigma(x)$ is a measure of the

“risk” in the nominal return:

$$\sigma(x) = \|L^T x\|_2.$$

We can formulate the problem of minimizing the risk subject to a constraint on the nominal return:

$$\min_x x^T \Sigma x : r^T x = \mu,$$

where μ is our target for the nominal return, and $\Sigma := LL^T$. This is again a [linearly constrained least-squares](#). Note that we obtain a problem that has exactly the same form as the stochastic model seen before.

Absorption spectrometry: using measurements at different light frequencies

Return to the absorption spectrometry setup described [here](#).

The Beer-Lambert law postulates that the logarithm of the ratio of the light intensities is a linear function of the concentrations of each gas in the mix. The log-ratio of intensities is thus of the form $y = a^T x$ for some vector $a \in \mathbf{R}^n$, where x is the vector of concentrations, and the vector $a \in \mathbf{R}^n$ contains the coefficients of absorption of each gas. This vector is actually also a function of the frequency of the light we illuminate the container with.

Now consider a container having a mixture of n “pure” gases in it. Denote by $x \in \mathbf{R}^n$ the vector of concentrations of the gases in the mixture. We now illuminate the container at different frequencies $\lambda_1, \dots, \lambda_m$. For each experiment, we record the corresponding log-ratio $y_i, i = 1, \dots, m$, of the intensities. If the Beer-Lambert law is to be believed, then we must have $y = Ax$ where A_{ij} is the coefficient of absorption of the j -th gas at frequency λ_i .

Since A_{ij} 's correspond to “pure” gases, they can be measured in the laboratory. We can then use the above model to infer the concentration of the gases in a mixture, given some observed light intensity log-ratio.

See also: [Absorption spectrometry: the Beer-Lambert law](#).

Largest singular value norm of a matrix

For a $m \times n$ matrix A , we define the *largest singular value* (or, LSV) norm of A to be the quantity

$$\|A\| := \max_x \|Ax\|_2 : \|x\|_2 = 1.$$

This quantity satisfies the conditions to be a norm (see [here](#)). The reason for which this norm is called this way is given [here](#).

By definition, for any vector $x \in \mathbf{R}^n$, with $\|x\|_2 = 1$, we have $\|Ax\|_2 \leq \|A\|$. Hence for any $x \in \mathbf{R}^n$, we have $\|Ax\|_2 \leq \|A\| \|x\|_2$.

The LSV norm can be computed as follows. Let us square the above. We obtain a representation of the squared LSV norm as a [Rayleigh quotient](#) of the matrix $A^T A$:

$$\|A\|^2 = \max_{x : \|x\|_2=1} x^T A^T A x.$$

This shows that the squared LSV norm is the largest eigenvalue of the (positive semi-definite) symmetric matrix $A^T A$, which is denoted λ_{\max} . That is:

$$\|A\| = \sqrt{\lambda_{\max}(A^T A)}.$$

Control of a unit mass

Consider the problem of transferring a unit mass at rest sliding on a plane from a point to another at a unit distance. We can exert a constant force of magnitude x_i on the mass at time intervals $i-1 < t \leq i, i = 1, \dots, 10$.

Denoting by y the position at the final instant $T = 10$, we can express via Newton's law the relationship between the force vector x and position/velocity vector y as $y = Ax$, where $A \in \mathbf{R}^{2 \times 10}$.

Now assume that we would like to find the smallest-norm (in the Euclidean sense) force that puts the mass at $y = (1, 0)$ at the final time. This is the problem of finding the minimum-norm solution to the equation $Ax = y$. The solution is $x_{\text{LN}} = A^T (AA^T)^{-1} y$.

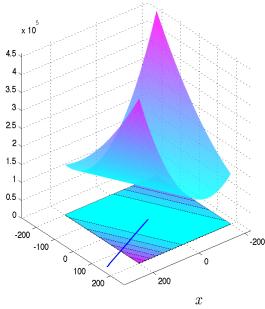
A squared linear function [Symmetric Matrices](#) > [Definitions](#) > Example

A squared linear function is a quadratic function $q : \mathbf{R}^n \rightarrow \mathbf{R}$ of the form

$$q(x) = (v^T x)^2,$$

for some vector $v \in \mathbf{R}^n$.

The function vanishes on the space orthogonal to v , which is the hyperplane defined by the single linear equation $v^T x = 0$. Thus, in effect this function is really one-dimensional: it varies only along the direction v .



Level sets and graph of a dyadic quadratic function, corresponding to the vector $v = (2, 1)$. The function is constant along hyperplanes orthogonal to v .

Control of a unit mass

Consider the problem of transferring a unit mass at rest sliding on a plane from a point to another at a unit distance. We can exert a constant force of magnitude x_i on the mass at time intervals $i-1 < t \leq i$, $i = 1, \dots, 10$.

Denoting by y_1 the position at the final instant $T = 10$, we can express via Newton's law the relationship between the force vector x and position/velocity vector y as $y = Ax$, where $A \in \mathbf{R}^{2 \times 10}$.

Now assume that we would like to find the smallest-norm (in the Euclidean sense) force that puts the mass at $y = (1, 0)$ at the final time. This is the problem of finding the minimum-norm solution to the equation $Ax = y$. The solution is $x_{\text{LN}} = A^T(AA^T)^{-1}y$.

Control of a unit mass

Consider the problem of transferring a unit mass at rest sliding on a plane from a point to another at a unit distance. We can exert a constant force of magnitude x_i on the mass at time intervals $i-1 < t \leq i$, $i = 1, \dots, 10$.

Denoting by y_1 the position at the final instant $T = 10$, we can express via Newton's law the relationship between the force vector x and position/velocity vector y as $y = Ax$, where $A \in \mathbf{R}^{2 \times 10}$.

Now assume that we would like to find the smallest-norm (in the Euclidean sense) force that puts the mass at $y = (1, 0)$ at the final time. This is the problem of finding the minimum-norm solution to the equation $Ax = y$. The solution is $x_{\text{LN}} = A^T(AA^T)^{-1}y$.

A 3×3 symmetric matrix [Symmetric Matrices > Definitions](#) > Example

The matrix

$$A := \begin{pmatrix} 4 & 3/2 & 2 \\ 3/2 & 2 & 5/2 \\ 2 & 5/2 & 2 \end{pmatrix}$$

is symmetric. The matrix

$$B := \begin{pmatrix} 4 & 3/2 & 2 \\ 3/2 & 2 & \mathbf{5/2} \\ 2 & \mathbf{5} & 2 \end{pmatrix}$$

is not, since it is not equal to its transpose. (Offending elements in bold.)

A 3×3 symmetric matrix [Symmetric Matrices > Definitions](#) > Example

The matrix

$$A := \begin{pmatrix} 4 & 3/2 & 2 \\ 3/2 & 2 & 5/2 \\ 2 & 5/2 & 2 \end{pmatrix}$$

is symmetric. The matrix

$$B := \begin{pmatrix} 4 & 3/2 & 2 \\ 3/2 & 2 & \mathbf{5/2} \\ 2 & \mathbf{5} & 2 \end{pmatrix}$$

is not, since it is not equal to its transpose. (Offending elements in bold.)

Hessian of a quadratic function [Symmetric Matrices > Definitions](#) > Example

For quadratic functions, the [Hessian](#) (matrix of second-derivatives) is a *constant* matrix, that is, it does not depend on the variable x .

As a specific example, consider the quadratic function

$$q(x) = 8x_1^2 + 6x_1x_2 + 4x_2^2 - 6x_1 + 9x_2 + 10.$$

The Hessian is given by

$$\frac{\partial^2 q}{\partial x_i \partial x_j}(x) = \begin{pmatrix} \frac{\partial^2 q}{\partial x_1^2}(x) & \frac{\partial^2 q}{\partial x_1 \partial x_2}(x) \\ \frac{\partial^2 q}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 q}{\partial x_2^2}(x) \end{pmatrix} = 2 \begin{pmatrix} 8 & 3 \\ 3 & 4 \end{pmatrix}.$$

Hessian of a quadratic function [Symmetric Matrices > Definitions](#) > Example

For quadratic functions, the [Hessian](#) (matrix of second-derivatives) is a *constant* matrix, that is, it does not depend on the variable x .

As a specific example, consider the quadratic function

$$q(x) = 8x_1^2 + 6x_1x_2 + 4x_2^2 - 6x_1 + 9x_2 + 10.$$

The Hessian is given by

$$\frac{\partial^2 q}{\partial x_i \partial x_j}(x) = \begin{pmatrix} \frac{\partial^2 q}{\partial x_1^2}(x) & \frac{\partial^2 q}{\partial x_1 \partial x_2}(x) \\ \frac{\partial^2 q}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 q}{\partial x_2^2}(x) \end{pmatrix} = 2 \begin{pmatrix} 8 & 3 \\ 3 & 4 \end{pmatrix}.$$

Representation of a two-variable quadratic function [Symmetric Matrices > Definitions](#) > Example

The quadratic function $q : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$q_1(x) = 4x_1^2 + 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 2,$$

can be represented via a symmetric matrix, as

$$q(x) = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}^T \begin{pmatrix} 4 & 3/2 & 2 \\ 3/2 & 2 & 5/2 \\ 2 & 5/2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}.$$

In short:

$$q(x) = \left(\begin{array}{c|c} x & \\ \hline 1 & \end{array} \right)^T \left(\begin{array}{c|c} A & b \\ \hline b^T & c \end{array} \right) \left(\begin{array}{c} x \\ 1 \end{array} \right),$$

where $x = (x_1, x_2)$, and

$$A = \left(\begin{array}{c|c} 4 & 3/2 \\ \hline 3/2 & 2 \end{array} \right) = A^T, \quad b = \left(\begin{array}{c} 2 \\ 5/2 \end{array} \right), \quad c = 2.$$

Representation of a two-variable quadratic function [Symmetric Matrices > Definitions](#) > Example

The quadratic function $q : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$q_1(x) = 4x_1^2 + 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 2,$$

can be represented via a symmetric matrix, as

$$q(x) = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}^T \begin{pmatrix} 4 & 3/2 & 2 \\ 3/2 & 2 & 5/2 \\ 2 & 5/2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}.$$

In short:

$$q(x) = \left(\begin{array}{c|c} x & \\ \hline 1 & \end{array} \right)^T \left(\begin{array}{c|c} A & b \\ \hline b^T & c \end{array} \right) \left(\begin{array}{c} x \\ 1 \end{array} \right),$$

where $x = (x_1, x_2)$, and

$$A = \left(\begin{array}{c|c} 4 & 3/2 \\ \hline 3/2 & 2 \end{array} \right) = A^T, \quad b = \left(\begin{array}{c} 2 \\ 5/2 \end{array} \right), \quad c = 2.$$

Quadratic Approximation of the Log-Sum-Exp Function [Symmetric Matrices > Definitions](#) > Example

As seen [here](#), the *log-sum-exp* function $\text{lse} : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$\text{lse}(x) := \log(e^{x_1} + e^{x_2})$$

admits the following gradient and Hessian at a point x :

$$\nabla \text{lse}(x) = \frac{1}{z_1 + z_2} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}, \quad \nabla^2 \text{lse}(x) = \frac{z_1 z_2}{(z_1 + z_2)^2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \text{ where } z_i := e^{x_i}, \quad i = 1, 2.$$

Hence, the quadratic approximation of the log-sum-exp function at a point $x = (x_1, x_2)$ is given by

$$\begin{aligned} \text{lse}(x + h) &\approx \text{lse}(x) + h^T \nabla \text{lse}(x) + \frac{1}{2} h^T \nabla^2 \text{lse}(x) h \\ &= \text{lse}(x) + \frac{h_1 e^{x_1} + h_2 e^{x_2}}{e^{x_1} + e^{x_2}} + \frac{e^{x_1+x_2}}{2(e^{x_1} + e^{x_2})^2} (h_1 - h_2)^2. \end{aligned}$$

A diagonal matrix and its associated quadratic form [Symmetric Matrices > Definitions > Example](#)

Define a diagonal matrix:

$$D := \text{diag}(1, 4, -3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -3 \end{pmatrix}.$$

For the matrix above, the associated quadratic form is

$$q(x) = x^T \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -3 \end{pmatrix} x = x_1^2 + 4x_2^2 - 3x_3^2.$$

[Quadratic Approximation of the Log-Sum-Exp Function](#) [Symmetric Matrices > Definitions > Example](#)

As seen [here](#), the *log-sum-exp* function $\text{lse} : \mathbf{R}^2 \rightarrow \mathbf{R}$, with values

$$\text{lse}(x) := \log(e^{x_1} + e^{x_2})$$

admits the following gradient and Hessian at a point x :

$$\nabla \text{lse}(x) = \frac{1}{z_1 + z_2} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}, \quad \nabla^2 \text{lse}(x) = \frac{z_1 z_2}{(z_1 + z_2)^2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \text{ where } z_i := e^{x_i}, \quad i = 1, 2.$$

Hence, the quadratic approximation of the log-sum-exp function at a point $x = (x_1, x_2)$ is given by

$$\begin{aligned} \text{lse}(x + h) &\approx \text{lse}(x) + h^T \nabla \text{lse}(x) + \frac{1}{2} h^T \nabla^2 \text{lse}(x) h \\ &= \text{lse}(x) + \frac{h_1 e^{x_1} + h_2 e^{x_2}}{e^{x_1} + e^{x_2}} + \frac{e^{x_1+x_2}}{2(e^{x_1} + e^{x_2})^2} (h_1 - h_2)^2. \end{aligned}$$

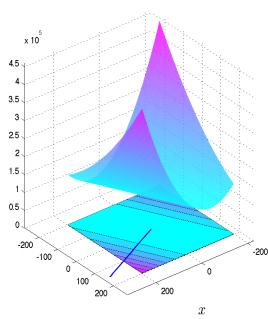
A squared linear function [Symmetric Matrices > Definitions > Example](#)

A squared linear function is a quadratic function $q : \mathbf{R}^n \rightarrow \mathbf{R}$ of the form

$$q(x) = (v^T x)^2,$$

for some vector $v \in \mathbf{R}^n$.

The function vanishes on the space orthogonal to v , which is the hyperplane defined by the single linear equation $v^T x = 0$. Thus, in effect this function is really one-dimensional: it varies only along the direction v .



Level sets and graph of a dyadic quadratic function, corresponding to the vector $v = (2, 1)$. The function is constant along hyperplanes orthogonal to v .

Theorem on positive semi-definite forms and eigenvalues

Theorem: A quadratic form $q(x) = x^T A x$, with $A \in \mathbf{S}^n$ is non-negative (resp. positive-definite) if and only if every eigenvalue of the symmetric matrix A is non-negative (resp. positive).

Proof: Let $A = U\Lambda U^T$ be the symmetric eigenvalue decomposition of A , as given by the [spectral theorem](#). Denote by u_i the i -th column of U and by λ_i the i -th eigenvalue of A .

If every eigenvalue of A is non-negative, that is, $\lambda_i \geq 0$ for every i , then for every x :

$$q_A(x) = x^T A x = \sum_{i=1}^n \lambda_i (u_i^T x)^2 \geq 0.$$

Conversely, if there exist i for which $\lambda_i < 0$, then choosing $x = u_i$ will result in $q(u_i) < 0$.

Likewise, a matrix A has all its eigenvalues positive if and only if q_A is a positive-definite function, that is, $q_A(x) \geq 0$ for every x , and $q_A(x) = 0$ if and only if $x = 0$. Indeed, when $\lambda_i > 0$ for every i , then the condition

$$q_A(x) = x^T A x = \sum_{i=1}^n \lambda_i (u_i^T x)^2 = 0$$

implies $u_i^T x = 0$ for every i , which can be written as $Ux = 0$. Since U is orthogonal, it is invertible, and we conclude that $x = 0$. Conversely, if $\lambda_i \leq 0$ for some i , we can achieve $q(x) \leq 0$ for some non-zero $x = u_i$.

See also: [spectral theorem](#).

Laplacian matrix of a graph

Another important symmetric matrix associated with a graph is the *Laplacian matrix*. This is the matrix $L = A^T A$, with A the arc-node incidence matrix, is another important symmetric matrix associated with the graph. It can be shown that the (i, j) element of the Laplacian matrix is given by $L_{ij} = \{$

$$\begin{matrix} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{matrix}$$

See also:

- [Arc-node incidence matrix of a graph](#).
- [Edge-weight matrix of a graph](#).

Singular value decomposition (SVD) theoremTheorem: Singular Value Decomposition (SVD)

$$A = \sum_{i=1}^r \sigma_i u_i v_i^T = U \tilde{S} V^T, \quad \tilde{S} := \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix},$$

An arbitrary matrix $A \in \mathbf{R}^{m \times n}$ admits a decomposition of the form $A = \sum_{i=1}^r \sigma_i u_i v_i^T$, where $U \in \mathbf{R}^{m \times m}$, $V \in \mathbf{R}^{n \times n}$ are both orthogonal matrices, and the matrix \tilde{S} is diagonal: $\tilde{S} = \text{diag}(\sigma_1, \dots, \sigma_r)$, where the positive numbers $\sigma_1 \geq \dots \geq \sigma_r > 0$ are unique, and are called the *singular values* of A . The number $r \leq \min(m, n)$ is equal to the rank of A , and the triplet (U, \tilde{S}, V) is called a *singular value decomposition* (SVD) of A . The first r columns of U : u_i , $i = 1, \dots, r$ (resp. V : v_i , $i = 1, \dots, r$) are called left (resp. right) singular vectors of A , and satisfy $Av_i = \sigma_i u_i$, $u_i^T A = \sigma_i v_i$, $i = 1, \dots, r$.

Proof: The matrix $n \times n A^T A$ is real and symmetric. According to the [spectral theorem](#), it admits an eigenvalue decomposition in the form $A^T A = V \Lambda V^T$, with V a $n \times n$ matrix whose columns form an orthonormal basis (that is, $V^T V = VV^T = I_n$), and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_r, 0, \dots, 0)$. Here, r is the rank of $A^T A$ (if $r = n$ then there are no trailing zeros in Λ). Since $A^T A$ is positive semi-definite, the λ_j 's are non-negative, and we can define the non-zero quantities $\sigma_j := \sqrt{\lambda_j}$, $j = 1, \dots, r$.

Note that when $j > r$, $Av_j = 0$, since then $\|Av_j\|_2^2 = v_j^T A^T Av_j = \lambda_j v_j^T v_j = 0$.

$$u_i = \frac{1}{\sigma_i} Av_i, \quad i = 1, \dots, r.$$

Let us construct an $m \times m$ orthogonal matrix U as follows. We set $u_i = \frac{1}{\sigma_i} Av_i$. These m -vectors are unit-norm, and mutually orthogonal, since v_j 's are eigenvectors of $A^T A$. Using (say) the [Gram-Schmidt orthogonalization procedure](#), we can complete (if necessary, that is in the case $r < m$) this set of vectors by u_{r+1}, \dots, u_m in order to form an orthogonal matrix $U := (u_1, \dots, u_m) \in \mathbf{R}^m$.

Let us check that U, V satisfy the conditions of the theorem, by showing that $U^T A V^T = \tilde{S} := \text{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0)$. We have

$$(U^T A V)_{ij} = u_i^T A v_j = \begin{cases} \sigma_j u_i^T u_j & \text{if } j \leq r \\ 0 & \text{otherwise,} \end{cases}$$

where the second line stems from the fact that $Av_j = 0$ when $j > r$. Thus, $U^T A V = \tilde{S}$, as claimed.

Nullspace of a 4×5 matrix via its SVD

Returning to [this example](#), involving a matrix with row size $m = 4$ and column size $n = 5$, and of rank $r = 3$. The nullspace is the span of the last $n - r = 2$ columns of the 5×5 matrix V : $\mathbf{N}(A) = \text{span}(v_4, v_5)$, with

$$v_4 := \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad v_5 := \begin{pmatrix} -\sqrt{0.8} \\ 0 \\ 0 \\ 0 \\ \sqrt{0.2} \end{pmatrix}$$

We can check that $Av_4 = Av_5 = 0$.

See also: [this example](#).

Nullspace of a 4×5 matrix via its SVD

Returning to [this example](#), involving a matrix with row size $m = 4$ and column size $n = 5$, and of rank $r = 3$. The nullspace is the span of the last $n - r = 2$ columns of the 5×5 matrix V : $\mathbf{N}(A) = \text{span}(v_4, v_5)$, with

$$v_4 := \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad v_5 := \begin{pmatrix} -\sqrt{0.8} \\ 0 \\ 0 \\ 0 \\ \sqrt{0.2} \end{pmatrix}$$

We can check that $Av_4 = Av_5 = 0$.

See also: [this example](#).

Linear Equations: Definition and Main Issues [Linear Equations](#) > Definitions | [Properties](#) | [Solution concepts](#)

- Definition
- Link with subspaces and affine sets
- Issues with linear equations

Definition

A *linear equation* in $x \in \mathbf{R}^n$ is an equation of the form $Ax = y$, where $A \in \mathbf{R}^{m \times n}$ and $y \in \mathbf{R}^m$ are given. Linear equations arise in many areas of engineering. (Note that a more rigorous terminology would call the above equation an *affine* equation, except in the special case $y = 0$.)

Example:

- [A simple \$3 \times 2\$ system](#).
- [CAT scan imaging](#).
- [Temperature distribution](#).

We define the *solution set* to be simply the set of solutions to the linear equation: $\mathbf{S} = \{x \in \mathbf{R}^n : Ax = y\}$. This is a subset of \mathbf{R}^n , where n is the column size of A .

Link with subspaces and affine sets

Recall the definition of an [affine set](#) as the span of a collection of vectors, possibly translated away from the origin. It turns out that solution sets of linear equations are always affine ([Proof](#)).

The relationship between linear equations and affine sets goes both ways: any affine set can be expressed as the solution set of some linear equation. Thus, the study of linear equations will help us understand basic geometric concepts such as subspaces and affine sets. In addition, we now know how to parametrize affine sets in two ways: either as the translated span of vectors, or as the solution set of a linear equation $Ax = y$.

Example: [hyperplane](#).

Important issues

Several important issues arise with linear equations.

Existence

First, do solutions *exist*? This question leads to the notion of *range* of a matrix, which characterizes those vectors y for which solutions exist. The notion of *rank* allows to determine the dimension of the affine set of solutions. If there are no solutions, we can try to solve the equation in approximate manner: this is what least-squares solutions, examined later, are about.

Unicity

Next we may be interested in the question of *unicity* of solutions (if any). The *nullspace* associated with A allows to describe the set of solutions, and characterizes the ambiguity about solutions. When the solution is not unique, we say that the system is *undetermined*.

The notion of nullspace allows to characterize systems for which the solution, if it exists, is unique. If there are many solutions, we can then define the notion of minimum-norm solution.

Solution concepts

If the solution set is empty, that is, the equation does not have any solution, then we may find an approximate solution by solving an optimization problem.

If the equation has many solutions to choose from, we can find a particular solution. The one with minimum norm is often of practical interest.

Sensitivity

A linear equation is often obtained from measurement data. In practice, the “data” in the equation (the matrix A and vector y in the linear equation $Ax = y$) can be noisy.

Sensitivity analysis is concerned with the study of the impact of errors in A, y , on a solution x . The linear algebra methods developed in the first part of this course allow to deal with noise in the measurement vector y . The analysis of the impact of noise in the matrix A is usually more difficult.

Pseudo-inverse of a 4×5 matrix via its SVD

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{pmatrix}.$$

Returning to [this example](#), the pseudo-inverse of the matrix can be computed via an SVD: $A = U\tilde{S}V^T$, with

$$U = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \quad \tilde{S} = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad V^T = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{pmatrix},$$

as follows.

$$\tilde{S}^\dagger = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 \\ 0 & 0 & 1/\sqrt{5} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

We first invert \tilde{S} , simply “inverting what can be inverted” and leaving zero values alone. We get

$$A^\dagger = V\tilde{S}^\dagger U^T = \begin{pmatrix} 0.2000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.2500 \\ 0 & 0.3333 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.4000 & 0 & 0 & 0 \end{pmatrix}.$$

obtained by exchanging the roles of U, V in the SVD:

See also: [this example](#).

Low-rank approximation of a 4×5 matrix via its SVD

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{pmatrix}.$$

Returning to [this example](#), involving a matrix with row size $m = 4$ and column size $n = 5$:

$$U = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \quad \tilde{S} = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad V^T = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{pmatrix}.$$

As seen here, the SVD is given by $A = U\tilde{S}V^T$, with

The matrix is rank $r = 3$. A rank-two approximation is given by zeroing out the smallest singular value, which produces

$$\begin{aligned} \hat{A}_2 &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 4 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{pmatrix}. \end{aligned}$$

We check that the Frobenius norm of the error $\|A - \hat{A}_2\|_F$ is the sum of singular values we have zeroed out, which here reduces to $\sigma_3 = \sqrt{5}$:

$$E := A - \hat{A}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \|E\|_F^2 = 1^2 + 2^2 = 5.$$

Pseudo-Inverse of a Matrix

The pseudo-inverse of a $m \times n$ matrix A is a matrix that generalizes to arbitrary matrices the notion of inverse of a square, invertible matrix. The pseudo-inverse can be expressed from the [singular value decomposition](#) (SVD) of A , as follows.

Let the SVD of A be

$$A = U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T,$$

where U, V are both orthogonal matrices, and S is a diagonal matrix containing the (positive) singular values of A on its diagonal.

Then the pseudo-inverse of A is the $n \times m$ matrix defined as

$$A^\dagger = V \begin{pmatrix} S^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T.$$

Note that A^\dagger has the same dimension as the transpose of A .

This matrix has many useful properties:

- If A is full column rank, meaning $\text{rank}(A) = n \leq m$, that is, $A^T A$ is not singular, then A^\dagger is a [left inverse](#) of A , in the sense that $A^\dagger A = I_n$. We have the closed-form expression

$$A^\dagger = (A^T A)^{-1} A^T.$$

- If A is full row rank, meaning $\text{rank}(A) = m \leq n$, that is, AA^T is not singular, then A^\dagger is a [right inverse](#) of A , in the sense that $AA^\dagger = I_m$. We have the closed-form expression

$$A^\dagger = A^T (AA^T)^{-1}.$$

- If A is square, invertible, then its [inverse](#) is $A^\dagger = A^{-1}$.

- The solution to the least-squares problem

$$\min_x \|Ax - y\|_2$$

with minimum norm is $x^* = A^\dagger y$.

Example: [pseudo-inverse of a \$4 \times 5\$ matrix](#).

Optimal set of Least-Squares via SVD Theorem: optimal set of ordinary least-squares

The optimal set of the OLS problem $p^* := \min_x \|Ax - y\|_2$ can be expressed as $\mathbf{X}^{\text{opt}} = A^\dagger y + \mathbf{N}(A)$, where A^\dagger is the [pseudo-inverse](#) of A , and $A^\dagger y$ is the minimum-norm point in the optimal set. If A is full column rank, the solution is unique, and equal to $x^* = A^\dagger y = (A^T A)^{-1} A^T y$. In general, the particular solution $A^\dagger y$ is the [minimum-norm solution](#) to the least-squares problem.

Proof: The following proof relies on the [SVD](#) of A , and the [rotational invariance](#) of the Euclidean norm.

Optimal value of the problem

Using the SVD we can find the optimal set to the least-squares optimization problem $p^* := \min_x \|Ax - y\|_2^2$. Indeed, if (U, \tilde{S}, V) is an SVD of A , the problem can be written $p^* = \min_x \|U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T x - y\|_2^2 = \min_x \left\| \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T x - U^T y \right\|_2^2$, where we have exploited the fact that the Euclidean norm is invariant under the orthogonal transformation U^T . With $\tilde{x} := V^T x$, and $\tilde{y} := U^T y$, and changing the variable x to \tilde{x} , we express the above as

$$p^* = \min_{\tilde{x}} \left\| \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} \tilde{x} - \tilde{y} \right\|_2^2.$$

Expanding the terms, and using the partitioned notations $\tilde{x} = (\tilde{x}_r, \tilde{x}_{n-r})$, $\tilde{y} = (\tilde{y}_r, \tilde{y}_{m-r})$, we obtain $p^* = \min_{\tilde{x}_r} \|S\tilde{x}_r - \tilde{y}_r\|_2^2 + \|\tilde{y}_{m-r}\|_2^2$. Since S is invertible, we can reduce the first term in the objective to zero with the choice $\tilde{x}_r = S^{-1}\tilde{y}_r$. Hence the optimal value is $p^* = \|\tilde{y}_{m-r}\|_2^2$. We observe that the optimal value is zero if and only if $y \in \mathbf{R}(A)$, which is exactly the same as $\tilde{y}_{m-r} = 0$.

Optimal set

Let us detail the optimal set for the problem. The variable \tilde{x} is partly determined, via its first r components: $\tilde{x}_r^* = S^{-1}\tilde{y}_r$. The remaining $n - r$ variables contained in \tilde{x}_{n-r} are free, as \tilde{x}_{n-r} does not appear in the objective function of the above problem.

Thus, optimal points are of the form $x = V\tilde{x}$, with $\tilde{x} = (\tilde{x}_r^*, \tilde{x}_{n-r})$, $\tilde{x}_r^* = S^{-1}\tilde{y}_r$, and \tilde{x}_{n-r} free.

To express this in terms of the original SVD of A , we observe that $x = V\tilde{x} = V(\tilde{x}_r, \tilde{x}_{n-r})$ means that $x = V_r \tilde{x}_r + V_{n-r} \tilde{x}_{n-r} = V_r S^{-1} \tilde{y}_r + V_{n-r} \tilde{x}_{n-r}$ where V is partitioned as $V = (V_r, V_{n-r})$, with $V_r \in \mathbf{R}^{n \times r}$ and $V_{n-r} \in \mathbf{R}^{n \times (n-r)}$. Similarly, the vector \tilde{y}_r can be expressed as $\tilde{y}_r = U_r^T y$, with U_r formed with the first r columns of U . Thus, any element x^* in the optimal set is of the form $x^* = x_{MN} + V_{n-r} z : z \in \mathbf{R}^{n-r}$, where $x_{MN} := V_r S^{-1} U_r^T y$. (We will soon explain the acronym appearing in the subscript.) The free components \tilde{x}_{n-r} correspond to the degrees of freedom allowed to by the nullspace of A .

Minimum-norm optimal point

The particular solution to the problem, x_{MN} , is the [minimum-norm solution](#), in the sense that it is the element of \mathbf{X}^{opt} that has the smallest Euclidean norm. This is best understood in the space of \tilde{x} -variables.

Indeed, the particular choice $\tilde{x} = (\tilde{x}_r^*, 0)$ corresponds to the element in the optimal set that has the smallest Euclidean norm. Indeed, the norm of \tilde{x} is the same as that of its rotated version, \tilde{x} : the first r elements in $\tilde{x} \tilde{x}_r^*$ are fixed, and since $\|\tilde{x}\|_2^2 = \|\tilde{x}_r\|_2^2 + \|\tilde{x}_{n-r}\|_2^2$, we see that the minimal norm is obtained with $\tilde{x}_{n-r} = 0$.

Optimal set via the pseudo-inverse

The matrix $V_r S^{-1} U_r^T$, which appears in the expression of the particular solution x_{MN} mentioned above, is nothing else than the [pseudo-inverse](#) of A , which is denoted

$A^\dagger = V \begin{pmatrix} S^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T$. With this convention, the minimum-norm optimal point is $A^\dagger y$. Recall that the last $n - r$ columns of V form a basis for the nullspace of A . Hence the optimal set of the problem is $\mathbf{X}^{\text{opt}} = A^\dagger y + \mathbf{N}(A)$. When A is full column rank ($r = n \leq m$, and $A^T A \succ 0$), the optimal set reduces to a singleton, as the nullspace is $\{0\}$. The unique optimal point expresses as $x_{LS} = A^\dagger y = (A^T A)^{-1} A^T y$.

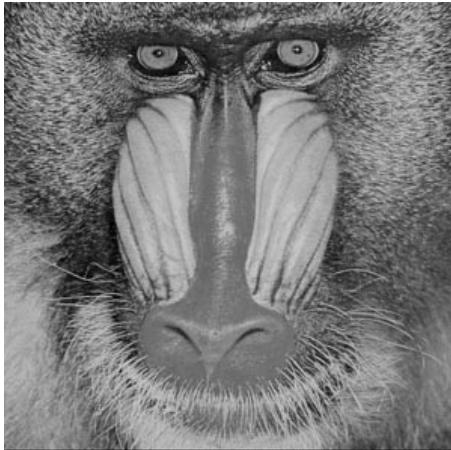
Applications of SVD: image compression [SVD](#) > [Applications](#) > Image compression | [Next](#)

- Images as matrices

- Low-rank approximation

Images as matrices

We can represent images as matrices, as follows. Consider an image having $n \times m$ pixels. For gray scale images, we need one number per pixel, which can be represented as a $n \times m$ matrix. For color images, we need three numbers per pixel, for each color: red, green and blue (RGB). Each color can be represented as a $n \times m$ matrix, and we can represent the full color image as a $n \times 3m$ matrix, where we stack each color's matrix column-wise alongside of each other, as $A = [A_{\text{red}}, A_{\text{green}}, A_{\text{blue}}]$.



The image on the left is a grayscale image, which can be represented as a 298×298 matrix containing the gray scale values stored as integers.

The image can be visualized in matlab as well. We must first transform the matrix from integer to double. In JPEG format, the image will be loaded into matlab as a three-dimensional array, one matrix for each color. For gray scale images, we only need the first matrix in the array.

Matlab syntax

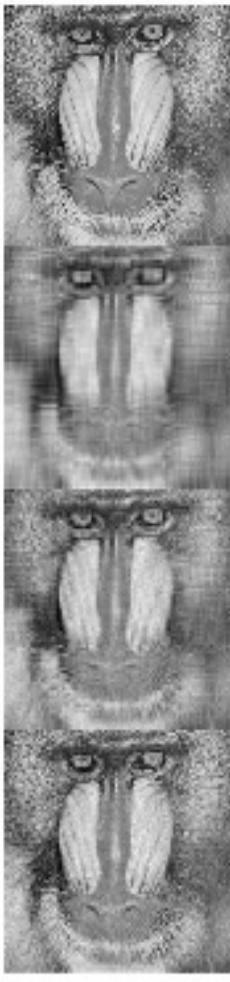
```
>> A = imread(baboon-grayscale.jpg); % loads image in integer format as a 3d-array
>> A = double(A); % transform to real values
>> A = A(:,:,1); % for gray-scale images, we consider only the first matrix in the array
```

Low-rank approximation

Using the [low-rank approximation via SVD](#) method, we can form the best rank- k approximations for the matrix. The matlab code for this is as follows.

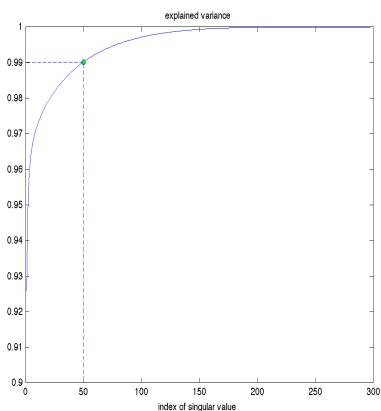
Matlab syntax

```
>> [U,S,V] = svd(A);
>> Ak = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
```



True and approximated images, with varying rank. We observe that with $k = 50$, the approximation is almost the same as the original picture, whose rank is $r = m = n = 298$.

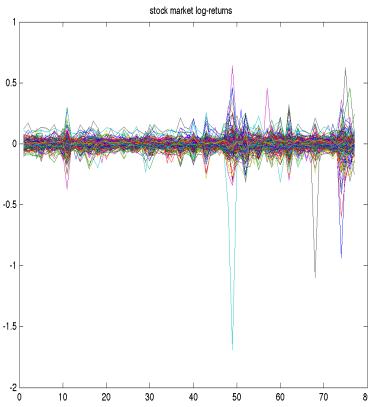
Recall that the [explained variance](#) of the rank- k approximation is the ratio between the squared norm of the error matrix, to the



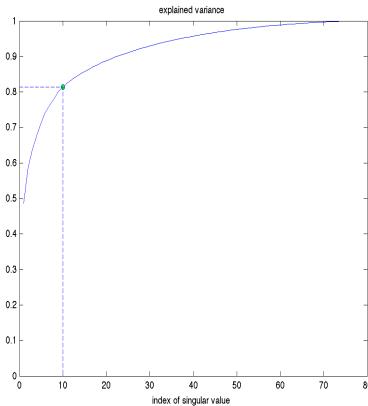
The explained variance for the various rank- k approximations to the original image. For $k = 50$, the approximation contains more than 99% of the total variance in the picture.

Applications of SVD: market data analysis [SVD](#) > [Applications](#) > [Back](#) | SVD of market data

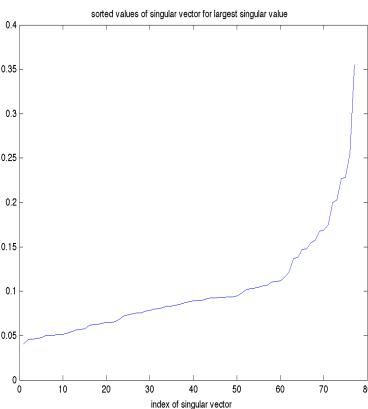
We consider the daily [log-returns](#) of a collection of 77 stocks chosen in the Fortune 100 companies over the time period from January 3, 2007, until December 31, 2008. We can represent this as a 77×504 matrix, with each column a day, and each row a time-series corresponding to a specific stock.



The image on the left represents the time series of the stock market mentioned above, shown as a collection of time-series. We note that the log-returns hover around a mean which appears to be close to zero.



We can form the SVD of the matrix of log-returns, and plot the explained variance. We see that the first 10 singular values explain more than 80% of the data's variance.



It is instructive to look at the singular vector corresponding to the largest singular value, arranged in increasing order. We observe that all the components have the same sign (which we can always assume is positive). This means we can interpret this vector as providing a weighted average of the market. As seen in the previous plot, the corresponding rank-one approximation roughly explains more than 80% of the variance in this market data, which justifies the phrase “the market average moves the market”. The five components with largest magnitude correspond to the following companies. Note that all are financial:

- FABC (Fidelity Advisor)
- FTU (Wachovia, bought by Wells Fargo)
- MER (Merrill Lynch, bought by Bank of America)
- AIG (don't need to elaborate)
- MS (Morgan Stanley)

Solution set of a linear equationTheorem

The solution set of a linear equation $\mathbf{A} = \{x \in \mathbf{R}^n : Ax = y\}$, where $A \in \mathbf{R}^{m \times n}$ and $y \in \mathbf{R}^m$ are given, is either empty, or an affine set.

Proof: Indeed, if it is not empty, we can express the condition $Ax = y$ as $A(x - x_0) = 0$, where x_0 is a particular solution. Hence, $x \in \mathbf{A}$ if and only if $x - x_0 \in \mathbf{L}$, where $\mathbf{L} = \{x : Ax = 0\}$ is a subspace. We write this as $\mathbf{A} = x_0 + \mathbf{L}$. Thus, \mathbf{A} is an affine subspace.

Solution Concepts [Linear Equations > Definitions | Properties](#) | Solution concepts

- Solution set
- Least-norm solutions
- Approximate solutions

Solution set

We consider the linear equation $Ax = y$, where $A \in \mathbf{R}^{m \times n}$ and $y \in \mathbf{R}^m$ are given. The associated solution set is the set of solutions $\mathbf{S} := \{x : Ax = y\}$. By construction, it is an affine set of \mathbf{R}^n .

Several cases may occur:

- If y is in the range of A , the equation has at least a solution, and the solution set is not empty. If the nullspace is not $\{0\}$, then this solution is not unique. In this case, we develop the notion of *minimum-norm solution*, which allows to select a particular solution among the many.

- If y is not in the range of A , then there are no solutions: the solution set is empty. In this case we introduce the notion of *approximate solution*, which is based on minimizing the norm of the “residual error”, $Ax - y$. We refer to a vector x that minimizes this error in norm, as a *minimum-residual solution* (although, strictly speaking, it is not a solution to the original equation).

For both the minimum-norm and minimum-residual problems, the qualitative behavior of the solution depends crucially on the norm used to measure the size of the vectors involved.

Minimum-norm solutions

In the case when $y \in \mathbf{R}(A)$, there may be many solutions to the linear equation $Ax = y$, namely if and only if the nullspace of A is not reduced to $\{0\}$. It is then of interest to select a specific solution. A popular choice is the minimum-norm solution, obtained via the optimization problem $\min_x \|x\| : Ax = y$. Again, the qualitative behavior of the solution depends heavily on the choice of the norm used. Depending on the choice for the norm $\|\cdot\|$, such problems may or may not be easy to solve. For the popular norms (ℓ_p , with $p \in \{1, 2, \infty\}$), they are.

A popular choice corresponds to the case when the norm in the objective function of the above problem is the Euclidean norm. When A is full row rank, meaning that there is always a solution irrespective of y , the minimum-Euclidean-norm solution has a closed-form expression $x_{MN} = A^T(AA^T)^{-1}y$. (Note that the inverse exist, since A is full row rank.) The geometric interpretation of this solution is that it is the *projection* of 0 on the set of solutions to the linear equation.

Example: [Control of a unit mass](#)

Other norms are often used. For example, the ℓ_1 norm is often used as it tends to produce *sparse* solution vectors x , meaning a vector x which has many zero components. This is in turn useful when one is trying to achieve some desired target $y = Ax$ by proper choice of x , with the smallest number of non-zero components.

Example: [Sparse image compression](#)

Approximate solutions via residual error minimizationBasic idea

When $y \notin \mathbf{R}(A)$, there exist no solutions to the equation $Ax = y$. This can be due to several factors: the linear equation may be an approximation of a non-linear one, resulting in *model errors*; or y might be a noisy measurement of the output of some linear process, resulting in *measurement errors*; or both. Such errors might make the solution set empty.

We can then try to find an approximate solution, by minimizing the distance from Ax to y by proper choice of x . We are led to problems of the form $\min_x \|Ax - y\|$. Again, the “solution” found this way depends heavily on the choice of norm. Also that choice has an impact on the computational complexity of the optimization problem above. As before for the popular norms (ℓ_p , with $p \in \{1, 2, \infty\}$), these problems are easy.

Least-squares

The most well-known case corresponds to the choice of the ℓ_2 (Euclidean) norm, because it is the only one of the three which leads to a closed-form solution. The corresponding problem $\min_x \|Ax - y\|_2^2$ is called a *least-squares* problem. We will see later that, when A is full column rank, the problem has a unique, closed-form solution, which is in the form $x_{LS} = (A^T A)^{-1} A^T y$.

Examples:

- [Auto-regressive models for time series prediction](#).
- [Power law model fitting](#).

Other norms

Other norms are often used. For example one can experimentally observe that using the ℓ_1 norm tends to produce solutions such that $(Ax)_i = y_i$ for many i :’s: the residual error vector $Ax - y$ is sparse (has many zero elements). Solving the corresponding problem can be efficiently done, but no “closed-form” solution is known, so we must use optimization solvers. We examine the corresponding problem [here](#).

Properties [Linear Equations > Definitions](#) | Properties | [Solution concepts](#)

- Existence: the range and rank of a matrix
- Unicity: the nullspace of a matrix
- Fundamental theorem of linear algebra
- Matrix inverses

Consider the linear equation in $x \in \mathbf{R}^n$: $Ax = y$, where $A \in \mathbf{R}^{m \times n}$ and $y \in \mathbf{R}^m$ are given.

Existence: range and rank of a matrixRange

The *range* (or, *image*) of a $m \times n$ matrix A is defined as the following subset of \mathbf{R}^m : $\mathbf{R}(A) := \{Ax : x \in \mathbf{R}^n\}$. The range describes the vectors $y = Ax$ that can be attained in the output space by an arbitrary choice of a vector x in the input space. The range is simply the span of the columns of A .

If $y \notin \mathbf{R}(A)$, we say that the linear equation $Ax = y$ is *infeasible*.

The matlab function `orth` accepts a matrix A as input, and returns a matrix, the columns of which span the range of the matrix A , and are mutually orthogonal. Hence, $U^T U = I_r$, where r is the dimension of the range.

Matlab syntax

```
>> U = orth(A); % columns of U span the range of A, and U'*U = identity
```

Example:

- [An infeasible linear system](#).

Rank

The dimension of the range is called the *rank* of the matrix. As we will see later, the rank cannot exceed any one of the dimensions of the matrix A : $r \leq \min(m, n)$. A matrix is said to be *full rank* if $r = \min(m, n)$.

Matlab syntax

```
r = rank(A); % r is the rank of A
```

Note that the rank is a very “brittle” notion, in that small changes in the entries of the matrix can dramatically change its rank. Random matrices, such as ones generated using the Matlab command `rand`, are full rank. We will develop [here](#) a better, more numerically reliable notion.

Examples:

- Range and rank of a simple matrix.
- [Rank-one matrices](#).
- [Rank properties of the arc-node incidence matrix](#).

Full row rank matrices

The matrix A is said to be *full row rank* (or, *onto*) if the range is the whole output space, \mathbf{R}^m . The name “full row rank” comes from the fact that the rank equals the row dimension of A .

An equivalent condition for A to be full row rank is that the square, $m \times m$ matrix AA^T is invertible, meaning that it has full rank, m . [Proof](#).

Unicity: nullspace of a matrixNullspace

The *nullspace* (or, *kernel*) of a $m \times n$ matrix A is the following subspace of \mathbf{R}^n : $\mathbf{N}(A) := \{x \in \mathbf{R}^n : Ax = 0\}$. The nullspace describes the ambiguity in x given $y = Ax$: any $z \in \mathbf{N}(A)$ will be such that $A(x + z) = y$, so x cannot be determined by the sole knowledge of y if the nullspace is not reduced to the singleton $\{0\}$.

The matlab function `null` accepts a matrix A as input, and returns a matrix, the columns of which span the nullspace of the matrix A , and are mutually orthogonal. Hence, $U^T U = I_r$, where r is the dimension of the range.

Matlab syntax

```
U = null(A); % columns of A span the nullspace of A, and U'*U = I
```

Example:

- Nullspace of a simple matrix.

Full column rank matrices

The matrix A is said to be *full column rank* (or, *one-to-one*) if its nullspace is the singleton $\{0\}$. In this case, if we denote by a_i the n columns of A , the equation $(Ax =) \sum_{i=1}^n a_i x_i = 0$ has $x = 0$ as the unique solution. Hence, A is one-to-one if and only if its columns are independent.

The term “one-to-one” comes from the fact that for such matrices, the condition $y = Ax$ uniquely determines x , since $Ax_1 = y$ and $Ax_2 = y$ implies $A(x_1 - x_2) = 0$, so that the solution is unique: $x_1 = x_2$. The name “full column rank” comes from the fact that the rank equals the column dimension of A .

An equivalent condition for A to be full column rank is that the square, $n \times n$ matrix $A^T A$ is invertible, meaning that it has full rank, n . [Proof](#)

Example: [Nullspace of a transpose incidence matrix](#).

Fundamental theorem of linear algebra

A basic result of linear algebra is that the nullspace of a $m \times n$ matrix and the range of the transpose matrix are *orthogonal* sets, in the sense that any two vectors, each chosen in one of the sets, are orthogonal. Further, their dimensions sum up to n . That is, the two sets form an orthogonal decomposition of the whole space.

Fundamental theorem of linear algebra

Let $A \in \mathbf{R}^{m \times n}$. The sets $\mathbf{N}(A)$ and $\mathbf{R}(A^T)$ form an *orthogonal decomposition* of \mathbf{R}^n , in the sense that any vector $x \in \mathbf{R}^n$ can be written as $x = y + z$, $y \in \mathbf{N}(A)$, $z \in \mathbf{R}(A^T)$, $y^T z = 0$.

Proof: the [proof](#) uses the [Singular Value Decomposition](#) seen later.

Example: XXX.

We will see later that the rank of a matrix is equal to that of its transpose. Thus, a consequence of the theorem is that $\text{Rank}(A) + \dim \mathbf{N}(A) = n$. The interpretation of the above is that n is the number of degrees of freedom in input x ; $\text{Rank}(A)$ gives the number of degrees of freedom that remain in the output, while $\dim \mathbf{N}(A)$ counts the number of dimensions of x that are “crushed” to zero by the mapping $x \rightarrow Ax$.

Matrix inversesLeft and right inverses

It can be shown that a $m \times n$ matrix A is full row rank if and only if it has a *right inverse*, that is, there exist a matrix B such that $AB = I_m$, where I_m is the $m \times m$ identity matrix.

It can be shown that a $m \times n$ matrix A is full column rank if and only if it has a *left inverse*, that is, there exist a matrix B such that $BA = I_n$, where I_n is the $n \times n$ identity matrix. Hence, for a one-to-one matrix, the equation $y = Ax$ has always a unique solution, $x = By$.

Invertible matrices

If square $n \times n$ matrix is full row rank if and only if it is full row rank, and vice-versa. In this case, we simply say that the matrix is *invertible*. Then, there exist a unique left- and right inverse, and both are equal to a matrix called the *inverse*, and denoted A^{-1} . The inverse satisfies $A \cdot A^{-1} = A^{-1} \cdot A = I_n$.

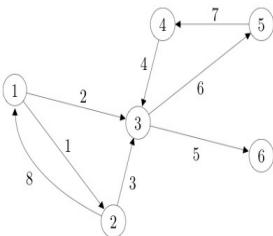
For an invertible $n \times n$ matrix, the nullspace is the zero subspace $\{0\}$, and the range is the whole space, \mathbf{R}^n . In addition, the equation $y = Ax$ then always has a unique solution for every y .

There is a closed-form expression of the inverse, based on the notion of determinant. This expression is useful for theoretical reasons but never used in practice. Later we will see how to compute the matrix inverse in a numerically more efficient way.

A useful property is the expression of the inverse of a product of two square, invertible matrices $A, B: (AB)^{-1} = B^{-1}A^{-1}$.

Edge weight matrix of a graph

A symmetric matrix is a way to describe a weighted, undirected graph: each edge in the graph is assigned a weight W_{ij} . Since the graph is undirected, the edge weight is independent of the direction (from i to j or vice-versa). Hence, W is symmetric.



To the graph in the figure, we can associate the corresponding undirected graph, obtained by ignoring the direction of the arrows. Assuming that all the edges have the same weight, the undirected graph has the edge weight matrix given by $W =$

See also:

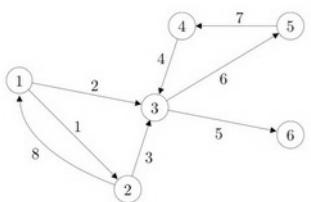
- [Arc-node incidence matrix of a graph](#)

Incidence matrix of a network

Mathematically speaking, a *network* is a graph of m nodes connected by n directed arcs. Here, we assume that arcs are ordered pairs, with at most one arc joining any two nodes; we also assume that there are no self-loops (arcs from a node to itself). We do not assume that the edges of the graph are weighted— they are all similar.

We can fully describe the network with the so-called *arc-node incidence matrix*, which is the $m \times n$ matrix defined as

$$A_{ij} = \begin{cases} 1 & \text{if arc } j \text{ starts at node } i \\ -1 & \text{if arc } j \text{ ends at node } i \\ 0 & \text{otherwise.} \end{cases}, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$



The figure shows the graph associated with the arc-node incidence matrix

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & -1 & -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}.$$

See also:[Network flow](#).

Definitions[Least-Squares](#) > Definitions | [Solution](#) | [Sensitivity](#) | [Limitations](#)

- [The LS problem](#)
- [Regularized LS](#)
- [Linearly-constrained LS and minimum-norm solutions to linear equations](#)

Rate of return of a financial portfolio

The rate of return r (or, return) of a financial asset over a given period (say, a year, or a day) is the interest obtained at the end of the period by investing in it. In other words, if, at the beginning of the period, we invest a sum S in the asset, we will earn $S_{\text{end}} := (1 + r)S$ at the end.

For n assets, we can define accordingly the vector $r \in \mathbf{R}^n$ of rates of return.

Assume that at the beginning of the period, we invest a sum S in all the assets, allocating a fraction x_i (in %) in the i -th asset. Here $x \in \mathbf{R}^n$ is a non-negative vector which

$$S_{\text{end}} := \sum_{i=1}^n (1 + r_i)x_i S.$$

sums to one. Then the portfolio we constituted this way will earn

The rate of return of the portfolio is the relative increase in wealth:

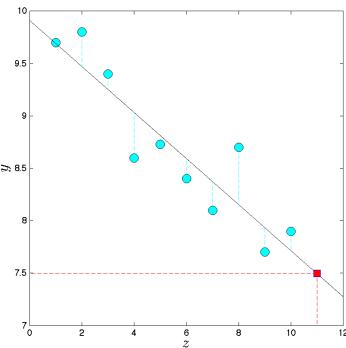
$$\frac{S_{\text{end}} - S}{S} = \sum_{i=1}^n (1 + r_i)x_i - 1 = \sum_{i=1}^n x_i - 1 + \sum_{i=1}^n r_i x_i = r^T x.$$

The rate of return is thus the scalar product between the vector of individual returns r and of the portfolio allocation weights x .

Note that, in practice, rates of return are never known in advance, and they can be negative (although, by construction, they are never less than -1).

Linear regression

A popular example of *least-squares problem* arises in the context of fitting a line through points. This is illustrated below in two dimensions.



In this example we seek to analyze how customers react to an increase in the price of a given item. We are given two-dimensional data points (z_i, y_i) , $i = 1, \dots, m$. The z_i 's contain the prices of the item, and the y_i 's the average number of customers who buy the item.

The generic equation of a non-vertical line is $y = x_1 z + x_2$, where $x = (x_1, x_2)$ contains the decision variables. The quality of the fit of a generic line is measured via the sum of the squares of the error in the component y (blue dotted lines). Thus, the best least-squares fit is obtained via the least-squares problem

$$\min_x \sum_{i=1}^m (x_1 z_i + x_2 - y_i)^2.$$

Once the line is found, it can be used to predict the value of the average number of customers buying the item (y) for a new price (z). The prediction is shown in red.

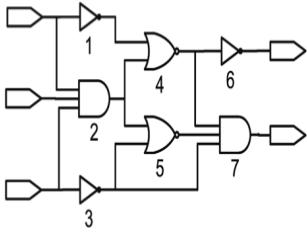
The linear regression approach can be extended to multiple dimensions, that is, to problems where the z -axis in the above problem contains more than one dimension (see [here](#)). It can also be extended to the problem of fitting non-linear curves.

Digital Circuit Design: Problem [GP](#) > [Standard Forms](#) | [Applications](#) > [Circuit Design](#) > Design problem | [Next](#)

- Circuit topology
- Design variables
- Design objective

Circuit Topology

The combinational circuit consists of n connected gates, with primary inputs and outputs. We assume that there are no loops in the corresponding graph. For each gate, we can define the *fan-in*, or set of predecessors of the gate in the circuit graph, and the *fan-out*, which is its set of successors.



Circuit topology: A digital circuit that consists of several “gates” (logical blocks with a few inputs and one output) connecting primary inputs, labeled 8, 9, 10, to primary outputs, labeled 11, 12. Each gate is represented by a symbol that specifies its type; for example, the gates labelled 1, 3 and 6 are inverters. For this circuit, the fan-in and fan-out of gate 2 is $\text{FI}(2) = \{8, 9, 10\}$, $\text{FO}(2) = \{4, 5\}$. By definition, the primary inputs have empty fan-in, and the primary outputs have empty fan-out.

Design Variables

The design variables in our models are the *scale factors*, which we denote by x_i , $i = 1, \dots, n$, which roughly determine the size of each gate. These scale factors satisfy $x_i \geq 1$, $i = 1, \dots, n$, where $x_i = 1$ corresponds to a minimum-sized gate, while a scale factor $x_i = 16$ corresponds to the case when all the devices in the gate have 16 times the widths of those in the minimum-sized gate.

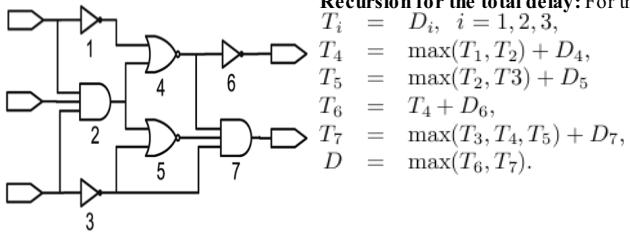
The scale factors determine the size, and various electrical characteristics, such as resistance and conductance, of the gates. These relationship can be well approximated as follows:

- The *area* of gate i is proportional to the scale factor x_i : $A_i(x) = a_i x_i$, with $a_i > 0$. (So, you can simply think of the scale factors as the areas.)
- The *intrinsic capacitance* of gate i is of the form $C_i^{\text{intr}}(x) = C_i^{\text{intr}} x_i$, with $c_i > 0$ positive coefficients.
- The *load capacitance* of gate i is a linear function of the scale factors of the gates in the fan-out of gate i : $C_i^{\text{load}}(x) = \sum_{j \in \text{FO}(i)} C_j^{\text{in}} x_j$, where $C_j^{\text{in}} > 0$ are positive coefficients.
- Each gate has a resistance that is *inversely* proportional to the scale factor (the larger the gate, the more current can pass through it): $R_i(x) = r_i/x_i$, with $r_i > 0$ positive coefficients.
- The *gate delay* is a measure of how fast the gate implements the logical operation it is supposed to perform; this delay can be approximated as $D_i(x) \approx 0.7 R_i(x)(C_i^{\text{intr}} + C_i^{\text{load}})$.

We observe that all the above parameters are actually *posynomials* in the (positive) design vector x .

Design Objective

A possible design objective is to minimize the total delay D for the circuit. We can express the total delay as $D = \max_{1 \leq i \leq n} T_i$, where T_i represents the latest time at which the output of gate i can transition, assuming that the primary inputs signals transition at $t = 0$. (That is, T_i is the maximum delay over all paths that start at primary input and end at gate i .) We can express T_i via the recursion $T_i = \max_{j \in \text{FI}(i)} (T_j + D_j)$. The operations involved in the computation of D involve only addition and point-wise maximum. Since each D_i is a posynomial in x , we can express the total delay as a [generalized posynomial](#) in x .



Recursion for the total delay: For the circuit on the left, the total delay D can be expressed as

$$\begin{aligned} T_1 &= D_i, \quad i = 1, 2, 3, \\ T_4 &= \max(T_1, T_2) + D_4, \\ T_5 &= \max(T_2, T_3) + D_5, \\ T_6 &= T_4 + D_6, \\ T_7 &= \max(T_3, T_4, T_5) + D_7, \\ D &= \max(T_6, T_7). \end{aligned}$$

Applications [Matrices](#) > [Basics](#) | [Matrix products](#) | [Linear maps](#) | Applications

- [Visualization of high-dimensional data](#).

Vectors and Matrices

Vectors are collections of numbers arranged in a column or a row. We review basic notion such as independence, span, subspaces, and dimension. Via the scalar product, we can establish a one-to-one correspondence between vectors and linear functions. Matrices are collections of vectors of same size. Again, via the matrix-vector product, we can interpret them as linear maps (vector-valued functions).

Outline

- [Vectors](#)
- [Scalar products and norms](#)
- [Matrices](#)
- [Linear functions and maps](#)
- [Applications](#)

Definitions [Least-Squares](#) > Definitions | [Solution](#) | [Sensitivity](#) | [Limitations](#)

- [The LS problem](#)
- [Regularized LS](#)
- [Linearly-constrained LS and minimum-norm solutions to linear equations](#)

A simple Example of A PSD Matrix [SDP](#) > [LMIs](#) > Example

Consider the symmetric matrix

$$F = \begin{pmatrix} 29 & 19 & -4 \\ 19 & 28 & 7 \\ -4 & 7 & 15 \end{pmatrix}.$$

We can check that F is positive-semidefinite by computing its eigenvalues.

Matlab syntax

```
>> s = eig(F)
s =
    0.3477
    2.1168
    4.8506
```

A simple Example of A PSD Matrix [SDP](#) > [LMIs](#) > Example

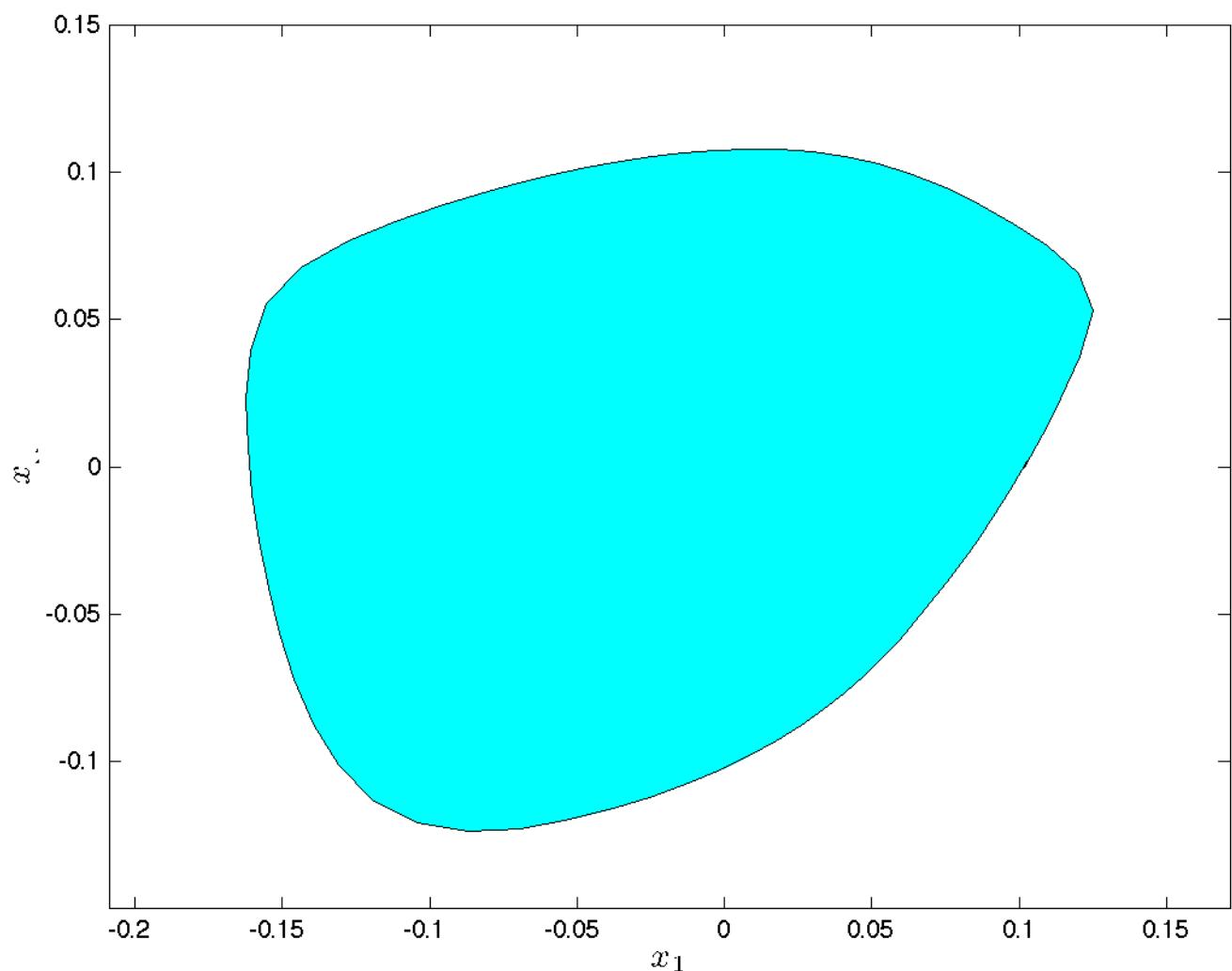
Consider the symmetric matrix

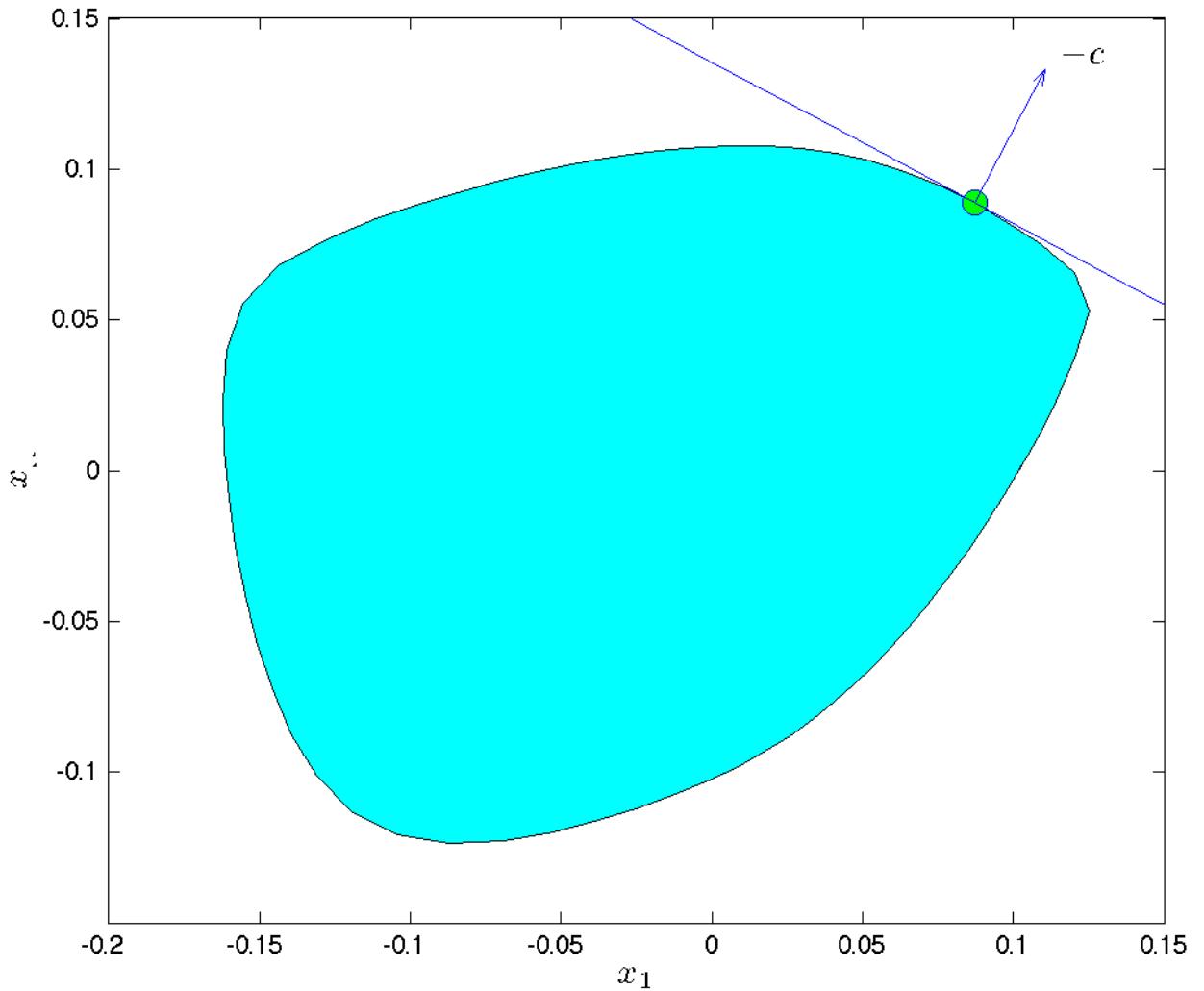
$$F = \begin{pmatrix} 29 & 19 & -4 \\ 19 & 28 & 7 \\ -4 & 7 & 15 \end{pmatrix}.$$

We can check that F is positive-semidefinite by computing its eigenvalues.

Matlab syntax

```
>> s = eig(F)
s =
    0.3477
    2.1168
```





Other Standard Forms

[Up](#) | [Back](#)

Inequality form

As seen [here](#), a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is affine if and only if it can be expressed via the scalar product, as $f(x) = a^T x - b$ for some appropriate vector a and scalar b .

Hence, a linear program can be expressed as $\min_x c^T x : a_i^T x \leq b_i, i = 1, \dots, m$, for some appropriate vectors $c, a_1, \dots, a_m \in \mathbf{R}^n$ and scalars $b_1, \dots, b_m \in \mathbf{R}$.

We can use the [component-wise inequality convention](#) and put the above problem into the *inequality standard form*: $\min_x c^T x : Ax \leq b$, where

$$A := \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix} \in \mathbf{R}^{m \times n}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \mathbf{R}^m.$$

Similarly, for QP a standard form is $\min_x c^T x + x^T Qx : Ax \leq b$,

QPs with equality constraints

In LP or QP models, we can have equality constraints as well. For example, the LP $\min_x c^T x : Ax \leq b, Cx = d$, where $C \in \mathbf{R}^{p \times n}, d \in \mathbf{R}^p$ define the equality constraints, can be put in standard inequality form, as $\min_x c^T x : Ax \leq b, Cx \leq d, -Cx \leq -d$.

Conic form

A problem of the form $\min_x c^T x : Ax = b, x \geq 0$ is an LP. Conversely, any LP can be put in the above form (Proof). A similar result holds for QP. (The reason of the term “conic” comes from the fact that the above problem is part of a more general class known as conic problems, in which the sign constraints on the variables are replaced with $x \in \mathbf{K}$, where \mathbf{K} is a cone.)

The above form is useful to develop theory and algorithms for LP, as it puts all the “data” of the problem into the linear objective and the linear *equality* constraints, while the inequalities have a very simple, data-independent structure.

Inequality form

As seen [here](#), a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is affine if and only if it can be expressed via the scalar product, as $f(x) = a^T x - b$ for some appropriate vector a and scalar b .

Hence, a linear program can be expressed as $\min_x c^T x : a_i^T x \leq b_i, i = 1, \dots, m$, for some appropriate vectors $c, a_1, \dots, a_m \in \mathbf{R}^n$ and scalars $b_1, \dots, b_m \in \mathbf{R}$.

We can use the [component-wise inequality convention](#) and put the above problem into the *inequality standard form*: $\min_x c^T x : Ax \leq b$, where

$$A := \begin{pmatrix} a_1^T \\ \vdots \\ a_m^T \end{pmatrix} \in \mathbf{R}^{m \times n}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \mathbf{R}^m.$$

Similarly, for QP a standard form is $\min_x c^T x + x^T Qx : Ax \leq b$,

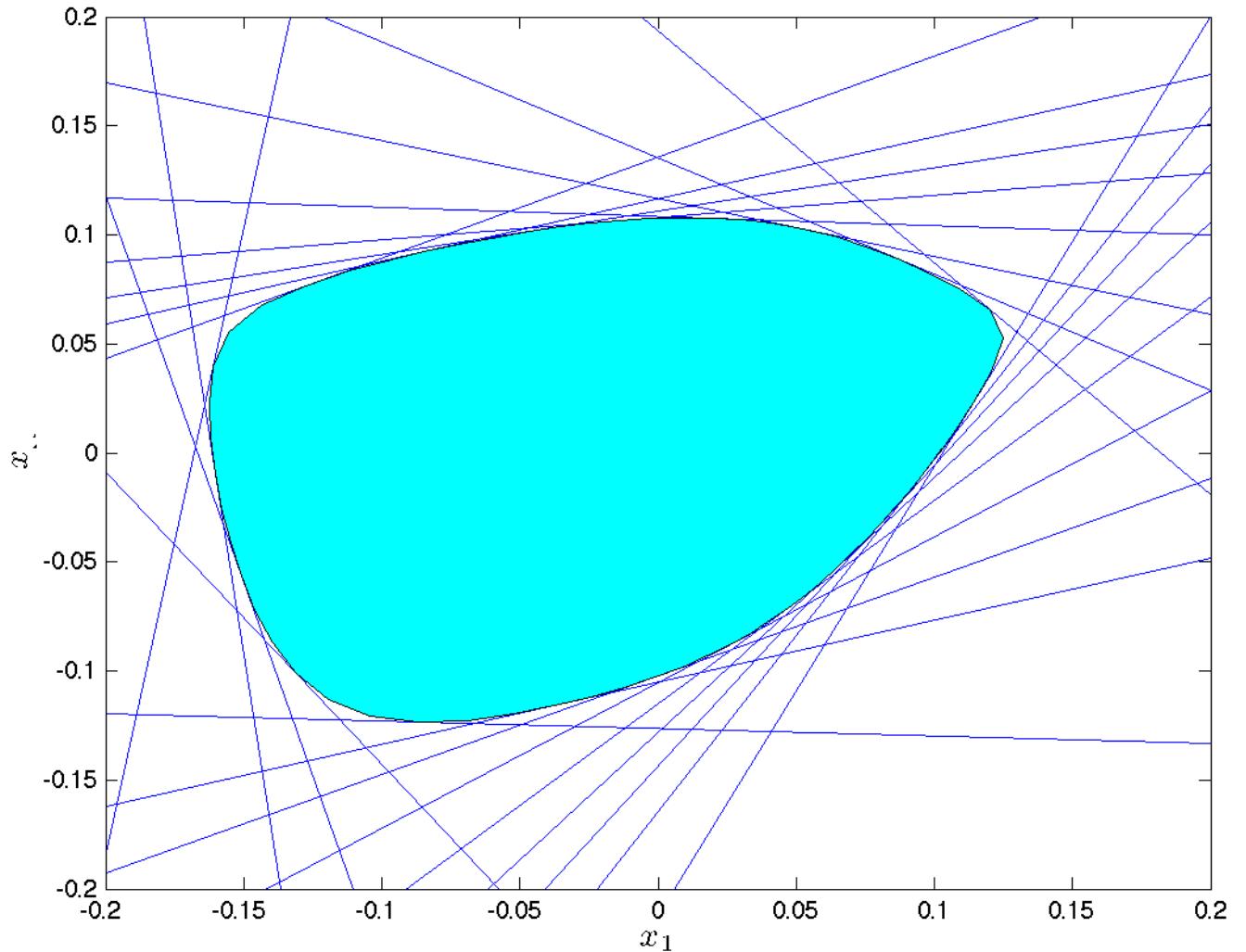
QPs with equality constraints

In LP or QP models, we can have equality constraints as well. For example, the LP $\min_x c^T x : Ax \leq b, Cx = d$, where $C \in \mathbf{R}^{p \times n}, d \in \mathbf{R}^p$ define the equality constraints, can be put in standard inequality form, as $\min_x c^T x : Ax \leq b, Cx \leq d, -Cx \leq -d$.

Conic form

A problem of the form $\min_x c^T x : Ax = b, x \geq 0$ is an LP. Conversely, any LP can be put in the above form (Proof). A similar result holds for QP. (The reason of the term “conic” comes from the fact that the above problem is part of a more general class known as conic problems, in which the sign constraints on the variables are replaced with $x \in \mathbf{K}$, where \mathbf{K} is a cone.)

The above form is useful to develop theory and algorithms for LP, as it puts all the “data” of the problem into the linear objective and the linear *equality* constraints, while the inequalities have a very simple, data-independent structure.



- Portfolio variance
- Variance with complete covariance information
- Bounding variance with incomplete covariance information

Portfolio Variance

Consider (see [here](#) for more background) a portfolio of n assets, which is described by a vector $x \in \mathbf{R}^n$, with x_i the amount (in, say, US dollars) invested in the i -th asset. We are interested in defining, and quantifying the risk associated with holding the position (as described by x) over a certain time period into the future.

We can define the vector $r \in \mathbf{R}^n$, with r_i the rate of return of the i -th asset at the end of the period considered. The return of the portfolio is $y = r^T x$.

Of course, r is unknown. Let us model this uncertainty by assuming that r is a random variable. Then, the return $y = r^T x$ is also a random variable. We define the *risk* of the portfolio x , and denote by $\sigma^2(x)$, the [variance](#) of its return y . That is:

$$\sigma^2(x) := \mathbf{E}(y - \hat{y})^2,$$

where $\hat{y} := \mathbf{E}(y)$ is the expected value of the portfolio's return, and \mathbf{E} denotes the expectation operator with respect to the distribution of the random variable r .

Evaluating the risk is useful to compare different portfolios. Some might have a higher expected return than others; this usually comes at the expense of higher values of risk. The risk defined above suffers from several practical limitations, and we explore one of them here.

Complete Covariance Information

With $y = r^T x$, and x not random (we just take a position at the beginning of the investment period, and hold it until the end), we have

$$\sigma_\Sigma^2(x) := \mathbf{E}(y - \hat{y})^2 = \mathbf{E}((r - \hat{r})^T x)^2 = x^T \Sigma x,$$

where $\Sigma := \mathbf{E}(r - \hat{r})(r - \hat{r})^T$ is the [covariance matrix](#) of the vector of returns.

If this matrix is known, the risk (as we defined it) is easily computed via the matrix-vector product. However, in practice, the covariance matrix is hard to estimate precisely.

Bounding Risk with Incomplete Covariance Information

Assume now that the covariance matrix is only partially specified. For example, in a case with $n = 3$ assets, we might describe our uncertainty as

$$\Sigma = \begin{pmatrix} 0.1 & + & - \\ + & 0.03 & ? \\ - & ? & 0.6 \end{pmatrix},$$

where the symbol $+$ denotes that we believe that the corresponding assets (in this case, assets 1 and 2) are positively correlated, while $-$ denotes negative correlation. Here, the symbol $?$ denotes complete uncertainty on the sign of the correlation. The above covariance information is only partial, with only the diagonal elements (variance of each asset) given hard numbers. Denote by Σ the set of symmetric matrices that satisfy the above pattern. This set is a polytope, which we will denote \mathbf{P} , in the space \mathbf{S}^n of symmetric matrices, since it is defined as the following ordinary inequalities:

$$\mathbf{P} := \left\{ \Sigma = \Sigma^T \in \mathbf{R}^{3 \times 3} : \begin{array}{l} \Sigma_{11} = 0.1, \Sigma_{22} = 0.3, \Sigma_{33} = 0.6, \\ \Sigma_{12} \geq 0, \Sigma_{13} \leq 0 \end{array} \right\}.$$

Of course, any symmetric matrix in the set \mathbf{P} is not necessarily a covariance matrix. In order for it to be a covariance matrix, it has to be positive semi-definite.

We define the *worst-case risk* as the largest variance of the portfolio that can be attained by some covariance matrix:

$$\max_{\Sigma} \sigma_\Sigma^2(x) : \Sigma \in \mathbf{P}, \Sigma \succeq 0.$$

The above is an SDP in (matrix) variable Σ . It is implementable in CVX as follows.

CVX syntax

```
vars = [0.1; 0.03; 0.6]; % given asset variances
cvx_begin
    variable S(n,n) symmetric
    maximize( x'*S*x )
    subject to
        diag(S) == vars;
        S(1,2) >= 0;
        S(1,3) <= 0;
        S == semidefinite(n);
cvx_end
```

Consider for example the case when the portfolio is given by the vector $x = (0.1, 0.5, 0.4)$. The maximum portfolio variance is $\sigma_{\max}^2 = 0.1584$. We can compare this number with the *best* (that is, smallest) portfolio achievable. Replacing the maximization by minimization (which is permitted in SDP, as the objective is linear), we obtain the most optimistic value of $\sigma_{\min}^2 = 0.0367$. Thus, under our uncertainty model for the covariance, the portfolio variance can change in relative amount by the staggering amount of

$$\frac{\sigma_{\max}^2 - \sigma_{\min}^2}{\sigma_{\max}^2} \approx 70\%.$$

- Motivation: maximum cut (MAXCUT) problem
- Boolean quadratic programming

- Rank-constrained formulation
- SDP relaxation
- Sub-optimal solution via the SDP relaxation

Motivation: MAXCUT Problem

We consider an undirected graph with n nodes, described by its weight matrix $W \in \mathbf{S}^n$, with $W_{ij} = W_{ji} \geq 0$ the weight of the arc joining node i and j (with the convention that the weight is zero if there are no edges connecting them). The *maximum cut* problem is to find a partition of the nodes in two complementary groups, so that the total weight of the edges that connect the two groups is maximized.

The max-cut problem can be formalized via the following boolean quadratic problem:

$$\max_x \frac{1}{2} \sum_{i < j} W_{ij}(1 - x_i x_j) : x_i^2 = 1, \dots, n.$$

Here, the boolean variable $x \in \{-1, 1\}^n$ corresponds to a particular assignment of each node into one of the two groups. The constraints in the above problem enforce the Boolean conditions on x . We check that the term $W_{ij}(1 - x_i x_j)$ is zero if and only if x_i and x_j are equal, which means that node i and node j are in the same group; otherwise, the cost is $2W_{ij}$.

Boolean QP

The above problem falls into the more general class of *Boolean quadratic programs*, which are of the form

$$\phi := \max_x x^T W x : x_i^2 = 1, \dots, n.$$

where $W \in \mathbf{S}^n$, with W_{ij} of arbitrary sign. Boolean QPs, as well as the special case of max-cut problems, are combinatorial, and hard to solve exactly. However, theory (based on SDP relaxations seen below) says that we can approximate the above quantity with a relative error of at most $\pi/2 - 1$ (the number falls to 0.87 in the case of MAX-CUT, which has special structure).

Rank-Constrained Formulation

We will introduce a semidefinite approximation to the problem, which is based on an expression of the original Boolean QP in terms of a *matrix* variable $X \in \mathbf{S}^n$, with components $X_{ij} = x_i x_j, 1 \leq i, j \leq n$. We can write the relationship between the vector x and the matrix X more compactly, as

$$X = xx^T.$$

We observe that X is positive semi-definite by construction.

We note that the objective of the Boolean QP, while quadratic in the original decision vector x , is actually linear in the matrix X , since

$$x^T W x = \text{Tr}(Wxx^T) = \text{Tr}(WX).$$

In the above, we have used the *Commutativity under trace* property, see [here](#). Likewise, the constraints can be written as linear functions of X :

$$x_i^2 = X_{ii}, \quad i = 1, \dots, n.$$

Finally, we note that a given symmetric matrix X has the form $X = xx^T$ for some vector $x \in \mathbf{R}^n$ if and only if it is positive semi-definite and its rank is one.

This means that we can formulate the Boolean QP in an equivalent, "rank-constrained" form:

$$\phi = \max_X \text{Tr}(WX) : X \succeq 0, \quad \text{rank}(X) = 1, \quad X_{ii} = 1, \quad i = 1, \dots, n. \quad \text{Semidefinite Relaxation}$$

The above rank-constrained problem is not convex, precisely due to the rank constraint. If we remove the constraint, we do obtain a convex problem—indeed, an SDP. This SDP is a *relaxation* of the original problem, that is, it is a new problem obtained by removing constraints. Since we are maximizing, the new problem yields an upper bound on the original one:

$$\phi \leq \psi := \max_X \text{Tr}(WX) : X \succeq 0, \quad X_{ii} = 1, \quad i = 1, \dots, n.$$

We can check that indeed the problem of computing ψ is an SDP:

- it involves a matrix variable that is constrained to the positive semi-definite cone;
- the other constraints are affine;
- the objective is linear.

The gap between ϕ (the true value of the combinatorial problem) and ψ (the SDP approximation) can be large, but not arbitrarily so. In fact, as mentioned above, it has been shown (by Nesterov in 1996) that the relative error satisfies

$$\frac{\psi - \phi}{\psi} \leq \frac{\pi}{2} - 1,$$

independent of problem size (and of data matrix W).

robust Stability of Linear Dynamical Systems [SDP](#) > [Conic problems](#) | [LMIs](#) | [Standard Forms](#) | [Applications](#) > [Back](#) | Robust Stability

- Motivation
- Lyapunov stability criterion
- Robust stability

- Robust linear control

Motivation

The time-behavior of many dynamical systems can be well modeled using a *linear* system of the form

$$x(t+1) = A(t)x(t) + B(t)u(t), \quad t = 0, 1, 2, \dots$$

where $x(t) \in \mathbf{R}^n$ is the state, which encapsulates the state of the system at time t , $u(t) \in \mathbf{R}^p$ contains control variables, and $A(t), B(t)$ are matrices of appropriate size. (For more details, see [here](#).) When $A(\cdot), B(\cdot)$ are constant, we say that the system is *time-invariant*.

A continuous-time version of the above takes the form

$$\dot{x}(t) := \frac{d}{dt}x(t) = A(t)x(t) + B(t)u(t), \quad t \geq 0.$$

We focus on these continuous-time models here.

A crucial problem in dynamical systems is asymptotic stability. Roughly speaking, we would like to know if, irrespective of the initial condition $x(0)$, and when there are no inputs ($u(t) = 0$ for every t), then $x(t) \rightarrow 0$ when $t \rightarrow +\infty$. If not, we would like to see how to act on the system with certain inputs that are functions of the state, so that the resulting system becomes stable.

Lyapunov stability criterion

Lyapunov invented (circa 1890) a method that provides a (in general, only sufficient) condition for asymptotic stability. Assume that there is a positive-definite function of the state, $V(x) = x^T P x$ that decreases strictly along every trajectory. Then x converges to zero as $t \rightarrow +\infty$. The physical interpretation of V is that it acts as a total energy for the system. For mechanical systems, V can be chosen to contain the kinetic and potential energy.

When $u \equiv 0$, and with $\dot{x}(t) = A(t)x(t)$, we have

$$\dot{V} = 2x^T P \dot{x} = 2x(t)^T (A(t)^T P + PA(t))x(t).$$

Thus a sufficient condition for stability is $\dot{V} < 0$ on any trajectory. Expressing this condition for $t = 0$ and for arbitrary $x(0)$ results in the following sufficient condition for asymptotic stability.

Lyapunov's sufficient condition for asymptotic stability: there exist $P = P^T \in \mathbf{R}^{n \times n}$ such that

1. P is positive definite;
2. for every $t \geq 0$, $A(t)^T P + PA(t)$ is negative definite.

It turns out that for time-invariant systems ($A(t)$ independent of t), the above is also necessary.

Robust stability

Now assume that $A(t)$ is not completely known, say it can take its values arbitrarily in a finite set $\mathbf{A} := \{A_1, \dots, A_L\}$. The matrices A_1, \dots, A_L can represent the behavior of the system under different operating conditions, and allow to model the uncertainty about the model's parameters. Based on Lyapunov's condition above, the following condition guarantees stability *irrespective* of the choice of $A(t)$ in \mathbf{A} .

Lyapunov's sufficient condition for robust asymptotic stability: there exist $P = P^T \in \mathbf{R}^{n \times n}$ such that

1. $P = P^T$ is positive definite;
2. for every $i = 1, \dots, L$, $A_i^T P + PA_i$ is negative definite.

The above is a semidefinite program, in matrix variable P . (There is no objective function here; this is a feasibility problem.)

Robust linear control

What if a system is not stable? One way to stabilize it is via *linear feedback*, using this time an input that is a linear function of the state (which we assume is measured, hence available). That is, we set $u(t) = Kx(t)$ where K is a matrix of *control parameters* (or gains). The closed-loop system becomes

$$\dot{x}(t) = (A(t) + B(t)K)x(t)$$

Applying the stability condition to the system above leads to a condition in P, K , that

$$R(t) := (A(t) + B(t)K)^T P + P(A(t) + B(t)K)$$

should be negative-definite for every $t \geq 0$. The above matrix is not affine in the variables P, K . However, the condition for a matrix R to be negative-definite is equivalent to the fact that $Q := XRX^T$ is, where X is an arbitrary invertible matrix. Applying this to $X := P^{-1}$, and letting $U := KX$, we obtain that if there exist X, U with X positive-definite, such that for every $t \geq 0$,

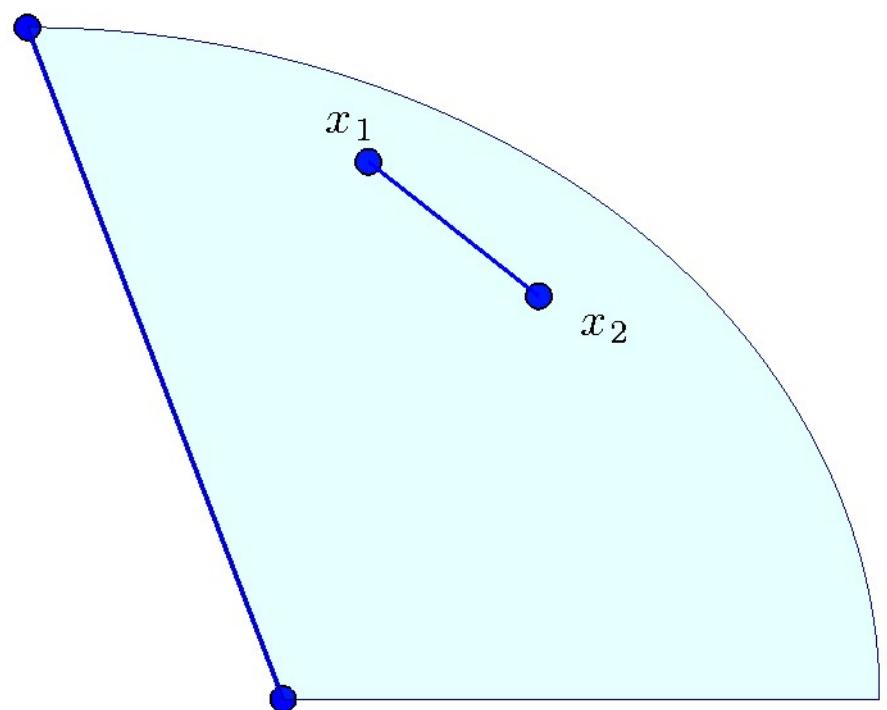
$$Q(t) := (A(t)X + B(t)U)^T P + P(A(t)X + B(t)U)$$

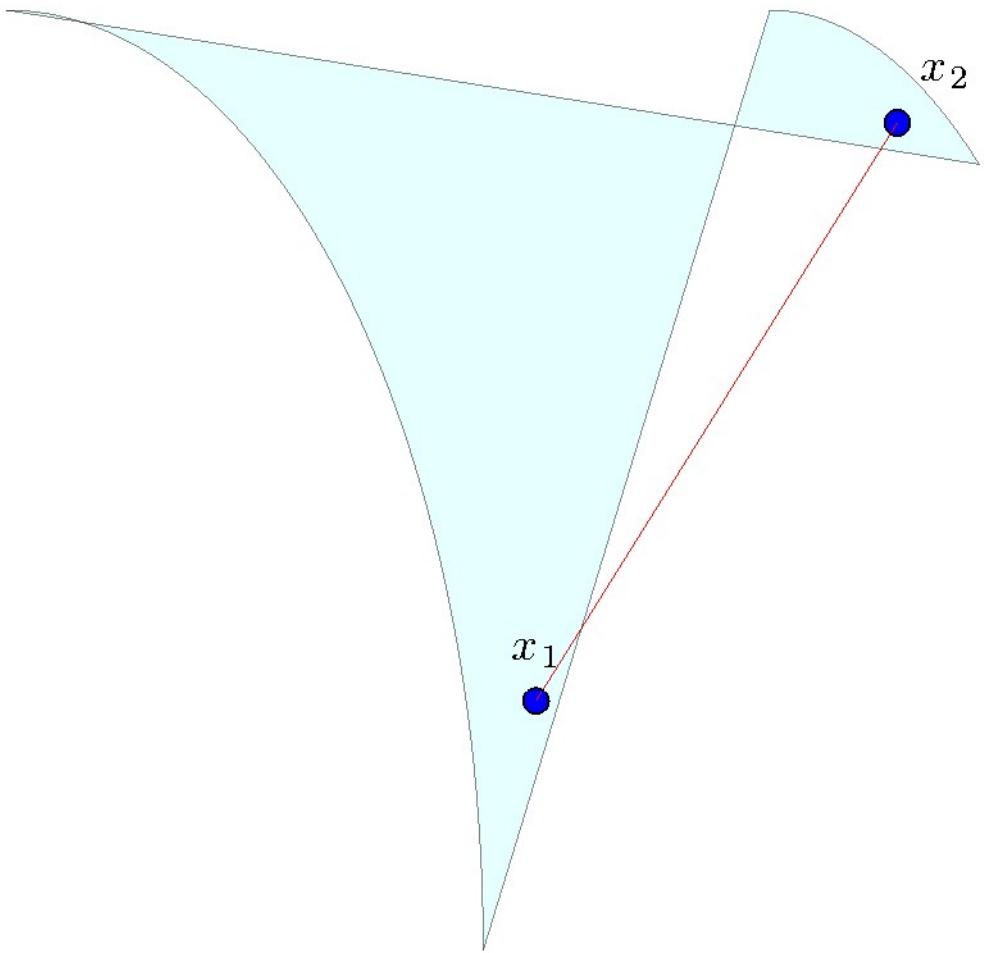
is negative-definite, then the system can be stabilized by linear feedback.

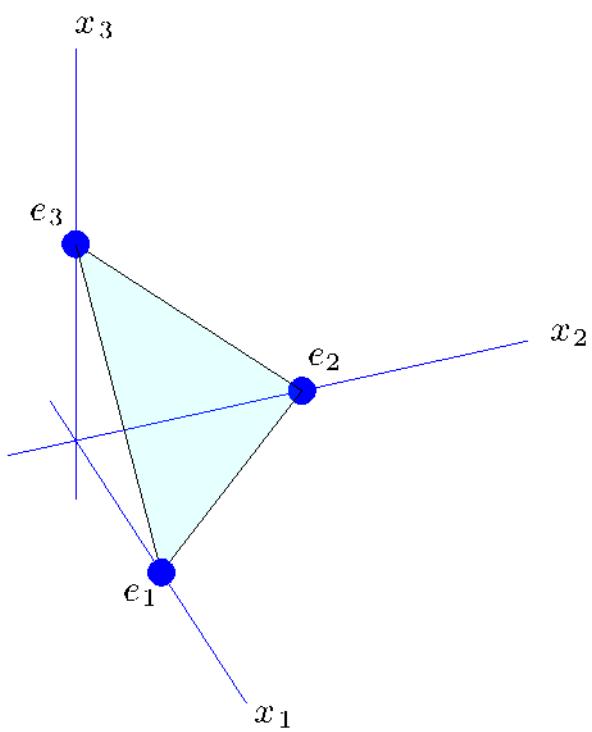
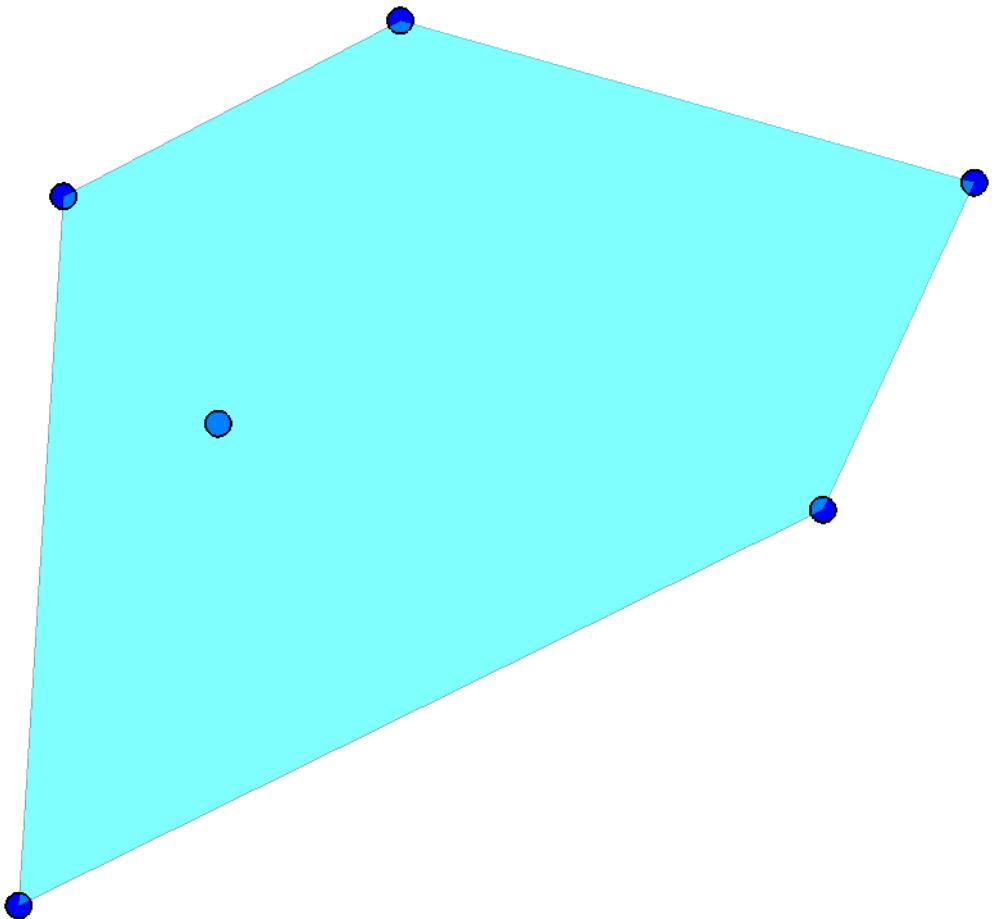
In the case when the matrix $[A(t), B(t)]$ is not completely known, say it can take its values arbitrarily in a finite set $\mathbf{C} := \{[A_1, B_1], \dots, [A_L, B_L]\}$, then the condition: there exist X, U with X positive-definite, such that:

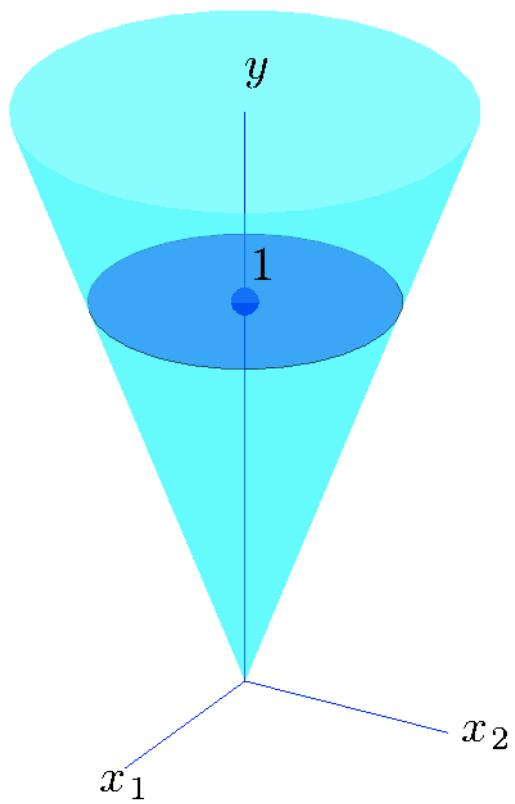
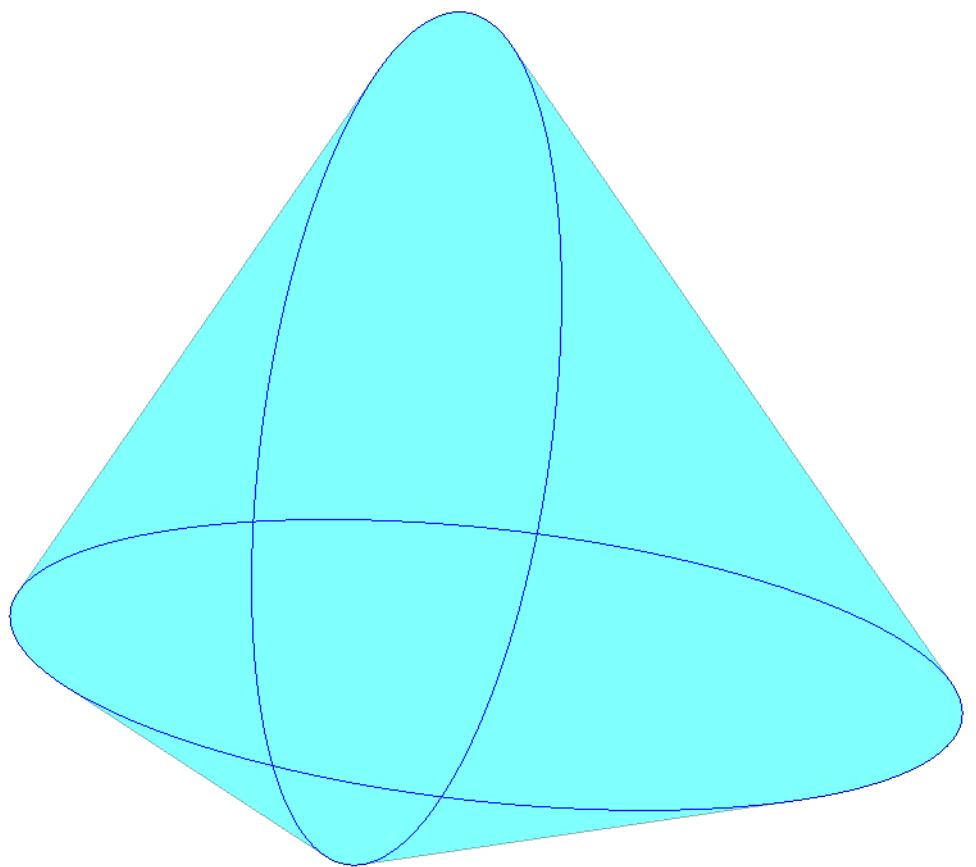
$$Q_i := (A_i X + B_i U)^T P + P(A_i X + B_i U)$$

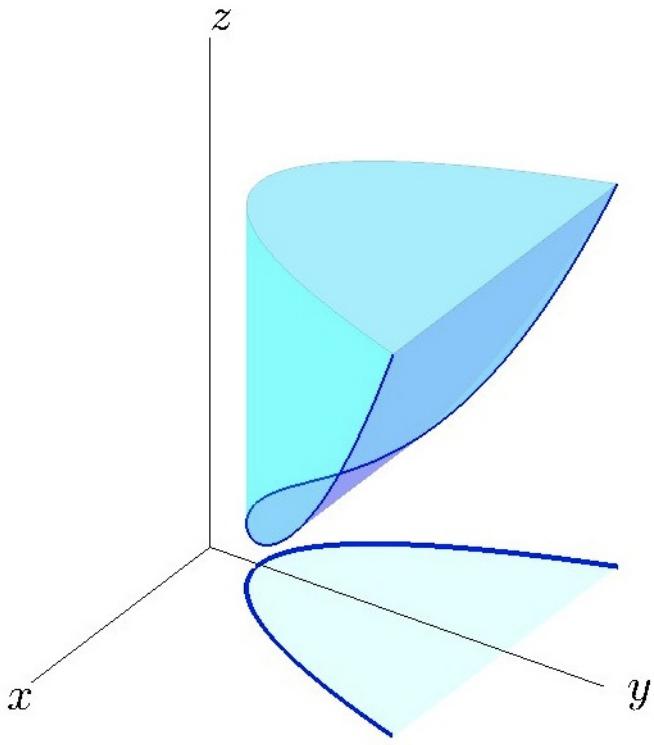
is negative definite, then the system is robustly stabilizable.









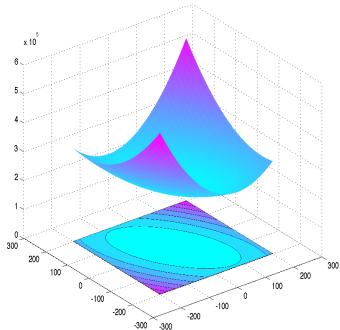


Quadratic functions in two variables

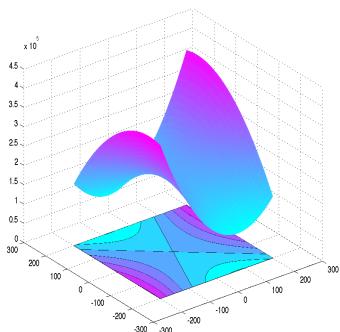
$$p(x) = 4x_1^2 + 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 2 \times 10^5,$$

$$\text{Two examples of quadratic function are } p, q : \mathbf{R}^2 \rightarrow \mathbf{R}, \text{ with values } q(x) = 4x_1^2 - 2x_2^2 + 3x_1x_2 + 4x_1 + 5x_2 + 2 \times 10^5.$$

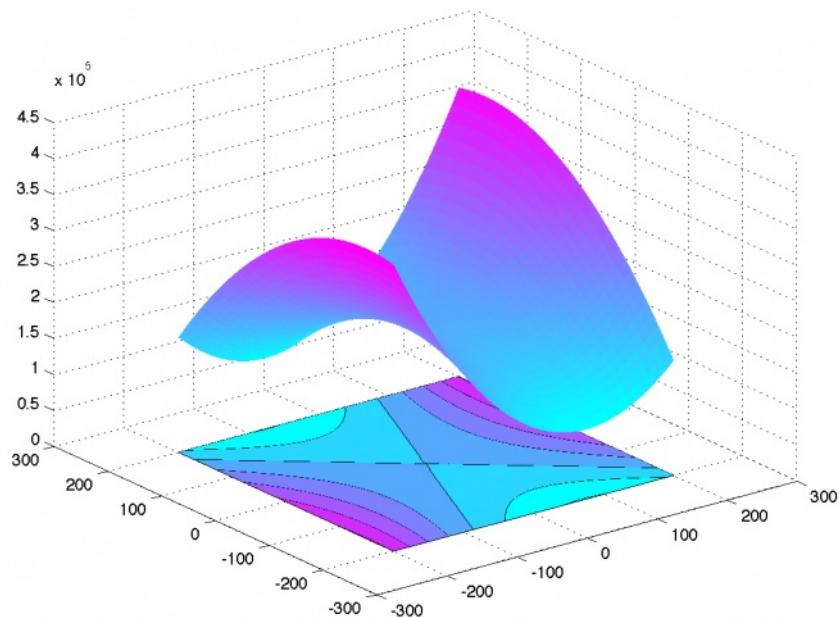
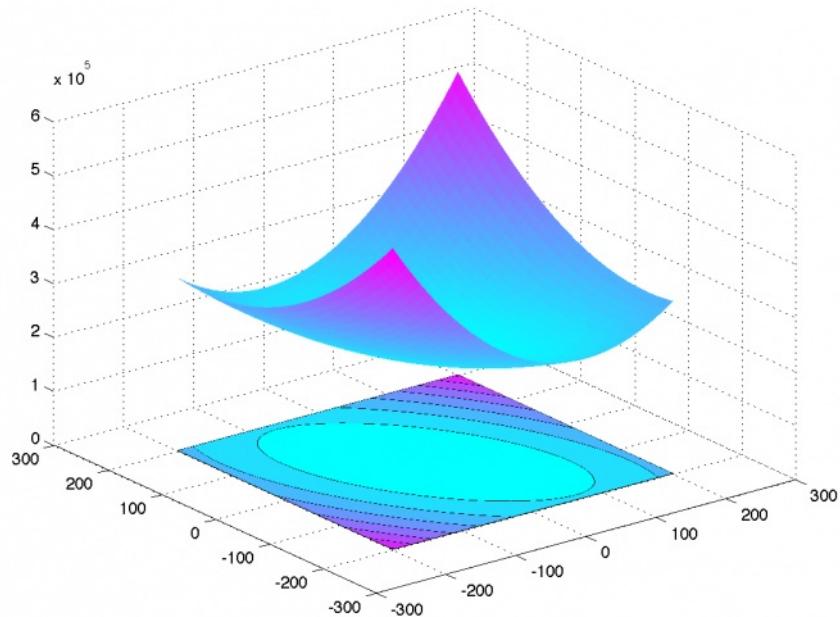
The function $r(x) = 4x_1^2 + 2x_2^2 + 3x_1x_2$ is a form, since it has no linear or constant terms in it.

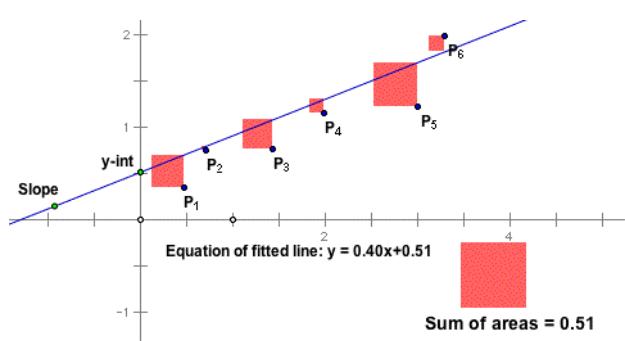
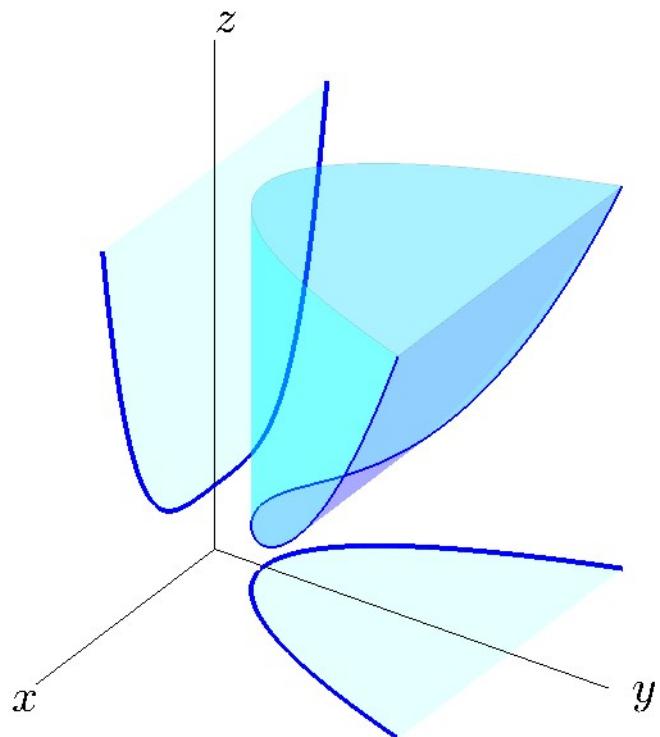


Level sets and graph of the quadratic function p . The epigraph is anything that extends above the graph in the z -axis direction. This function is “bowl-shaped”, or convex.

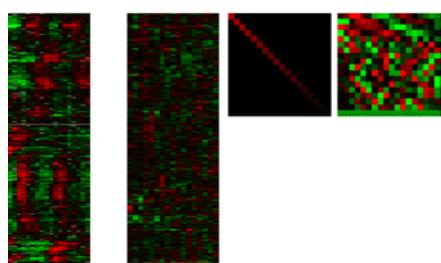


Level sets and graph of the quadratic function q .





$$A = U \cdot W \cdot V^T$$



A popular model for the prediction of time series is based on the so-called auto-regressive (AR) model

$$y_t = \theta_1 y_{t-1} + \dots + \theta_m y_{t-m}, \quad t = 1, \dots, m,$$

where θ_i 's are constant coefficients, and m is the *memory length* of the model. The interpretation of the model is that the next output is a linear function of the past. Elaborate variants of auto-regressive models are widely used for prediction of time series arising in finance and economics.

To find the coefficient vector theta in \mathbf{R}^m , we collect observations $(y_t)_{0 \leq t \leq T}$ (with $T \geq m$) of the time series, and try to minimize the total squared error in the above equation:

$$\min_{\theta} : \sum_{t=m}^T (y_t - \theta_1 y_{t-1} - \dots - \theta_m y_{t-m})^2.$$

This can be expressed as a linear least-squares problem, with appropriate data A, y .

See also: [Linear regression via Least-Squares](#).

T

- [Tomography](#)
- [Trace of a matrix](#)
- [Triangle inequality](#)
- [Triangular matrices](#)
- [Triangular systems of linear equations](#), see also [backward substitution](#) algorithm

N

- Norms: [general definition](#), for [vectors](#), for [matrices](#); see also dual norm
- [Nullspace](#) of a matrix

O

- [Optimal point, optimal value, optimal set](#)
- Orthogonal: [vectors](#), [matrices](#)

Vectors [Vectors, Matrices](#) > Vectors | [Scalar products](#) | [Matrices](#) | [Linear functions](#) | [Applications](#)

- Definitions
- Independence
- Subspaces, span, affine sets
- Basis, dimension

Definitions Vectors

Assume we are given a collection of n real numbers, x_1, \dots, x_n . We can represent them as n locations on a line. Alternatively, we can represent the collection as a single point in a n -dimensional space. This is the *vector* representation of the collection of numbers; each number is called a *component* of the vector.

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

Vectors can be arranged in a column, or a row; we usually write vectors in column format: If $x \in \mathbf{R}^n$ denotes a vector, we use subscripts to denote components, so that x_i is the i -th component of x . Sometimes we use the in-line notation $x = (x_1, \dots, x_n)$.

Example: The vector $x = (2, 1)$ in \mathbf{R}^2 .

Examples:

- [Temperatures at different airports](#).
- [Bag-of-words representation of text](#).

Transpose

If x is a column vector, x^T denotes the corresponding row vector, and vice-versa. Hence, if x is the column vector above: $x^T = (x_1 \dots x_n)$.

A column vector $x = (2, 3, 1, 4)$ and its transpose y can be declared in Matlab's workspace as follows. We can also declare y as a row vector directly.

Matlab syntax

```
>> x = [2; 3.1; -4]; % declare a column vector using ;
>> y = x'; % ' transposes the vector
>> y = [2,3.1,-4]; % can also declare a row vector with commas.
```

Independence

$$\sum_{i=1}^m \lambda_i x_i = 0 \quad \text{implies } \lambda = 0.$$

A set of vectors $\{x_1, \dots, x_n\}$ in \mathbf{R}^n , $i = 1, \dots, m$ is said to be *independent* if and only if the following condition on a vector $\lambda \in \mathbf{R}^m$: $\sum_{i=1}^m \lambda_i x_i = 0$ implies $\lambda = 0$. This means that no vector in the set can be expressed as a linear combination of the others.

Example: the vectors $x_1 = [1, 2, 3]$ and $x_2 = [3, 6, 9]$ are *not* independent, since $x_1 - 3x_2 = 0$.

Subspace, span, affine sets

A *subspace* of \mathbf{R}^n is a subset that is closed under addition and scalar multiplication. Geometrically, subspaces are “flat” (like a line or plane in 3D) and pass through the origin. A subspace \mathbf{S} can always be represented as the *span* of a set of vectors $x_i \in \mathbf{R}^n$, $i = 1, \dots, m$, which is the set

$$\mathbf{S} = \text{span}(x_1, \dots, x_m) := \left\{ \sum_{i=1}^m \lambda_i x_i : \lambda \in \mathbf{R}^m \right\}.$$

An *affine set* is a translation of a subspace—it is “flat” but does not necessarily pass through 0 , as a subspace would. think for example of a line, or a plane, that does not go through the origin. So an affine set \mathbf{A} can always be represented as the translation of the subspace spanned by some vectors:

$$\mathbf{A} = \left\{ x_0 + \sum_{i=1}^m \lambda_i x_i : \lambda \in \mathbf{R}^m \right\}, \quad \text{for some vectors } x_0, x_1, \dots, x_m. \text{ In shorthand notation, we write } \mathbf{A} = x_0 + \mathbf{S}.$$

$$u = \begin{bmatrix} -1 \\ 2 \\ 0.5 \end{bmatrix}, \quad v := \begin{bmatrix} 1 \\ 3 \\ 0.1 \end{bmatrix}$$

Example: In \mathbf{R}^3 , the span \mathbf{S} of the two vectors

When \mathbf{S} is the span of a single non-zero vector, the set \mathbf{A} is called a *line* passing through the point x_0 . Thus, lines have the form $\{x_0 + tu : t \in \mathbf{R}\}$ where u determines the direction of the line, and x_0 is a point through which it passes.

Example: A line in \mathbf{R}^2 passing through the point $x_0 = (2, 0)$, with direction $u = (0.8944, 0.4472)$.

Basis, dimensionBasis in \mathbf{R}^m

$$x = \sum_{i=1}^n \lambda_i u_i$$

A *basis* of \mathbf{R}^n is a set of n independent vectors. If the vectors u_1, \dots, u_n form a basis, we can express any vector as a linear combination of the u_i 's: for appropriate numbers $\lambda_1, \dots, \lambda_n$.

The *standard basis* (alternatively, natural basis) in \mathbf{R}^n consists of the vectors e_i , where e_i 's components are all zero, except the i -th, which is equal to 1. In \mathbf{R}^3 , we have

$$e_1 := \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad e_2 := \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad e_3 := \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Example: [A basis in \$\mathbf{R}^3\$](#) .

Basis of a subspace

The basis of a given subspace $\mathbf{S} \subseteq \mathbf{R}^n$ is any *independent* set of vectors whose span is \mathbf{S} . If the vectors u_1, \dots, u_r form a basis of \mathbf{S} , we can express any vector as a linear combination of the u_i 's: for appropriate numbers $\lambda_1, \dots, \lambda_r$.

The number of vectors in the basis is actually independent of the choice of the basis (for example, in \mathbf{R}^3 you need two independent vectors to describe a plane containing the origin). This number is called the *dimension* of \mathbf{S} . We can accordingly define the dimension of an affine subspace, as that of the linear subspace it is a translation of.

Examples:

- The dimension of a line is 1, since a line is of the form $x_0 + \text{span}(x_1)$ for some non-zero vector x_1 .
- [Dimension of an affine subspace](#).

Matrices [Vectors, Matrices](#) | [Vectors](#) | [Scalar products](#) | Matrices | [Linear functions](#) | [Applications](#)

- Matrices as collections of vectors
- Matrix-vector product
- Matrix-matrix product
- Trace, scalar product
- Some special matrices

Matrices as collections of vectors

Matrices can be viewed simply as a collection of vectors of same size, that is, as a collection of points in a high-dimensional space.

Matrices as collections of columns

Matrices can be described in column-wise fashion: given n vectors a_1, \dots, a_n in \mathbf{R}^m , we can define the $m \times n$ matrix A with a_j 's as columns:
 $A = [\begin{array}{ccc} a_1 & \dots & a_n \end{array}]$. Geometrically, A represents n points in a m -dimensional space.

Transpose

The notation A_{ij} denotes the element of A sitting in row i and column j . The *transpose* of a matrix A , denoted by A^T , is the matrix with (i, j) element A_{ji} , $i = 1, \dots, m$, $j = 1, \dots, n$.

Matrices as collections of rows

Similarly, we can describe a matrix in row-wise fashion: given m vectors b_1, \dots, b_m in \mathbf{R}^n , we can define the $m \times n$ matrix B with the transposed vectors b_i^T as rows:

$$B = \begin{bmatrix} b_1^T \\ \vdots \\ b_m^T \end{bmatrix}.$$

Geometrically, B represents m points in a n -dimensional space.

The notation $\mathbf{R}^{m \times n}$ denotes the set of $m \times n$ matrices.

Examples:

- [A simple \$3 \times 2\$ matrix](#).
- [Arc-node incidence matrix of a network](#).
- [Matrix of votes in the US Senate, 2004-2006](#).

Matrix-vector product

We define the matrix-vector product between a $m \times n$ matrix and a n -vector x , and denote by Ax , the m -vector with i -th component

$$(Ax)_i = \sum_{j=1}^n A_{ij} x_j, \quad i = 1, \dots, m.$$

If the columns of A are given by the vectors a_i , $i = 1, \dots, n$, so that $A = (a_1, \dots, a_n)$, then Ax can be interpreted as a linear combination of these columns, with

$$Ax = \sum_{i=1}^n x_i a_i.$$

weights given by the vector x :

$$A = \begin{bmatrix} a_1^T \\ \vdots \\ a_m^T \end{bmatrix}, \quad Ax = \begin{bmatrix} a_1^T x \\ \vdots \\ a_m^T x \end{bmatrix}.$$

Alternatively, if the rows of A are the row vectors a_i^T , $i = 1, \dots, m$:

$$\text{then } Ax \text{ is the vector with elements } a_i^T x, \quad i = 1, \dots, m;$$

Examples:

- Return to the [network example](#), involving a $m \times n$ incidence matrix. We note that, by construction, the columns of A sum to zero, which can be compactly written as $\mathbf{1}^T A = 0$, or $A^T \mathbf{1} = 0$.
- [Network flow](#).
- [Absorption spectrometry: using measurements at different light frequencies](#).
- [Average of votes in the US Senate, 2004-2006](#).

Matrix-matrix productDefinition

We can extend matrix-vector product to matrix-matrix product, as follows. If $A \in \mathbf{R}^{m \times n}$ and $B \in \mathbf{R}^{n \times p}$, the notation AB denotes the $m \times p$ matrix with i, j element

$$(AB)_{ij} = \sum_{k=1}^n A_{ik} B_{kj}.$$

given by It can be shown that transposing a product changes the order, so that $(AB)^T = B^T A^T$.

Column-wise interpretation

If the columns of B are given by the vectors b_i , $i = 1, \dots, n$, so that $B = [b_1, \dots, b_n]$, then AB can be written as

$$AB = A [\begin{array}{ccc} b_1 & \dots & b_n \end{array}] = [\begin{array}{ccc} Ab_1 & \dots & Ab_n \end{array}].$$

In other words, AB results from transforming each column b_i of B into Ab_i .

Row-wise interpretation

The matrix-matrix product can also be interpreted as an operation on the rows of A . Indeed, if A is given by its rows a_i^T , $i = 1, \dots, m$, then AB is the matrix obtained

$$AB = \begin{bmatrix} a_1^T \\ \vdots \\ a_m^T \end{bmatrix} B = \begin{bmatrix} a_1^T B \\ \vdots \\ a_m^T B \end{bmatrix}.$$

by transforming each one of these rows via B , into $a_i^T B$, $i = 1, \dots, m$. (Note that $a_i^T B$'s are indeed row vectors, according to our matrix-vector rules.)

Matrix-matrix products by blocks

Matrix algebra generalizes to blocks, provided block sizes are consistent. To illustrate this, consider the matrix-vector product between a $m \times n$ matrix A and a n -vector

$$A = [\begin{array}{cc} A_1 & A_2 \end{array}], \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

where A, x are partitioned in blocks, as follows: where A_i is $m \times n_i$, $x_i \in \mathbf{R}^{n_i}$, $i = 1, 2$, $n_1 + n_2 = n$. Then

$Ax = A_1 x_1 + A_2 x_2$. Likewise, if a $n \times p$ matrix B is partitioned into two blocks B_i , each of size n_i , $i = 1, 2$, with $n_1 + n_2 = n$, then

$$AB = [\begin{array}{cc} A_1 & A_2 \end{array}] \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = A_1 B_1 + A_2 B_2.$$

Example: [Gram matrix](#).

Trace, scalar product

The *trace* of a square $n \times n$ matrix A , denoted by $\text{Tr} A$, is the sum of its diagonal elements: $\text{Tr} A = \sum_{i=1}^n A_{ii}$.

Scalar product

$$\langle A, B \rangle = \text{Tr} A^T B = \sum_{i=1}^m \sum_{j=1}^m A_{ij} B_{ij}.$$

We can define the scalar product between two $m \times n$ matrices A, B via We can interpret the above scalar product as the (vector) scalar product between two long vectors of length mn each, obtained by stacking all the columns of A, B on top of each other.

Special matrices

Important classes of matrices include the following.

Identity matrix

The $n \times n$ identity matrix (often denoted I_n , or simply I , if context allows), has ones on its diagonal and zeros elsewhere. It is diagonal, symmetric, and orthogonal, and satisfies $A \cdot I_n = A$ for every matrix A with n columns.

Square matrices

Square matrices are matrices that have the same number of rows as columns.

Diagonal matrices

Diagonal matrices are square matrices A with $A_{ij} = 0$ when $i \neq j$.

Symmetric matrices

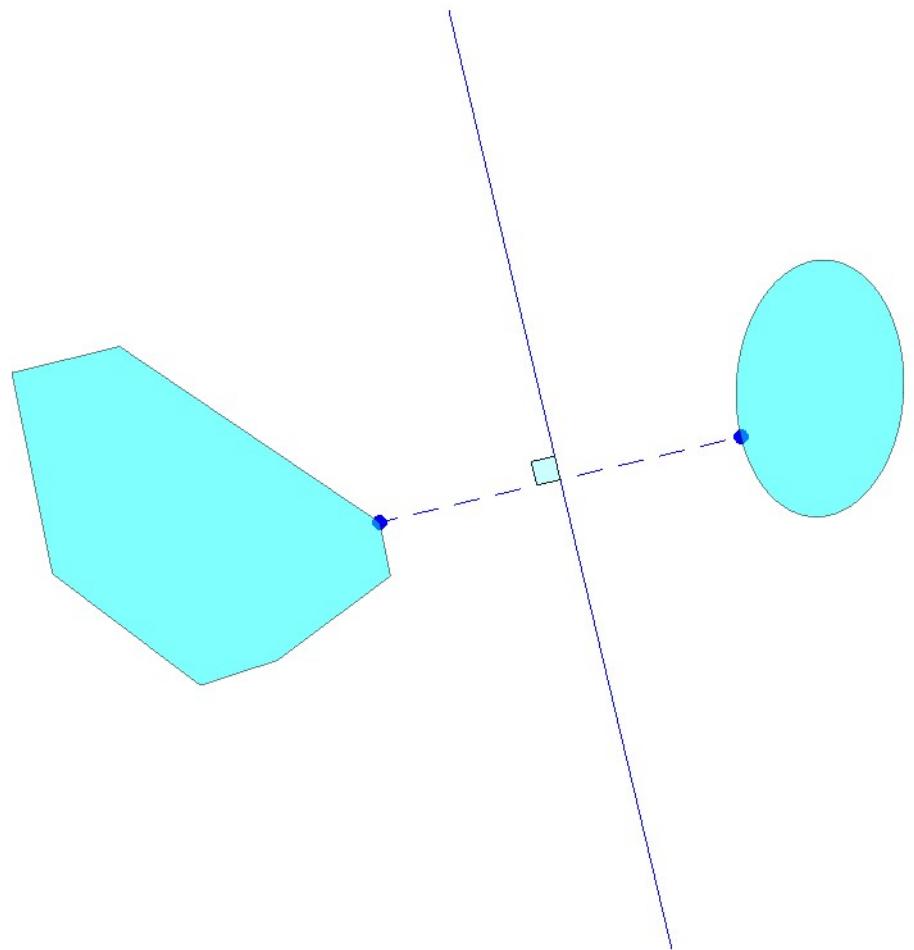
Symmetric matrices are square matrices that satisfy $A_{ij} = A_{ji}$ for every pair (i, j) . An entire [topic](#) is devoted to symmetric matrices.

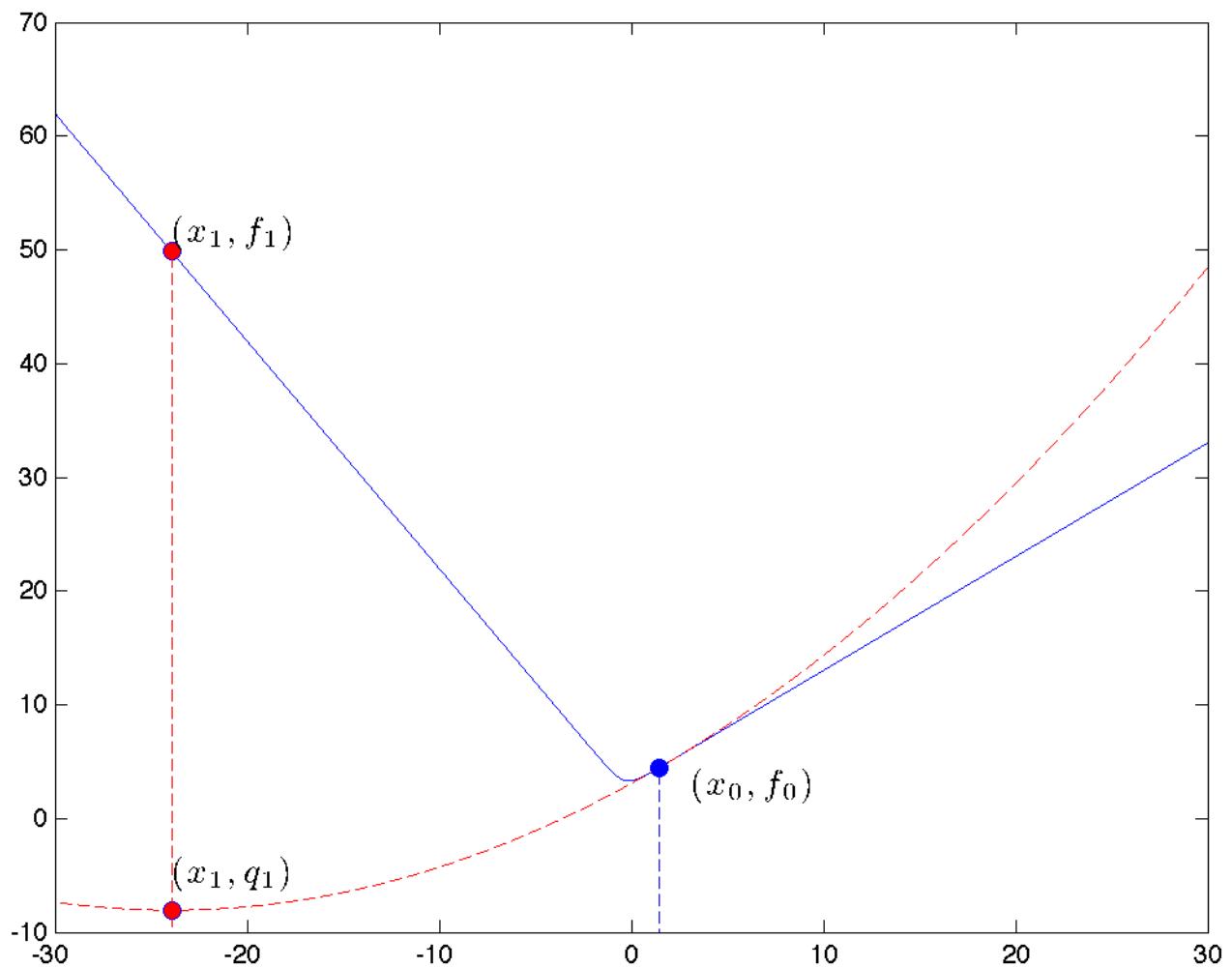
Orthogonal matrices

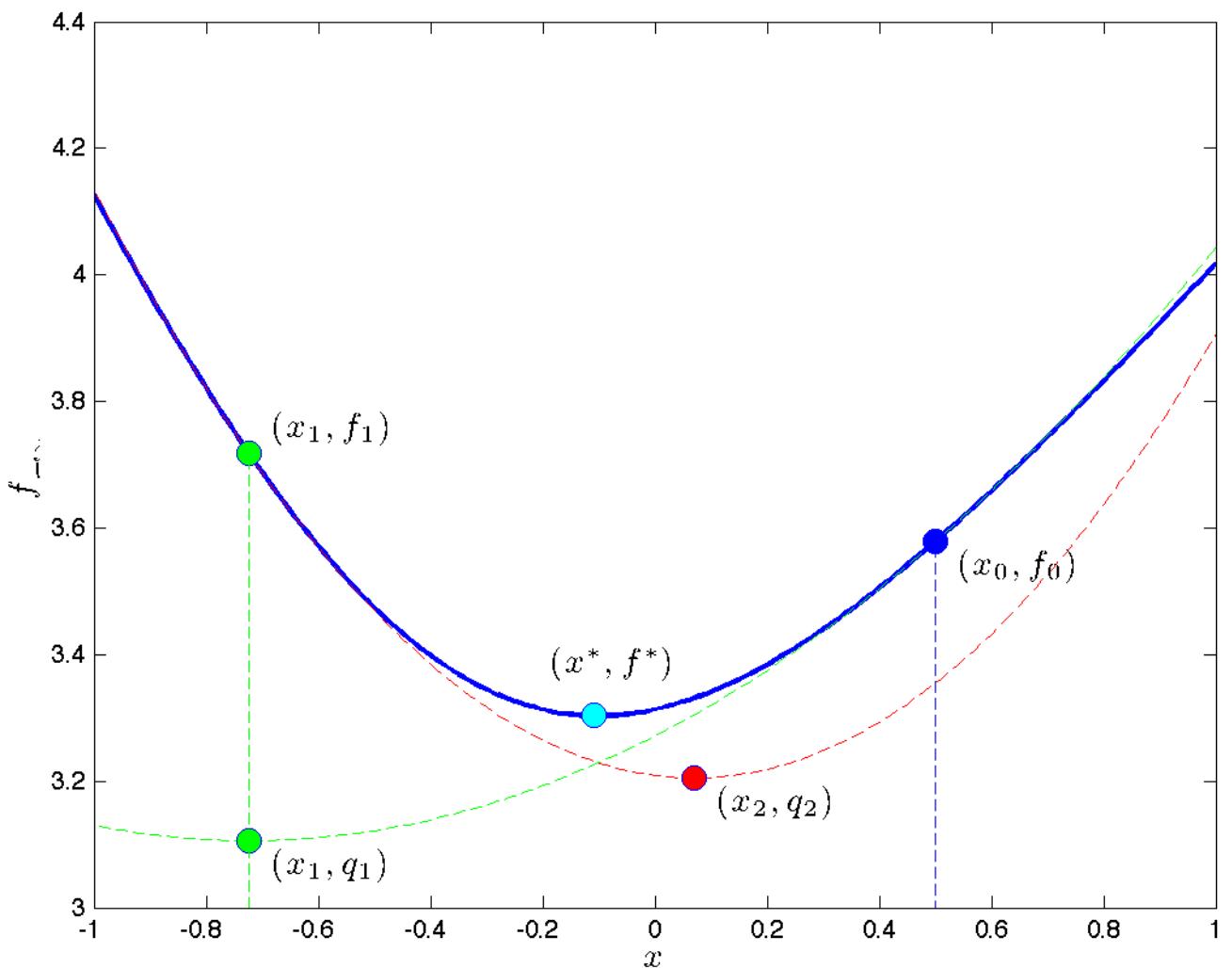
Orthogonal matrices are square matrices, such that the columns form an orthonormal basis. If $U = [u_1, \dots, u_n]$ is an orthogonal matrix, then $u_i^T u_i = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$ Thus, $U^T U = I_n$. Similarly, $U U^T = I_n$.

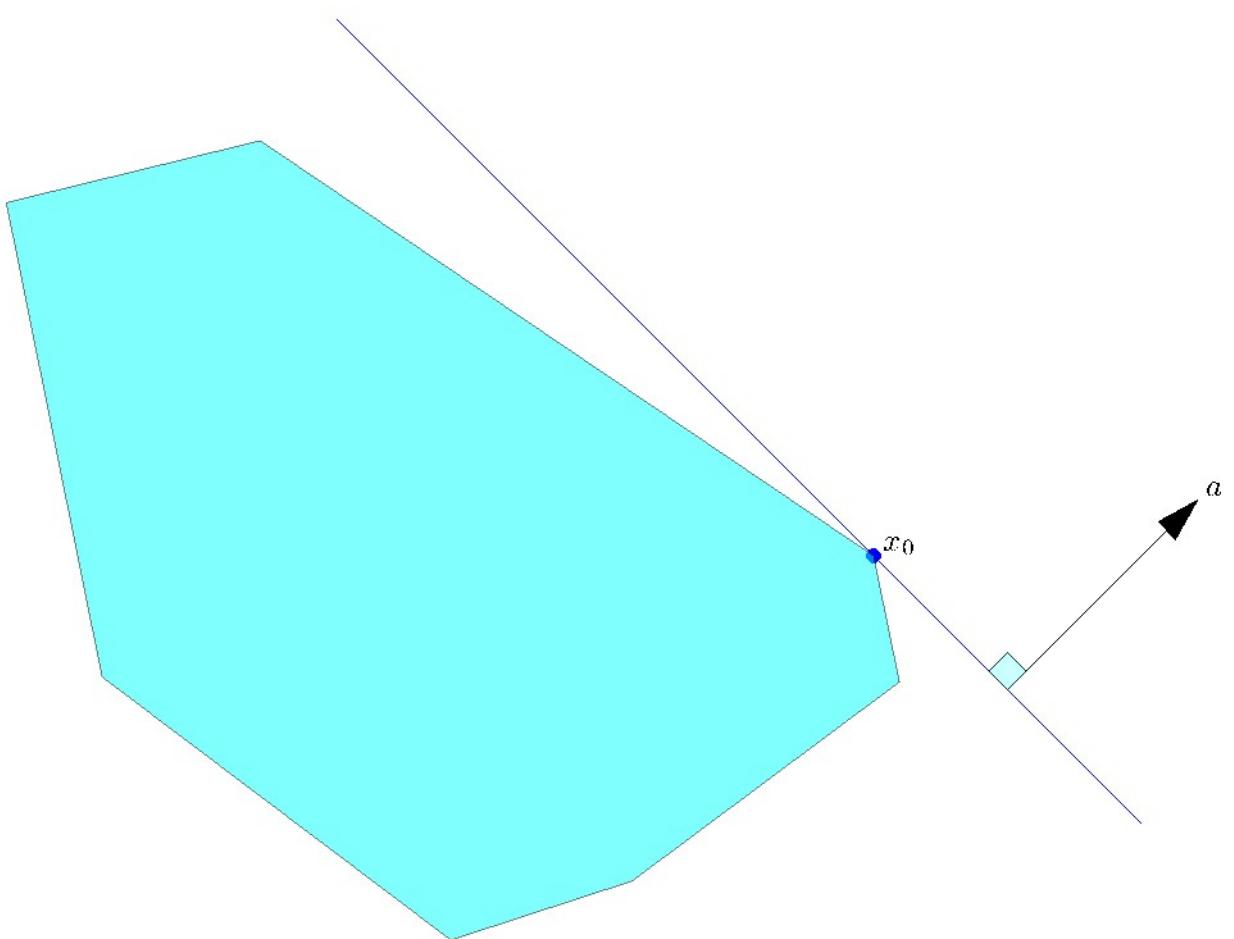
Orthogonal matrices correspond to bases that are a rotation of the standard basis. Their effect on a vector is to rotate it, leaving its length (Euclidean norm) invariant: for every vector x , $\|Ux\|_2^2 = (Ux)^T(Ux) = x^T U^T U x = x^T x = \|x\|_2^2$.

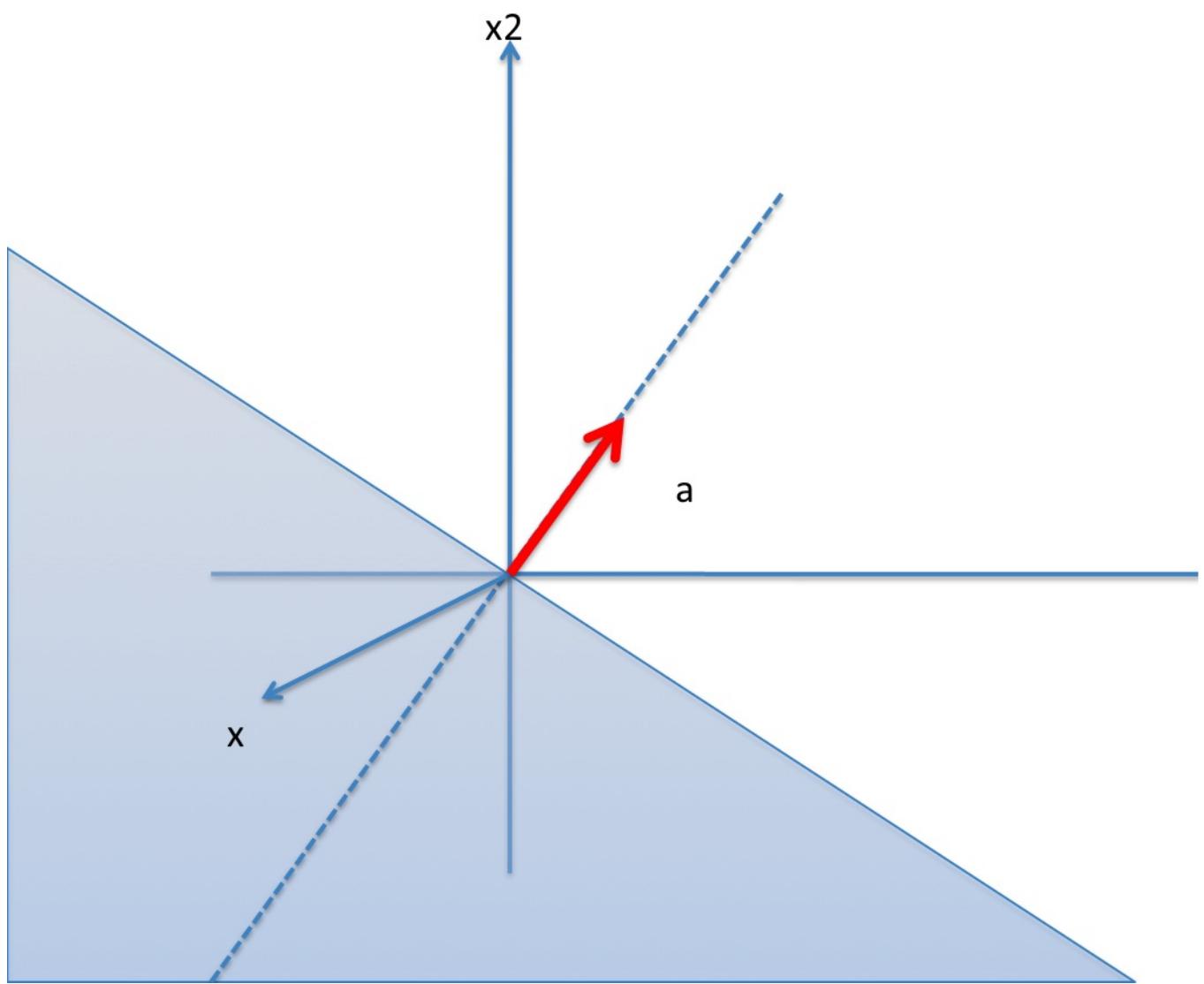
Example: [A \$2 \times 2\$ orthogonal matrix.](#)

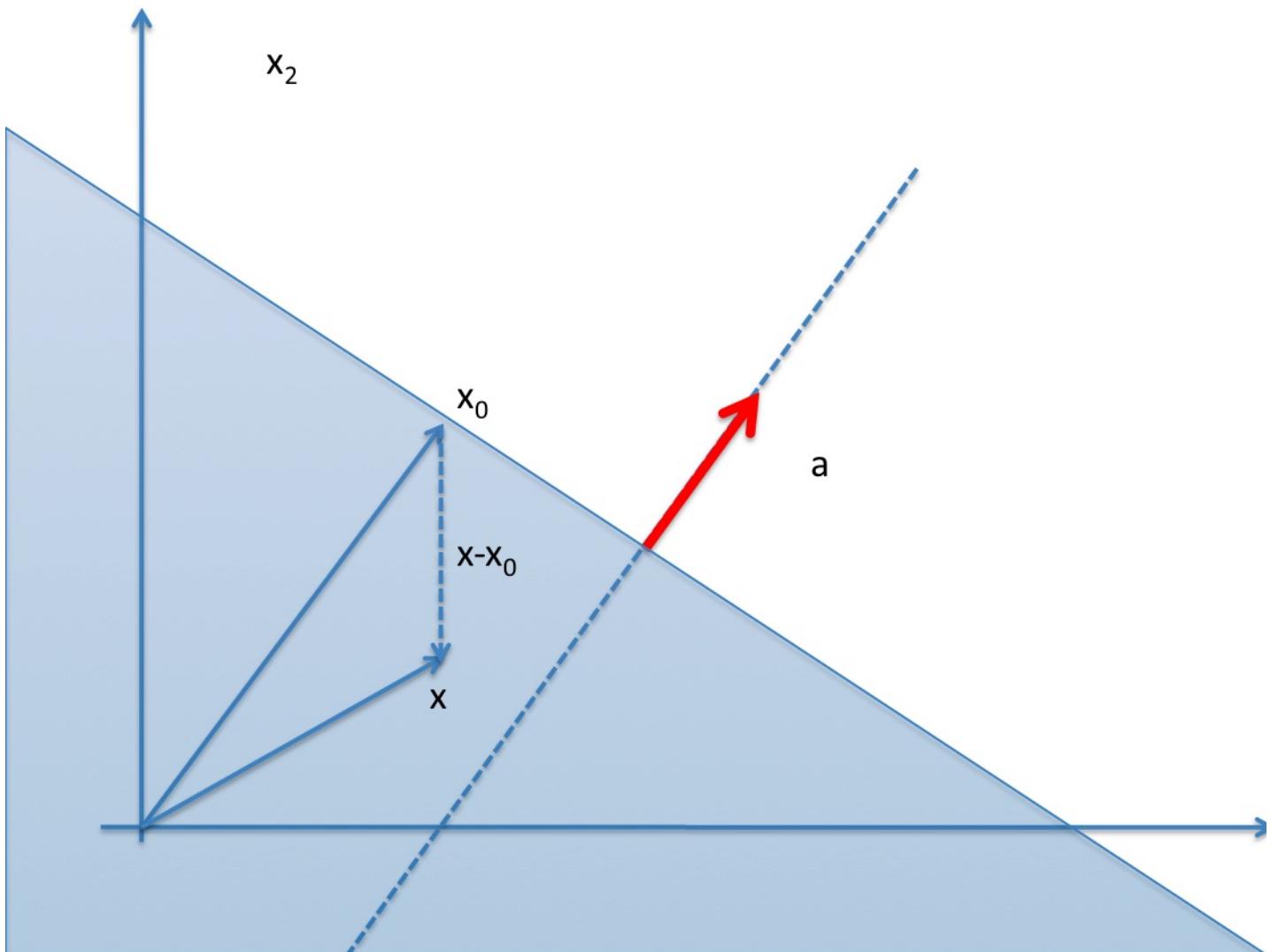


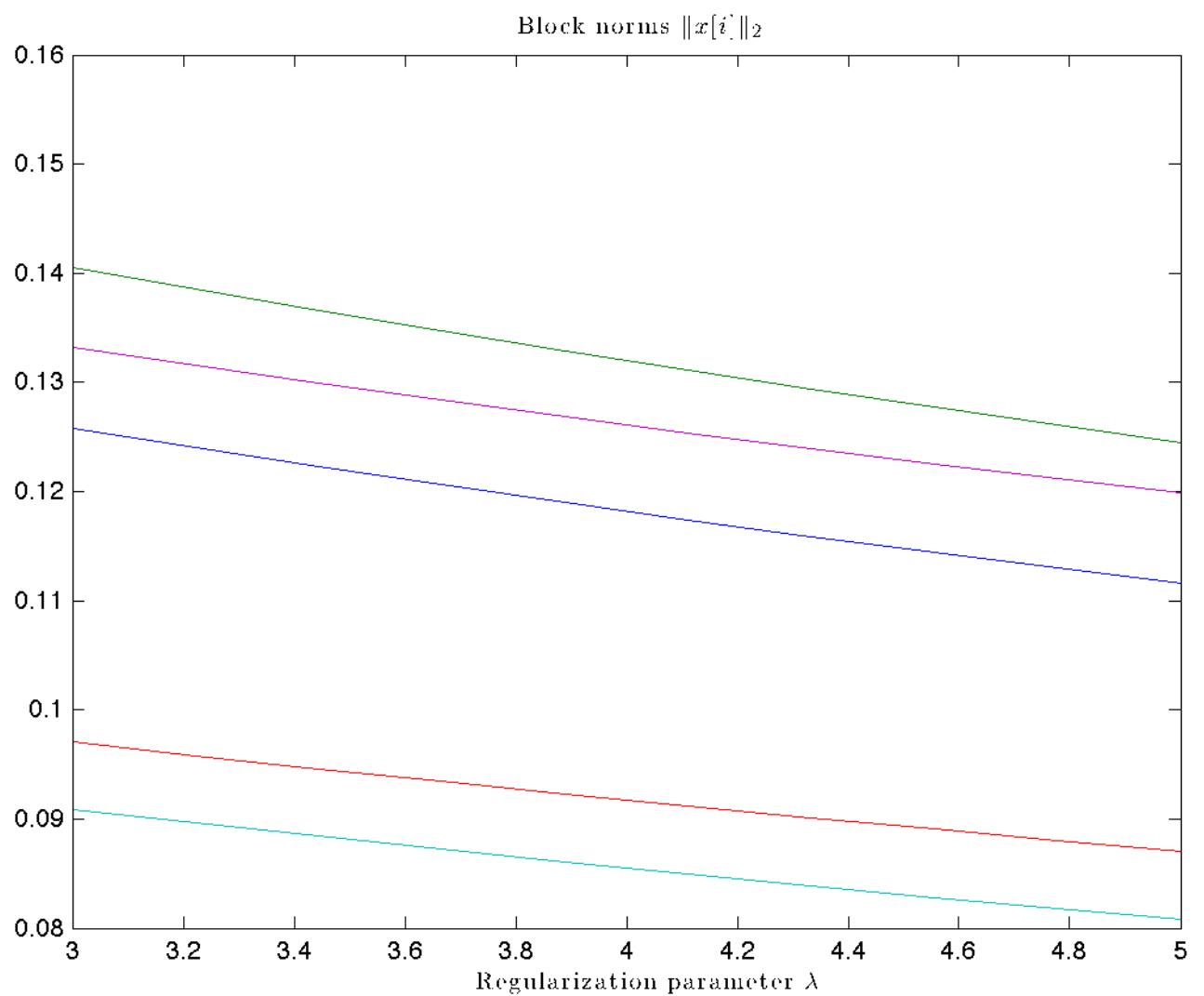


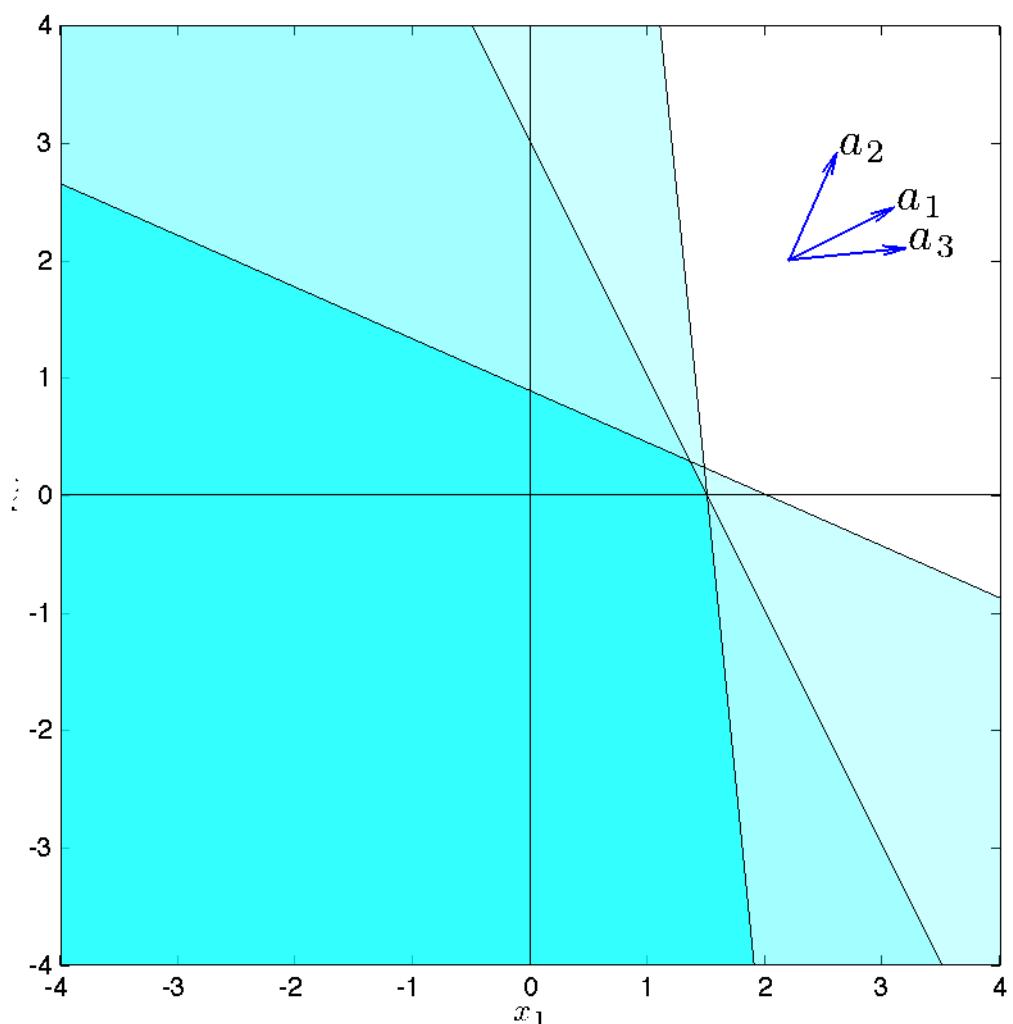


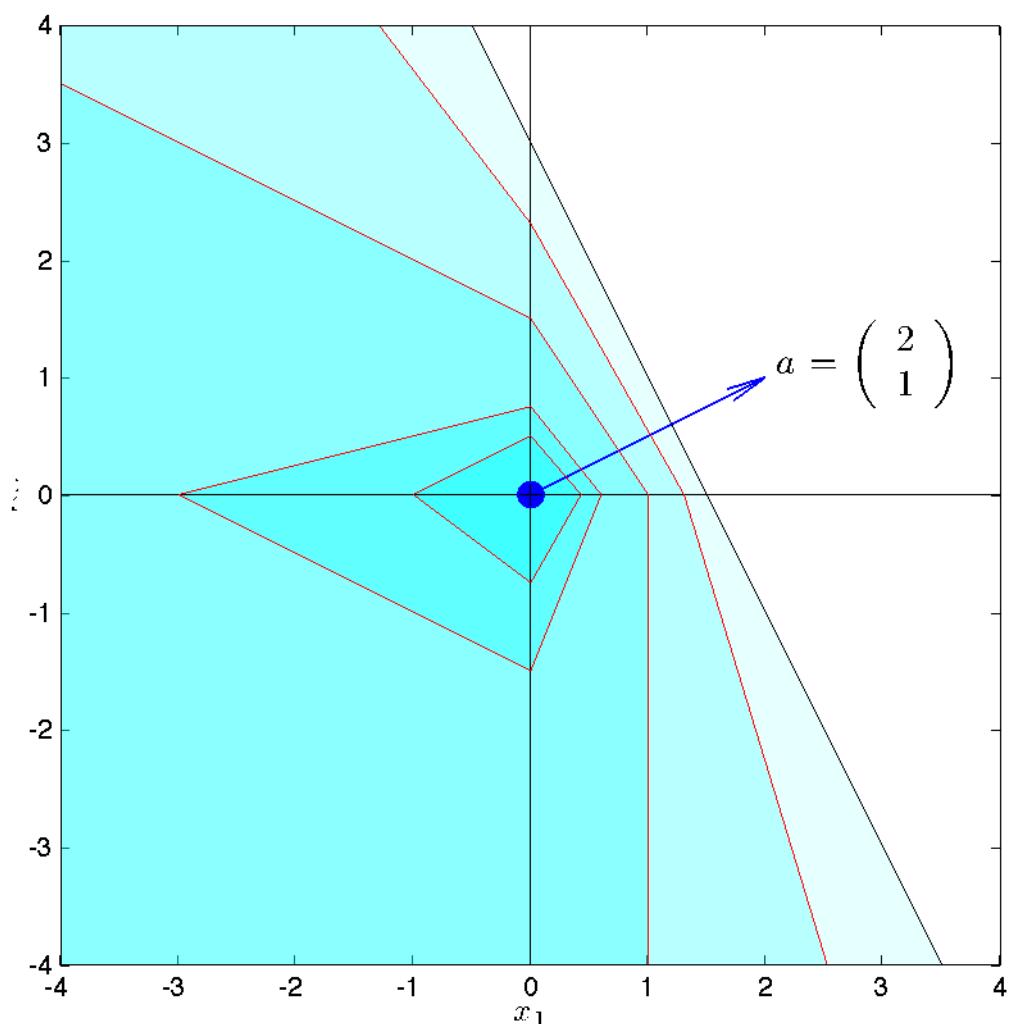


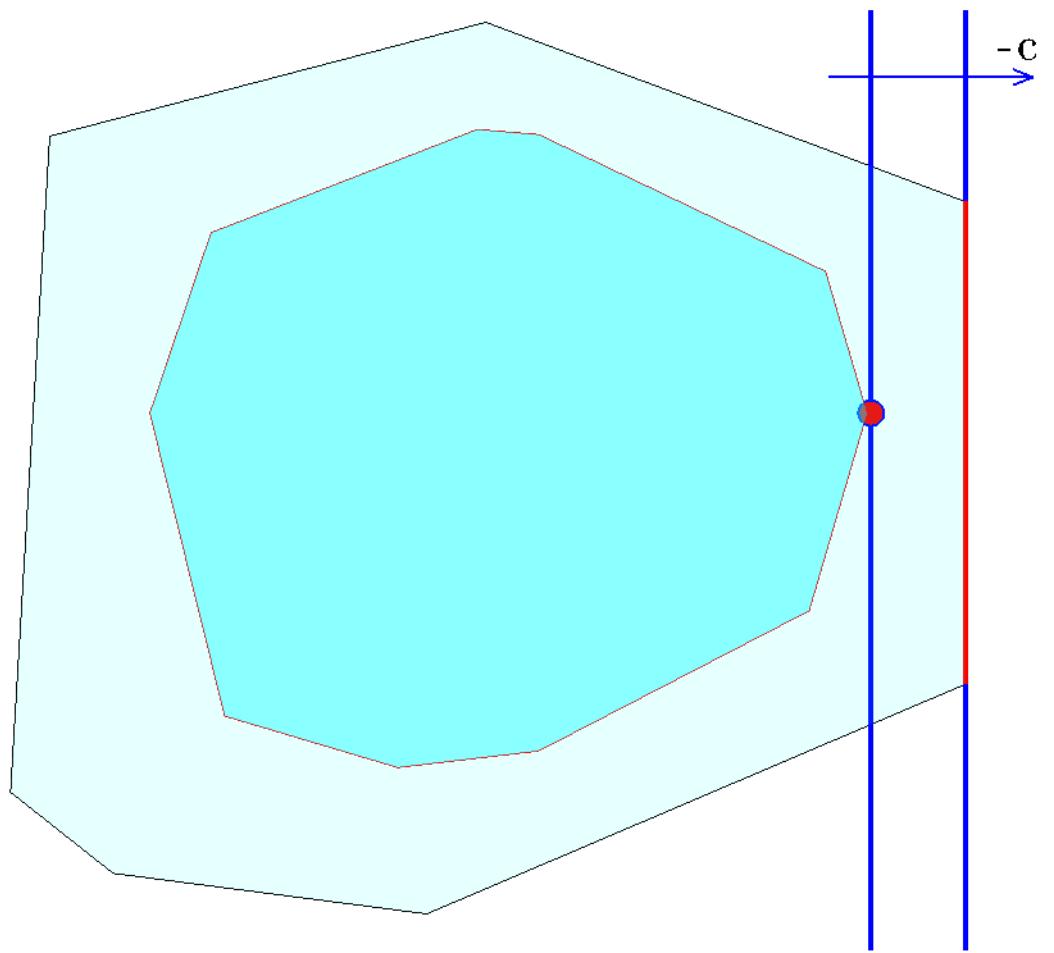


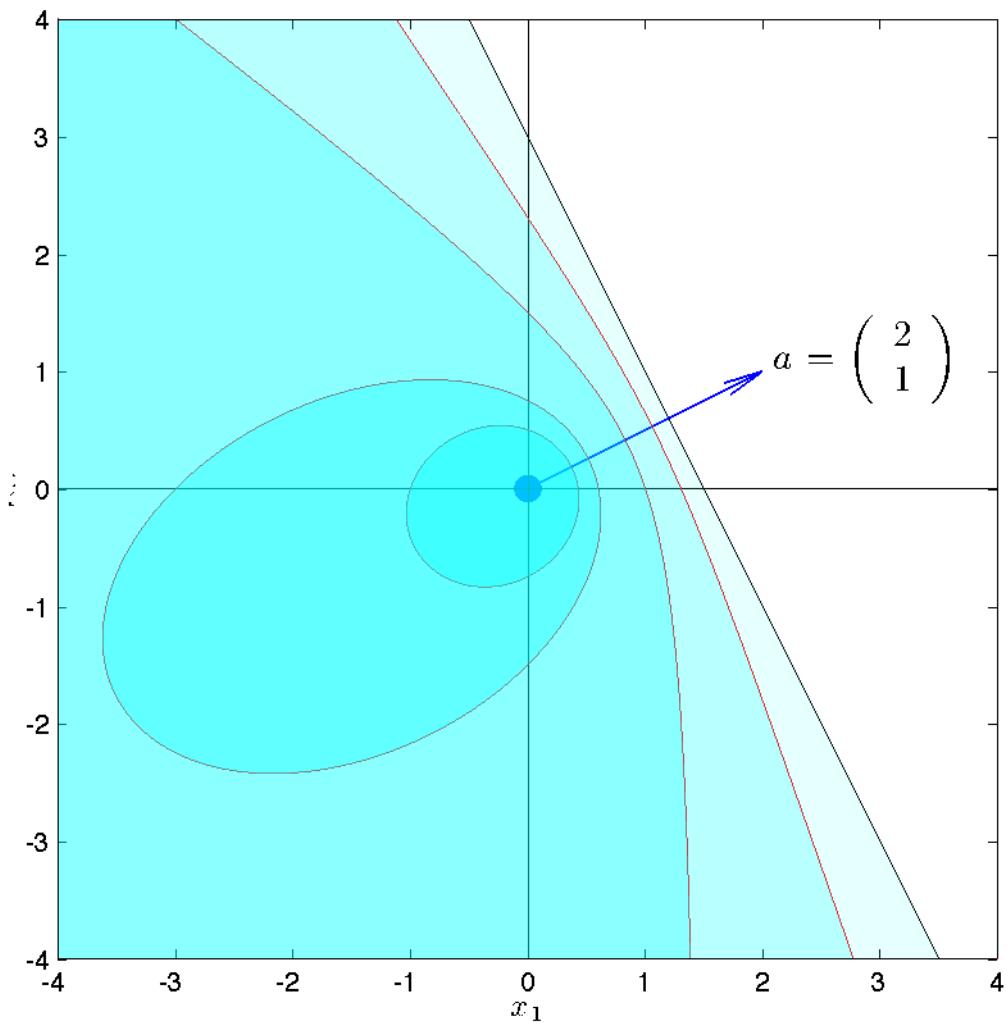












A

- [Absorbtion spectrometry](#)
 - [Affine function](#)
 - [Affine set](#)
 - [Angle](#) between vectors
 - [Auto-regressive models](#) for time-series prediction

B

- Backward substition method for solving triangular systems of linear equations
 - Bag-of-word representation of text
 - Basis of a subspace
 - Beer-Lambert law

C

- CAT scan imaging
 - Cauchy-Schwartz inequality
 - Cardinality of a vector
 - Cardinality minimization, see also l_1 -norm
 - Circuit design via Geometric Programming

- [Combinational logic](#) in circuit design
- [Componentwise inequality](#) between vectors
- [Condition number](#) of a matrix
- [Convex function](#)
- [Convex optimization problem](#)
- [Covariance matrix](#)
- [CVX](#)

D

- [Determinant](#) of a square matrix
- [Dimension of an affine set](#)
- [Domain of a function](#)
- Duality: [weak](#), [strong](#)
- [Dual norm](#)
- [Dual problem](#)
- [Dual function](#)
- [Dual norm](#), see also [norm](#)
- [Dyad](#)

E

- [Eigenvalue](#) decomposition (EVD) of a square, symmetric matrix
- [Epigraph of a function](#)
- [Expected value](#) of a random variable

Spectral Theorem [Symmetric Matrices](#) > [Definitions](#) | Spectral theorem | [PSD matrices](#) | [PCA](#) | [Applications](#)

- Eigenvalues and eigenvectors of symmetric matrices
- The symmetric eigenvalue decomposition theorem
- Rayleigh quotients

Eigenvalues and eigenvectors of symmetric matrices

Let A be a square, $n \times n$ symmetric matrix. A real scalar λ is said to be an *eigenvalue* of A if there exist a non-zero vector $u \in \mathbf{R}^n$ such that

$$Au = \lambda u.$$

The vector u is then referred to as an *eigenvector* associated with the eigenvalue λ . The eigenvector u is said to be *normalized* if $\|u\|_2 = 1$. In this case, we have

$$u^T Au = \lambda u^T u = \lambda.$$

The interpretation of u is that it defines a direction along A behaves just like scalar multiplication. The amount of scaling is given by λ . (In German, the root “eigen”, means “self” or “proper”). The eigenvalues of the matrix A are characterized by the *characteristic equation*

$$\det(\lambda I - A) = 0,$$

where the notation \det refers to the [determinant](#) of its matrix argument. The function with values $t \rightarrow p(t) := \det(tI - A)$ is a polynomial of degree n called the *characteristic polynomial*.

From the fundamental theorem of algebra, any polynomial of degree n has n (possibly not distinct) complex roots. For symmetric matrices, the eigenvalues are real, since $\lambda = u^T Au$ when $Au = \lambda u$, and u is normalized.

Spectral theorem

An important result of linear algebra, called the *spectral theorem*, or *symmetric eigenvalue decomposition* (SED) theorem, states that for any symmetric matrix, there are exactly n (possibly not distinct) eigenvalues, and they are all real; further, that the associated eigenvectors can be chosen so as to form an orthonormal basis. The result offers a simple way to decompose the symmetric matrix as a product of simple transformations.

Theorem: Symmetric eigenvalue decomposition

We can decompose any symmetric matrix $A \in \mathbf{S}^n$ with the *symmetric eigenvalue decomposition* (SED)

$$A = \sum_{i=1}^n \lambda_i u_i u_i^T = U \Lambda U^T, \quad \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n).$$

where the matrix of $U := [u_1, \dots, u_n]$ is orthogonal (that is, $U^T U = U U^T = I_n$), and contains the eigenvectors of A , while the diagonal matrix Λ contains the eigenvalues of A .

Here is a [proof](#). The SED provides a decomposition of the matrix in simple terms, namely [dyads](#).

We check that in the SED above, the scalars λ_i are the eigenvalues, and u_i 's are associated eigenvectors, since

$$A u_j = \sum_{i=1}^n \lambda_i u_i u_i^T u_j = \lambda_j u_j, \quad j = 1, \dots, n.$$

The eigenvalue decomposition of a symmetric matrix can be efficiently computed with standard software, in time that grows proportionately to its dimension n as n^3 . Here is the matlab syntax, where the first line ensure that matlab knows that the matrix A is exactly symmetric.

Matlab syntax

```
>> A = triu(A)+tril(A', -1);
>> [U, D] = eig(A);
```

Example:

- [Eigenvalue decomposition of a \$2 \times 2\$ symmetric matrix](#).

Rayleigh quotients

Given a symmetric matrix A , we can express the smallest and largest eigenvalues of A , denoted λ_{\min} and λ_{\max} respectively, in the so-called *variational* form

$$\lambda_{\min}(A) = \min_x \{x^T A x : x^T x = 1\}, \quad \lambda_{\max}(A) = \max_x \{x^T A x : x^T x = 1\}.$$

For a proof, see [here](#).

The term “variational” refers to the fact that the eigenvalues are given as optimal values of optimization problems, which were referred to in the past as variational problems. Variational representations exist for all the eigenvalues, but are more complicated to state.

The interpretation of the above identities is that the largest and smallest eigenvalues is a measure of the range of the quadratic function $x \rightarrow x^T A x$ over the unit Euclidean ball. The quantities above can be written as the minimum and maximum of the so-called *Rayleigh quotient* $x^T A x / x^T x$.

Historically, David Hilbert coined the term “spectrum” for the set of eigenvalues of a symmetric operator (roughly, a matrix of infinite dimensions). The fact that for symmetric matrices, every eigenvalue lies in the interval $[\lambda_{\min}, \lambda_{\max}]$ somewhat justifies the terminology.

Example: [Largest singular value norm of a matrix](#).

Functions and Maps [Optimization Models](#) > [Gauss' Bet](#) | Functions and Maps | [Standard Forms](#) | [Nomenclature](#) | [Problem Classes](#) | [Complexity](#) | [History](#)

- Definitions: function, domain, map
- Graph and epigraph
- Level and sub-level sets

Definitions Functions

In this course we define *functions* as objects which take an argument in \mathbf{R}^n , and return a value in \mathbf{R} . We use the notation

$f : \mathbf{R}^n \rightarrow \mathbf{R}$,

to refer to a function with “input” space \mathbf{R}^n . The “output” space for functions is \mathbf{R} .

Example: The function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$ with values

$$f(x) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

gives the distance from the point (x_1, x_2) to a given point (y_1, y_2) .

Domain

We allow functions to take infinity values. The *domain* of a function f , denoted $\text{dom } f$, is defined as the set of points where the function is finite.

Example: Define the logarithm function as the function $f : \mathbf{R} \rightarrow \mathbf{R}$, with values $f(x) = \log x$ if $x > 0$, and $-\infty$ otherwise. The domain of the function is thus \mathbf{R}_{++} (the set of positive reals).

Two functions can differ not by their formal expression, but because they have different domains.

Example: The functions f, g defined as

$$f(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ +\infty & \text{otherwise,} \end{cases}, \quad g(x) = \begin{cases} 1/x & \text{if } x > 0 \\ +\infty & \text{otherwise,} \end{cases}$$

have the same formal expression inside their respective domains. However, they are not the same functions, since their domain is different. ♠

Maps

We reserve the term *map* to refer to vector-valued functions. That is, maps are functions which return more than a single value. We use the notation

$f : \mathbf{R}^n \rightarrow \mathbf{R}^m$,

to refer to a map with input space \mathbf{R}^n and output space \mathbf{R}^m . The *components* of the map f are the (scalar-valued) functions $f_i, i = 1, \dots, m$.

Example:[a map](#)

Graph and Epigraph

Consider a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$. We can define two sets relevant to f : the *graph* and the *epigraph*. Both are subsets of \mathbf{R}^{n+1} .

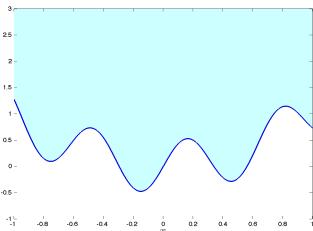
Graph

The *graph* of f is the set of input-output pairs that f can attain, that is:

$$G(f) := \{(x, f(x)) \in \mathbf{R}^{n+1} : x \in \mathbf{R}^n\} \cdot \text{Epigraph}$$

The *epigraph*, denoted $\text{epi } f$, describes the set of input-output pairs that f can achieve, as well as “anything above” (*epi* in Greek means “above”):

$$\text{epi } f := \{(x, t) \in \mathbf{R}^{n+1} : x \in \mathbf{R}^n, t \geq f(x)\}.$$



The function $f : \mathbf{R} \rightarrow \mathbf{R}$, with domain $(-1, 1)$ and value inside the domain $f(x) = x^2 + (1/2) \sin(10x)$. The graph corresponds to the points in blue, and the epigraph is in light blue. The epigraph extends to infinity above the graph. Points outside the domain are not shown.

Level and Sub-level Sets

Level and sub-level sets correspond to the notion of contour of a function. Both are indexed on some scalar value t .

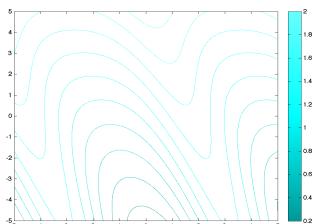
Level sets

A *level set* is simply the set of points that achieve exactly some value for the function f . For $t \in \mathbf{R}$, the t -level set of the function f is defined as

$$\mathbf{L}_t(f) := \{x \in \mathbf{R}^n : x \in \mathbf{R}^n, t = f(x)\} \cdot \text{Sub-level sets}$$

A related notion is that of *sub-level set*. This is now the set of points that achieve at most a certain value for f , or below:

$$\mathbf{S}_t(f) := \{x \in \mathbf{R}^n : x \in \mathbf{R}^n, t \geq f(x)\}.$$



Level and sub-level sets of a function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$, with domain \mathbf{R}^2 itself, and values on the domain given by

$$f(x) = \log \left(e^{\sin(x_1 + 0.3x_2 - 0.1)} + e^{2x_2 + 0.7} \right).$$

Sample and weighted mean, expected value

The *sample mean* (or, average) of given numbers x_1, \dots, x_n , is defined as

$$\hat{x} := \frac{1}{n}(x_1 + \dots + x_n).$$

The sample average can be interpreted as a scalar product:

$$\hat{x} = p^T x,$$

where $x = (x_1, \dots, x_n)$ is the vector containing the samples, and $p = (1/n)\mathbf{1}$, with $\mathbf{1}$ the vector of ones.

More generally, for any vector $p \in \mathbf{R}^n$, with $p_i \geq 0$ for every i , and $p_1 + \dots + p_n = 1$, we can define the corresponding *weighted average* as $p^T x$. The interpretation of p is in terms of a discrete probability distribution of a random variable X , which takes the value x_i with probability $p_i, i = 1, \dots, n$. The weighted average is then simply the *expected value* (or, mean) of X under the probability distribution p . The expected value is often denoted $\mathbf{E}_p(X)$, or $\mathbf{E}(X)$ if the distribution p is clear from context.

F

- [Feasible point, feasible set](#)
- [First-order approximation](#)
- [Frobenius norm of a matrix](#)
- [Function](#)
- [Fundamental theorem of linear algebra](#)

G

- [Geometric program](#) (GP)
- [Global optimum](#)
- [Gradient of a function](#)
- [Gradient methods](#), for solving convex programs
- [Gram matrix](#)
- [Gram-Schmidt \(GS\) procedure](#)
- [Graph of a function](#)

H

- [Half-space](#)
- [Hessian of a function](#)
- [Hyperplane](#)

I

- [Image compression](#)
- [Incidence matrix of a graph](#)
- [Independent set of vectors](#)
- [Interior-point methods](#), for solving convex programs
- [Inverse](#) of a matrix

K

- [Kernel matrix](#), kernel trick

L

- [Laplacian of a graph](#)
- [Lagragian](#) of an optimization problem
- [Laplace formula](#) for the determinant of a matrix
- [Largest singular value](#) (LSV) norm of a matrix
- [Least-squares](#)
- [Leibnitz formula](#) for the determinant of a matrix
- [Left inverse](#) of a matrix
- [Linear function](#)
- [Linear matrix inequality](#)
- [Linear program](#) (LP)
- [Linear regression](#)
- [Local optimum](#)
- [Log-return](#) of a financial asset
- [Log-sum-exp function](#)
- l_p [norms](#) of a vector: l_1 norm, l_2 -norm (also called Euclidean norm), l_∞ norm.
- l_0 [norm](#), or cardinality, of a vector.

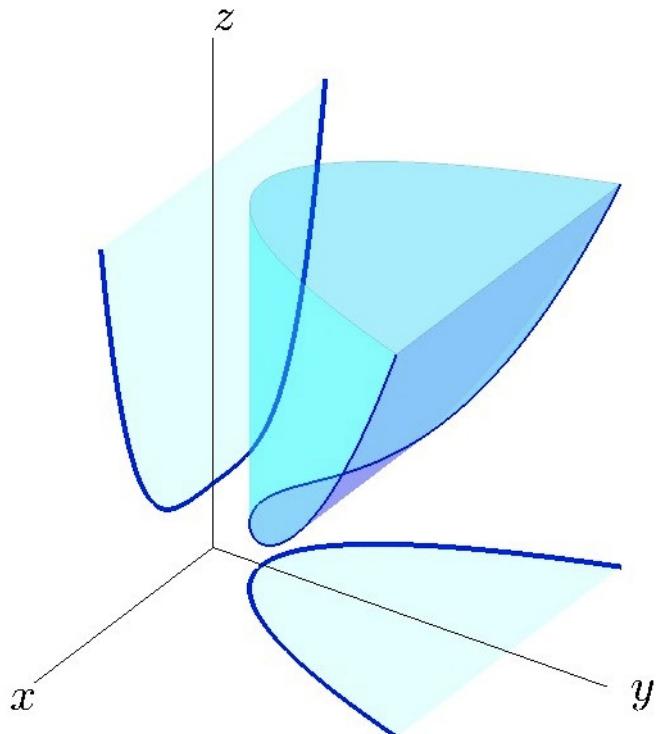
M

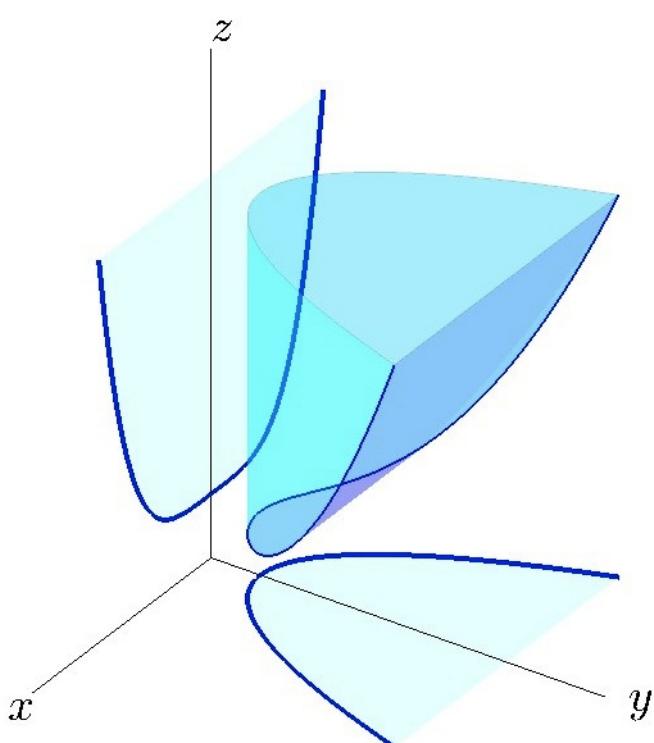
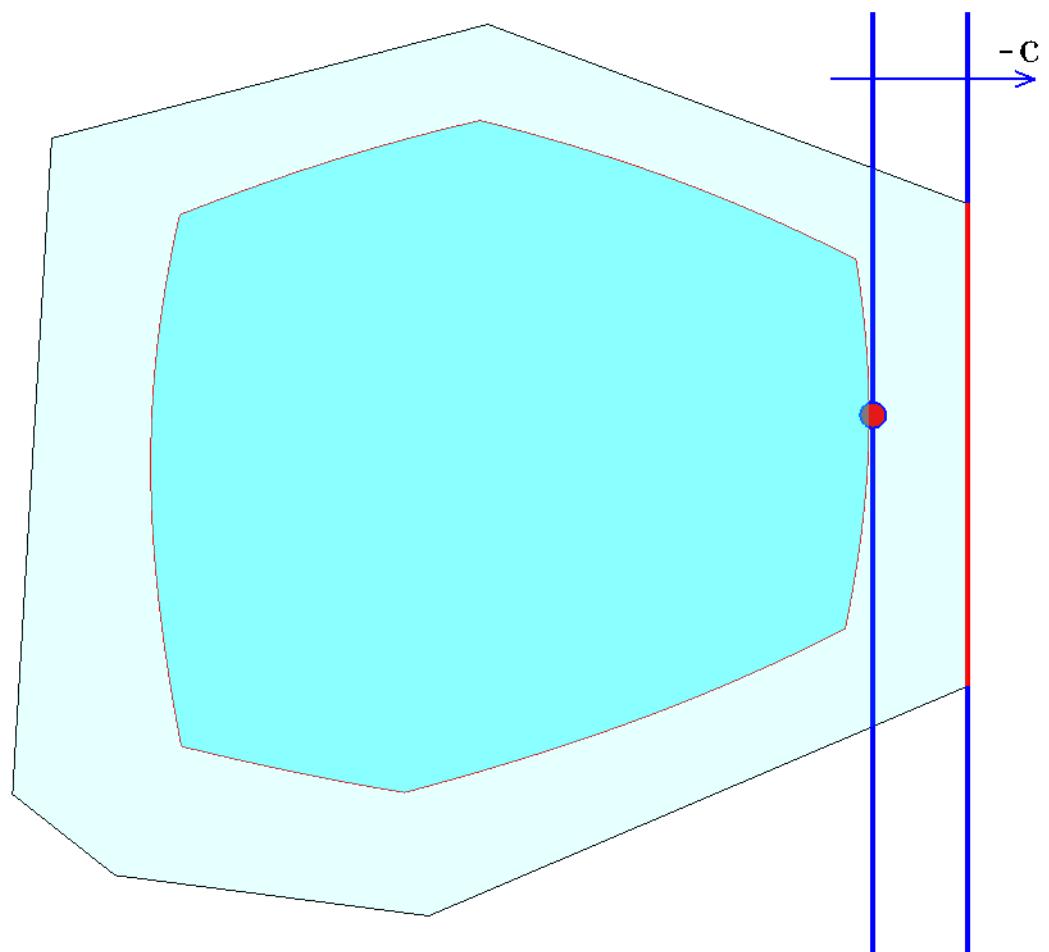
- [Maps](#)

- [Matrix](#)
- [Matrix-vector product](#)
- Sample [Mean](#)
- [Minimax inequality](#)

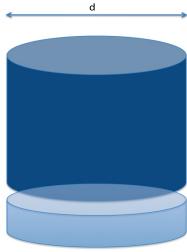
P

- [Permutation matrix](#)
- [Point-wise maximum](#) of functions
- [Polyhedron](#)
- [Polyhedral function](#)
- [Positive-definite](#)
- [Power laws](#)
- [Principal Component Analysis \(PCA\)](#)
- [Probability simplex](#)
- Projection: on a [line](#)
- [Pseudo-inverse of a matrix](#)





This is a simple example of a posynomial function, involving the construction and operating costs of a cylindrical liquid (say, oil, or water) storage tank.



A cylindrical water tank, with height h and diameter d . The tank includes a base, which is made of different material than the tank itself. In our model, the base's height does not depend on the tank's height. This is a reasonable approximation for heights not exceeding a certain value.

The parameters of the tank are its diameter d and height h . The costs to manufacture, and then operate during a given period of time (say, a year) the tank, include the following.

- *Filling costs* represent the costs associated with supplying a volume of water in the given time period. These costs depends on the ratio $V_{\text{supp}}/V_{\text{tank}}$, where V_{supp} is the volume to be supplied, and $V_{\text{tank}} = (\pi/4)hd^2$ is the volume of the tank (the smaller the volume of the tank is with respect to the volume to be supplied, the more often we have to refill the tank, and the larger the cost). Thus, the filling cost is inversely proportional to the tank's volume:

$$C_{\text{fill}}(d, h) = \alpha_1 \frac{V_{\text{supp}}}{V_{\text{tank}}} = c_1 h^{-1} d^{-2},$$

where α_1 is some positive constant, expressed in (say) dollars, and $c_1 := 4\alpha_1 V_{\text{supp}}/\pi$.

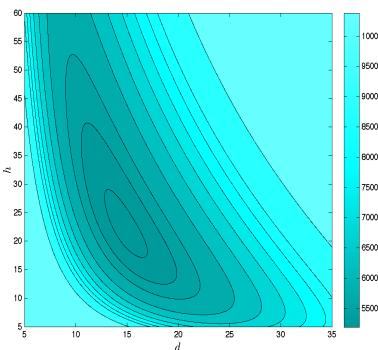
- *Construction costs* include the costs associated with building a base for the tank, and costs associated with building the tank itself. In our model, the first type of costs depends only on the base area $\pi d^2/4$, while the second depends on the surface of the tank, πdh . (This assumes that we can use the same base height for a variety of tank heights.) Thus the construction cost function can be written

$$C_{\text{constr}}(d, h) = C_{\text{base}}(d, h) + C_{\text{tank}}(d, h) = c_2 d^2 + c_3 dh,$$

where the constants $c_2 = \alpha_2 \pi/4$, $c_3 = \alpha_3 \pi$, with $\alpha_2 > 0, \alpha_3 > 0$ constants expressed in dollars per square meters.

The total manufacturing and operating cost function is thus the posynomial function

$$C_{\text{total}}(d, h) = C_{\text{fill}}(d, h) + C_{\text{constr}}(d, h) = c_1 h^{-1} d^{-2} + c_2 d^2 + c_3 dh.$$



Level sets of the cost function of the problem, corresponding to the values

$$V_{\text{supp}} = 800,000 L, \quad \alpha_1 = 10 \$, \\ \alpha_2 = 6 \$/m^2, \quad \alpha_3 = 2 \$/m^2.$$

The function clearly appears not to be convex, since some level sets are not.

See also: [Optimization of a water tank](#).

Signal to Interference plus Noise Ratio (SINR) in Wireless Communications

Consider a cellular wireless network with n transmitter/receiver pairs. Transmit powers are denoted as p_1, \dots, p_n . Transmitter i is supposed to transmit to receiver i , but due to interference, signal from the other transmitters is also present. In addition, there is (self-) noise power in each receiver. To measure this, we form the Signal to Interference plus Noise Ratio (SINR) at each receiver. This takes the form

$$\gamma_i := \frac{S_i}{I_i + \sigma_i}, \quad i = 1, \dots, n$$

where S_i is a measure of the (desired) signal power received from transmitter i , I_i is the total signal power received from all the other receivers, and $\sigma_i > 0$ is a measure of the receiver noise.

The SINR is a (in general, complicated) function of the power used at the transmitters.

A Linear Model

We can express the SINRs at the receivers in terms of the powers p_1, \dots, p_n more explicitly by assuming that the received powers S_i, I_i are linear functions of the transmit powers p_1, \dots, p_n .

The model (also known as the *Rayleigh fading* model) states that

$$S_i = G_{ii} p_i, \quad i = 1, \dots, n,$$

and

$$I_i = \sum_{j \neq i} G_{ij} p_j$$

where the coefficients G_{ij} , $1 \leq i, j \leq n$ are known as the *path gains* from transmitter j to receiver i .

A posynomial function

The SINR functions

$$\gamma_i(p) := \frac{S_i}{I_i + \sigma_i} = \frac{G_{ii} p_i}{\sigma_i + \sum_{j \neq i} G_{ij} p_j}, \quad i = 1, \dots, n,$$

are not posynomials, but their inverse

$$\frac{1}{\gamma_i(p)} = \frac{\sigma_i}{G_{ii}} p_i^{-1} + \sum_{j \neq i} \frac{G_{ij}}{G_{ii}} p_j p_i^{-1}, \quad i = 1, \dots, n,$$

are.

Why do we take the log in GP?

For a *posynomial*^f with values

$$f(x) = \sum_{k=1}^K c_k x^{a_k},$$

where $c > 0$, we have

$$\log f(x) = \log \left(\sum_{k=1}^K e^{a_k^T y + b_k} \right) = \text{lse}(Ay + b),$$

where $b_k := \log c_k$, $k = 1, \dots, K$, and A is the $K \times n$ matrix with rows a_1, \dots, a_K .

We may ask, why take the logarithm of f ? The function with values

$$g(y) = \sum_{k=1}^K e^{a_k^T y + b_k}$$

is convex already.

The answer is simply that the function g might take huge values, which might raise numerical problems. Taking its logarithm allows to recover smaller values.

Note that the logarithm is concave, while g is convex. *A priori*, $\log g$ is not convex. It turns out that “the exponential beats the logarithm”, in the sense that the latter functions’ concavity is not enough to destroy the convexity of the exponential.

Optimization of a Water Tank[GP > Posynomials | Standard Forms](#) > Example: Water Tank Optimization | [Applications](#)

This is a simple example of a geometric programming problem, involving the construction and operating costs of a cylindrical liquid (say, oil, or water) storage tank. The parameters are the diameter d of the tank, and its height, h .

As seen [here](#), the cost function of this problem is of the form

$$C_{\text{total}}(d, h) = C_{\text{fill}}(d, h) + C_{\text{constr}}(d, h) = c_1 h^{-1} d^{-2} + c_2 d^2 + c_3 d h,$$

where $c_1 := 4\alpha_1 V_{\text{supp}}/\pi$, $c_2 = \alpha_2 \pi/4$, $c_3 = \alpha_3 \pi$, and $\alpha_1, \alpha_2, \alpha_3$ positive constants.

The constraints involve upper and lower bounds on the variables:

$$0 < d \leq d_{\max}, \quad 0 < h \leq h_{\max}.$$

We might also have an upper bound κ_{\max} on the *aspect ratio* of the tank (the aspect ratio constraint is useful to take into account structural resistance to wind):

$$h \leq \alpha_{\max} d.$$

Our GP takes the standard form

$$\min_{d>0, h>0} c_1 h^{-1} d^{-2} + c_2 d^2 + c_3 d h \quad : \quad 0 < d_{\max}^{-1} d \leq 1, \quad 0 < h_{\max}^{-1} h \leq 1 \\ \kappa_{\max}^{-1} h d^{-1} \leq 1.$$

A CVX syntax for this problem is as follows.

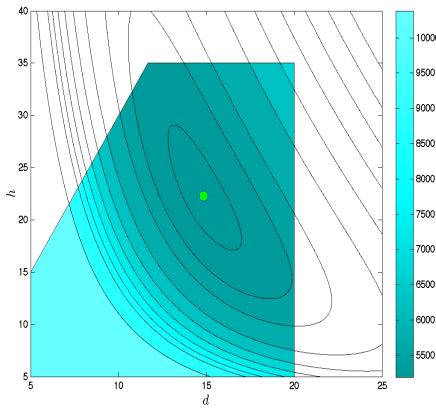
CVX syntax

```
cvx_begin gp
    variables h d
    % objective function is cost
    minimize( c1/(h*d^2) + c2*d^2 + c3*d*h )
    subject to
```

```

d <= dmax;
h <= hmax;
h/d <= alphamax;
cvx_end

```



The (non-convex) cost function of the problem, corresponding to the values

$$V_{\text{supp}} = 800,000L, \quad \alpha_1 = 10 \$, \\ \alpha_2 = 6 \$/m^2, \quad \alpha_3 = 2 \$/m^2, \\ d_{\max} = 20m, \quad h_{\max} = 30m, \\ \kappa_{\max} = 3.$$

The optimum variables are $d^* = 14.8430m, h^* = 22.2656m$ (optimal point shown in green on the plot). The optimal cost is $C^* = 5,1912\$$.

The design constraints of the problem are pictured as a forbidden region, in white. The problem is effectively unconstrained, as the bounds on variables and aspect ratio are not active at optimum.

See also:

- [Construction and operating costs for a water tank](#).
- [Convex form of the water tank optimization problem](#).

Optimization of a Water Tank: Convex Form of the GP [GP](#) > [Posynomials | Standard Forms](#) > Example: Water Tank Optimization | [Applications](#)

The GP described [here](#) for the optimization of the shape of a water tank is

$$\min_{d>0, h>0} C(d, h) := c_1 h^{-1} d^{-2} + c_2 d^2 + c_3 d h : \quad 0 < d_{\max}^{-1} d \leq 1, \quad 0 < h_{\max}^{-1} h \leq 1 \\ \kappa_{\max}^{-1} h d^{-1} \leq 1.$$

Let us define the new variables:

$$y_1 = \log d, \quad y_2 = \log h.$$

The objective function is transformed into

$$C(d, h) = \exp(-2y_1 - y_2 + \log c_1) + \exp(2y_1 + \log c_2) + \exp(y_1 + y_2 + \log c_3).$$

Taking logarithms, the new objective function is

$$f_0(y) := \log(\exp(-2y_1 - y_2 + \log c_1) + \exp(2y_1 + \log c_2) + \exp(y_1 + y_2 + \log c_3)) \\ = \text{lse}(-2y_1 - y_2 + \log c_1, 2y_1 + \log c_2, y_1 + y_2 + \log c_3),$$

where **lse** denotes the (convex) [log-sum-exp](#) function.

The constraints all involve monomials, so they will translate into linear or affine constraints. For example, the aspect ratio constraint on (d, h) :

$$\kappa_{\max}^{-1} h d^{-1} \leq 1$$

is equivalent to

$$\exp(-y_1 + y_2 - \log \kappa_{\max}) \leq 1,$$

which, after taking logarithms, becomes the affine constraint:

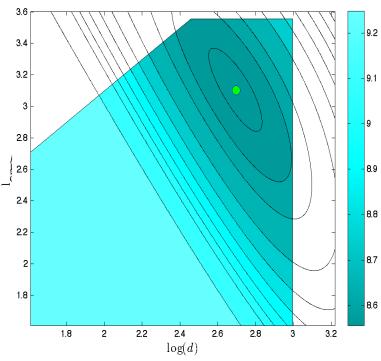
$$-y_1 + y_2 - \log \kappa_{\max} \leq 0.$$

Thus, the original GP takes the standard convex form:

$$\min_y \log(\exp(-2y_1 - y_2 + \log c_1) + \exp(2y_1 + \log c_2) + \exp(y_1 + y_2 + \log c_3))$$

$$\text{s.t. } y_1 \leq \log d_{\max}, \quad y_2 \leq \log h_{\max}, \quad -y_1 + y_2 - \log \kappa_{\max} \leq 0.$$

The water tank optimization problem in convex form, corresponding to the values



$$V_{\text{supp}} = 800,000L, \quad \alpha_1 = 10 \$, \quad \alpha_2 = 6 \$/m^2, \quad \alpha_3 = 2 \$/m^2, \\ d_{\max} = 20m, \quad h_{\max} = 30m, \quad \kappa_{\max} = 3.$$

The design constraints of the problem are pictured as a forbidden region, in white. We observe that the cost function in this formulation is convex, contrary to that of the original problem.

Mean and Covariance Matrix of a Random Variable

If X is a vector random variable, the *mean* of X is $\hat{X} := \mathbf{E}(X)$, where \mathbf{E} denotes the expectation operator with respect to the distribution that X follows. The *covariance matrix* of X is defined as

$$\Sigma := \mathbf{E}(X - \mathbf{E}(X))(X - \mathbf{E}(X))^T.$$

Note that the covariance matrix is always a positive semi-definite matrix.

Assume that the distribution is discrete, with p_i the probability that the random variable X takes a certain value $x_i \in \mathbf{R}^n$, $i = 1, \dots, N$. Then the expected value of X is

$$\hat{X} := \mathbf{E}(X) = \frac{1}{N} \sum_{i=1}^N x_i,$$

and the covariance matrix is

$$\Sigma = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{X})(x_i - \hat{X})^T.$$

Example: if X takes the two values x_1, x_2 , with

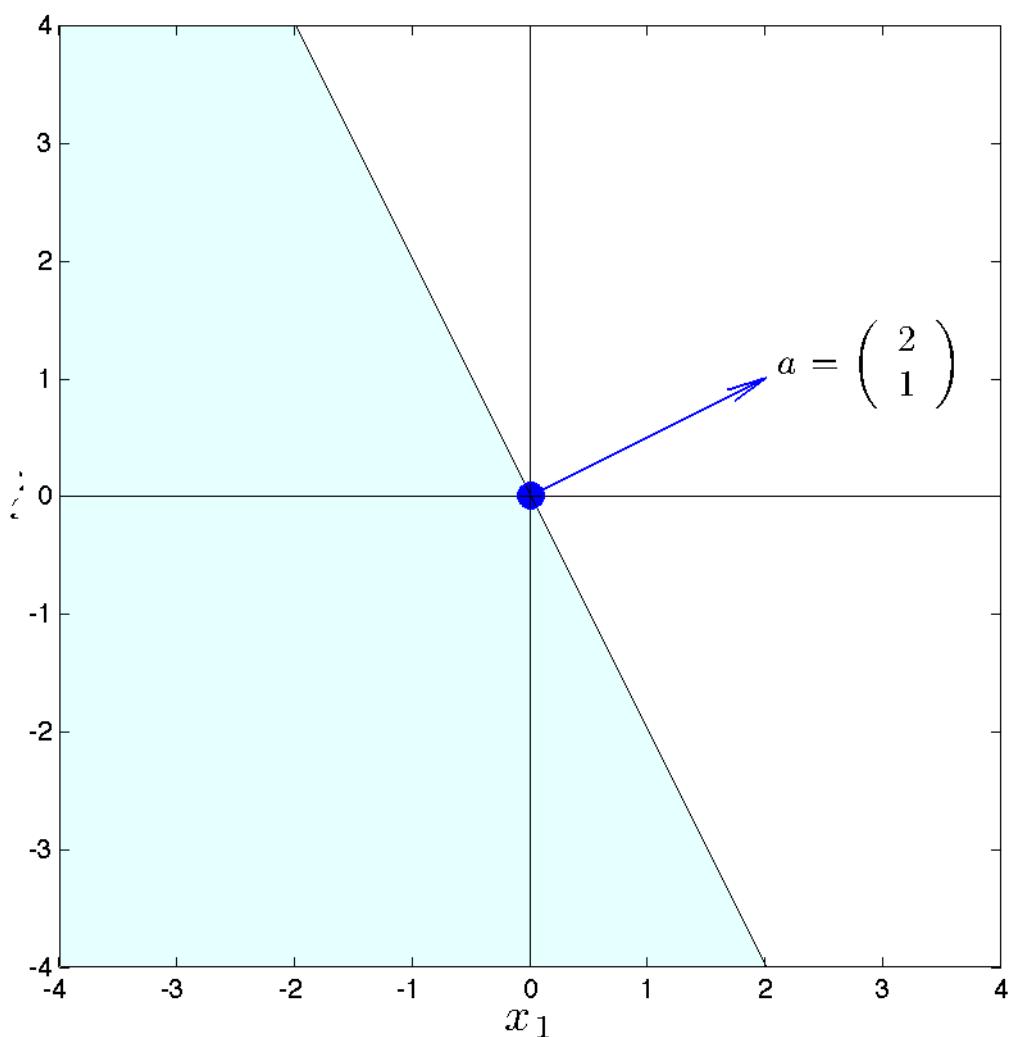
$$x_1 = \begin{pmatrix} 1 \\ -2 \end{pmatrix}, \quad x_2 = \begin{pmatrix} -1 \\ 3 \end{pmatrix},$$

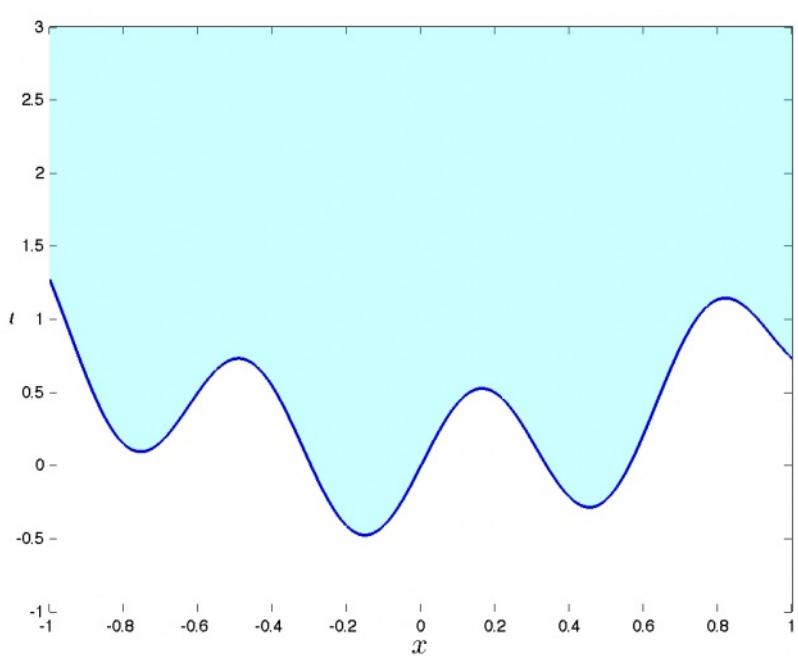
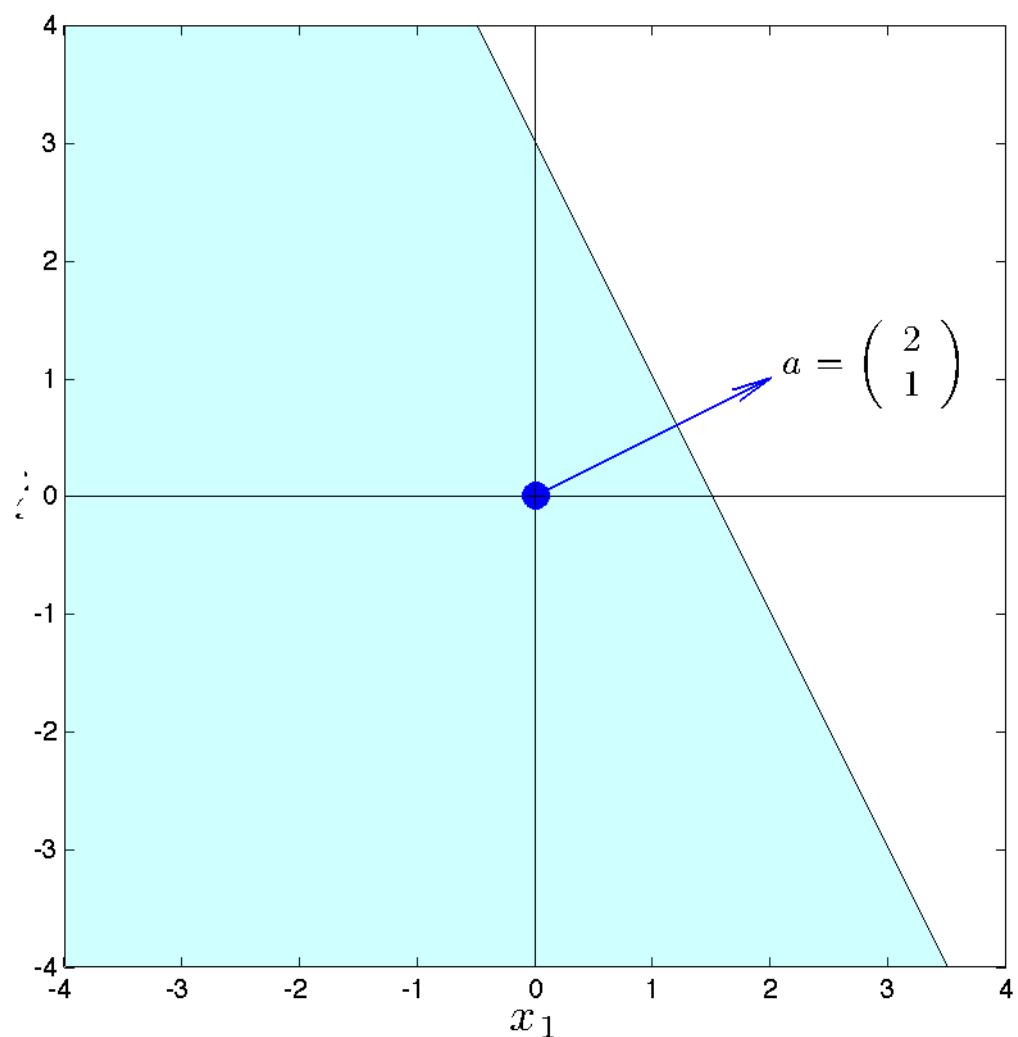
with the probability of x_1 (resp. x_2) being $p_1 = 0.3$ (resp. $p_2 = 0.7$), then the mean of X is

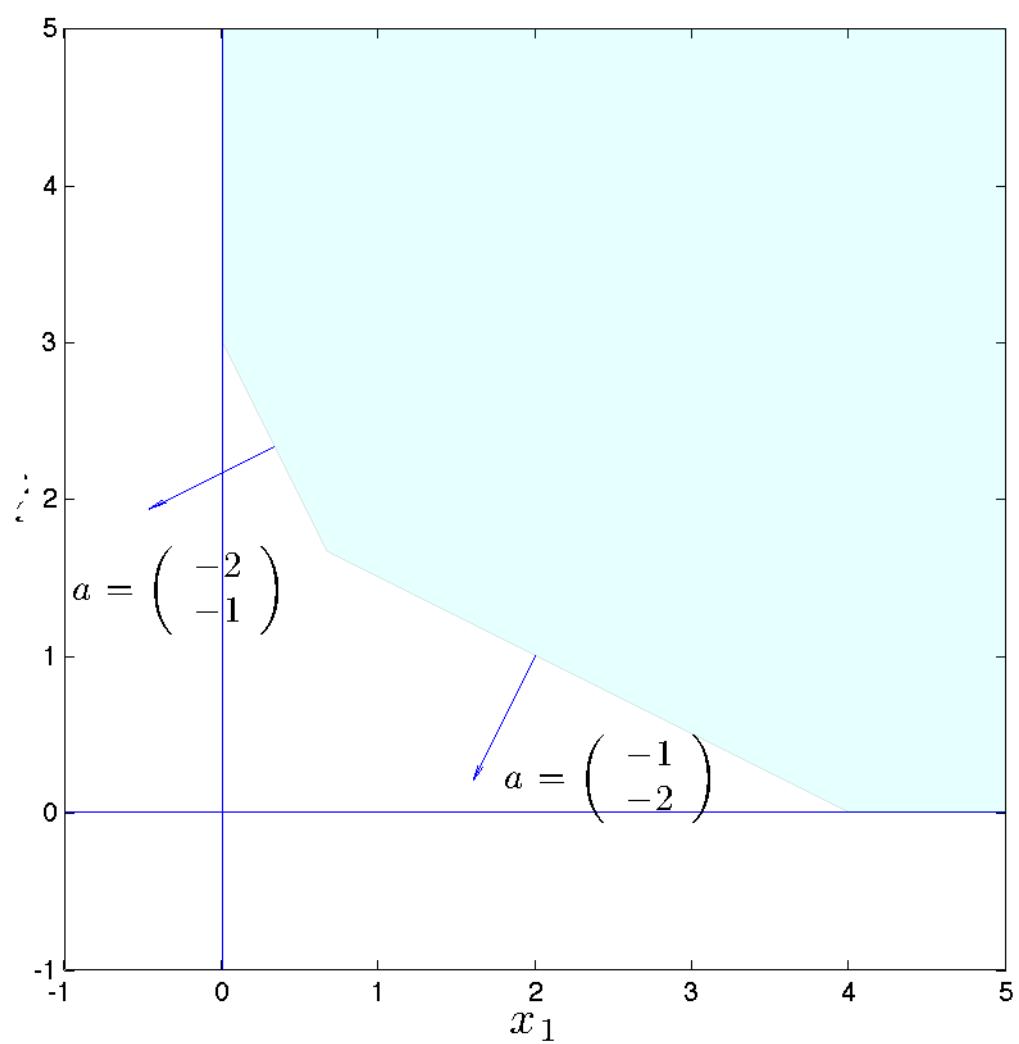
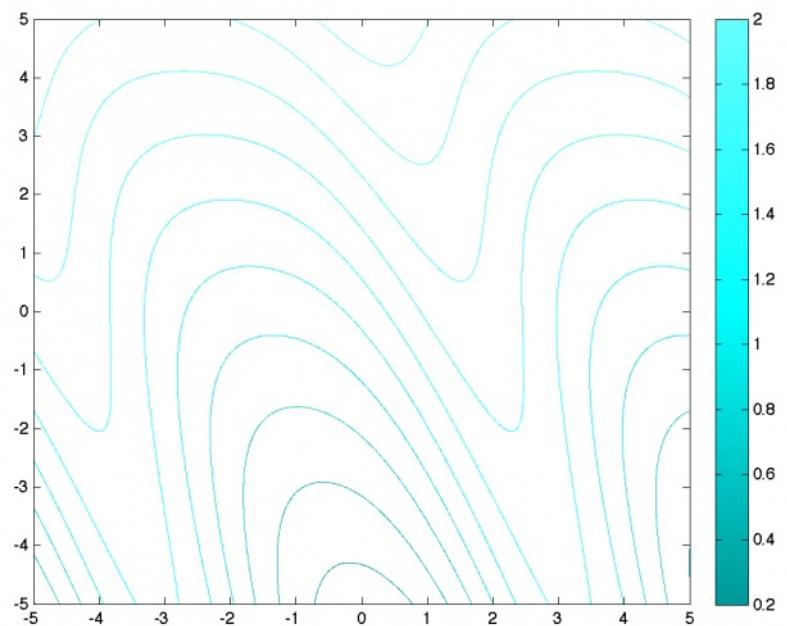
$$\hat{X} = \mathbf{E}(X) = p_1 x_1 + p_2 x_2 = 0.3 \begin{pmatrix} 1 \\ -2 \end{pmatrix} + 0.7 \begin{pmatrix} -1 \\ 3 \end{pmatrix} = \begin{pmatrix} -0.4 \\ 1.5 \end{pmatrix},$$

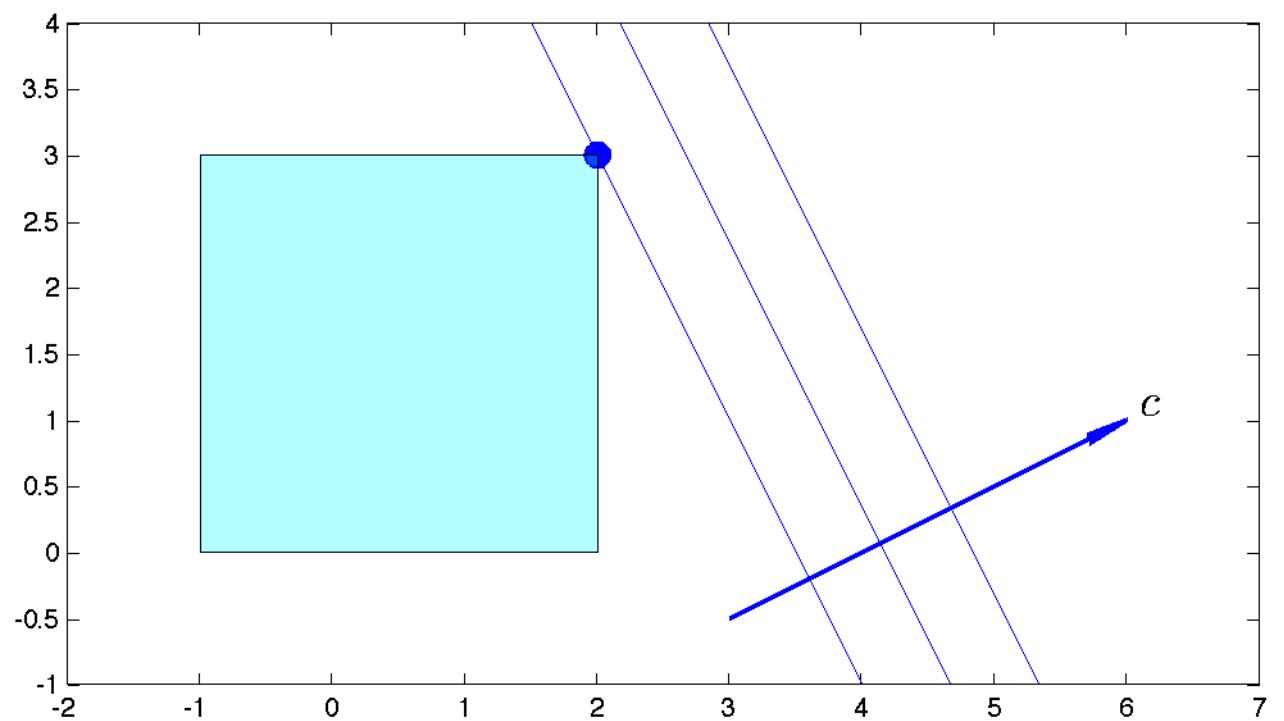
and its covariance matrix is

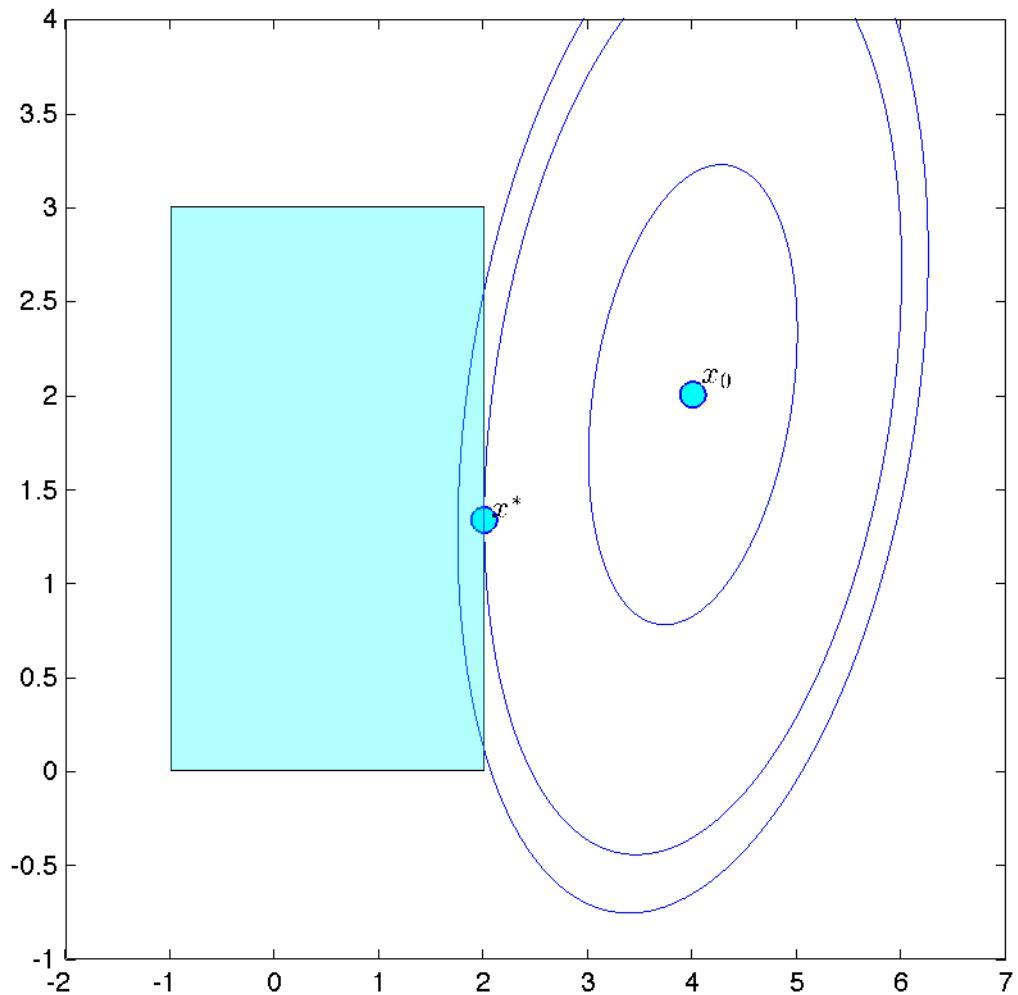
$$\Sigma = p_1(x_1 - \hat{X})(x_1 - \hat{X})^T + p_2(x_2 - \hat{X})(x_2 - \hat{X})^T = \begin{pmatrix} 0.84 & -2.1 \\ -2.1 & 5.25 \end{pmatrix}.$$











Cardinality minimization: the l_1 -norm trick [LP and QP](#) > [Half-Spaces](#) | [Polyhedra](#) | [Polyhedral Functions](#) | [Applications](#) > Cardinality Minimization | [Next](#)

- Cardinality minimization problem
- The l_1 -norm heuristic
- Application: piece-wise constant fitting

Cardinality Minimization

Many problems in engineering can be cast as

$$\min_x \mathbf{Card}(x) : x \in \mathbf{P},$$

where \mathbf{P} is a polytope (or, more generally, a convex set), and $\mathbf{card}(x)$ denotes the [cardinality](#) (number of non-zero elements) of the vector x . Such problems seek a “sparse” solution, one with many zeroes in it.

A related problem is a penalized version of the above, where we seek to trade-off an objective function against cardinality:

$$\min_x f(x) + \lambda \mathbf{Card}(x) : x \in \mathbf{P},$$

where f is some “cost” function, and $\lambda > 0$ is a penalty parameter.

Cardinality minimization is a hard problem in general, but it appears in many areas.

The l_1 -norm heuristic

The l_1 -norm heuristic consists in replacing the (non-convex) cardinality function $\mathbf{Card}(x)$ with a polyhedral (hence, convex) one, involving the l_1 -norm. This heuristic leads to replace the problem at the top with

$$\min_x \|x\|_1 : x \in \mathbf{P},$$

which is an LP (provided \mathbf{P} is a polyhedron).

If \mathbf{P} is described via affine inequalities, as $\mathbf{P} = \{x : Ax \leq b\}$, with A a matrix and b a vector existing in the matlab workspace, then the following CVX snippet solves

the above problem.

CVX implementation

```
cvx_begin
    variable x(n,1)
    minimize( norm(x,1) )
    subject to
        Ax <= b;
cvx_end
```

Applications Piece-wise Constant Fitting

We observe a noisy time-series (encoded in a vector x) which is almost piece-wise constant. The goal in piece-wise constant fitting is to find what the constant levels are, as they have a biological interpretation.

Unfortunately, the signal is not exactly piece-wise constant, but a noisy version of such a signal. Thus, we seek to obtain an estimate of the signal, say \hat{x} , such that \hat{x} has as few changes in it as possible. We model the latter by minimizing the cardinality of the “difference” vector Dx , where D is the *difference* matrix

$$D = \begin{bmatrix} -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & \dots & 0 \\ & & \ddots & & \\ \dots & & 0 & -1 & 1 \end{bmatrix}.$$

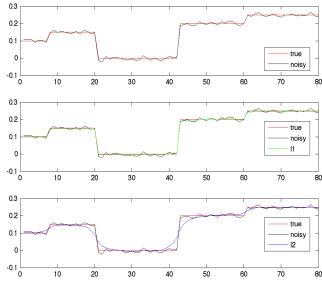
We are led to the problem

$$\min_{\hat{x}} \|x - \hat{x}\|_2^2 : \text{Card}(Dx) \leq K,$$

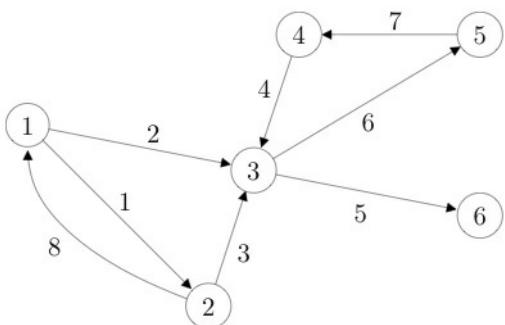
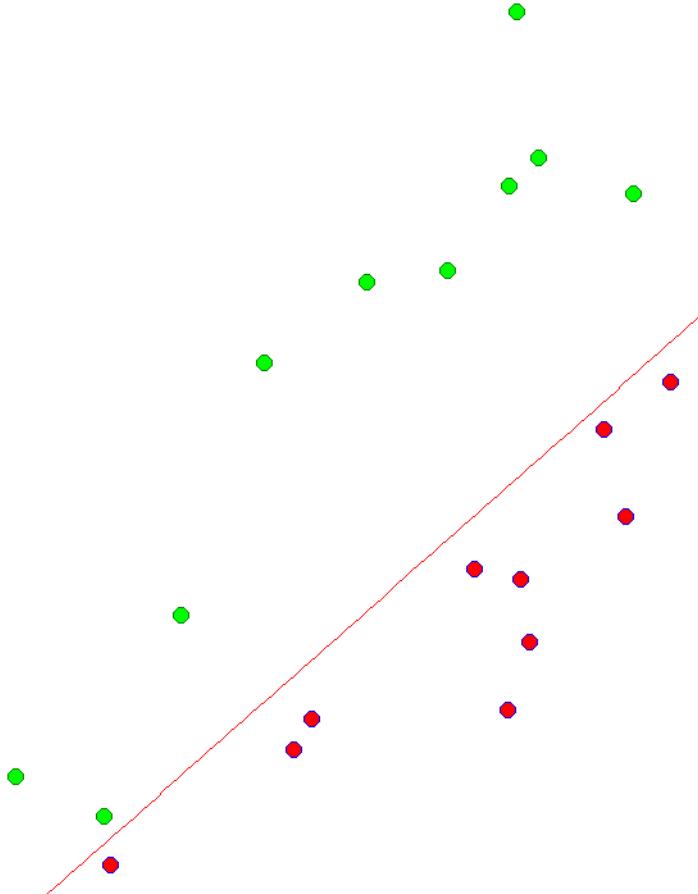
where K is an estimate on the number of jumps in the signal. Here, objective function in the problem is a measure of the error between the noisy measurement and the estimate \hat{x} .

The l_1 -norm heuristic consists in replacing the top (hard) problem by

$$\min_{\hat{x}} \|x - \hat{x}\|_2^2 : \|Dx\|_1 \leq K,$$



Piece-wise constant fitting. Top panel: the true signal and its noisy version. Middle: l_1 -constrained fitting, showing a good fit with the true (unknown) signal. Bottom: l_2 -constrained fitting.



Sum of largest components in a vector

- Definition
- Convexity
- Efficient representation of the epigraph
- Proof

Definition

For $x \in \mathbf{R}^n$, the sum of the k -largest components in x is written as

$$s_k(x) := \sum_{i=1}^k x_{[i]},$$

where $x[i]$ is the i -th largest component of x .

Convexity

The function $s_k(x)$ is convex, since it is the [point-wise maximum](#) of linear (hence, convex) functions:

$$s_k(x) = \max_{(i_1, \dots, i_k) \in \{1, \dots, n\}^k} x_{i_1} + \dots + x_{i_k}.$$

Since the functions inside the maximum are all linear, the function s_k is actually [polyhedral](#).

The convex constraint $s_k(x) \leq \alpha$ can be implemented in CVX by invoking `sum_largest`. (See [here](#) for an example.)

Efficient representation of the epigraph

The above representation is a very cumbersome representation, and is not used by CVX internally. Indeed, there are $C_{n,k}$ (n choose k) linear functions involved in the maximum, all obtained by choosing a particular subset of k indices in $\{1, \dots, n\}$. For example, with $k = 2$ and $n = 4$:

$$s_2(x) = \max(x_1 + x_2, x_2 + x_3, x_3 + x_1, x_1 + x_4, x_2 + x_4, x_3 + x_4).$$

Hence, to represent the constraint $s_k(x) \leq \alpha$ based on the above, we'd have to list 6 constraints:

$$x_1 + x_2 \leq \alpha, \quad x_2 + x_3 \leq \alpha, \quad x_3 + x_1 \leq \alpha, \quad x_1 + x_4 \leq \alpha, \quad x_2 + x_4 \leq \alpha, \quad x_3 + x_4 \leq \alpha.$$

The list of constraints needed grows very quickly with n, k : for $n = 100, k = 10$, we'd have to list more than 10^{13} linear constraints!

A more efficient representation is based in the following expression, proven below:

$$s_k(x) = \min_t kt + \sum_{i=1}^n \max(0, x_i - t).$$

In this form, a constraint of the form $s_k(x) \leq \alpha$ can be expressed as: there exist a scalar t and a n -vector u such that

$$kt + \sum_{i=1}^n u_i \leq \alpha, \quad u \geq 0, \quad u_i \geq x_i - t, \quad i = 1, \dots, n.$$

The above proves that s_k is convex, since the set of points (x, α) such that the above holds for some u, t is a polyhedron (hence the function has a convex epigraph). The above representation is much more efficient, as it involves a polyhedron with $2n + 1$ constraints. The price to pay is a moderate increase in the number of *variables*, which is now $2n + 1$ instead of n .

The lesson here is that a polyhedron in a n -dimensional space, with an exponential number of facets, can be represented as a polyhedron in a higher dimensional space with a moderate number of facets. By adding just a few dimensions to the problem we are able to deal (implicitly) with a very high number of constraints.

Proof

A proof based on the powerful concept of [duality](#) is [here](#).

We can assume without loss of generality that the elements of x are ordered in decreasing fashion: x_1, \dots, x_n . Then $s_k(x) = x_1 + \dots + x_k$. Now choose t such that $x_k \geq t \geq x_{k+1}$. We have

$$kt + \sum_{i=1}^n \max(0, x_i - t) = kt + \sum_{i=1}^k (x_i - t) = \sum_{i=1}^k x_i = s_k(x).$$

Since $s_k(x)$ is attained for a *particular* choice of t , we obtain that $s_k(x)$ is bounded below by the minimum over t :

$$s_k(x) \geq \min_t kt + \sum_{i=1}^n \max(0, x_i - t).$$

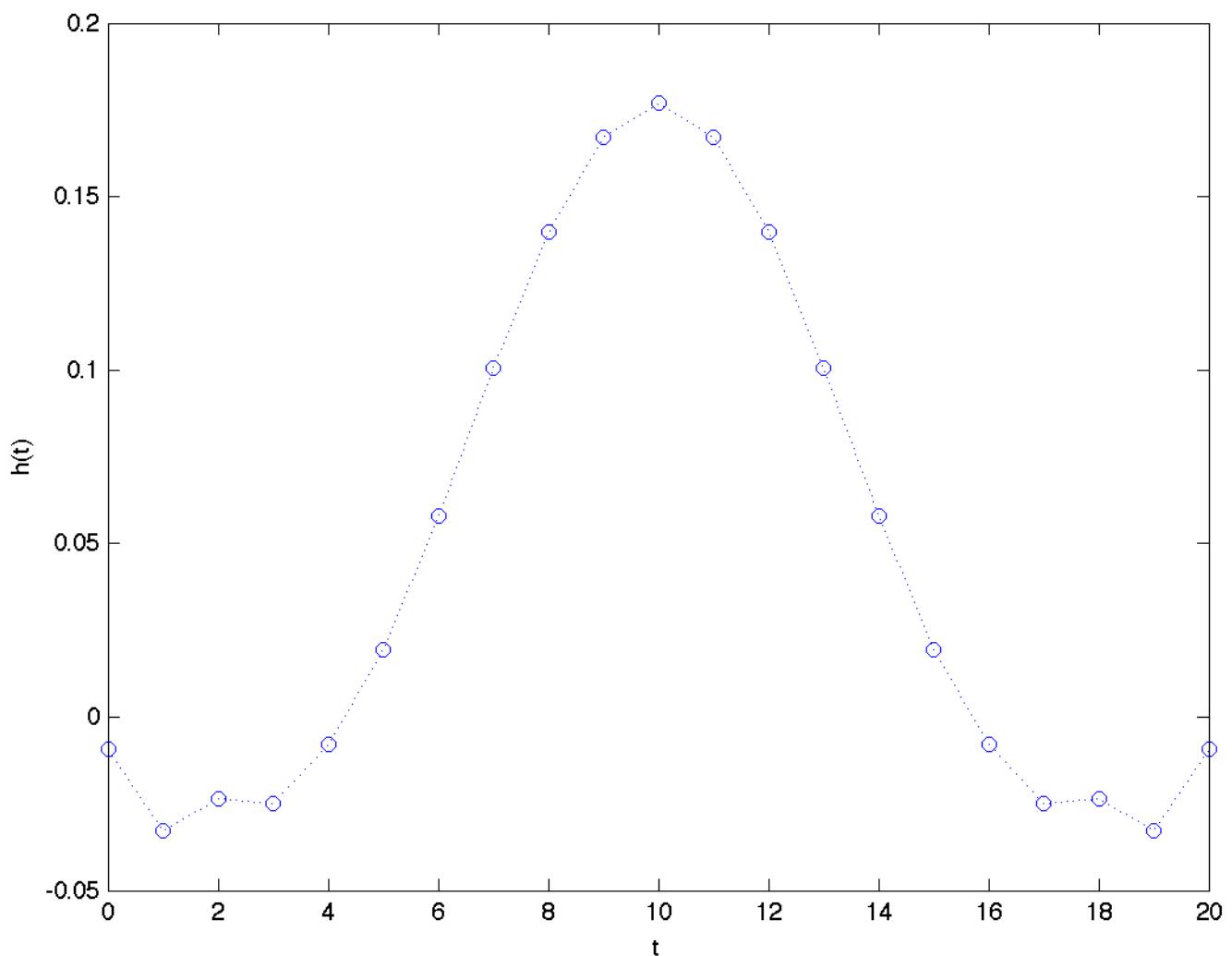
On the other hand, for every t , we have

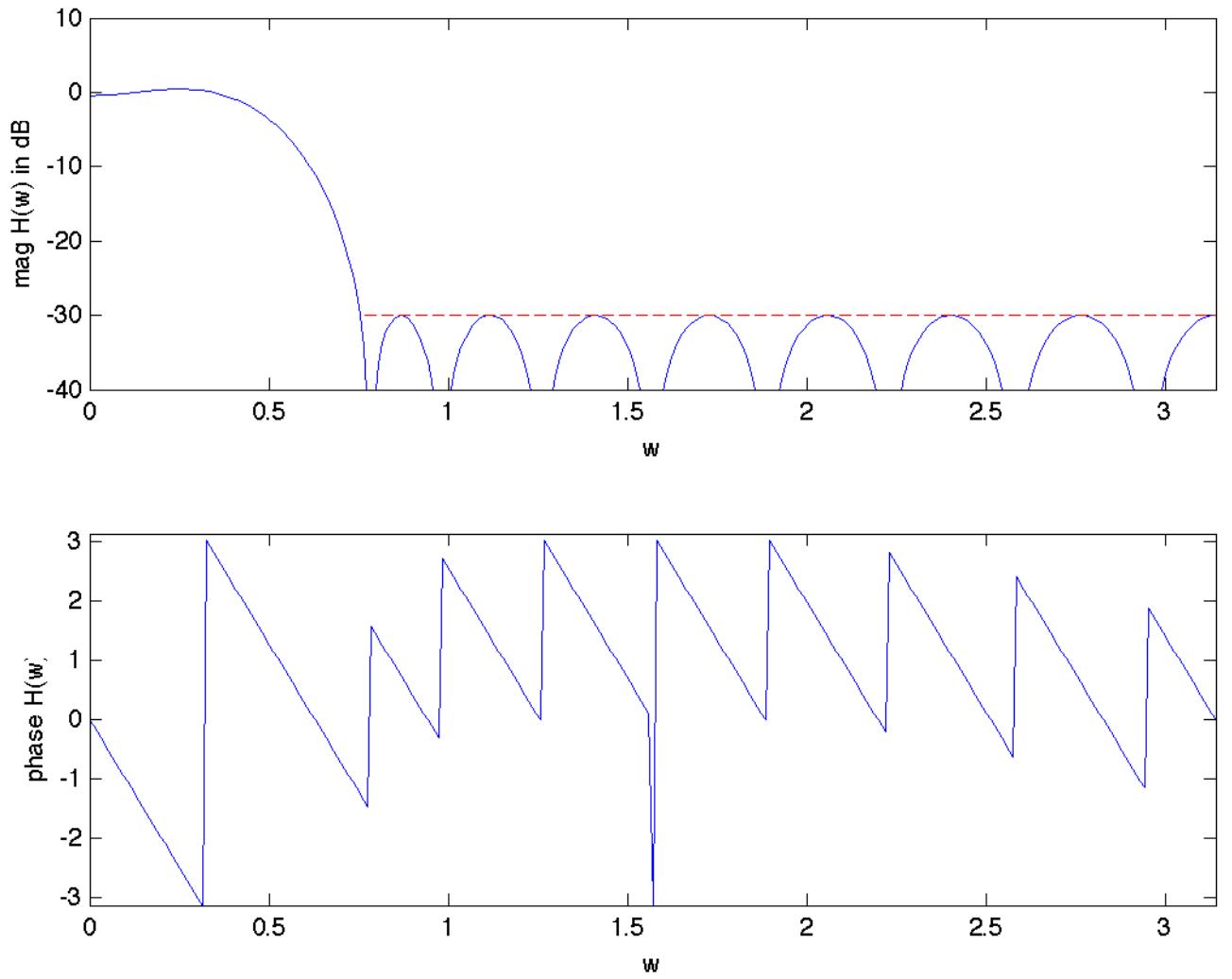
$$\begin{aligned} s_k(x) &= \sum_{i=1}^k (x_i - t + t) = kt + \sum_{i=1}^k (x_i - t) \\ &\leq kt + \sum_{i=1}^k \max(0, x_i - t) \leq kt + \sum_{i=1}^n \max(0, x_i - t). \end{aligned}$$

Since the upper bound above is valid for *every* t , it remains valid when minimizing over t , and we have

$$s_k(x) \leq \min_t kt + \sum_{i=1}^n \max(0, x_i - t).$$

This concludes the proof.





Surface Area

Consider a surface in \mathbf{R}^3 that is described by a function from the square $C := [0, 1] \times [0, 1]$ to \mathbf{R} . The corresponding *surface area* is

$$A(f) := \int_C \sqrt{1 + \|\nabla f(x, y)\|_2^2} dx dy.$$

We can obtain this formula using a grid of the square C , with grid points (ih, jh) , $0 \leq i, j \leq K$, where $h = 1/K$ is a small increment. With this discretization, to a point on the grid, we associate a rectangle on the plane $z = 0$. The area of the rectangle is h^2 , with $h = 1/K$ the spacing of the grid. However, the area of the corresponding surface is not the same, as it is not a rectangle but a parallelogram.

This parallelogram is defined by its two adjacent vectors

$$u = h \begin{pmatrix} 1 \\ 0 \\ \frac{\partial f}{\partial x}(ih, jh) \end{pmatrix}, \quad v = h \begin{pmatrix} 0 \\ 1 \\ \frac{\partial f}{\partial y}(ih, jh) \end{pmatrix}.$$

The formula for the area of a parallelogram defined by two adjacent vectors $u, v \in \mathbf{R}^3$ is the magnitude of the cross-product between the two vectors:

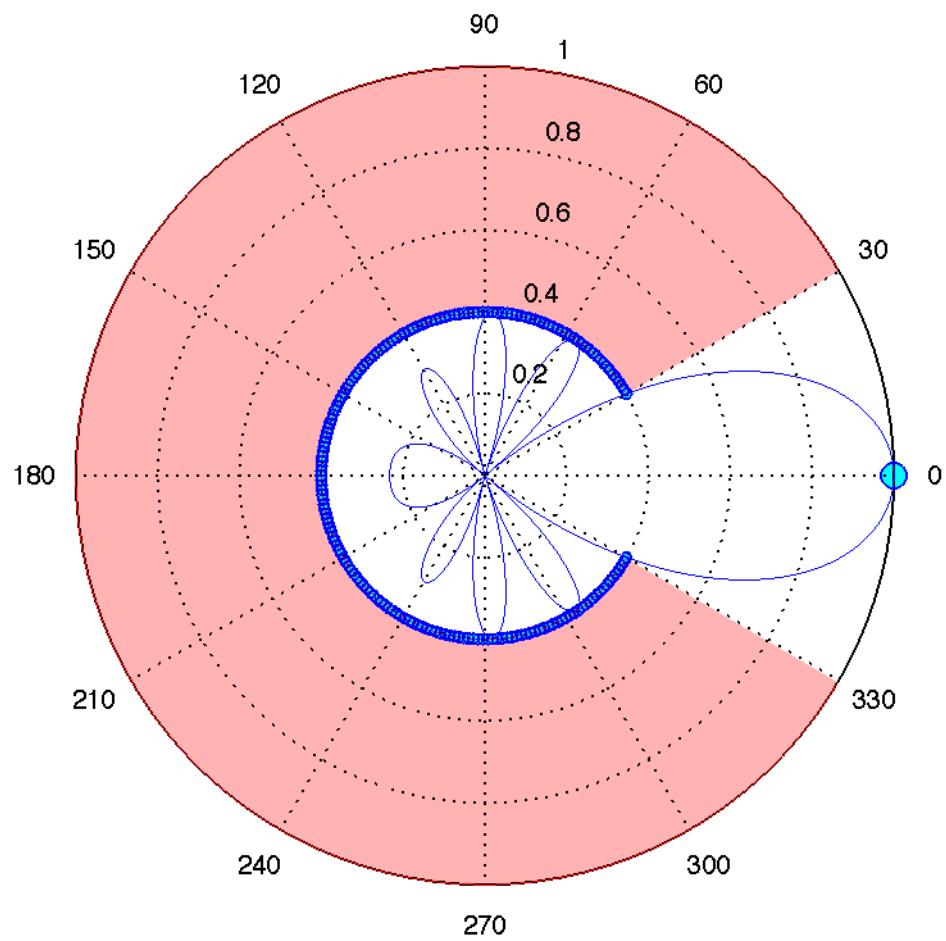
$$|u \times v| = \sqrt{\|u\|_2^2 \cdot \|v\|_2^2 - (u^T v)^2} = \|u\|_2 \|v\|_2 \sin(\theta),$$

where θ is the angle between the two vectors.

Using the expressions for u, v given above, after some algebra we obtain the formula for the area of the parallelogram:

$$h^2 \sqrt{1 + \frac{\partial f}{\partial x}(ih, jh)^2 + \frac{\partial f}{\partial y}(ih, jh)^2},$$

which proves the formula above.



J

- [Jacobian matrix](#) of a non-linear map

Q

- [Quadratic program](#) (QP)
- [Quadratic form, quadratic function](#)

R

- [Range of a matrix](#)
- [Rank of a matrix](#)
- [Rate of return](#) of a financial asset
- [Rayleigh quotient](#)
- Regression: [Linear](#)
- [Right inverse](#) of a matrix
- [Robust linear program](#)

S

- [Sample mean, sample standard deviation, sample covariance matrix](#)

- Scalar product: for [vectors](#), for [matrices](#)
- [Second-order approximation of a function](#)
- [Second-order cone](#)
- [Second-order cone program](#) (SOCP)
- [Semidefinite program](#) (SDP)
- [Singular value](#) of a general matrix
- [Singular value decomposition](#) (SVD)
- [Slater's condition for strong duality](#)
- [Spectral theorem](#), or symmetric eigenvalue decomposition (SED) theorem
- [Symmetric matrix](#)

U

- [Unconstrained optimization](#)
- [Unitary](#) matrix (see also [Orthogonal](#) matrix)

V

- [Vectors](#)
- Sample [Variance](#)

W

X

Y

Z

O

- [Optimal point, optimal value, optimal set](#)
- Orthogonal: [vectors](#), [matrices](#)

Special Matrices [Matrices](#) > [Basics](#) | [Matrix products](#) | Special matrices | [QR](#) | [Matrix inverses](#) | [Linear maps](#) | [Matrix norms](#) | [Applications](#)

- Square Matrices
 - Identity and diagonal matrices
 - Triangular matrices
 - Symmetric matrices
 - Orthogonal Matrices
- Dyads

Some special square matrices

Square matrices are matrices that have the same number of rows as columns. The following are important instances of square matrices.

Identity matrix

The $n \times n$ identity matrix (often denoted I_n , or simply I , if context allows), has ones on its diagonal and zeros elsewhere. It is square, diagonal and symmetric. This matrix satisfies $A \cdot I_n = A$ for every matrix A with n columns, and $I_n \cdot B = B$ for every matrix B with n rows.

Matlab syntax

```
>> I3 = eye(3); % the 3x3 identity matrix
>> A = eye(3,4); % a 3x4 matrix having the 3x3 identity in its first 3 columns
```

Diagonal matrices

Diagonal matrices are square matrices A with $A_{ij} = 0$ when $i \neq j$. A diagonal $n \times n$ matrix A can be denoted as $A = \text{diag}(a)$, with $a \in \mathbf{R}^n$ the vector containing the elements on the diagonal. We can also write

$$A = \begin{pmatrix} a_1 & & \\ & \ddots & \\ & & a_r \end{pmatrix},$$

where by convention the zeros outside the diagonal are not written.

Matlab syntax

```
>> A = diag([1 2 3]); % a diagonal matrix with 1,2,3 on the diagonal
>> A = spdiags([1 2 3]',0,3,3); % the same matrix declared as a sparse matrix
```

Symmetric matrices

Symmetric matrices are square matrices that satisfy $A_{ij} = A_{ji}$ for every pair (i, j) . An entire [section](#) is devoted to symmetric matrices.

Triangular matrices

A square matrix $A \in \mathbf{R}^{m \times n}$ is *upper triangular* if $A_{ij} = 0$ when $i < j$. Here are a few examples:

$$A_1 = \begin{pmatrix} 1 & -1 \\ 0 & 2 \\ 0 & 0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 3 & 8 & 3 \\ 0 & 6 & -1 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 0 & 8 & 3 \\ 0 & 0 & -1 \end{pmatrix}.$$

A matrix is lower triangular if its transpose is upper triangular. For example:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 8 & -9 & 0 \\ 1 & -2 & 3 \end{pmatrix}.$$

Orthogonal (or, *unitary*) matrices are square matrices, such that the columns form an orthonormal basis. If $U = [u_1, \dots, u_n]$ is an orthogonal matrix, then

$$u_i^T u_j = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $U^T U = I_n$. Similarly, $U U^T = I_n$.

Orthogonal matrices correspond to rotations or reflections across a direction: they preserve length and angles. Indeed, for every vector x ,

$$\|Ux\|_2^2 = (Ux)^T (Ux) = x^T U^T U x = x^T x = \|x\|_2^2.$$

Thus, the underlying linear map $x \rightarrow Ux$ preserves the length (measured in Euclidean norm). This is sometimes referred to as the *rotational invariance* of the Euclidean norm.

In addition, angles are preserved: if x, y are two vectors with unit norm, then the angle θ between them satisfies $\cos \theta = x^T y$, while the angle θ' between the rotated vectors $x' = Ux, y' = Uy$ satisfies $\cos \theta' = (x')^T y'$. Since

$$(Ux)^T (Uy) = x^T U^T U y = x^T y,$$

we obtain that the angles are the same. (The converse is true: any square matrix that preserves lengths and angles is orthogonal.)

Geometrically, orthogonal matrices correspond to rotations (around a point) or reflections (around a line passing through the origin).

Examples:

- [A \$2 \times 2\$ orthogonal matrix](#).
- [Permutation matrices](#).

Dyads

Dyads are a special class of matrices, also called rank-one matrices, for reasons seen later.

Definition

A matrix $A \in \mathbf{R}^{m \times n}$ is a *dyad* if it is of the form $A = uv^T$ for some vectors $u \in \mathbf{R}^m, v \in \mathbf{R}^n$. The dyad acts on an input vector $x \in \mathbf{R}^n$ as follows:

$$Ax = (uv^T)x = (v^T x)u.$$

In terms of the associated linear map, for a dyad, the output always points in the *same* direction u in output space (\mathbf{R}^m), no matter what the input x is. The output is thus always a simple scaled version of u . The amount of scaling depends on the vector v , via the linear function $x \rightarrow v^T x$.

Example:[Single-factor models of financial data](#)

Normalized dyads

We can always *normalize* the dyad, by assuming that both u, v are of unit (Euclidean) norm, and using a factor to capture their scale. That is, any dyad can be written in *normalized* form:

$$A = uv^T = (\|u\|_2 \cdot \|v\|_2) \cdot \left(\frac{u}{\|u\|_2}\right) \left(\frac{v}{\|v\|_2}\right)^T = \sigma \tilde{u} \tilde{v}^T,$$

where $\sigma > 0$, and $\|\tilde{u}\|_2 = \|\tilde{v}\|_2 = 1$.

Basics [Vectors](#) > Basics | [Scalar product, Norms](#) | [Projection on a line](#) | [Orthogonalization](#) | [Hyperplanes](#) | [Linear functions](#) | [Application](#)

- Definitions
- Independence
- Subspaces, span, affine sets
- Basis, dimension

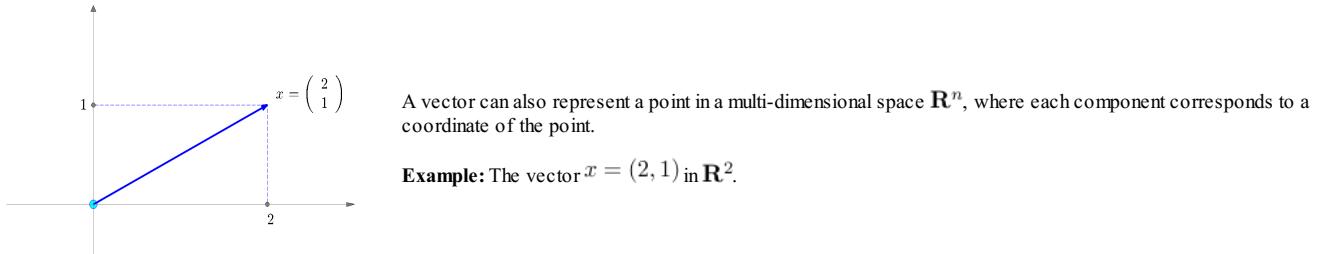
Definitions Vectors

Assume we are given a collection of n real numbers, x_1, \dots, x_n . We can represent them as n locations on a line. Alternatively, we can represent the collection as a single point in a n -dimensional space. This is the *vector* representation of the collection of numbers; each number x_i is called a *component* or *element* of the vector.

Vectors can be arranged in a column, or a row; we usually write vectors in column format:

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

We denote by \mathbf{R}^n denotes the set of real vectors with n components. If $x \in \mathbf{R}^n$ denotes a vector, we use subscripts to denote components, so that x_i is the i -th component of x . Sometimes the notation $x(i)$ is used to denote the i -th component.



Examples:

- [Temperatures at different airports.](#)
- [Bag-of-words representation of text.](#)

Transpose

If x is a column vector, x^T denotes the corresponding row vector, and vice-versa. Hence, if x is the column vector above:

$$x^T = (x_1 \dots x_n).$$

Sometimes we use the looser, in-line notation $x = (x_1, \dots, x_n)$, to denote a row or column vector, the orientation being understood from context.

Matlab syntax

A column vector $x = (2, 3.1, -4)$ and its transpose y can be declared in Matlab's workspace as follows. Here, no room for loose notation: we use a semicolon to separate the components of a column vector, while we use commas for row vectors.

Matlab syntax: declare and transpose a vector

```
>> x = [2; 3.1; -4]; % declare a column vector using ";".
>> y = x'; % the prime operator ' transposes the vector.
>> y = [2,3.1,-4]; % can also declare a row vector with commas.
>> x(2) % this produces the second component of x.
>> x([1,3]) % this produces the 2-vector with the first and the third component of x.
```

Independence

A set of vectors $\{x_1, \dots, x_n\}$ in \mathbf{R}^n , $i = 1, \dots, m$ is said to be *independent* if and only if the following condition on a vector $\lambda \in \mathbf{R}^m$:

$$\sum_{i=1}^m \lambda_i x_i = 0$$

implies $\lambda = 0$. This means that no vector in the set can be expressed as a linear combination of the others.

Example: the vectors $x^1 = [1, 2, 3]$ and $x^2 = [3, 6, 9]$ are *not* independent, since $3x^1 - x^2 = 0$.

Subspace, span, affine sets

A *subspace* of \mathbf{R}^n is a subset that is closed under addition and scalar multiplication. Geometrically, subspaces are “flat” (like a line or plane in 3D) and pass through the origin.

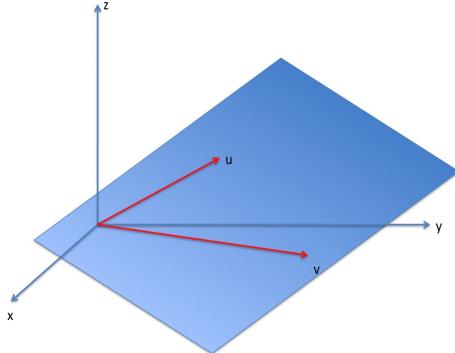
An important result of linear algebra, which we will prove later, says that a subspace \mathbf{S} can always be represented as the *span* of a set of vectors $x_i \in \mathbf{R}^n$, $i = 1, \dots, m$, that is, as a set of the form

$$\mathbf{S} = \text{span}(x_1, \dots, x_m) := \left\{ \sum_{i=1}^m \lambda_i x_i : \lambda \in \mathbf{R}^m \right\}.$$

An *affine set* is a translation of a subspace—it is “flat” but does not necessarily pass through 0 , as a subspace would. (Think for example of a line, or a plane, that does not go through the origin.) So an affine set \mathbf{A} can always be represented as the translation of the subspace spanned by some vectors:

$$\mathbf{A} = \left\{ x_0 + \sum_{i=1}^m \lambda_i x_i : \lambda \in \mathbf{R}^m \right\},$$

for some vectors x_0, x_1, \dots, x_m . In shorthand notation, we write $\mathbf{A} = x_0 + \mathbf{S}$.



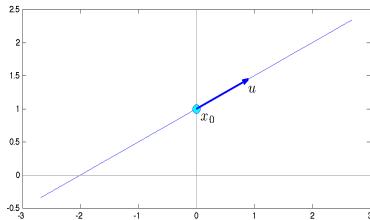
Example: In \mathbf{R}^3 , the span \mathbf{S} of the two vectors

$$u = \begin{bmatrix} -1 \\ 2 \\ 0.5 \end{bmatrix}, \quad v := \begin{bmatrix} 1 \\ 3 \\ 0.1 \end{bmatrix}$$

is the plane passing through the origin pictured in blue.

When \mathbf{S} is the span of a single non-zero vector, the set \mathbf{A} is called a *line* passing through the point x_0 . Thus, lines have the form $\{x_0 + tu : t \in \mathbf{R}\}$.

where u determines the direction of the line, and x_0 is a point through which it passes.



Example: A line in \mathbf{R}^2 passing through the point $x_0 = (2, 0)$, with direction $u = (0.8944, 0.4472)$.

Basis, dimensionBasis in \mathbf{R}^m

A *basis* of \mathbf{R}^n is a set of n independent vectors. If the vectors u_1, \dots, u_n form a basis, we can express any vector as a linear combination of the u_i 's:

$$x = \sum_{i=1}^n \lambda_i u_i$$

for appropriate numbers $\lambda_1, \dots, \lambda_n$.

The *standard basis* (alternatively, natural basis) in \mathbf{R}^n consists of the vectors e_i , where e_i 's components are all zero, except the i -th, which is equal to 1 . In \mathbf{R}^3 , we have

$$e_1 := \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad e_2 := \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad e_3 := \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Example: A basis in \mathbf{R}^3 .

Basis of a subspace

The basis of a given subspace $\mathbf{S} \subseteq \mathbf{R}^n$ is any *independent* set of vectors whose span is \mathbf{S} . If the vectors u_1, \dots, u_r form a basis of \mathbf{S} , we can express any vector as a linear combination of the u_i 's:

$$x = \sum_{i=1}^r \lambda_i u_i$$

for appropriate numbers $\lambda_1, \dots, \lambda_r$.

The number of vectors in the basis is actually independent of the choice of the basis (for example, in \mathbf{R}^3 you need two independent vectors to describe a plane containing the origin). This number is called the *dimension* of \mathbf{S} . We can accordingly define the dimension of an affine subspace, as that of the linear subspace of which it is a translation.

Examples:

- The dimension of a line is 1, since a line is of the form $x_0 + \text{span}(x_1)$ for some non-zero vector x_1 .
- [Dimension of an affine subspace.](#)

Sample variance and standard deviation

The *sample variance* of given numbers x_1, \dots, x_n , is defined as

$$\sigma^2 := \frac{1}{n} ((x_1 - \hat{x})^2 + \dots + (x_n - \hat{x})^2),$$

where \hat{x} is the [sample average](#) of x_1, \dots, x_n . The sample variance is a measure of the deviations of the numbers x_i with respect to the average value \hat{x} .

The *sample standard deviation* is the square root of the sample variance, σ^2 . It can be expressed in terms of the Euclidean norm of the vector $x = (x_1, \dots, x_n)$, as

$$\sigma = \frac{1}{\sqrt{n}} \|x - \hat{x}\|_2,$$

where $\|\cdot\|_2$ denotes the Euclidean norm.

More generally, for any vector $p \in \mathbf{R}^n$, with $p_i \geq 0$ for every i , and $p_1 + \dots + p_n = 1$, we can define the corresponding *weighted variance* as

$$\sum_{i=1}^n p_i (x_i - \hat{x})^2.$$

The interpretation of p is in terms of a discrete probability distribution of a random variable X , which takes the value x_i with probability p_i , $i = 1, \dots, n$. The weighted variance is then simply the *expected value* of the squared deviation of X from its mean $\mathbf{E}X$, under the probability distribution p .

See also: [sample and weighted average](#).

Algorithms for Convex Optimization [Convex Optimization](#) > [Convex sets](#) | [Convex functions](#) | [Convex optimization problems](#) | Algorithms | [DCP](#)

- From Least-Squares to convex minimization
- Unconstrained minimization via Newton's method
- Interior-point methods
- Gradient methods

From Least-Squares to convex minimization

We have seen how [ordinary least-squares](#) (OLS) problems can be solved using linear algebra (e.g. [SVD](#)) methods. Using OLS, we can minimize convex, quadratic functions of the form

$$q(x) = \frac{1}{2} x^T H x + g^T x$$

when $H = H^T \succeq 0$. This last requirement ensures that the function q is convex. For such convex quadratic functions, as for any convex functions, any local minimum is global. In fact, when $H \succ 0$, then the unique minimizer is $x^* = -H^{-1}g$.

It turns out one can leverage the approach to minimizing more general functions, using an iterative algorithm, based on a *local quadratic* approximation of the function at the current point. The approach can then be extended to problems with constraints, by replacing the original constrained problem with an unconstrained one, in which the constraints are *penalized* in the objective.

Unconstrained minimization: Newton's method

We consider an unconstrained minimization problem, where we seek to minimize a function twice-differentiable function f . Let us assume that the function under consideration is *strictly convex*, which is to say that its Hessian is positive definite everywhere. (If f is not convex, we might run into a [local minima](#).)

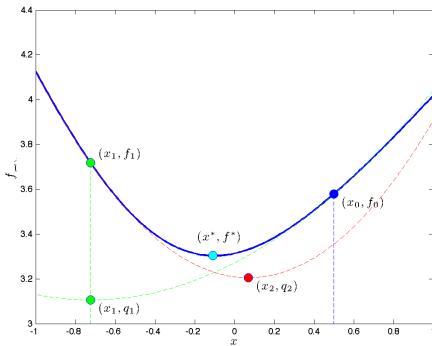
For minimizing convex functions, an iterative procedure could be based on a simple quadratic approximation procedure known as Newton's method. We start with initial guess x_0 . At each step t , we update our current guess x_t by minimizing the second-order approximation \tilde{f} of f at x_t , which is the quadratic function (see [here](#))

$$\tilde{f}(x) := f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2} (x - x_t)^T \nabla^2 f(x_t) (x - x_t),$$

where $\nabla f(x_t)$ denotes the [gradient](#), and $\nabla^2 f(x_t)$ the [Hessian](#), of f at x_t . Our next guess x_{t+1} , will be set to be a solution to the problem of minimizing \tilde{f} . Since the function is strictly convex, we have $\nabla^2 f(x_0) \succ 0$, so that the problem we are solving at each step has a unique solution, which corresponds to the global minimum of \tilde{f} . The basic Newton iteration is thus

$$x_{t+1} = x_t - (\nabla^2 f(x_t))^{-1} \nabla f(x_t)$$

Two initial steps of Newton's method to minimize the function $f : \mathbf{R}^2 \rightarrow \mathbf{R}$ with domain the whole \mathbf{R}^2 ,

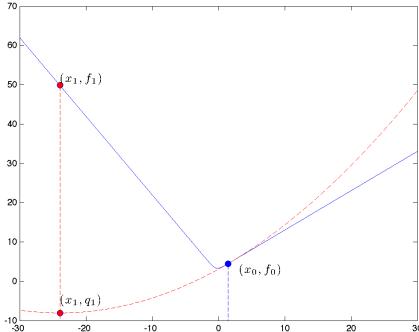


and values

$$f(x) = \log(\exp(x-3) + \exp(-2x+2)).$$

A first local quadratic approximation at the initial point $x_0 = 0.5$ is formed (dotted line in green). The corresponding minimizer is the new iterate, x_1 . The Newton algorithm proceeds to form a new quadratic approximation of the function at that point (dotted line in red), leading to the second iterate, x_2 . Although x_1 turns out to be further away from the global minimizer x^* (in light blue), x_2 is closer, and the method actually converges quickly.

This idea will fail for general (non-convex) functions. It might even fail for some convex functions. However, for a large class of convex functions, known as *self-concordant functions*, a variation on the Newton method works extremely well, and is guaranteed to find the global minimizer of the function f . For such functions, the Hessian does not vary too fast, which turns out to be a crucial ingredient for the success of Newton's method.



Failure of the Newton method to minimize the above convex function. The initial point is chosen too far from the global minimizer x^* , in a region where the function is almost linear. As a result, the quadratic approximation is almost a straight line, and the Hessian is close to zero, sending the first iterate of Newton's method to a relatively large negative value. The method quickly diverges in this case, with a second iterate at $x_2 \approx 2e^{30}$.

Constrained minimization: interior-point methods

The method above can be applied to the more general context of convex optimization problems of standard form:

$$\min_x f_0(x) \text{ subject to } f_i(x) \leq 0, \quad i = 1, \dots, m,$$

where every function involved is twice-differentiable, and convex.

The basic idea behind interior-point methods is to replace the constrained problem by an unconstrained one, involving a function that is constructed with the original problem functions. One further idea is to use a *logarithmic barrier*: in lieu of the original problem, we address

$$\min_x f(x) := f_0(x) - \mu \sum_{i=1}^m \log(-f_i(x)),$$

where $\mu > 0$ is a small parameter. The function f turns out to be convex, as long as f_0, \dots, f_m are.

For μ large, solving the above problem results in a point well “inside” the feasible set, an “interior point”. As $\mu \rightarrow 0$ the solution converges to a global minimizer for the original, constrained problem. In fact, the theory of convex optimization says that if we set $\mu = m/\epsilon$, then a minimizer to the above function is ϵ -suboptimal. In practice, algorithms do not set the value of μ so aggressively, and update the value of μ a few times.

For a large class of convex optimization problems, the function f is self-concordant, so that we can safely apply Newton's method to the minimization of the above function. In fact, for a large class of convex optimization problems, the method converges in time polynomial in m .

The interior-point approach is limited by the need to form the gradient and Hessian of the function f above. For extremely large-scale problems, this task may be too daunting.

Gradient methods Unconstrained case

Gradient methods offer an alternative to interior-point methods, which is attractive for large-scale problems. Typically, these algorithms need a considerably larger number of iterations compared to interior-point methods, but each iteration is much cheaper to process.

Perhaps the simplest algorithm to minimizing a convex function involves the iteration

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t),$$

where $\alpha_t > 0$ is a parameter. The interpretation of the algorithm is that it tries to decrease the value of the function by taking a step in the direction of the negative gradient. For small enough value of α_t , indeed we have $f(x_{t+1}) \leq f(x_t)$.

Depending on the choice of the parameter α_t (as a function of the iteration number t), and some properties on the function f , convergence can be rigorously proven.

Constrained case

The gradient method can be adapted to constrained problems, via the iteration

$$x_{t+1} = P(x_t - \alpha_t \nabla f(x_t)),$$

where P is the projection operator, which to its argument z associates the point closest (in Euclidean norm sense) to z in \mathbb{C} . Depending on problem structure, this projection may or may not be easy to perform.

Example: Box-constrained least-squares.

- Symmetric matrices and quadratic functions
- Second-order approximation of non-linear functions
- Special symmetric matrices

Symmetric matrices and quadratic functions Symmetric matrices

A square matrix $A \in \mathbf{R}^{n \times n}$ is *symmetric* if it is equal to its transpose. That is,

$$A_{ij} = A_{ji}, \quad 1 \leq i, j \leq n.$$

The set of symmetric $n \times n$ matrices is denoted \mathbf{S}^n . This set is a subspace of $\mathbf{R}^{n \times n}$.

Examples:

- [A \$3 \times 3\$ example](#).
- [Representation of a weighted, undirected graph](#).
- [Laplacian matrix of a graph](#).
- [Hessian of a function](#).
- [Gram matrix](#) of data points.

Quadratic functions

A function $q : \mathbf{R}^n \rightarrow \mathbf{R}$ is said to be a *quadratic function* if it can be expressed as

$$q(x) = \sum_{i=1}^n \sum_{j=1}^n A_{ij} x_i x_j + 2 \sum_{i=1}^n b_i x_i + c,$$

for numbers A_{ij} , b_i , and c , $i, j \in \{1, \dots, n\}$. A quadratic function is thus an affine combination of the x_i 's and all the “cross-products” $x_i x_j$. We observe that the coefficient of $x_i x_j$ is $(A_{ij} + A_{ji})$.

The function is said to be a *quadratic form* if there are no linear or constant terms in it: $b_i = 0, c = 0$.

Note that the [Hessian](#) (matrix of second-derivatives) of a quadratic function is constant.

Examples:

- [Quadratic functions of two variables](#).
- [Hessian of a quadratic function](#).

Link between quadratic functions and symmetric matrices

There is a natural relationship between symmetric matrices and quadratic functions. Indeed, any quadratic function $q : \mathbf{R}^n \rightarrow \mathbf{R}$ can be written as

$$q(x) = \begin{pmatrix} x \\ 1 \end{pmatrix}^T \begin{pmatrix} A & b \\ b^T & c \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} = x^T A x + 2b^T x + c,$$

for an appropriate symmetric matrix $A \in \mathbf{S}^n$, vector $b \in \mathbf{R}^n$ and scalar $c \in \mathbf{R}$. Here, A_{ii} is the coefficient of x_i^2 in q ; for $i \neq j$, $2A_{ij}$ is the coefficient of the term $x_i x_j$ in q ; $2b_i$ is that of x_i ; and c is the constant term, $q(0)$. If q is a quadratic form, then $b = 0, c = 0$, and we can write $q(x) = x^T A x$ where now $A \in \mathbf{S}^n$.

Examples:

- [Two-dimensional example](#).

Second-order approximations of non-quadratic functions

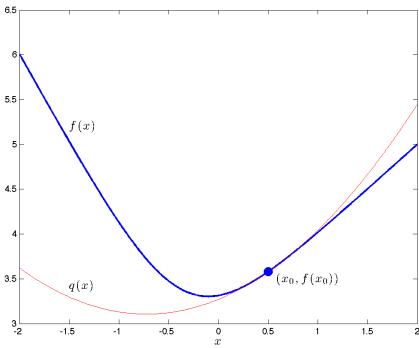
We have seen [here](#) how linear functions arise when one seeks a simple, linear approximation to a more complicated non-linear function. Likewise, quadratic functions arise naturally when one seeks to approximate a given non-quadratic function by a quadratic one.

One-dimensional case

If $f : \mathbf{R} \rightarrow \mathbf{R}$ is a twice-differentiable function of a *single* variable, then the *second order approximation* (or, second-order Taylor expansion) of f at a point x_0 is of the form

$$f(x) \approx q(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2,$$

where $f'(x_0)$ is the first derivative, and $f''(x_0)$ the second derivative, of f at x_0 . We observe that the quadratic approximation q has the same value, derivative, and second-derivative as f , at x_0 .



Example: The figure shows a second-order approximation q of the univariate function $f : \mathbf{R} \rightarrow \mathbf{R}$, with values

$$f(x) = \log(\exp(x-3) + \exp(-2x+2)),$$

at the point $x_0 = 0.5$ (in red).

Multi-dimensional case

In multiple dimensions, we have a similar result. Let us approximate a twice-differentiable function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ by a quadratic function q , so that f and q coincide up and including to the second derivatives.

The function q must be of the form

$$q(x) = x^T Ax + 2b^T x + c,$$

where $A \in \mathbf{S}^n$, $b \in \mathbf{R}^n$, and $c \in \mathbf{R}$. Our condition that q coincides with f up and including to the second derivatives shows that we must have

$$\nabla^2 q(x) = 2A = \nabla^2 f(x_0), \quad \nabla q(x) = 2(Ax_0 + b) = \nabla f(x_0), \quad x_0^T Ax_0 + 2b^T x_0 + c = f(x_0),$$

where $\nabla^2 f(x_0)$ is the [Hessian](#), and $\nabla f(x_0)$ the gradient, of f at x_0 .

Solving for A, b, c we obtain the following result:

Second-order expansion of a function. The second-order approximation of a twice-differentiable function f at a point x_0 is of the form

$$f(x) \approx q(x) = f(x_0) + \nabla f(x_0)^T(x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0),$$

where $\nabla f(x_0) \in \mathbf{R}^n$ is the [gradient](#) of f at x_0 , and the symmetric matrix $\nabla^2 f(x_0)$ is the [Hessian](#) of f at x_0 . \diamond

Example:[Second-order expansion of the log-sum-exp function](#).

Special symmetric matrices
Diagonal matrices

Perhaps the simplest special case of symmetric matrices is the class of diagonal matrices, which are non-zero only on their diagonal.

If $\lambda \in \mathbf{R}^n$, we denote by $\text{diag}(\lambda_1, \dots, \lambda_n)$, or $\text{diag}(\lambda)$ for short, the $n \times n$ (symmetric) diagonal matrix with λ on its diagonal. Diagonal matrices correspond to quadratic functions of the form

$$q(x) = \sum_{i=1}^n \lambda_i x_i^2 = x^T \text{diag}(\lambda) x.$$

Such functions do not have any “cross-terms” of the form $x_i x_j$ with $i \neq j$.

Example:[A diagonal matrix and its associated quadratic form](#).

Symmetric dyads

Another important class of symmetric matrices is that of the form uu^T , where $u \in \mathbf{R}^n$. The matrix has elements $u_i u_j$, and is symmetric. Such matrices are called [symmetric dyads](#). (If $\|u\|_2 = 1$, then the dyad is said to be normalized.)

Symmetric dyads corresponds to quadratic functions that are simply squared linear forms: $q(x) = (u^T x)^2$.

Example:[A squared linear form](#).

Spectral Theorem [Symmetric Matrices](#) > [Definitions](#) | Spectral theorem | [PSD matrices](#) | [PCA](#) | [Applications](#)

- Eigenvalues and eigenvectors of symmetric matrices
- The symmetric eigenvalue decomposition theorem
- Rayleigh quotients

Eigenvalues and eigenvectors of symmetric matrices

Let A be a square, $n \times n$ symmetric matrix. A real scalar λ is said to be an *eigenvalue* of A if there exist a non-zero vector $u \in \mathbf{R}^n$ such that

$$Au = \lambda u.$$

The vector u is then referred to as an *eigenvector* associated with the eigenvalue λ . The eigenvector u is said to be *normalized* if $\|u\|_2 = 1$. In this case, we have $u^T A u = \lambda u^T u = \lambda$.

The interpretation of u is that it defines a direction along A behaves just like scalar multiplication. The amount of scaling is given by λ . (In German, the root “eigen”, means

“self” or “proper”). The eigenvalues of the matrix A are characterized by the *characteristic equation*

$$\det(\lambda I - A) = 0,$$

where the notation \det refers to the [determinant](#) of its matrix argument. The function with values $t \rightarrow p(t) := \det(tI - A)$ is a polynomial of degree n called the *characteristic polynomial*.

From the fundamental theorem of algebra, any polynomial of degree n has n (possibly not distinct) complex roots. For symmetric matrices, the eigenvalues are real, since $\lambda = u^T A u$ when $Au = \lambda u$, and u is normalized.

Spectral theorem

An important result of linear algebra, called the *spectral theorem*, or *symmetric eigenvalue decomposition* (SED) theorem, states that for any symmetric matrix, there are exactly n (possibly not distinct) eigenvalues, and they are all real; further, that the associated eigenvectors can be chosen so as to form an orthonormal basis. The result offers a simple way to decompose the symmetric matrix as a product of simple transformations.

Theorem: Symmetric eigenvalue decomposition

We can decompose any symmetric matrix $A \in \mathbf{S}^n$ with the *symmetric eigenvalue decomposition* (SED)

$$A = \sum_{i=1}^n \lambda_i u_i u_i^T = U \Lambda U^T, \quad \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n).$$

where the matrix of $U := [u_1, \dots, u_n]$ is orthogonal (that is, $U^T U = U U^T = I_n$), and contains the eigenvectors of A , while the diagonal matrix Λ contains the eigenvalues of A .

Here is a [proof](#). The SED provides a decomposition of the matrix in simple terms, namely [dyads](#).

We check that in the SED above, the scalars λ_i are the eigenvalues, and u_i 's are associated eigenvectors, since

$$Au_j = \sum_{i=1}^n \lambda_i u_i u_i^T u_j = \lambda_j u_j, \quad j = 1, \dots, n.$$

The eigenvalue decomposition of a symmetric matrix can be efficiently computed with standard software, in time that grows proportionately to its dimension n as n^3 . Here is the matlab syntax, where the first line ensure that matlab knows that the matrix A is exactly symmetric.

Matlab syntax

```
>> A = triu(A)+tril(A',-1);
>> [U,D] = eig(A);
```

Example:

- [Eigenvalue decomposition of a \$2 \times 2\$ symmetric matrix](#).

Rayleigh quotients

Given a symmetric matrix A , we can express the smallest and largest eigenvalues of A , denoted λ_{\min} and λ_{\max} respectively, in the so-called *variational* form

$$\lambda_{\min}(A) = \min_x \{x^T A x : x^T x = 1\}, \quad \lambda_{\max}(A) = \max_x \{x^T A x : x^T x = 1\}.$$

For a proof, see [here](#).

The term “variational” refers to the fact that the eigenvalues are given as optimal values of optimization problems, which were referred to in the past as variational problems. Variational representations exist for all the eigenvalues, but are more complicated to state.

The interpretation of the above identities is that the largest and smallest eigenvalues is a measure of the range of the quadratic function $x \rightarrow x^T A x$ over the unit Euclidean ball. The quantities above can be written as the minimum and maximum of the so-called *Rayleigh quotient* $x^T A x / x^T x$.

Historically, David Hilbert coined the term “spectrum” for the set of eigenvalues of a symmetric operator (roughly, a matrix of infinite dimensions). The fact that for symmetric matrices, every eigenvalue lies in the interval $[\lambda_{\min}, \lambda_{\max}]$ somewhat justifies the terminology.

Example: [Largest singular value norm of a matrix](#).

Slater Condition for Strong Duality [Strong Duality](#) > Slater Condition | [Optimality conditions](#) | [Sensitivity](#) | [Applications](#)

- Primal and dual problem
- Strong duality
- Slater's theorem
- Geometry

Primal and Dual Problems

We consider a *convex* constrained optimization problem in [standard form](#): $p^* := \min_x f_0(x) : Ax = b, f_i(x) \leq 0, i = 1, \dots, m$, where $f_0, \dots, f_m : \mathbf{R}^n \rightarrow \mathbf{R}$ are convex functions, $A \in \mathbf{R}^{p \times n}$, $b \in \mathbf{R}^p$ define the affine inequality constraints.

To this problem we associate the Lagrangian, which is the function $L : \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R}^p \rightarrow \mathbf{R}$ with values

$$g(\lambda, \nu) = \min_x f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \nu^T (Ax - b).$$

corresponding dual function is the function $g : \mathbf{R}^m \times \mathbf{R}^p \rightarrow \mathbf{R}$ with values

Recall that the function g is

concave, and that it can assume $-\infty$ values. Its domain is $\text{dom } g := \{(\lambda, \nu) : g(\lambda, \nu) > -\infty\}$.

Finally, the dual problem reads $d^* = \max_{\lambda \geq 0, \nu} g(\lambda, \nu)$. Note that the sign constraints are imposed only on the dual variables corresponding to inequality constraints. Note also that there are (possibly) implicit constraints in the above problem, since we must have $(\lambda, \nu) \in \text{dom } g$.

Strong duality

The theory of weak duality seen here states that $p^* \geq d^*$. This is true always, even if the original problem is not convex. We say that *strong duality* holds if $p^* = d^*$.

Slater's sufficient condition for strong duality

Slater's theorem provides a *sufficient condition* for strong duality to hold. Namely, if

- The primal problem is convex;
- It is strictly feasible, that is, there exists $x_0 \in \mathbf{R}^n$ such that $Ax_0 = b$, $f_i(x_0) < 0$, $i = 1, \dots, m$, then, strong duality holds: $p^* = d^*$, and the dual problem is attained. (Proof)

Example:

- [Minimum distance to an affine subspace](#).
- [Dual of LP](#).
- [Dual of QP](#).

Geometry

The geometric interpretation of weak duality shows why strong duality holds for a convex, strictly feasible problem. For simplicity again, we consider the case with no equality constraints, and a single convex constraint.

[Recall](#) that we can express the primal problem with two new scalar variables u, t , as follows: $p^* = \min_{u,t} t : (u, t) \in \mathbf{A}, u \leq 0$, where $\mathbf{A} := \{(u, t) \in \mathbf{R}^2 : \exists x, u \geq f_1(x), t \geq f_0(x)\}$. Since the primal problem is convex, that is, f_0 and f_1 are convex functions, the above set is convex.

Strict primal feasibility means that the set \mathbf{A} cuts “inside” the right-half of the (u, t) -plane. If that property holds, then we can attain the optimal point $(0, p^*)$ by a tangent with a finite strictly negative slope. One implication is that $d^* = p^*$, that is, strong duality holds. This slope is precisely the optimal dual variable, λ^* ; thus the dual problem is attained.

Sample covariance matrixDefinition

For a vector $z \in \mathbf{R}^m$, the [sample variance](#) σ^2 measures the average deviation of its coefficients around the [sample average](#) \hat{x} :

$$\hat{z} := \frac{1}{n}(z(1) + \dots + z(m)), \quad \sigma^2 := \frac{1}{n}((z(1) - \hat{z})^2 + \dots + (z(m) - \hat{z})^2),$$

Now consider a matrix $X = [x_1, \dots, x_m] \in \mathbf{R}^{n \times m}$, where each column x_i represents a data point in \mathbf{R}^n . We are interested in describing the amount of variance in this data set. To this end, we look at the numbers we obtain by [projecting the data along a line](#) defined by the direction $u \in \mathbf{R}^n$. This corresponds to the (row) vector in \mathbf{R}^m

$$z = (u^T x_1, \dots, u^T x_m) = u^T X \in \mathbf{R}^m.$$

The corresponding sample mean and variance are

$$\hat{z} = u^T \hat{x}, \quad \sigma^2(u) := \frac{1}{m} \sum_{k=1}^m (u^T x_k - u^T \hat{x})^2,$$

where $\hat{x} := (1/m)(x_1 + \dots + x_m) \in \mathbf{R}^n$ is the sample mean of the vectors x_1, \dots, x_m .

The sample variance along direction u can be expressed as a [quadratic form](#) in u :

$$\sigma^2(u) = \frac{1}{n} \sum_{k=1}^n [u^T (x_k - \hat{x})]^2 = u^T \Sigma u,$$

where Σ is a $n \times n$ symmetric matrix, called the *sample covariance matrix* of the data points:

$$\Sigma := \frac{1}{m} \sum_{k=1}^m (x_k - \hat{x})(x_k - \hat{x})^T.$$

Properties

The covariance matrix satisfies the following properties.

- The sample covariance matrix allows to find the variance along any direction in data space.
- The diagonal elements of Σ give the variances of each vector in the data.
- The trace of Σ gives the sum of all the variances.
- The matrix Σ is [positive semi-definite](#), since the associated quadratic form $u \rightarrow u^T \Sigma u$ is non-negative everywhere.

Matlab syntax

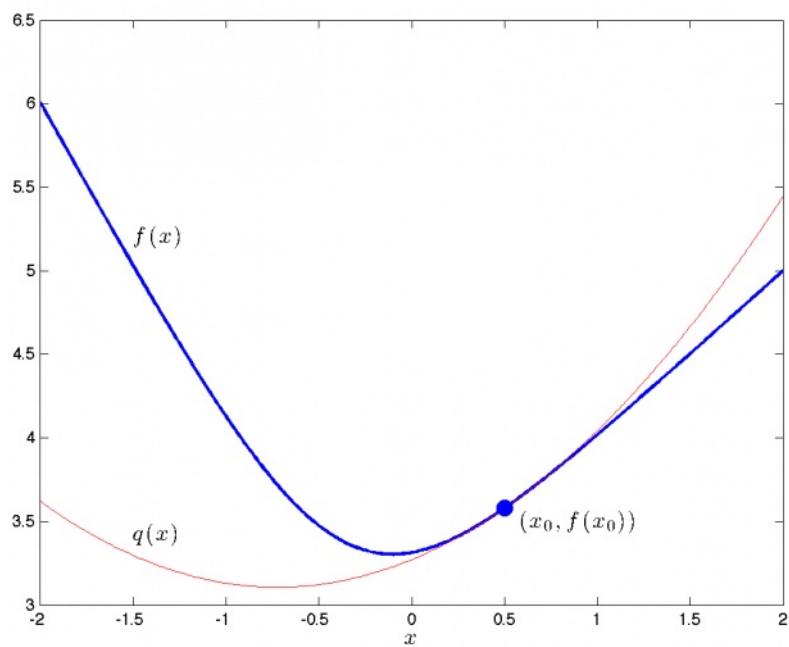
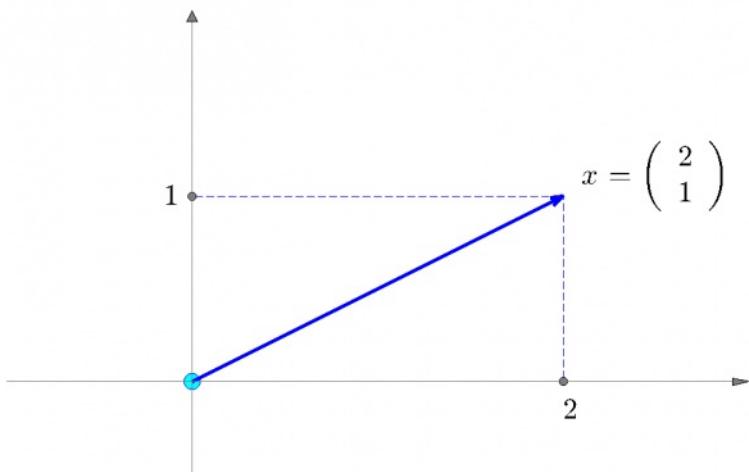
The following matlab syntax assumes that the m data points in \mathbf{R}^n are collected in a $n \times m$ matrix $X: X = [x_1, \dots, x_m]$

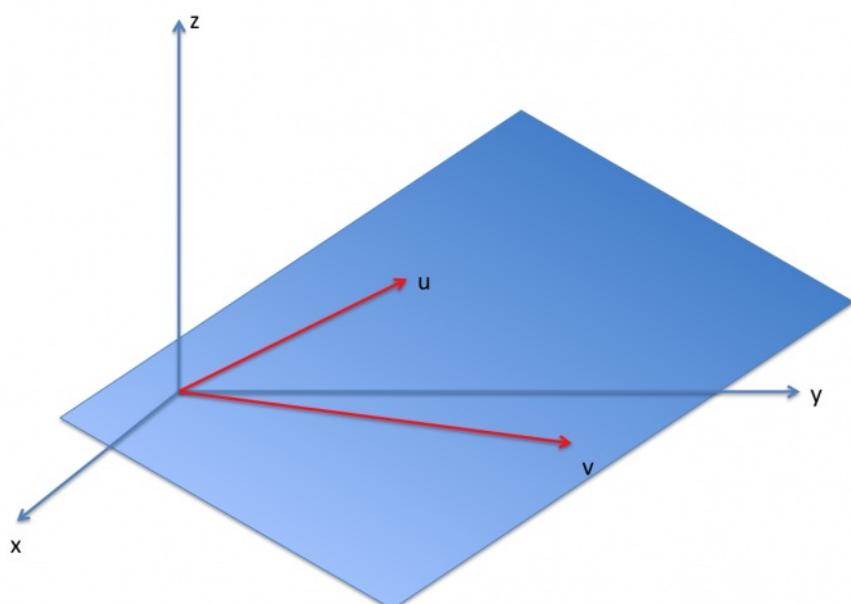
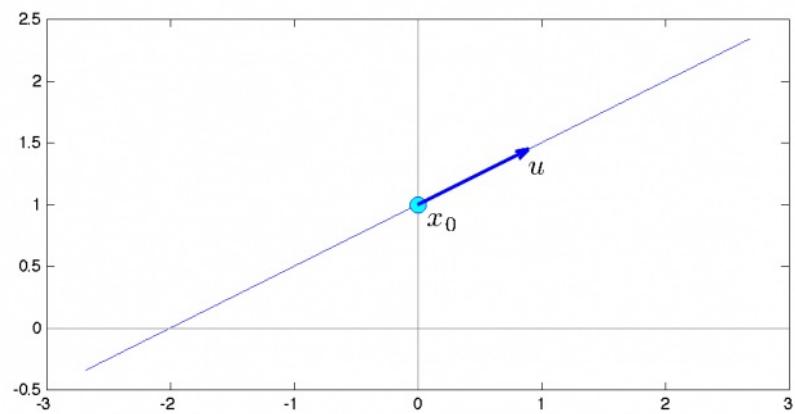
Matlab syntax

```

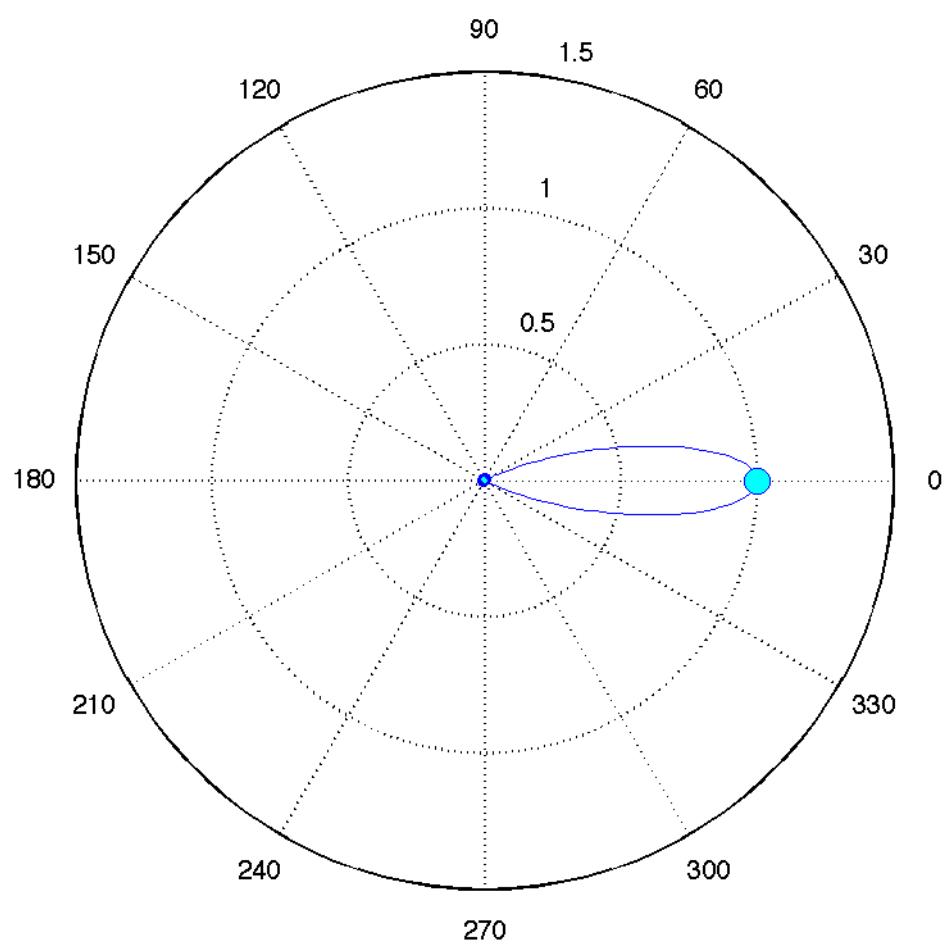
>> xhat = mean(X,2); % mean of columns of matrix X
>> Xc = X-xhat*ones(1,m); % centered data matrix
>> Sigma = (1/m)*Xc'*Xc; % covariance matrix
>> Sigma = cov(X',1); % built-in command produces the same thing

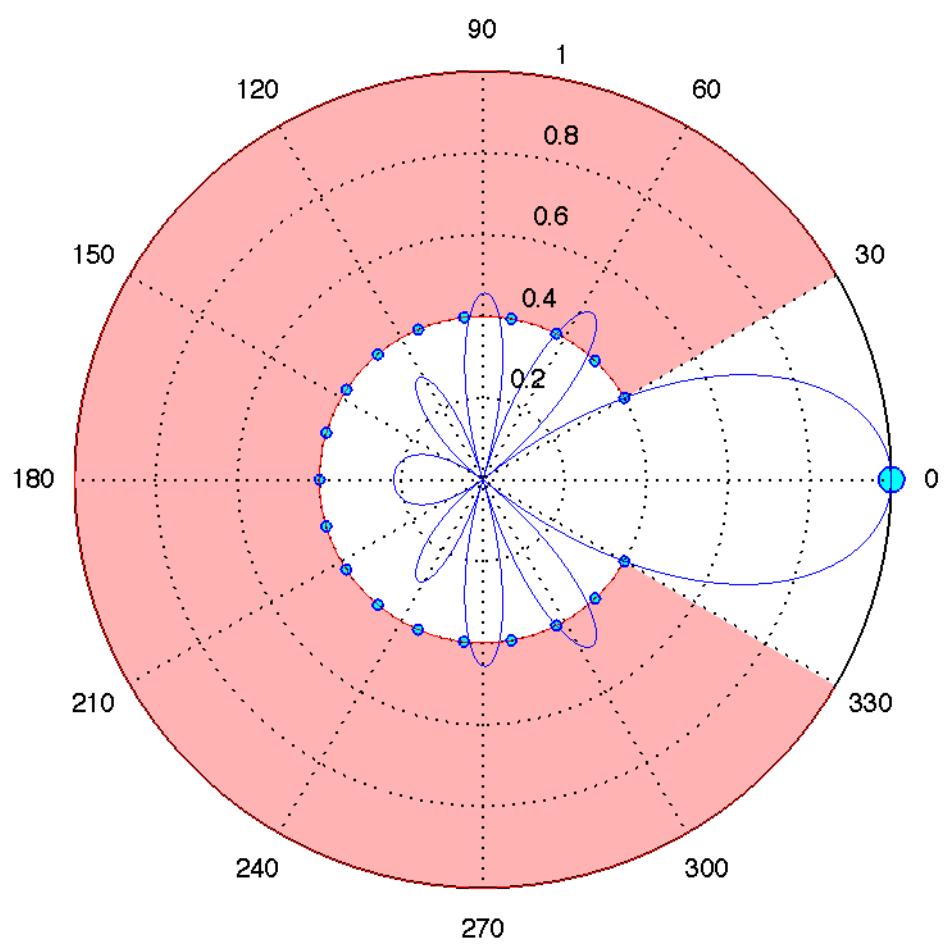
```

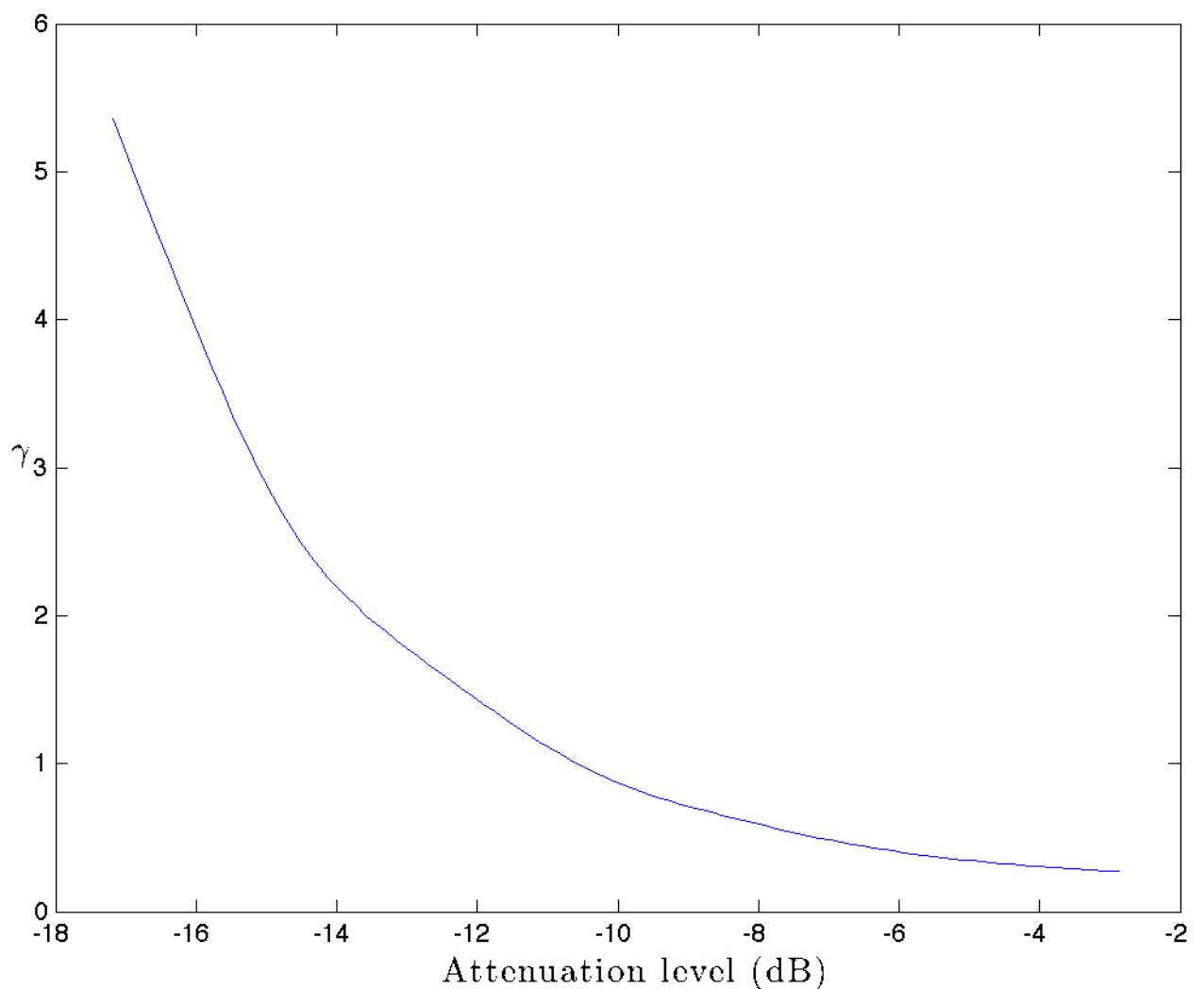


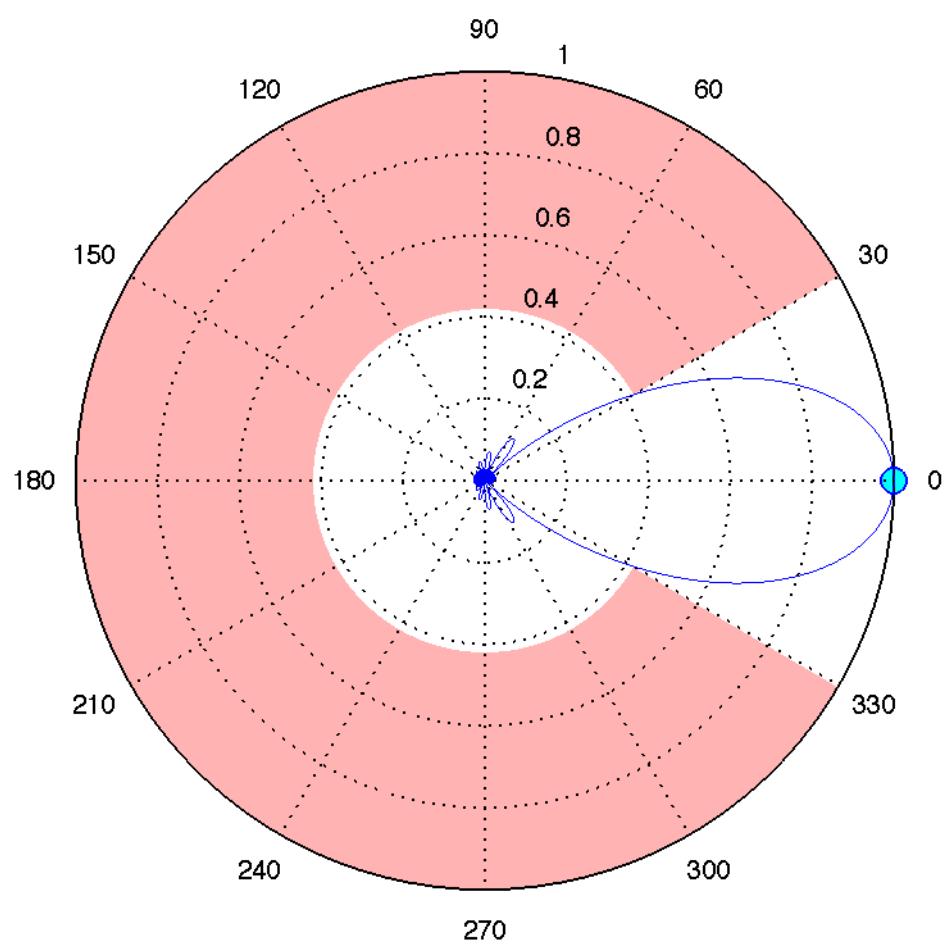


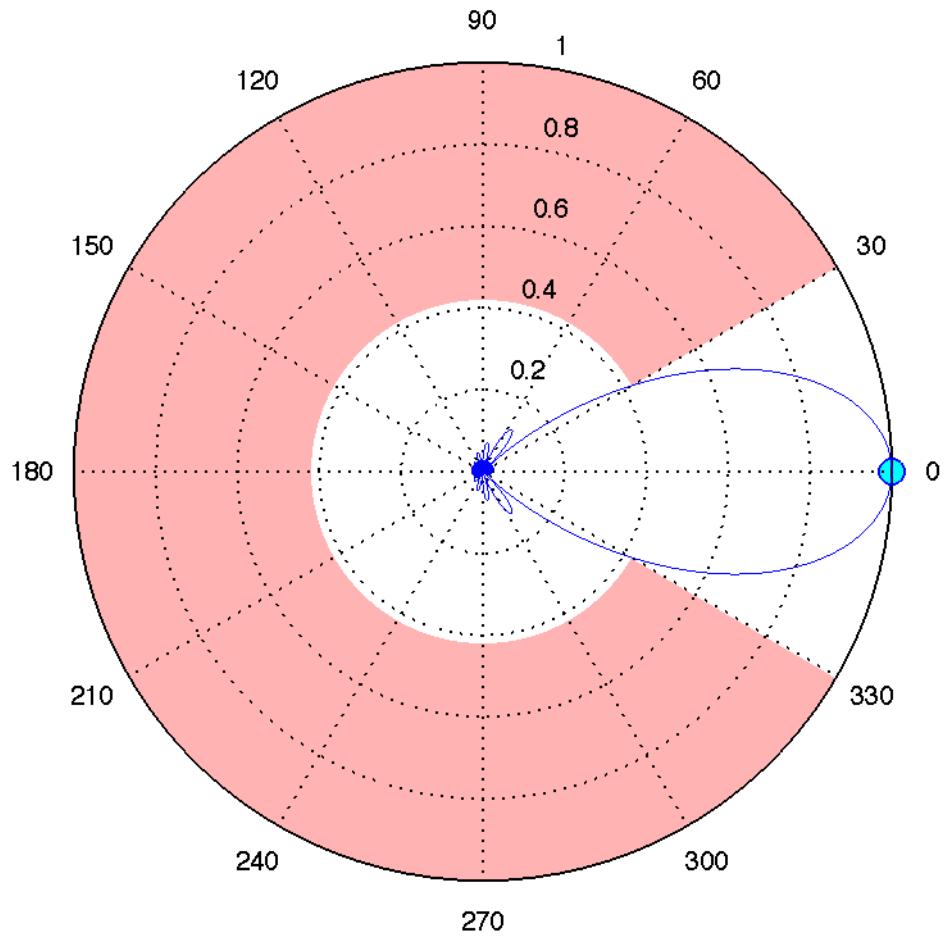
SHELBY--SESSIONS
STEVENS--MURKOWSKI
MCCAIN-----KYL
LINCOLN--PRYOR
FEINSTEIN--BOXER
SALAZAR--ALLARD
LIEBERMAN--DODD
CARPER--BIDEN
NELSON--MARTINEZ
ISAKSON--CHAMBLISS
INOUE--AKAKA
CRAPO--CRAIG
OBAMA--DURBIN
LUGAR--BAYH
HARKIN--GRASSLEY
ROBERTS--BROWNBACK
MCCONNELL--BURNING
LANDRIEU--VITTER
SNOWE--COLLINS
SARBANES--MIKULSKI
KERRY--KENNEDY
LEVIN--STABENOW
COLEMAN--DAYTON
LOTT--TALENT
BOND--BAUCUS
BURNS--HAGEL
NELSON--ENSIGN
REID--GREGG
SUNUNU--CORZINE
MENENDEZ--LAUTENBERG
BINGAMAN--DOMENICI
CLINTON--SCHUMER
BURR--DOLE
CONRAD--DORGAN
DEWINE--VOINOVICH
INHOFE--COBURN
SMITH--WYDEN
SANTORUM--SPECTER
CHAFFEE--REED
DEMINT--GRAHAM
THUNE--JOHNSON
FRIST--ALEXANDER
CORNYN--HUTCHISON
BENNETT--HATCH
JEFFORDS--LEAHY
ALLEN--WARNER
CANTWELL--MURRAY
BYRD--ROCKEFELLER
FEINGOLD--KOHL
ENZI--THOMAS











Linearly Constrained Least-Squares Problems [Least-Squares](#) > [Definitions](#) >> [[LS](#)|[Regularized LS](#)] | [Constrained LS](#)] | [Solution](#) | [Sensitivity](#) | [Limitations](#)

- Linearly-constrained least-squares
- Minimum-norm solutions to linear equations

Linearly constrained least-squares Definition

An interesting variant of the ordinary least-squares problem involves equality constraints on the decision variable x :

$$\min_x \|Ax - y\|_2^2 : Cx = d,$$

where $C \in \mathbf{R}^{p \times n}$, and $d \in \mathbf{R}^p$ are given.

Examples:

- [Minimum-variance portfolio](#).

Solution

We can express the solution by first computing the nullspace of C . Assuming that the feasible set of the constrained LS problem is not empty, it can be expressed as

$$\mathbf{X} = \{x_0 + Nz : z \in \mathbf{R}^k\},$$

where k is the dimension of the nullspace of C , and x_0 is a particular solution to the equation $Cx = d$.

Expressing x in terms of the free variable z , we can write the constrained problem as an unconstrained one:

$$\min_z \|\tilde{A}z - \tilde{y}\|_2,$$

where $\tilde{A} := AN$, and $\tilde{y} = y - Ax_0$.

Minimum-norm solution to linear equations

A special case of linearly constrained LS is

$$\min_x \|x\|_2 : Ax = y,$$

in which we implicitly assume that the linear equation in x : $Ax = y$, has a solution, that is, y is in the range of A .

The above problem allows to select a particular solution to a linear equation, in the case when there are possibly many, that is, the linear system $Ax = y$ is under-determined.

As seen [here](#), when A is full row rank, the matrix AA^T is invertible, and the above has the closed-form solution

$$x^* = A^T(AA^T)^{-1}y.$$

Examples: [Control positioning of a mass](#).

Some Limitations of OLS [Least-Squares](#) > [Ordinary least-squares](#) | [Optimal set](#) | [Sensitivity analysis](#) | Limitations

- Limitations on constraints
- Limitations on residual norm
- Limitations on regularization norms

Sensitivity Analysis [Least-Squares](#) > [Ordinary LS](#) | [Variants](#) | Sensitivity | [Applications](#)

- Relative error analysis
- Condition number

We consider the OLS problem:

$$\min_x \|Ax - y\|_2.$$

with

- $A \in \mathbf{R}^{m \times n}$ the data matrix (known);
- $y \in \mathbf{R}^m$ is the measurement (known);
- $x \in \mathbf{R}^n$ is the vector to be estimated (unknown).

We assume that A is full column rank. Then, the solution \hat{x}_{LS} to the OLS problem is unique, and can be written as a *linear* function of the measurement vector y :

$$\hat{x}_{\text{LS}} = A^\dagger y,$$

with A^\dagger the pseudo-inverse of A . Again, since A is full column rank,

$$A^\dagger = (A^T A)^{-1} A^T.$$

We are interested in analyzing the impact of perturbations in the vector y , on the resulting solution \hat{x}_{LS} . We begin by analyzing the absolute errors in the estimate, and then turn to the analysis of *relative* errors.

Bound on the absolute error

We first try to find a bound on the error in the solution given a bound on the error in the input. Let us assume a simple model of potential perturbations: we assume that δy belongs to a unit ball: $\|\delta y\|_2 \leq \alpha$, where $\alpha > 0$ is given. We will assume $\alpha = 1$ for simplicity; the analysis is easily extended to any $\alpha > 0$.

Let us define $\hat{x}_{\text{LS}} + \delta x$ to be the estimate corresponding to the measurement $y + \delta y$, that is:

$$\hat{x}_{\text{LS}} + \delta x = A^\dagger(y + \delta y).$$

Since $\hat{x}_{\text{LS}} = A^\dagger y$, we have

$$\delta x = A^\dagger \delta y.$$

We obtain that

$$\|\delta x\|_2 \leq \|A^\dagger\| \cdot \|\delta y\|_2,$$

where $\|\cdot\|$ is the [matrix norm](#), which is defined for a matrix B as

$$\|B\| := \max_{x : \|x\|_2 \leq 1} \|Bx\|_2.$$

In other words, $\|B\|$ is simply the maximum output norm that we can attain with a unit-norm input. (The matrix norm can be computed from an [SVD](#) of the matrix B .)

The interpretation is that $\|A^\dagger\|$ is the largest absolute error that can result from an absolute error in the input y that bounded by 1.

Bound on the relative error analysis

We now turn to the problem of analyzing the impact of relative errors in the measurement, on the relative error in the solution.

To simplify, let us assume that the matrix A is square ($m = n$) and full rank (invertible). The development before simplifies to

$$\delta x = (A^T A)^{-1} A^T y = A^{-1} \delta y,$$

so that

$$\|\delta x\|_2 \leq \|A^{-1}\| \cdot \|\delta y\|_2.$$

Likewise, the equation $Ax = y$ allows to bound $\|x\|_2$ with respect to $\|y\|_2$:

$$\|y\|_2 \leq \|A\| \cdot \|x\|_2.$$

Combining the two inequalities we obtain:

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq (\|A\| \cdot \|A^{-1}\|) \cdot \frac{\|\delta y\|_2}{\|y\|_2}. \quad \text{Condition number}$$

As seen above, when A is square, invertible, the quantity

$$\kappa(A) := \|A\| \cdot \|A^{-1}\|$$

measures the sensitivity to relative errors in the system $Ax = y$.

The quantity above is called the [condition number](#) of A . The condition number is always greater than 1. The closer to 1, the more confident we can be about the solution x when y is subject to small relative errors. We can generalize this notion to non-necessarily square matrices that have full rank.

Many practical algorithms to solve linear equations (or OLS) use the condition number (or an estimate) to evaluate the quality of the solution.

Regularized Least-Squares Problem [Least-Squares > Definitions](#) >> [[LS](#)] Regularized LS | [Constrained LS](#)] | [Solution](#) | [Sensitivity](#) | [Limitations](#)

- Regularized least-squares
- Expression as an ordinary LS problem
- Solution

Regularized least-squares

In the case when the matrix A in the OLS problem is not full column rank, the closed-form solution cannot be applied. A remedy often used in practice is to transform the original problem into one where the full column rank property holds.

The *regularized* least-squares problem has the form

$$\min_x \|Ax - y\|_2^2 + \lambda \|x\|_2^2,$$

where $\lambda > 0$ is a (usually small) parameter.

Expression as an ordinary LS problem

The regularized problem can be expressed as an ordinary least-squares problem, where the data matrix is full column rank. Indeed, the above problem can be written as the ordinary LS problem

$$\min_x \|\tilde{A}x - \tilde{y}\|_2^2,$$

where

$$\tilde{A} := \begin{pmatrix} A \\ \sqrt{\lambda} I_n \end{pmatrix}, \quad \tilde{y} := \begin{pmatrix} y \\ 0 \end{pmatrix}.$$

The presence of the identity matrix in the $(m+n) \times n$ matrix \tilde{A} ensures that it is full (column) rank.

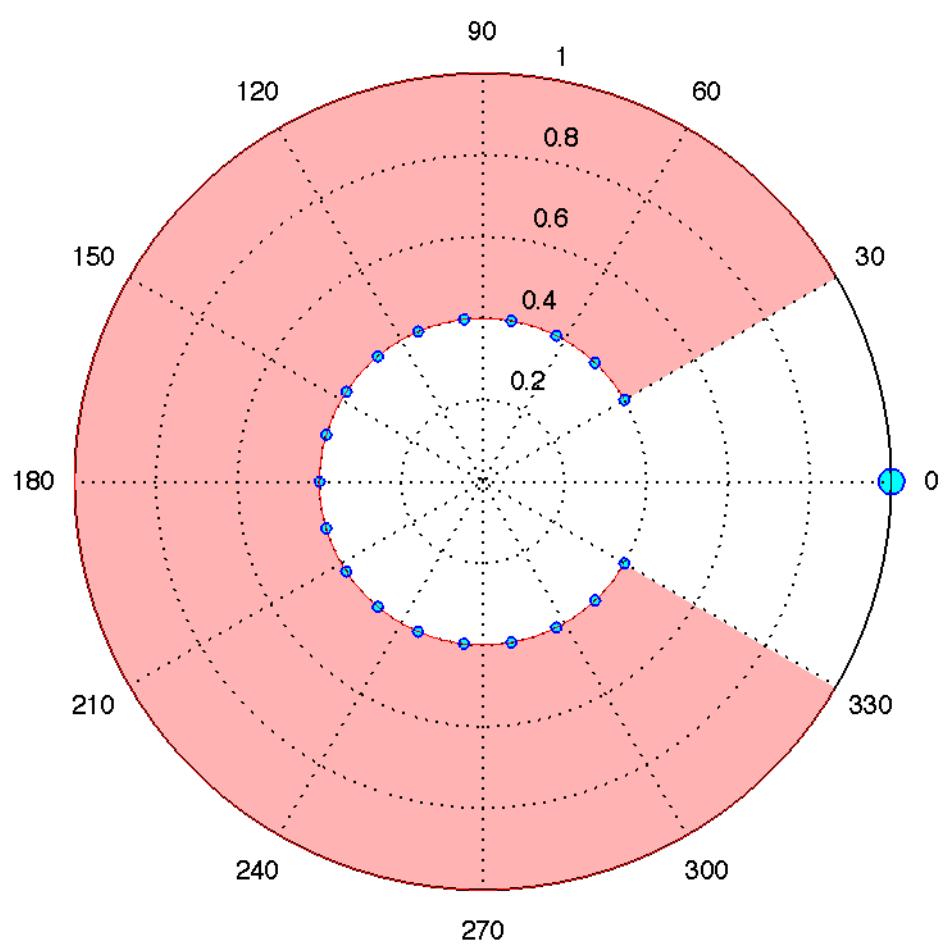
Solution

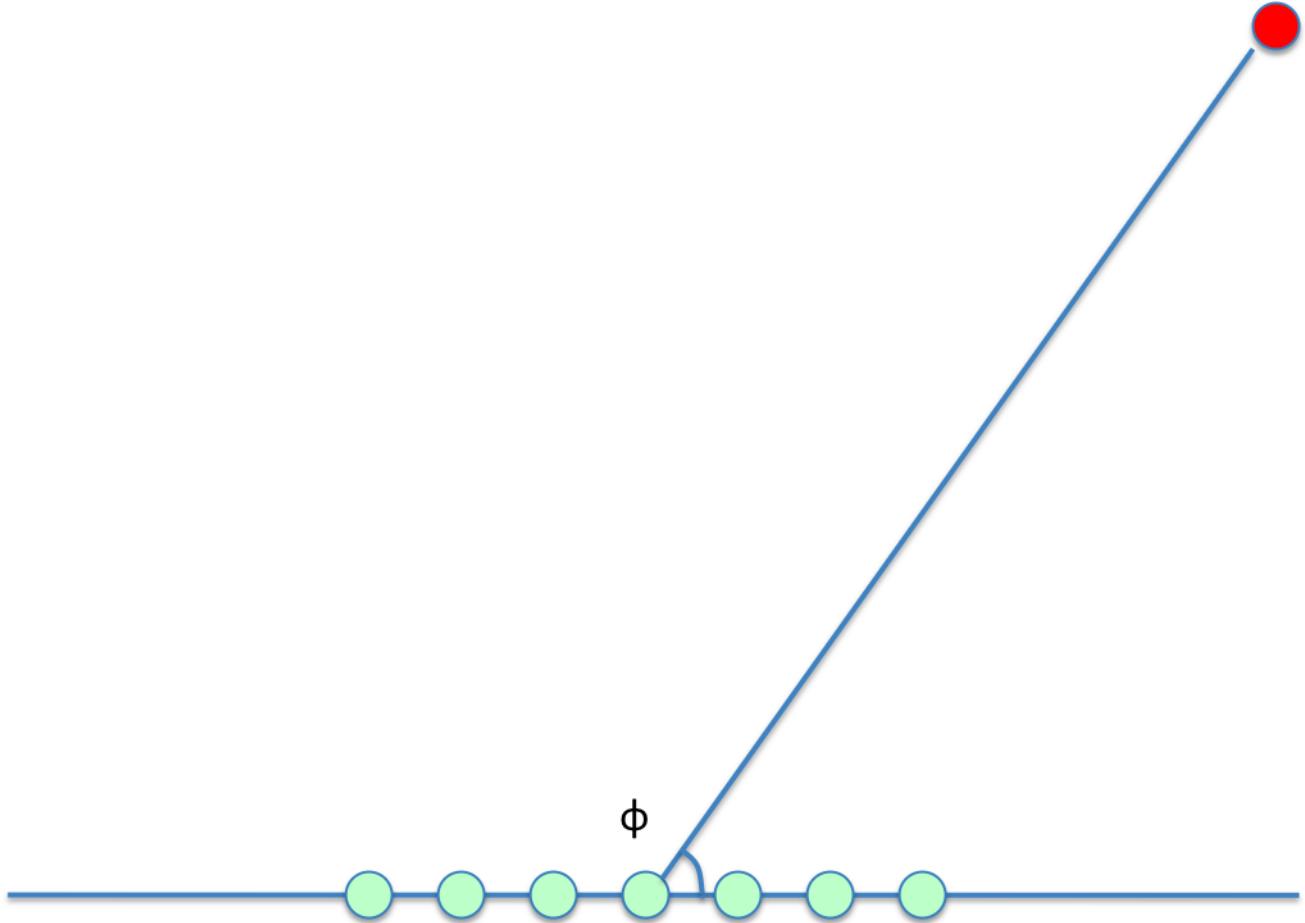
Since the data matrix in the regularized LS problem has full column rank, the formula seen [here](#) applies. The solution is unique, and given by

$$x^* = (\tilde{A}^T \tilde{A})^{-1} \tilde{A}^T \tilde{y} = (A^T A + \lambda I_n)^{-1} A^T y.$$

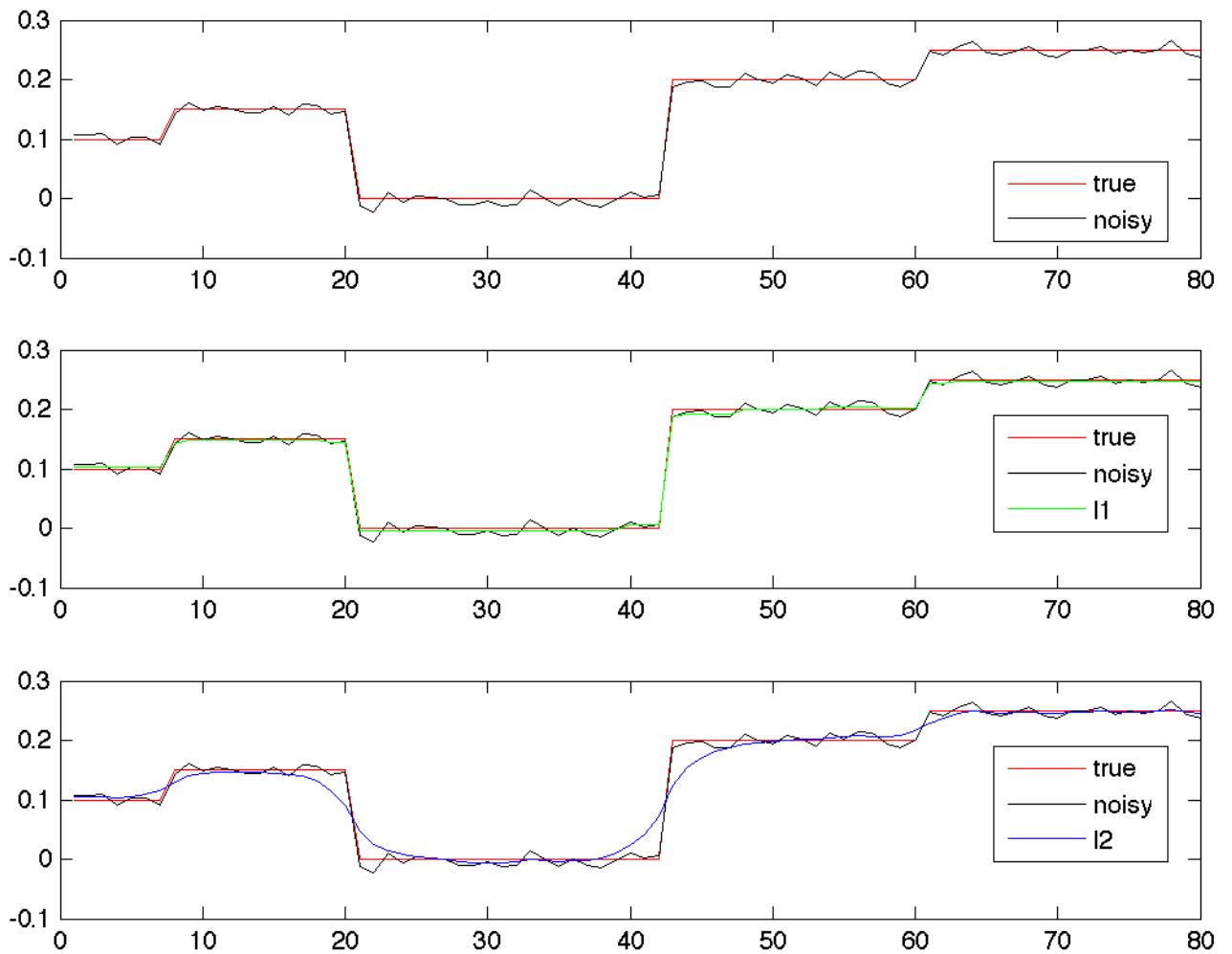
For $\lambda = 0$, we recover the ordinary LS expression that is valid when the original data matrix is full rank.

The above formula explains one of the motivations for using regularized least-squares in the case of a rank-deficient matrix A : if $\lambda > 0$, but is small, the above expression is still defined, even if A is rank-deficient.





This section under development



Half-Spaces [LP and QP](#) > Half-Spaces | [Polyhedra](#) | [Standard Forms](#) | [Polyhedral Functions](#) | [Applications](#)

- Definition
- Geometry
- Link with linear functions

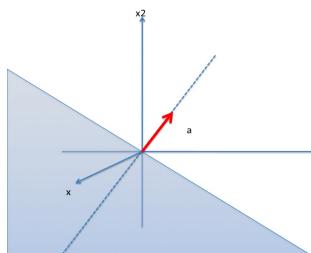
Definition

A *half-space* is a set defined by a single affine inequality. Precisely, a half-space in \mathbf{R}^n is a set of the form

$$\mathbf{H} = \{x : a^T x \leq b\},$$

where $a \in \mathbf{R}^n$, $b \in \mathbf{R}$. A half-space is a [convex set](#), the boundary of which is a [hyperplane](#).

A half-space separates the whole space in two halves. The complement of the half-space is the open half-space $\{x : a^T x > b\}$.



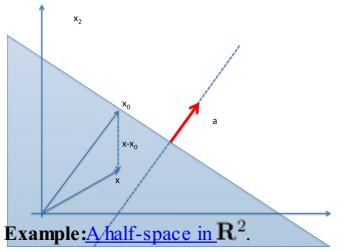
When $b = 0$, the half-space

$$\mathbf{H} = \{x : a^T x \leq 0\},$$

is the set of points which form an obtuse angle (between 90° and 270°) with the vector a . The boundary of this set is a subspace, the hyperplane of vectors orthogonal to a .

When $b \neq 0$, the corresponding half-space can be written as

$$\mathbf{H} = \{x : a^T(x - x_0) \leq 0\},$$



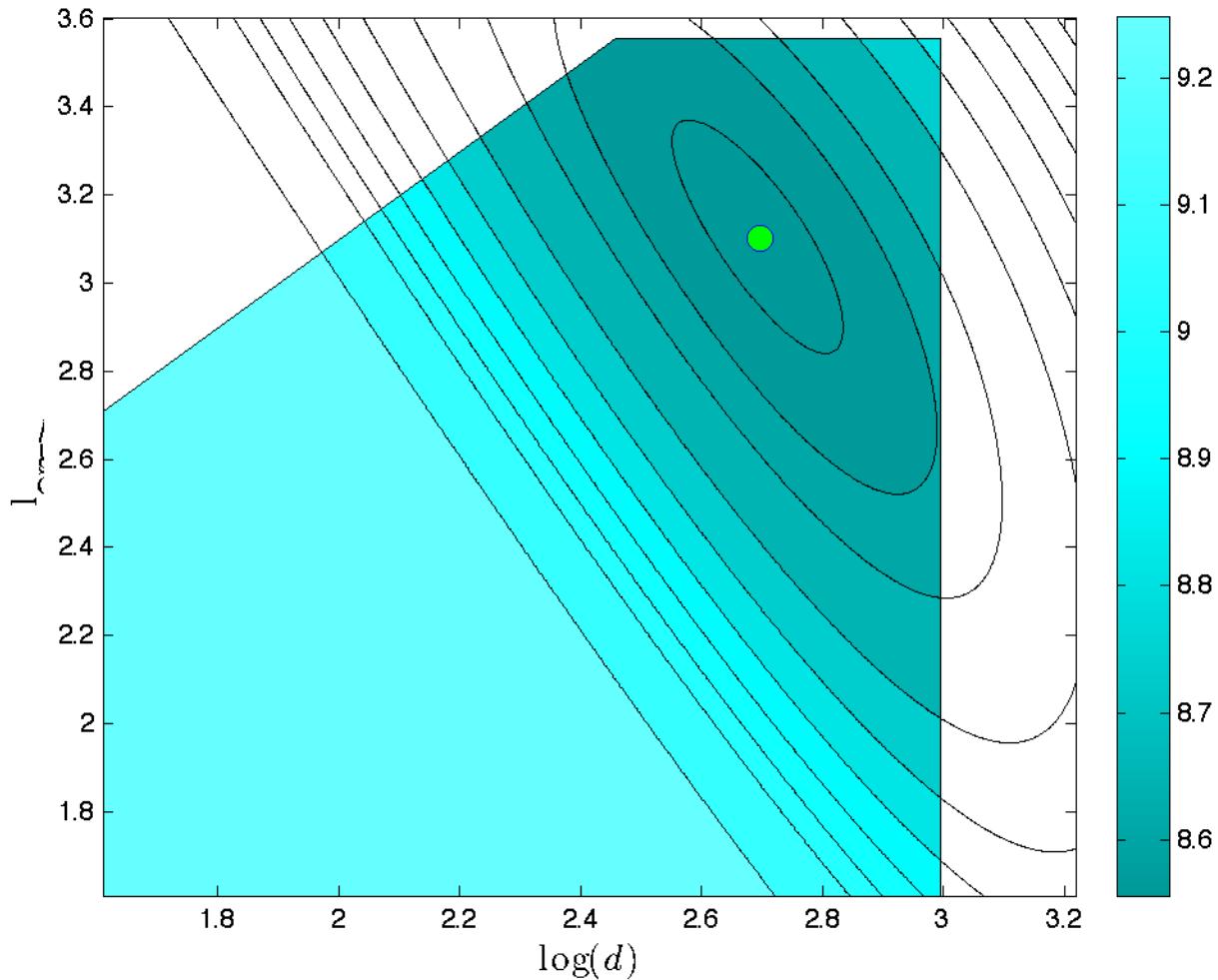
where x_0 is chosen such that $a^T x_0 = b$. For example, $x_0 = b/\|a\|_2^2$ is such a point on the boundary of the half-space (this particular choice corresponds to the minimum-norm solution to the equation $a^T x = b$). Thus, the half-space above corresponds to the set of points such that $x - x_0$ (shown in dotted) forms an obtuse angle with the vector a . The vector a points outwards from the boundary.

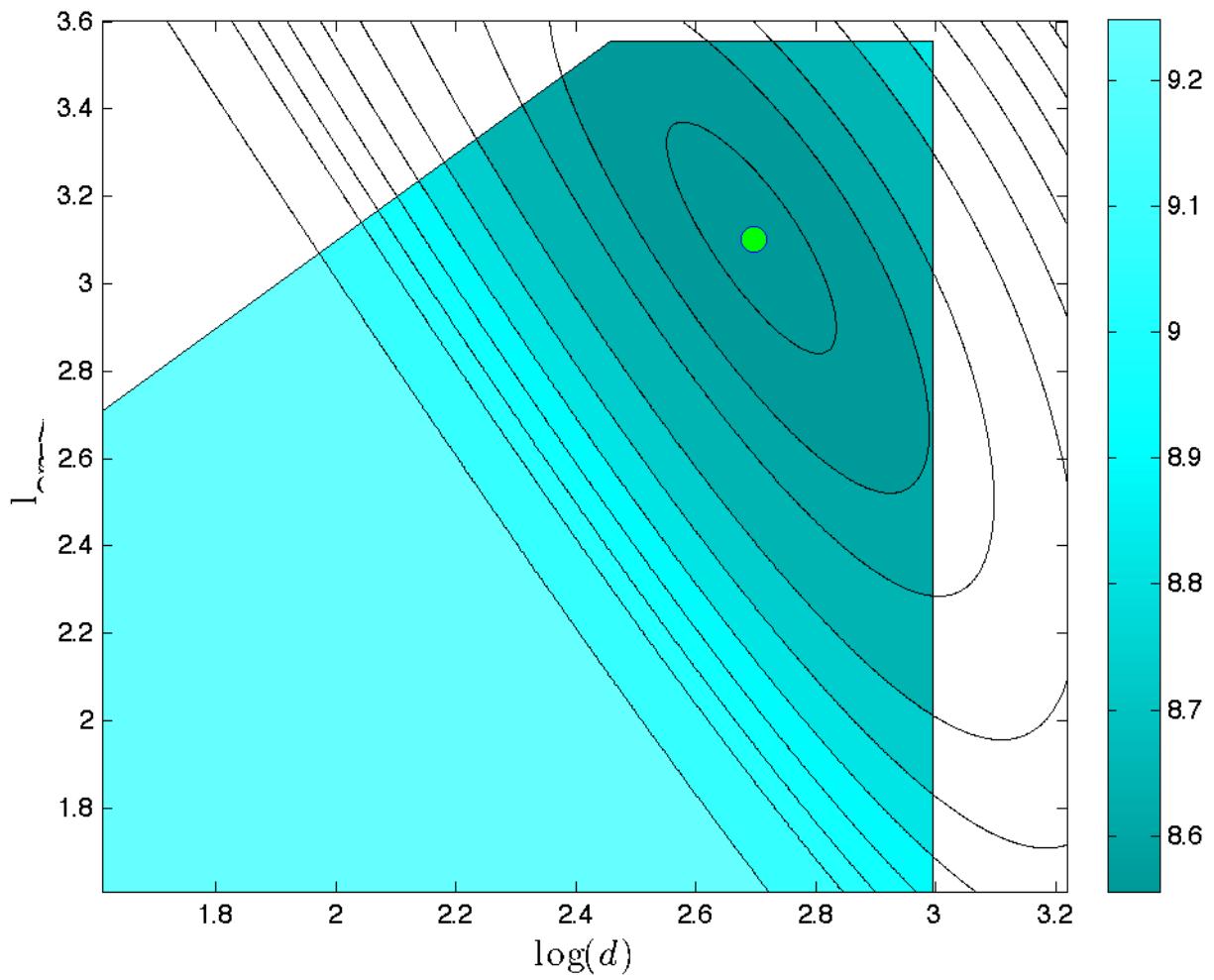
Example: A half-space in \mathbb{R}^2 .

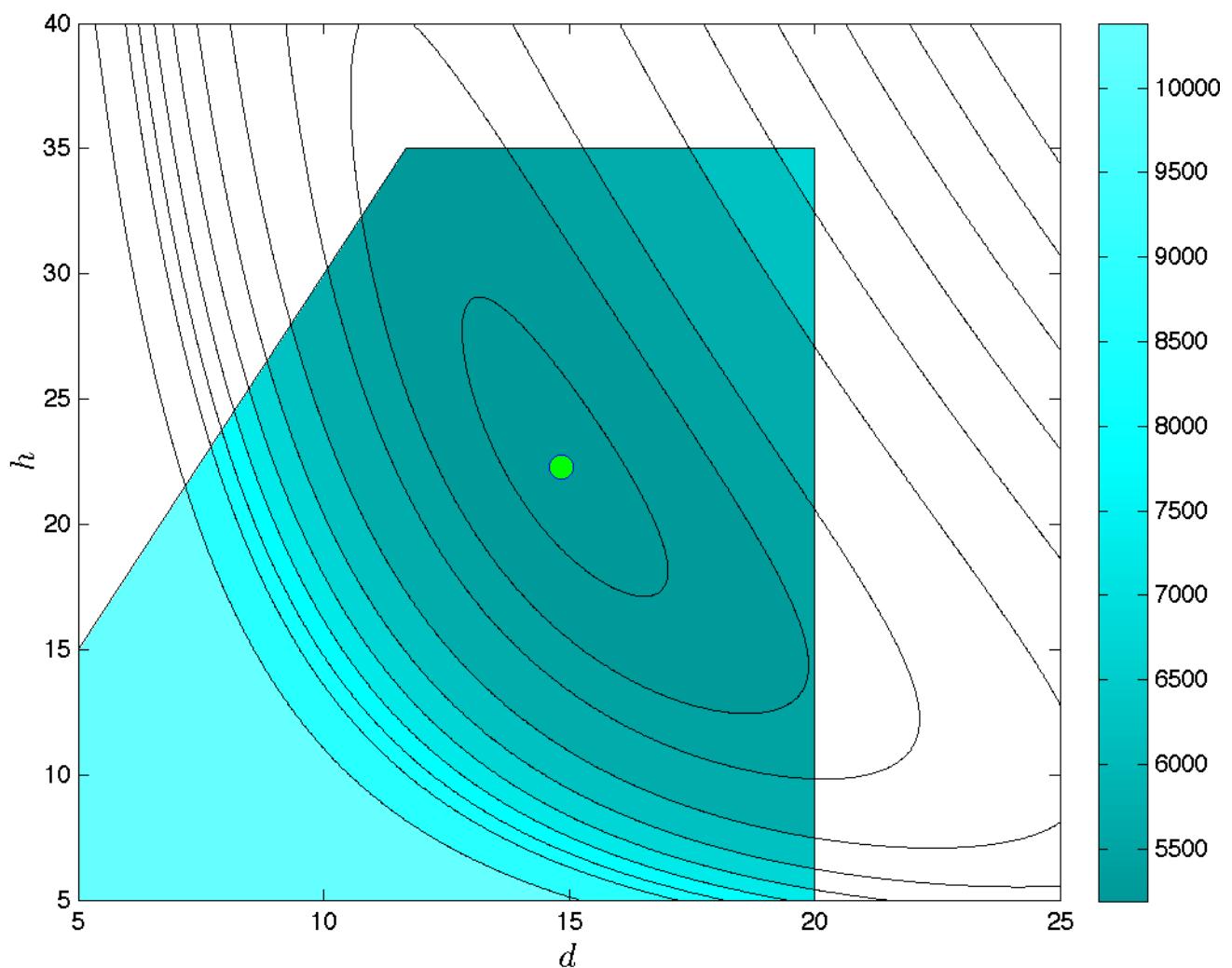
Link with linear functions

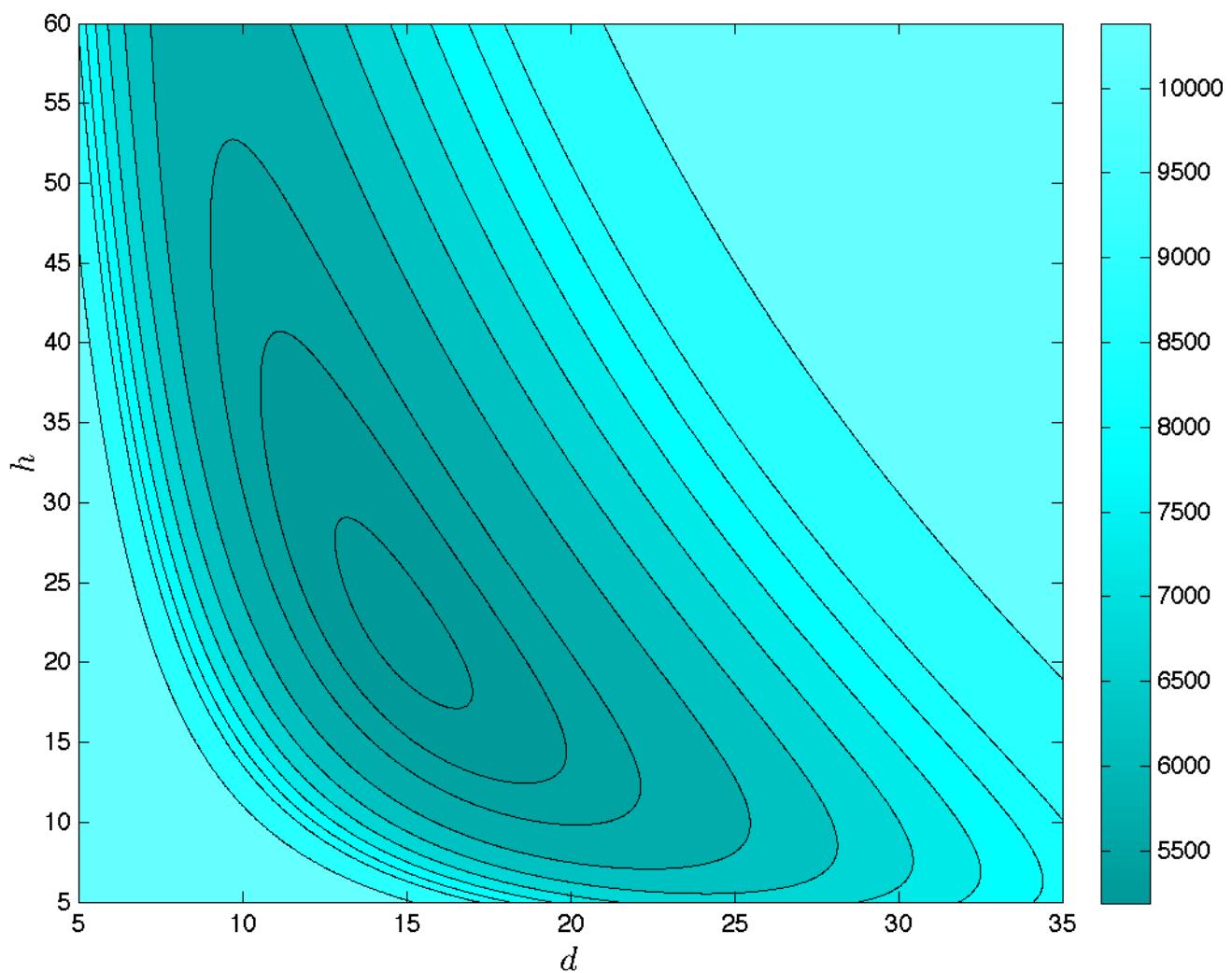
Hyperplanes correspond to [level sets](#) of linear functions.

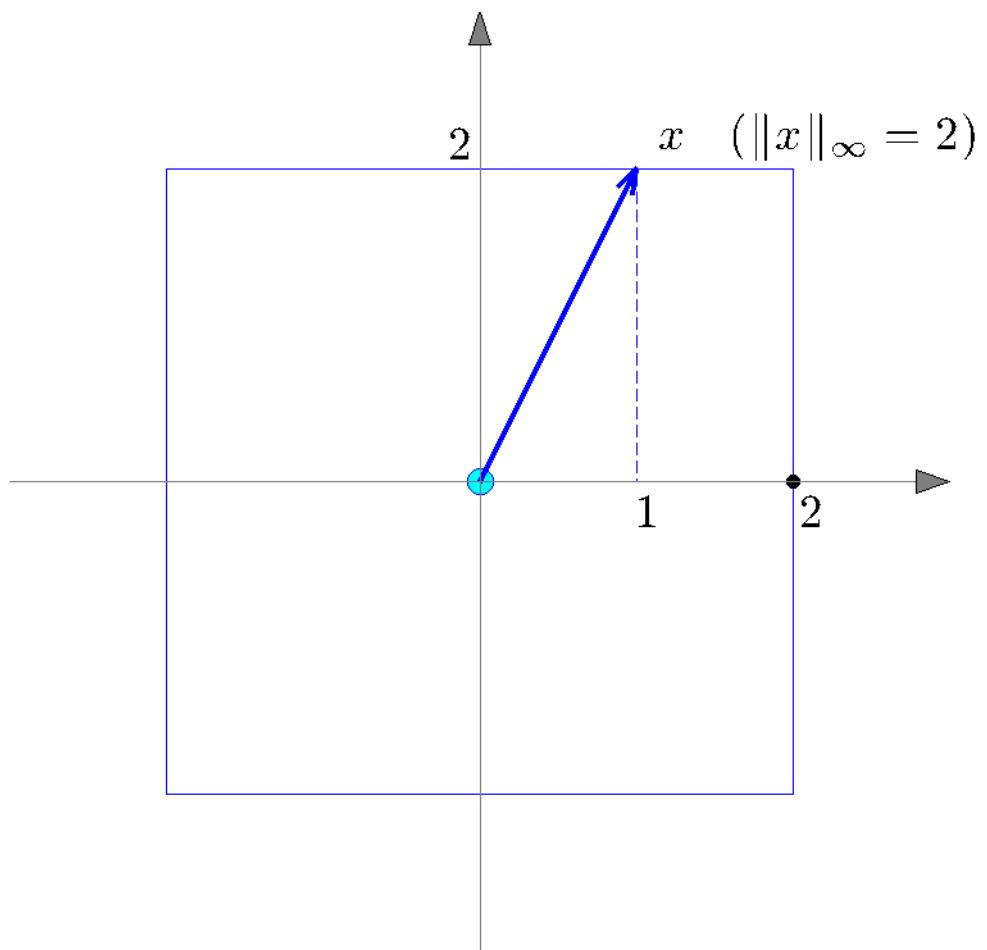
Half-spaces represent [sub-level sets](#) of linear functions: the half-space above describes the set of points such that the linear function $x \rightarrow a^T x$ achieves the value b , or less. A quick way to check which half of the space the half-space describe, is to look at where the origin is: if $b \geq 0$, then $x = 0$ is in the half-space.

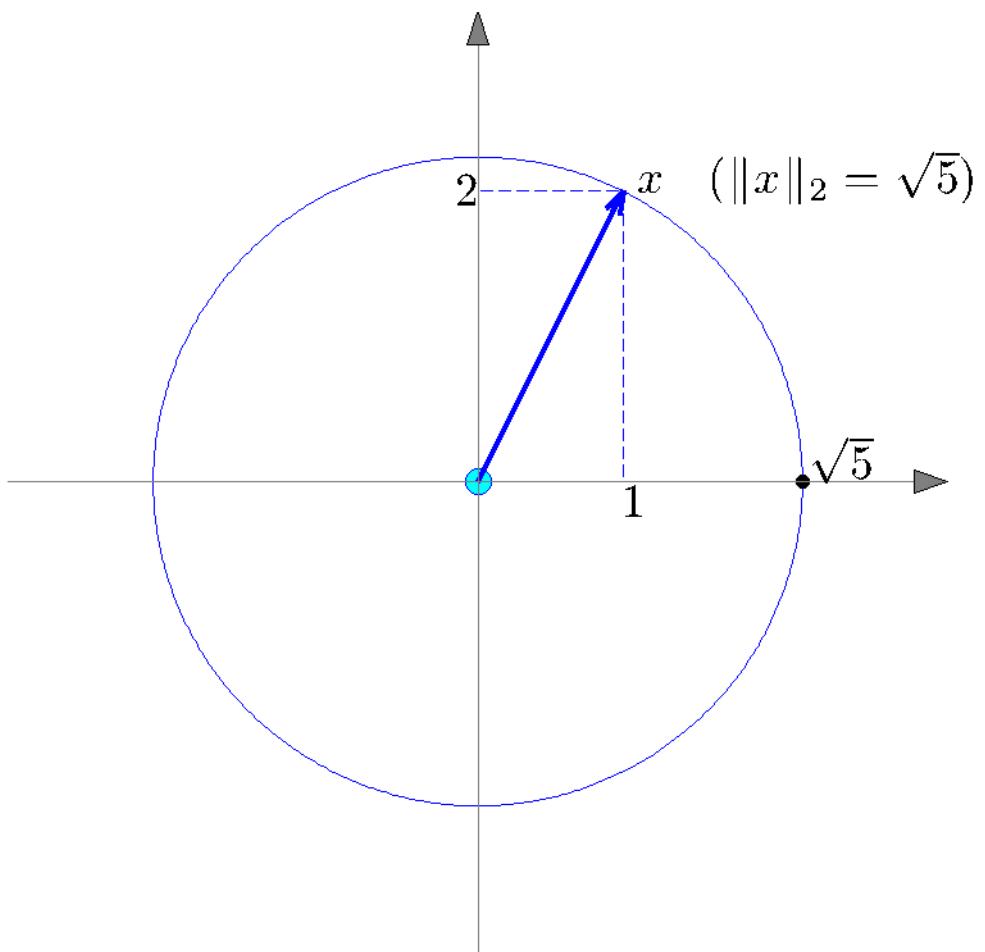


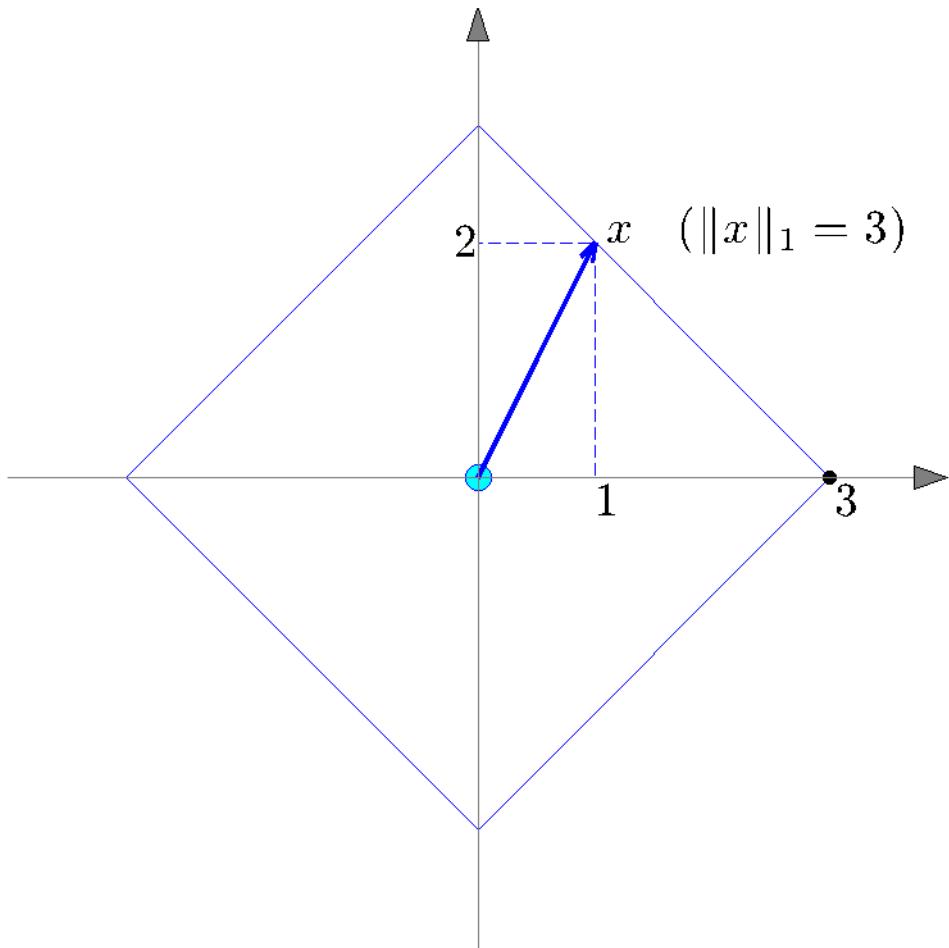












Digital Circuit Optimization via Geometric Programming S. Boyd, S.-J. Kim, D. Patil, and M. Horowitz

Operations Research, 53(6): 899-932, 2005.

- [gp_digital_ckt.pdf](#)

Related material:

- [A Tutorial on Geometric Programming](#)
- [Geometric Programming Applications to EDA Problems \(DATE 2005 tutorial\)](#)
- [A Heuristic for Optimizing Stochastic Activity Networks with Applications to Statistical Digital Circuit Sizing](#)
- [GGPLAB](#), a simple Matlab toolbox for geometric programming

This tutorial paper concerns a method for digital circuit sizing based on formulating the problem as a geometric program (GP), a special type of mathematical optimization problem that can be very efficiently solved. We start with a basic gate scaling problem, with delay modeled as a simple RC time constant, and then add various layers of complexity and modeling accuracy, such as accounting for differing fall and rise times, effects of signal transition times, and so on. We then consider more complex formulations such as design over corners, multi-scenario design, and statistical design. Finally, we consider problems in which threshold and power supply voltage are also variables to be chosen. In all cases our focus is on formulating the problem as a GP, or an extension of GP called generalized geometric programming (GGP).

A 2×2 orthogonal matrix

The matrix $U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ is orthogonal.

The vector $x = (2, 1)$ is transformed by the orthogonal matrix above into $Ux = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 3 \end{pmatrix}$. Thus, U corresponds to a rotation of angle 45° counter-clockwise.

We consider the data matrix containing the votes of the Senators in the 2004-2006 US Senate (2004-2006) introduced [here](#).

Scoring function

We are interested in assigning a “score” to each Senator, and thus represent all the Senators as a single value on a line, using the method described [here](#). We will project the data along a vector in the “bill” space, which is \mathbf{R}^m . That is, we are going to form linear combinations of the bills, so that the $n = 542$ votes for each Senator are reduced to a single number, or “score”.

Our scoring function takes the form $f(x) = u^T(x - \hat{x})$ where, \hat{x} is the m -vector of average votes (across Senators). Here, without loss of generality, the direction $u \in \mathbf{R}^m$ is normalized ($\|u\|_2 = 1$).

Computing the scores

To compute the scores, we first center the data: $X_{\text{cent}} = (x_1 - \hat{x} \dots x_n - \hat{x}) = X - \hat{x}\mathbf{1}_n^T$, where $\mathbf{1}_n$ is the vector of ones in \mathbf{R}^n . Then, we compute the (row) vector of scores: $f = u^T X_{\text{cent}}$.

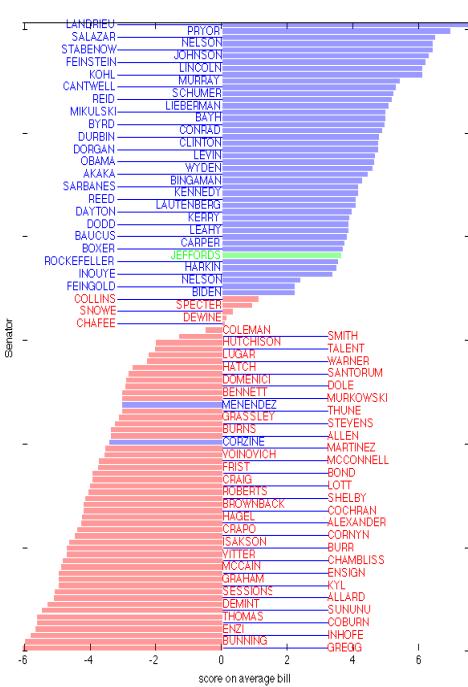
Example: behavior with respect to average bill

$$u = \frac{1}{\sqrt{m}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbf{R}^m.$$

(Our normalization ensures that the Euclidean norm of u is 1, and allows to compare several directions. The scaling factor is actually irrelevant here.)

The direction u corresponds to trying to understand the voting behaviors of the Senators in terms a single, synthetic, “average” bill. The scores we get this way correspond to an “extremism/conformism index”, since Senators with a low score tend to vote along the average (on the “average” bill), while those with a high score would tend to vote opposite to it.

With that choice, here is what we get:



Average vs. extreme scores on average bill: This image shows the values of the projections of the Senator’s votes $x_j - \hat{x}$ (that is, with average across Senators removed) on the (normalized) “average bill” direction defined above.

The party affiliation of each Senator is shown, with names of Democrats in blue and those of Republicans in red. Based on this crude analysis, we could be led to conclude that the former tend to vote more according to the average, while those of the latter tend to vote less according to the average.

Network flow

We describe a *flow* (of goods, traffic, charge, information, etc) across the network as a vector $x \in \mathbf{R}^n$, which describes the amount flowing through any given arc. By convention, we use positive values when the flow is in the direction of the arc, and negative ones in the opposite case. The total flow leaving a given node i is then

$$\sum_{j=1}^n A_{ij} x_j = (Ax)_i,$$

(remember our convention that the index j spans the arcs) with $(Ax)_i$ our notation for the i -th component of vector Ax . Now we define the *external supply* as a vector $b \in \mathbf{R}^m$, with negative b_i representing an external demand at node i , and positive b_i a supply. We assume that the total supply equals the total demand, which means $\mathbf{1}^T b = 0$.

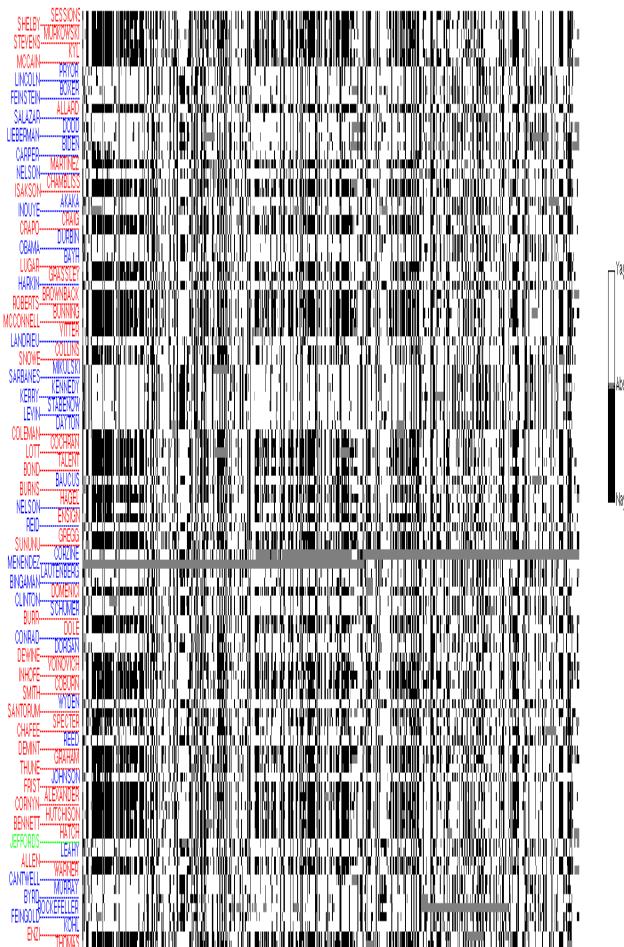
The *balance equations* for the supply vector b are $Ax = b$. These equations represent constraints the flow vector must satisfy in order to satisfy the external supply/demand represented by b .

See also:[Incidence matrix of a network](#)

Senate Voting Data Matrix

The data consists of the votes of $n = 100$ Senators in the 2004-2006 US Senate (2004-2006), for a total of $m = 542$ bills. “Yay” (“Yes”) votes are represented as 1’s, “Nay” (“No”) as -1’s, and the other votes are recorded as 0. (A number of complexities are ignored here, such as the possibility of pairing the votes.)

The data can be represented here as a $m \times n$ “voting” matrix $X = [x_1, \dots, x_n]$, with elements taken from $\{-1, 0, 1\}$. Each column of the voting matrix x_j , $j = 1, \dots, n$ contains the votes of a single Senator for all the bills; each row contains the votes of all Senators on a particular bill.



Senate voting matrix: “Nay” votes are in black, “Yay” ones in white, and the others in grey. The transpose voting matrix is shown. The picture becomes has many gray areas, as some Senators are replaced over time. Simply plotting the raw data matrix is often not very informative.

Source: [VoteWorld](#).

A simple 3×2 matrix

$$A = \begin{pmatrix} 3 & 4.5 \\ 2 & 1.2 \\ -0.1 & 8.2 \end{pmatrix}.$$

Consider the 3×2 matrix A . The matrix can be interpreted as the collection of two column vectors: $A = (a_1, a_2)$, where a_j 's contain the columns of A :

$$a_1 = \begin{pmatrix} 3 \\ 2 \\ -0.1 \end{pmatrix}, \quad a_2 = \begin{pmatrix} 4.5 \\ 1.2 \\ 8.2 \end{pmatrix},$$

Geometrically, A represents 2 points in a 3-dimensional space.

$$A = \begin{pmatrix} b_1^T \\ b_2^T \\ b_3^T \end{pmatrix},$$

Alternatively, we can interpret A as a collection of 3 row vectors in \mathbf{R}^2 :

$$b_1 = \begin{pmatrix} 3 \\ 4.5 \end{pmatrix}, \quad b_2 = \begin{pmatrix} 2 \\ 1.2 \end{pmatrix}, \quad b_3 = \begin{pmatrix} -0.1 \\ 8.2 \end{pmatrix}.$$

Geometrically, A represents 3 points in a 2-dimensional space.

Dimension of an affine subspace

The set in \mathbf{R}^3 defined by the linear equations $x_1 - 13x_2 + 4x_3 = 2$, $3x_2 - x_3 = 9$ is an affine subspace of dimension 1. The corresponding linear subspace is defined by the linear equations obtained from the above by setting the constant terms to zero: $x_1 - 13x_2 + 4x_3 = 0$, $3x_2 - x_3 = 0$. We can solve for x_3 and get

$$x = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} x_2.$$

$x_1 = x_2$, $x_3 = 3x_2$. We obtain a representation of the linear subspace as the set of vectors $x \in \mathbf{R}^3$ such that the vector $(1, 1, 3)$, and is of dimension 1. Hence the linear subspace is the span of

Basis in \mathbf{R}^3

$$x_1 := \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}, \quad x_3 = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix},$$

The set of three vectors in \mathbf{R}^3 : x_1, x_2, x_3 is not independent, since $x_1 - x_2 + x_3 = 0$, and its span has dimension 2. Since x_1, x_2 are independent (the equation $\lambda_1 x_1 + \lambda_2 x_2 = 0$ has $\lambda = 0$ as the unique solution), a basis for that span is, for example, $\{x_1, x_2\}$. In contrast, the collection $\{x_1, x_2, x_3 - e_1\}$ spans the whole space \mathbf{R}^3 , and thus forms a basis of that space.

Representing temperatures at different airports as a vector

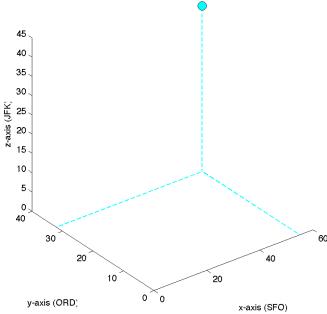
We record the temperatures at three different airports, and obtain the following table.

airport temperature ($^{\circ}\text{F}$)

SFO	55
ORD	32
JFK	43

We can represent the temperatures on a single temperature axis. This is known as the *dot* representation of a vector.

The dot representation could become very confusing if there were three temperatures to plot for each day in the year.



Alternatively, we can view the triple of temperatures as a *point* in a three-dimensional space. Each axis corresponds to temperatures at a specific location. The vector representation is still legible if we have more than one triplet of temperatures to represent. The vector representation cannot be viewed in more than three dimensions, that is, if we have more than three cities involved. However, the data can still be represented as a collection of vectors.

Sample variance and standard deviation

The *sample variance* of given numbers x_1, \dots, x_n , is defined as $\sigma^2 := \frac{1}{n} ((x_1 - \hat{x})^2 + \dots + (x_n - \hat{x})^2)$, where \hat{x} is the sample average of x_1, \dots, x_n . The sample variance is a measure of the deviations of the numbers x_i with respect to the average value \hat{x} .

The *sample standard deviation* is the square root of the sample variance, σ^2 . It can be expressed in terms of the Euclidean norm of the vector $x = (x_1, \dots, x_n)$, as $\sigma = \frac{1}{\sqrt{n}} \|x - \hat{x}\mathbf{1}\|_2$, where $\|\cdot\|_2$ denotes the Euclidean norm.

More generally, for any vector $p \in \mathbf{R}^n$, with $p_i \geq 0$ for every i , and $p_1 + \dots + p_n = 1$, we can define the corresponding *weighted variance* as $\sum_{i=1}^n p_i(x_i - \hat{x})^2$. The interpretation of p is in terms of a discrete probability distribution of a random variable X , which takes the value x_i with probability p_i , $i = 1, \dots, n$. The weighted variance is then simply the *expected value* of the squared deviation of X from its mean $\mathbf{E}X$, under the probability distribution P .

See also: [sample and weighted average](#).

Sample covariance matrix

The average of N given real numbers $x^{(1)}, \dots, x^{(N)}$ is the quantity $\hat{x} := (1/N)(x^{(1)} + \dots + x^{(N)})$. Their variance is the non-negative number $\sigma_x^2 := (1/N) \sum_{k=1}^N (x^{(k)} - \hat{x})^2$, and is a measure of the variability of the numbers around their average; in particular, the variance is zero if and only if all the numbers are equal.

Now consider two series of real numbers, $x^{(k)}, y^{(k)}, k = 1, \dots, N$. We can define the means \hat{x}, \hat{y} and variances σ_x^2, σ_y^2 of these numbers as before. In addition, we can define the covariance between these two collections of numbers as the value

$$\sigma_{xy}^2 := (1/N) \sum_{k=1}^N (x^{(k)} - \hat{x})(y^{(k)} - \hat{y}).$$

The covariance is a measure of closeness between the two series of numbers: if the covariance is large and positive, then in average, when $x^{(k)}$'s are far above or below their mean, then so is $y^{(k)}$; the series tend to move in the same direction simultaneously. If the covariance is large and negative, then the two series tend to move in opposite directions. If the covariance is zero, then

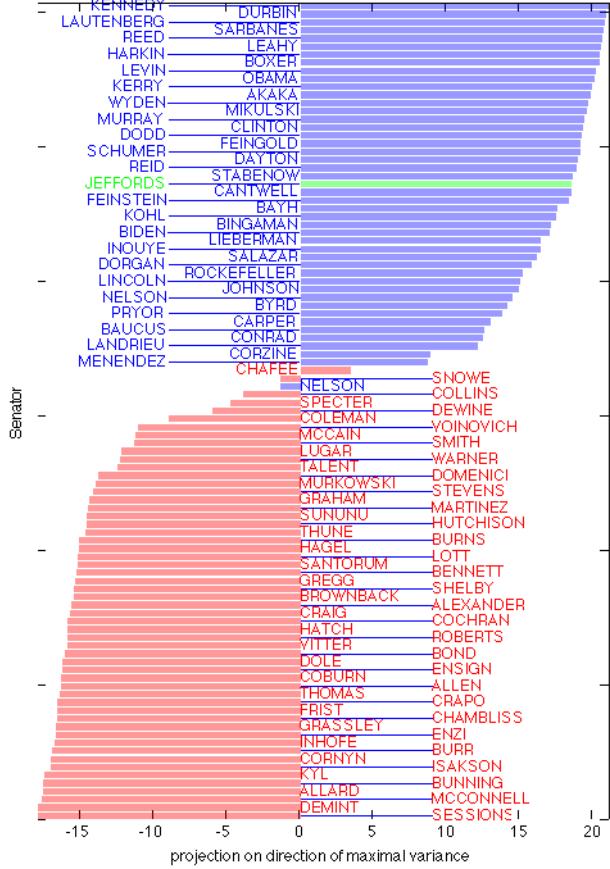
$$\Sigma_{xy} := \begin{pmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_y^2 \end{pmatrix}$$

the excursions of the x -series above or below its mean tend to compensate those of the y -series. The symmetric matrix

Consider now a collection of vectors $x^{(k)} \in \mathbf{R}^n$, $k = 1, \dots, N$. We can define, as before, the average as the vector $\hat{x} = (1/N) \sum_{i=1}^N x^{(i)}$. We then define the covariance matrix as the $n \times n$ matrix Σ formed with the covariance between the series $x_i^{(k)}, y_j^{(k)}$, $k = 1, \dots, N$, for $i, j = 1, \dots, n$. More compactly:

$$\Sigma := \frac{1}{N} \sum_{k=1}^N (x^{(k)} - \hat{x})(x^{(k)} - \hat{x})^T.$$

A diagonal element of Σ , Σ_{ii} , captures the individual variance of the i -th component of the vectors, which is the series $x_i^{(k)}$, $k = 1, \dots, N$. The covariance matrix is symmetric by construction. In addition, it is positive semi-definite, since it is a non-negative sum of symmetric dyads.



Sample and weighted average, expected value

$$\hat{x} := \frac{1}{n}(x_1 + \dots + x_n).$$

The *sample average* of given numbers x_1, \dots, x_n , is defined as

The sample average can be interpreted as a scalar product: $\hat{x} = p^T x$, where $x = (x_1, \dots, x_n)$ is the vector containing the samples, and $p = (1/n)\mathbf{1}$, with $\mathbf{1}$ the vector of ones.

More generally, for any vector $p \in \mathbf{R}^n$, with $p_i \geq 0$ for every i , and $p_1 + \dots + p_n = 1$, we can define the corresponding *weighted average* as $p^T x$. The interpretation of p is in terms of a discrete probability distribution of a random variable X , which takes the value x_i with probability p_i , $i = 1, \dots, n$. The weighted average is then simply the *expected value* (or, mean) of X under the probability distribution p . The expected value is often denoted $E_p(X)$, or $E(X)$ if the distribution p is clear from context.

Visualization of High-Dimensional Data [Vectors, Matrices > Applications > Visualization](#)

- Visualization problem
- Projecting data on a line
- Projecting data on a plane

Visualization Problem

Consider a data set of n points x_j , $j = 1, \dots, n$ in \mathbf{R}^m . Each point can represent the a chemical experiment under m specific conditions; the response of a specific gene to a number of m different drugs; the votes of a particular citizen on an array of m issues; m atmospheric readings (temperature, pressure, humidity, etc) at a specific location; m past prices of a single asset; etc.

We can represent this data set as a $m \times n$ matrix $X = [x_1, \dots, x_n]$, where each x_j is a n -vector. Simply plotting the raw matrix is often not very informative.

Example: Raw data matrix for the US Senate, 2004-2006.

We can try to visualize the data set, by projecting each data point (each row or column of the matrix) on (say) a 1D-, 2D- or 3D-space. Each “view” corresponds to a particular projection, that is, a particular one-, two- or three-dimensional subspace on which we choose to project the data. The *visualization problem* consists of choosing an appropriate projection.

There are many ways to formulate the visualization problem, and none dominates the others. Here, we focus on the basics of that problem.

Projecting on a line

To simplify, let us first consider the simple problem of representing the high-dimensional data set on a simple line, using the method described [here](#).

Specifically we would like to assign a single number, or “score”, to each column of the matrix. We choose a direction $u \in \mathbf{R}^m$, and a scalar $v \in \mathbf{R}$. This corresponds to the affine “scoring” function $f : \mathbf{R}^m \rightarrow \mathbf{R}$, which, to a generic column $x \in \mathbf{R}^m$ of the data matrix, assigns the value $f(x) = u^T x + v$. We thus obtain a vector of values $f \in \mathbf{R}^n$, with $f_j = u^T x_j + v$, $j = 1, \dots, n$. It is often useful to center these values around zero. This can be done by choosing v such that

$$0 = \sum_{j=1}^n (u^T x_j + v) = u^T \left(\sum_{j=1}^n x_j \right) + n \cdot v, \quad \hat{x} := \frac{1}{n} \sum_{j=1}^n x_j \in \mathbf{R}^m$$

that is: $v = -u^T \hat{x}$, where \hat{x} is the vector of [sample averages](#) across the columns of the matrix (that is, data points). The vector \hat{x} can be interpreted as the “average response” across experiments.

The values of our scoring function can now be expressed as $f(x) = u^T (x - \hat{x})$. In order to be able to compare the relative merits of different directions, we can assume, without loss of generality, that the vector u is normalized (so that $\|u\|_2 = 1$).

It is convenient to work with the “centered” data matrix, which is $X_{\text{cent}} = (x_1 - \hat{x} \ \dots \ x_n - \hat{x}) = X - \hat{x} \mathbf{1}_n^T$, where $\mathbf{1}_n$ is the vector of ones in \mathbf{R}^n .

In matlab, we can compute the centered data matrix as follows.

Matlab syntax

```
>> xhat = mean(X, 2);  
>> [m, n] = size(X);  
>> Xcent = X - xhat * ones(1, n);
```

We can compute the (row) vector scores using the simple [matrix-vector product](#): $f = u^T X_{\text{cent}} \in \mathbf{R}^{1 \times m}$. We can check that the average of the above row vector is zero: $f \mathbf{1}_n = u^T X_{\text{cent}} \mathbf{1}_n = u^T (X - \hat{x} \mathbf{1}_n^T) \mathbf{1}_n = u^T (X \mathbf{1}_n - n \cdot \hat{x}) = 0$.

Example: Senator scores on average bill.

Projection on a plane

We can also try to project the data on a plane, which involves assigning *two* scores to each data point.

This corresponds to the affine “scoring” map $f : \mathbf{R}^m \rightarrow \mathbf{R}$, which, to a generic column $x \in \mathbf{R}^m$ of the data matrix, assigns the two-dimensional value $f(x) = \begin{pmatrix} u_1^T x + v_1 \\ u_2^T x + v_2 \end{pmatrix} = U^T x + v$, where $u_1, u_2 \in \mathbf{R}^m$ are two vectors, and v_1, v_2 two scalars, while $U = [u_1, u_2] \in \mathbf{R}^{m \times 2}$, $v \in \mathbf{R}^2$.

The affine map f allows to generate n two-dimensional data points (instead of m -dimensional) $f_j = U^T x_j + v$, $j = 1, \dots, n$. As before, we can require that the f_j ’s be centered:

$0 = \sum_{j=1}^n f_j = \sum_{j=1}^n (U^T x_j + v)$, by choosing the vector v to be such that $v = -U^T \hat{x}$, where $\hat{x} \in \mathbf{R}^m$ is the “average response” defined above. Our (centered) scoring map takes the form $f(x) = U^T (x - \hat{x})$.

We can encapsulate the scores in the $2 \times n$ matrix $F = [f_1, \dots, f_n]$. The latter can be expressed as the matrix-matrix product $F = U^T X_{\text{cent}} = \begin{pmatrix} u_1^T X_{\text{cent}} \\ u_2^T X_{\text{cent}} \end{pmatrix}$, with X_{cent} the centered data matrix defined above.

Example: Visualizing Senate voting on a plane.

Robust Principal Component Analysis [SDP](#) > [Conic problems](#) | [LMIs](#) | [Standard Forms](#) | [Applications](#) > [Back](#) | Robust PCA

Sparse Principal Component Analysis [SDP](#) > [Conic problems](#) | [LMIs](#) | [Standard Forms](#) | [Applications](#) > [Back](#) | SPCA

Standard Forms and Geometry

[Up](#) | [Next](#)

Linear Programs

A *linear program* (or LP, for short) is an optimization problem in the standard form introduced [here](#): $\min_x f_0(x) : f_i(x) \leq 0$, $i = 1, \dots, m$, where the objective function f_0 , and the constraint functions f_i , $i = 1, \dots, m$, are all [affine](#).

Since affine functions are linear plus constant, and constant terms in the objective do not matter, we can always assume that f_0 is actually linear.

Examples:

- [An LP in two variables](#).
- [A drug production problem](#).

Quadratic programs

A *quadratic program* (or QP, for short) is an optimization problem in the standard form above, where:

- the constraint functions $f_i, i = 1, \dots, m$, are all [affine](#), as in LP;
- the objective function f_0 is quadratic convex, that is, its values can be expressed as $f_0(x) = c^T x + x^T Qx$ for some vector $c \in \mathbf{R}^n$ and $Q = Q^T \succeq 0$ (Q is symmetric, and all of its eigenvalues are non-negative).

Geometry

Each constraint in an LP is a single affine inequality in the decision vector x . Hence, each constraint says that the decision vector x should lie in a [half-space](#). Taken together, the constraints require that x should lie in the intersection of those half-spaces, that is, a [polyhedron](#).

The geometry of LP is to minimize a linear function over a polyhedron. The geometry of QP is to minimize a convex (bowl-shaped) quadratic function over a polyhedron.

Examples:

- [A quadratic program in two variables](#).

Visualization of High-Dimensional Data

[Matrices](#) > [Applications](#) > Visualization

- Visualization problem
- Projecting data on a line
- Projecting data on a plane

Visualization Problem

Consider a data set of n points $x_j, j = 1, \dots, n$ in \mathbf{R}^m . Each point can represent the a chemical experiment under m specific conditions; the response of a specific gene to a number of m different drugs; the votes of a particular citizen on an array of m issues; m atmospheric readings (temperature, pressure, humidity, etc) at a specific location; m past prices of a single asset; etc.

We can represent this data set as a $m \times n$ matrix $X = [x_1, \dots, x_n]$, where each x_j is a n -vector. Simply plotting the raw matrix is often not very informative.

Example: [Raw data matrix for the US Senate, 2004-2006](#).

We can try to visualize the data set, by projecting each data point (each row or column of the matrix) on (say) a 1D-, 2D- or 3D-space. Each “view” corresponds to a particular projection, that is, a particular one-, two- or three-dimensional subspace on which we choose to project the data. The *visualization problem* consists of choosing an appropriate projection.

There are many ways to formulate the visualization problem, and none dominates the others. Here, we focus on the basics of that problem.

Projecting on a line

To simplify, let us first consider the simple problem of representing the high-dimensional data set on a simple line, using the method described [here](#).

Specifically we would like to assign a single number, or “score”, to each column of the matrix. We choose a direction $u \in \mathbf{R}^m$, and a scalar $v \in \mathbf{R}$. This corresponds to the affine “scoring” function $f : \mathbf{R}^m \rightarrow \mathbf{R}$, which, to a generic column $x \in \mathbf{R}^m$ of the data matrix, assigns the value

$$f(x) = u^T x + v.$$

We thus obtain a vector of values $f \in \mathbf{R}^n$, with $f_j = u^T x_j + v, j = 1, \dots, n$. It is often useful to center these values around zero. This can be done by choosing v such that

$$0 = \sum_{j=1}^n (u^T x_j + v) = u^T \left(\sum_{j=1}^n x_j \right) + n \cdot v,$$

that is: $v = -u^T \hat{x}$, where

$$\hat{x} := \frac{1}{n} \sum_{j=1}^n x_j \in \mathbf{R}^m$$

is the vector of [sample averages](#) across the columns of the matrix (that is, data points). The vector \hat{x} can be interpreted as the “average response” across experiments.

The values of our scoring function can now be expressed as

$$f(x) = u^T (x - \hat{x}).$$

In order to be able to compare the relative merits of different directions, we can assume, without loss of generality, that the vector u is normalized (so that $\|u\|_2 = 1$).

It is convenient to work with the “centered” data matrix, which is

$$X_{\text{cent}} = (x_1 - \hat{x} \ \dots \ x_n - \hat{x}) = X - \hat{x} \mathbf{1}_n^T,$$

where $\mathbf{1}_n$ is the vector of ones in \mathbf{R}^n .

In Matlab, we can compute the centered data matrix as follows.

Matlab syntax

```
>> xhat = mean(X, 2);
>> [m, n] = size(X);
```

```
>> Xcent = X-xhat*ones(1,n);
```

We can compute the (row) vector scores using the simple [matrix-vector product](#):

$$f = u^T X_{\text{cent}} \in \mathbf{R}^{1 \times m}.$$

We can check that the average of the above row vector is zero:

$$f\mathbf{1}_n = u^T X_{\text{cent}} \mathbf{1}_n = u^T (X - \hat{x}\mathbf{1}_n^T) \mathbf{1}_n = u^T (X\mathbf{1}_n - n \cdot \hat{x}) = 0.$$

Example: [Senator scores on average bill](#).

Projection on a plane

We can also try to project the data on a plane, which involves assigning *two* scores to each data point.

This corresponds to the affine “scoring” map $f : \mathbf{R}^m \rightarrow \mathbf{R}$, which, to a generic column $x \in \mathbf{R}^m$ of the data matrix, assigns the two-dimensional value

$$f(x) = \begin{pmatrix} u_1^T x + v_1 \\ u_2^T x + v_2 \end{pmatrix} = U^T x + v,$$

where $u_1, u_2 \in \mathbf{R}^m$ are two vectors, and v_1, v_2 two scalars, while $U = [u_1, u_2] \in \mathbf{R}^{m \times 2}$, $v \in \mathbf{R}^2$.

The affine map f allows to generate n *two-dimensional* data points (instead of m -dimensional) $f_j = U^T x_j + v$, $j = 1, \dots, n$. As before, we can require that the f_j ’s be centered:

$$0 = \sum_{j=1}^n f_j = \sum_{j=1}^n (U^T x_j + v),$$

by choosing the vector v to be such that $v = -U^T \hat{x}$, where $\hat{x} \in \mathbf{R}^m$ is the “average response” defined above. Our (centered) scoring map takes the form

$$f(x) = U^T (x - \hat{x}).$$

We can encapsulate the scores in the $2 \times n$ matrix $F = [f_1, \dots, f_n]$. The latter can be expressed as the matrix-matrix product

$$F = U^T X_{\text{cent}} = \begin{pmatrix} u_1^T X_{\text{cent}} \\ u_2^T X_{\text{cent}} \end{pmatrix},$$

with X_{cent} the centered data matrix defined above.

Example: [Visualizing Senate voting on a plane](#).

Digital Circuit Design via GP [GP](#) > [Applications](#) > Circuit Design | [Next](#)

We consider the problem of designing a kind of digital circuit known as a *combinational logic block*. In such a circuit, the basic building block is a *gate*: a circuit that implements some simple boolean function. A gate could be for example an *inverter*, which performs logical inversion, or a more complicated function such as a “not-AND”, or NAND, which takes two boolean inputs A, B to produce the inversion of (A and B).

The basic idea is to design the gates in such a way that the circuit operates fast, yet occupies a small area. There are other factors, such as power, involved in the design problem but we will not discuss this here.

The design variables are scale factors that determine the size of each gate, as well as its basic electrical parameters. Together with the (here, fixed) topology of the circuit, these parameters in turn influence the speed of the circuit.

- [Digital circuit design problem](#)
- [GP representation](#)
- [Example](#)
- [Notes and references](#)

Digital Circuit Design: GP Representation [GP](#) > [Standard Forms](#) | [Applications](#) > [Circuit Design](#) > [Back](#) | GP Form | [Next](#)

- Generalized posynomial representation
- Explicit posynomial representation

Generalized Posynomial Representation

We now consider the problem of choosing the scale factors to minimize the total delay, subject to an area constraint.
 $\min_x D(x) : A_i(x) \leq A_{\max}, \quad x_i \geq 1, \quad i = 1, \dots, n$, where A_{\max} is an upper bound on the area of each gate.

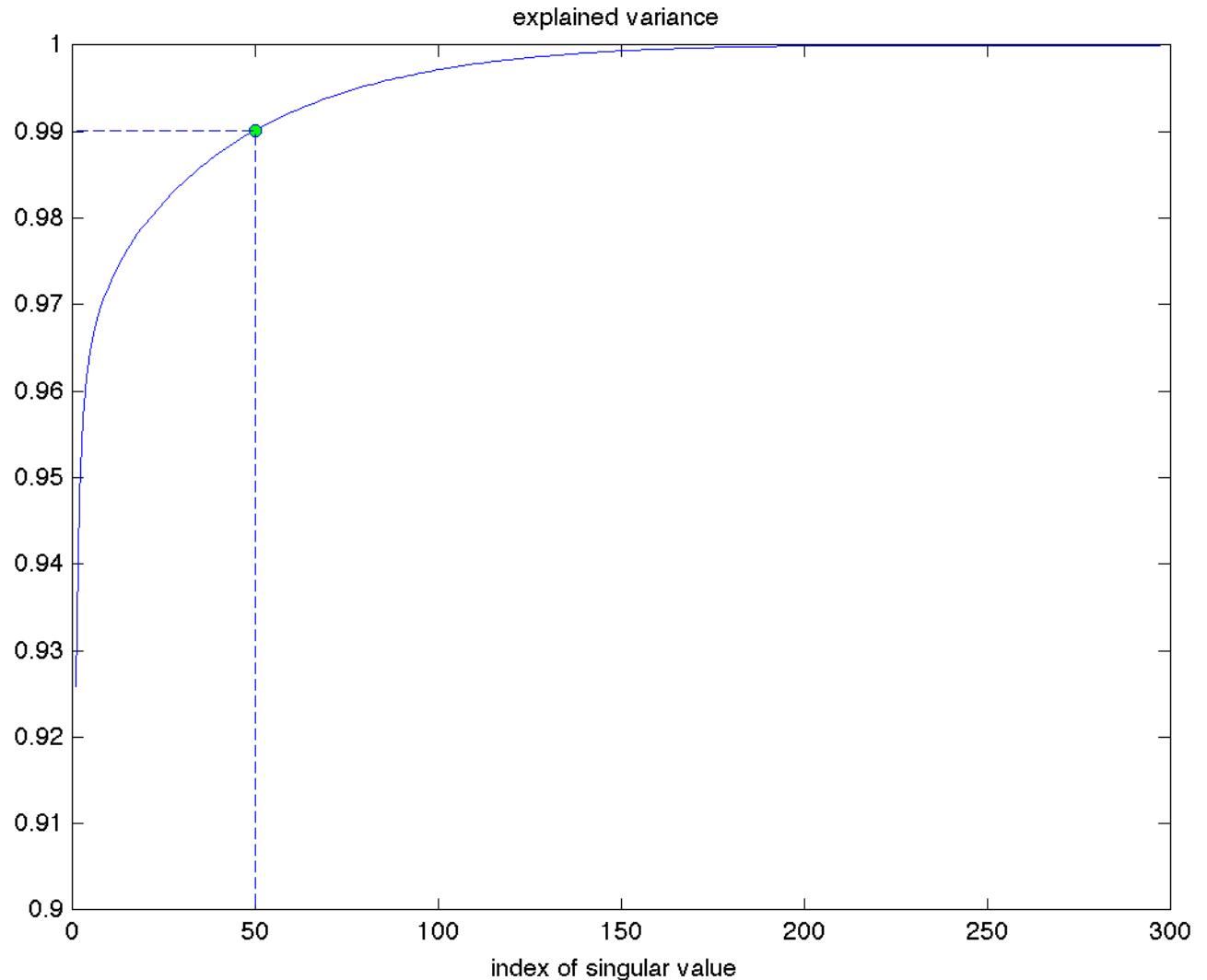
Since T_i is a maximum of function of D_i , itself a posynomial in x , we can express the total delay as a posynomial in x . Hence the above is a GP.

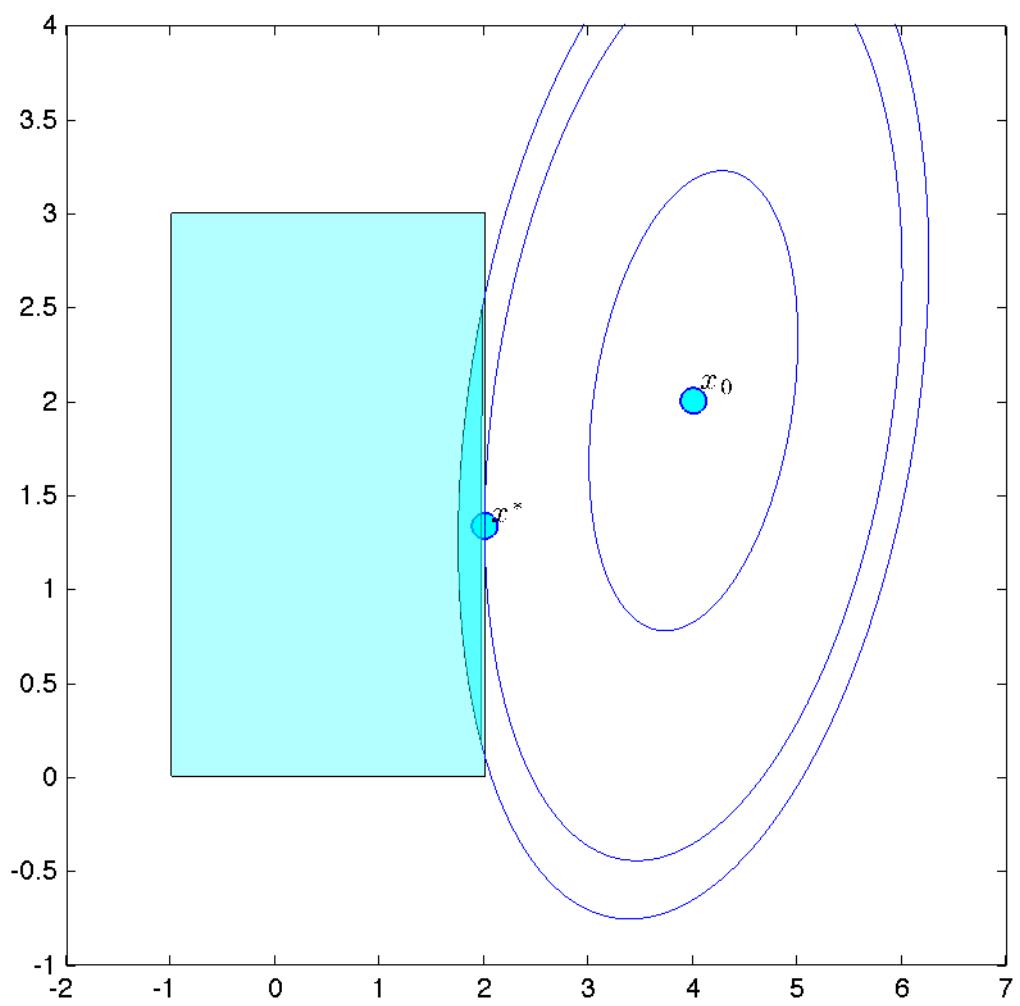
Explicit Posynomial Representation

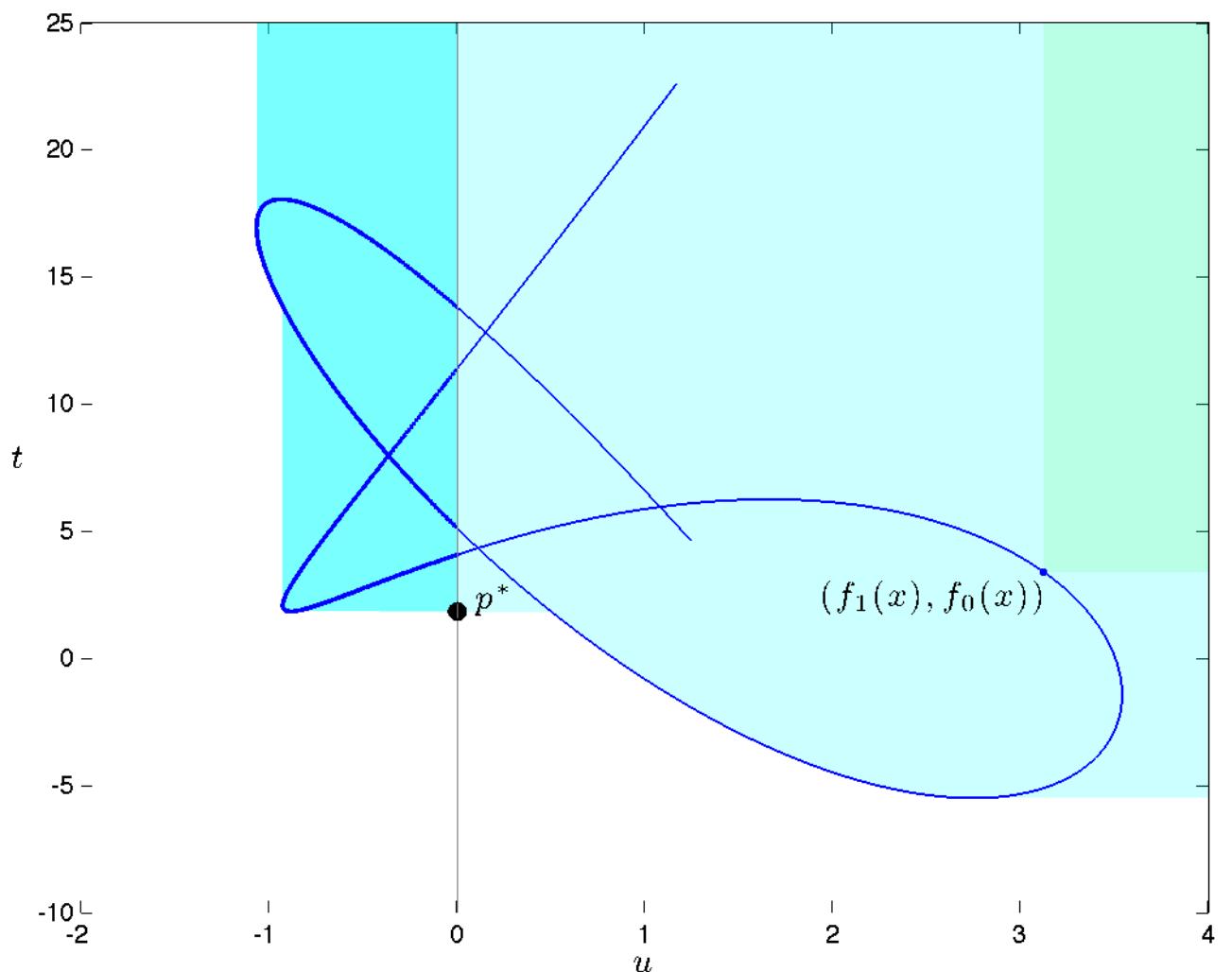
Let us form the GP in a more explicit way, using the intermediate variables T_i , which we encountered in our definition of the delay. The basic idea is to replace the equality defining these intermediate variables, e.g. $T_4 = \max(T_1, T_2) + D_4$ by inequalities: $T_4 \geq T_1 + D_4$, $T_4 \geq T_2 + D_4$. It turns out we “lose nothing” in this process; that is, at optimum, equality holds.

$T_i = \max_{j \in \text{FI}(i)} (T_j + D_i).$
 More generally, we replace the constraint $T_i \geq (T_j + D_i), j \in \text{FI}(i)$. by a relaxed version: $T_i \geq \max_{j \in \text{FI}(i)} (T_j + D_i).$ The above can be written:

We obtain an explicit representation of the previous GP:
 $\min_{x, T > 0} D : A_i(x) \leq A_{\max}, x_i \geq 1, i = 1, \dots, n,$
 $D \geq T_i, i = 1, \dots, n,$
 $T_i \geq (T_j + D_i(x)), i = 1, \dots, n, j \in \text{FI}(i).$ The above is also a GP. At the expense of adding n new variables and also adding constraints, we have obtained a sparser problem.



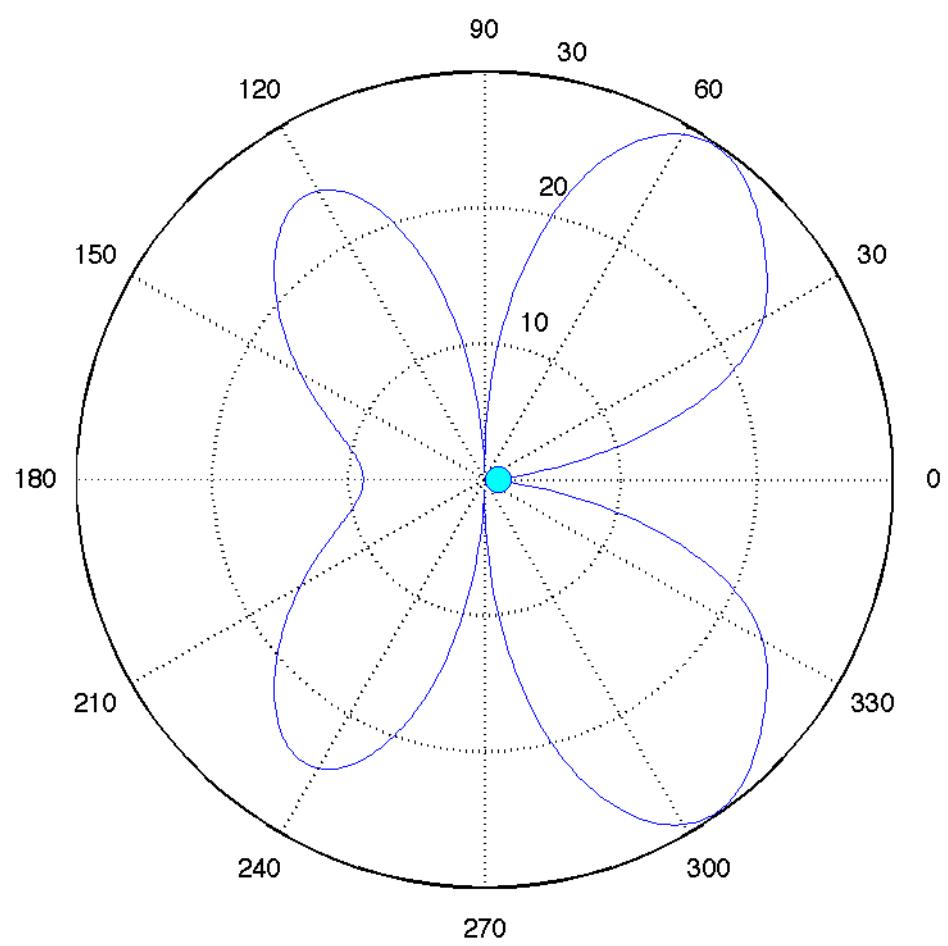


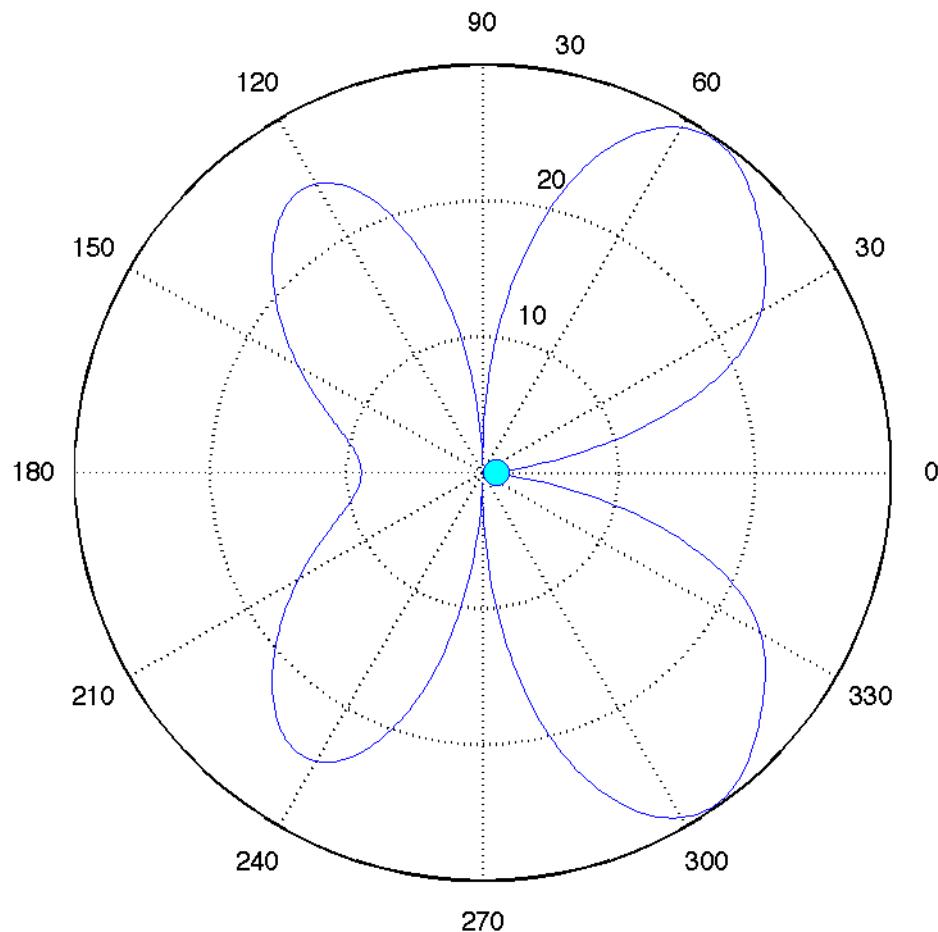


Two orthogonal vectors

$$x = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}, \quad y = \begin{pmatrix} 4 \\ -1 \\ -1 \end{pmatrix}$$

The two vectors in \mathbf{R}^3 : $x^T y = \underbrace{1 \times 4}_{x_1 \times y_1} + \underbrace{1 \times (-1)}_{x_2 \times y_2} + \underbrace{(-1) \times 3}_{x_3 \times y_3} = 0.$





Example [SOC](#) > [SOC inequalities](#) | [Standard Forms](#) | [Applications](#) > [Antenna Array Design](#) > Back | Example

- Data
- Minimum thermal noise power for given sidelobe level
- Minimum sidelobe level attenuation

Data

For a particular example, we have the following parameters:

- Number of antennas: $n = 16$.
- Wavelength: $\lambda = 8$.
- Pass-band size: $\Delta = \pi/6$. In what follows, we use the following matlab applet to define the data of our problem.

Matlab data

```
N = 10; % discretization parameter
n = 16; % number of antennas
lambda = 8; % wavelength
Phi = pi/6; % sidelobe parameter
Angles = linspace(Phi,pi,N); % angles in the stop band
a = 2*pi*sqrt(-1)/lambda; % intermediate parameter
```

Minimum thermal noise power for given sidelobe level

We first seek to minimize the thermal noise power subject to a sidelobe level constraint $\delta = .4$. Measured in decibels (dB): $20 \log_{10}(\delta) = -7.9588$ dB. This problem

$$\min_{z \in \mathbb{C}^n, \delta} \sum_{i=1}^n |z_i|^2 : \operatorname{Re}(D_z(0)) \geq 1, |D_z(\phi_i)| \leq \delta, i = 1, \dots, m.$$

can be cast as an SOCP: A CVX implementation of this problem is given below. Note that CVX understands the magnitude of a complex variable and transforms the corresponding constraint into a second-order cone one internally.

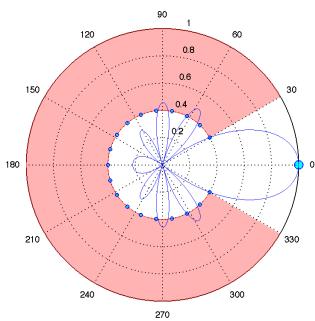
CVX implementation

```
cvx_begin
variable z(n,1) complex;
```

```

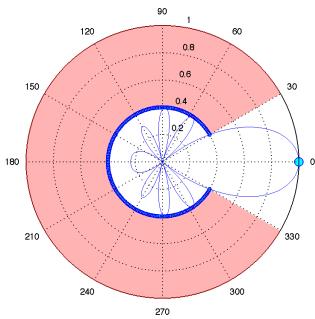
minimize( norm(z,2) )
subject to
  for i = 1:N,
    abs(exp(a*cos(Angles(i))*(1:n))*z) <= delta;
  end
  real( exp(a*(1:n))*z) >= 1;
cvx_end

```



Antenna array design: minimal thermal noise power given a sidelobe level constraint of -7.98 dB, enforced at $N = 10$ points. Due to our coarse discretization, the sidelobe constraints are only enforced at specific angles (in blue), but not everywhere satisfied.

The coarse discretization level of $N = 10$ may be an issue. This is readily solved by using a higher number, say $N = 90$.



Antenna array design: minimal thermal noise power given a sidelobe level constraint, enforced at $N = 90$ points. With a finer discretization, the sidelobe constraints are everywhere satisfied.

Minimum sidelobe level attenuation

Our goal is now to minimize the sidelobe attenuation level, δ , given the normalization requirement $\text{Re}(D_z(0)) \geq 1$.

$\min_{z \in \mathbb{C}^n, \delta} \delta : \text{Re}(D_z(0)) \geq 1, |D_z(\phi_i)| \leq \delta, i = 1, \dots, m.$

This can be cast as the SOCP

A CVX implementation is given below.

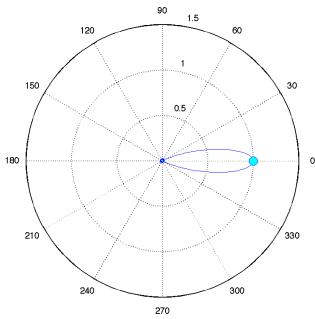
CVX implementation

```

cvx_begin
variable z(n,1) complex;
variable delta(1)
minimize( delta )
subject to
  for i = 1:N,
    abs(exp(a*cos(Angles(i))*(1:n))*z) <= delta;
  end
  real( exp(a*(1:n))*z) >= 1;
cvx_end

```

The result shows an optimal attenuation level in the stop band of $\delta^* =$, which, in decibels, is: $20 \log_{10}(\delta^*) = -46.4218$ dB.



Antenna array design: the optimal magnitude diagram. The attenuation is excellent in the stop band, so that the stop-band magnitude is not distinguishable from zero in this plot.

- Array of oscillators
- Diagram of a linear array

Harmonic oscillators

The basic unit in a transmitting antenna is a isotropic harmonic oscillator, which emits an spherical, monochromatic wave at wavelength λ and frequency ω .

The oscillator generates an electromagnetic field with electrical component at a certain point P located at a distance d from the antenna in space is given by $\frac{1}{d} \mathbf{Re} \left[z \exp \left(j(\omega t - \frac{2\pi d}{\lambda}) \right) \right]$, where $z \in \mathbf{C}$ is a design parameter that allows to scale and change the phase of the electrical field. We refer to this complex number as the *weight* of the antenna.

Array of oscillators

We now place n such oscillators at the locations $p_k \in \mathbf{R}^3$, $k = 1, \dots, n$. Each oscillator is associated with a complex weight z_k , $k = 1, \dots, n$. Then, the total

$$E = \mathbf{Re} \left[\exp(j\omega t) \cdot \sum_{k=1}^n \frac{1}{d_k} z_k \exp\left(-\frac{2\pi j d_k}{\lambda}\right) \right],$$

electrical field received at a point $p \in \mathbf{R}^3$ is then the sum, where $d_k := \|p - p_k\|_2$ is the distance from p to p_k , $k = 1, \dots, n$.

Diagram of a linear array

To simplify, let us assume that the oscillators form a *linear array*: they are placed on an equidistant grid of points placed on the x -axis, that is, $p_k = k e_1$, $k = 1, \dots, n$, with $e_1 = (1, 0, 0)$ the first unit vector. Let us further assume that the point p under consideration is far away: $p = r u$, with $u \in \mathbf{R}^3$ a unit-norm vector that specifies the direction, and the distance to the origin, r , is large.

For a linear array, the electrical field E depends only on the angle between the array and the far away point under consideration, ϕ . In fact, a good approximation to E is

of the form $E \approx \mathbf{Re} \left(\frac{1}{r} \exp(j\omega t - \frac{2\pi r}{\lambda}) \right) D_z(\phi)$ where ϕ denotes the angle between the vector u and e_1 , and the function $D_z : [0, 2\pi] \rightarrow \mathbf{C}$, with complex values $D_z(\phi) := \sum_{k=1}^n z_k \exp\left(\frac{2\pi j k \cos(\phi)}{\lambda}\right)$ is called the *diagram* of the antenna. We have used the subscript “ z ” to emphasize that the diagram depends on our choice of the complex weight vector, $z = (z_1, \dots, z_n)$.

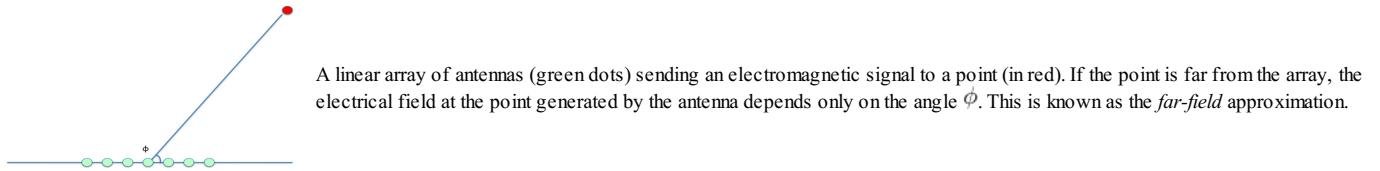


Image Annotation via Group Sparsity [SOCP > Applications > Back](#) | Image annotation | [Next](#)

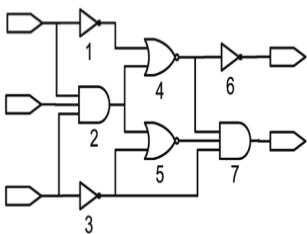
In progress; see [here](#).

Digital Circuit Design: Notes and References [GP > Applications > Circuit Design > Back](#) | Notes and Refs

GPs have a long history in circuit design. The presentation given here, as well as the example, is taken from the [paper](#) by S. Boyd, S.-J. Kim, D. Patil, and M. Horowitz; the paper contains many references.

Digital Circuit Design: Example [GP > Standard Forms | Applications > Circuit Design > Back](#) | Example | [Next](#)

We return to the example pictured before.



Design example: For the circuit, the gates have the following parameters.

Gate	C^{in}	C^{intr}	r	a
1, 3, 6	3	3	0.48	3
2, 7	4	6	0.48	8
4, 5	5	6	0.48	10

$$\min_{x, T > 0} D : \begin{aligned} A_i(x) &\leq A_{\max}, \quad x_i \geq 1, \quad i = 1, \dots, n, \\ D &\geq T_i, \quad i = 1, \dots, n, \end{aligned}$$

The design problem is $T_i \geq (T_j + D_i(x))$, $i = 1, \dots, n$, $j \in \text{FI}(i)$, where $A_i(x) = a_i x_i$, and the delay functions D_i are defined [here](#).

Senate Voting: Visualizing Senators on a Plane

We consider the data matrix containing the votes of the Senators in the 2004-2006 US Senate (2004-2006) introduced [here](#). We have seen [here](#) how to score, that is,

assign a single value to, each Senator.

Here, we are interested in projecting the data matrix on a plane, instead on a single line.

Projecting the data on a plane amounts to provide *two* scores for each Senators, $f_k(x) = u_k^T(x - \hat{x})$, where $x \in \mathbf{R}^m$ is the vector of votes of the Senator, and $k = 1, 2$ represents the axis of the plane. We can represent the data by plotting the scores $(f_1(x_j), f_2(x_j))$, $j = 1, \dots, n$. The vector \vec{f} can be expressed as

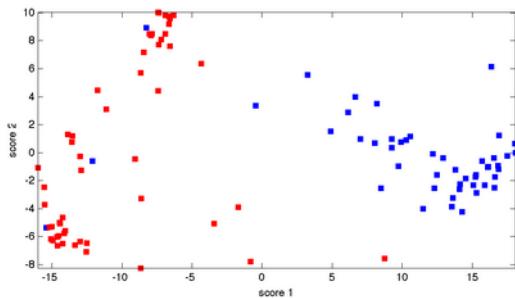
$$\vec{f} = U^T X_{\text{cent}},$$

where $U = [u_1, u_2] \in \mathbf{R}^{m \times 2}$ is the projection matrix.

Clearly, depending on which plan we choose to project on, we get a very different pictures. Some planes seem to be more “informative” than others. We return to this issue here.



Two-dimensional projection of the Senate voting matrix: This particular projection does not seem to be very informative. Notice in particular, the scale of the vertical axis. The data is all but crushed to a line, and even on the horizontal axis, the data does not show much variation.



Two-dimensional projection of the Senate voting matrix: This particular projection seems to allow to cluster the Senators along party line, and is therefore more informative. In XXX we explain how choose such a direction.

Senate Voting: Scoring Senators [Matrices](#) > [Matrix products](#) > Example

We consider the data matrix containing the votes of the Senators in the 2004-2006 US Senate (2004-2006) introduced [here](#). The data is stored in a $m \times n$ “voting” matrix $X = [x_1, \dots, x_n]$, with elements taken from $\{-1, 0, 1\}$. Each column (“data point”) of the voting matrix $x_j \in \mathbf{R}^m$, $j = 1, \dots, n$ contains the votes of a single Senator for all the bills; each row contains the votes of all Senators on a particular bill.

Scoring function

We are interested in assigning a “score” to each Senator, and thus represent all the data points as a single value on a line. This is the same as projecting the points on a line, as seen [here](#).

We take our scoring function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ to be affine, with values

$$f(x) = u^T x + v,$$

where $u \in \mathbf{R}^m$ represents a direction in the “bill” space, which is \mathbf{R}^m , and $v \in \mathbf{R}$. That is, we form *affine* combinations of the votes on each bill, so that the $n = 542$ votes for each Senator are reduced to a single number, or “score”.

Evaluated on the data points x_1, \dots, x_n , the scoring function takes the values $f_j = u^T x_j + v$, $j = 1, \dots, n$. Without loss of generality, we can always choose the constant (“bias”) v such that the average of the scores is zero:

$$0 = \sum_{i=1}^n f_i = \sum_{i=1}^n (u^T x_i + v) = u^T \left(\sum_{i=1}^n x_i \right) + n \cdot v,$$

which leads to the choice $v = u^T \hat{x}$, where

$$\hat{x} := \frac{1}{n} \sum_{i=1}^n x_i \in \mathbf{R}^m$$

represents the *sample average* of the Senators (that is, the vectors of votes of an “average” Senator).

Our zero-mean scoring function now takes the form

$$f(x) = u^T (x - \hat{x}).$$

Here, without loss of generality, the direction $u \in \mathbf{R}^m$ is normalized ($\|u\|_2 = 1$).

Computing the scores

To compute the scores, we first center the data:

$$X_{\text{cent}} = \begin{pmatrix} x_1 - \hat{x} & \dots & x_n - \hat{x} \end{pmatrix} = X - \hat{x}\mathbf{1}_n^T,$$

where $\mathbf{1}_n$ is the vector of ones in \mathbf{R}^n . Then, we compute the (row) vector of scores, via the matrix-vector product:

$$f = u^T X_{\text{cent}} = (X_{\text{cent}}^T u)^T$$

Example: behavior with respect to average bill

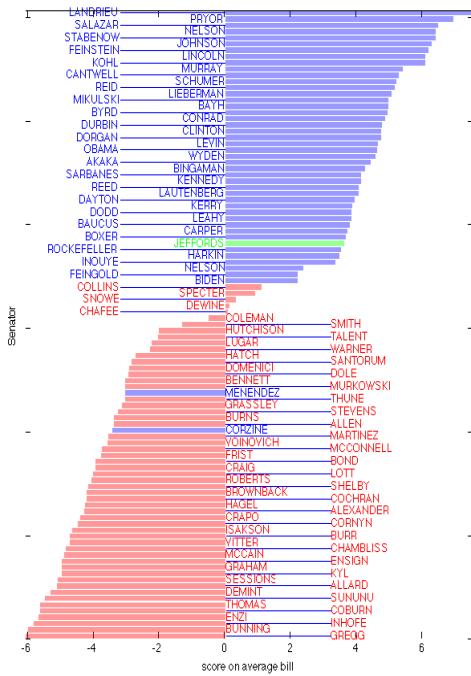
Choosing u to be a (normalized) vector of ones:

$$u = \frac{1}{\sqrt{m}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbf{R}^m.$$

(Our normalization ensures that the Euclidean norm of u is 1, and allows to compare several directions. The scaling factor is actually irrelevant here.)

The direction u corresponds to trying to understand the voting behaviors of the Senators in terms a single, synthetic, “average” bill. The scores we get this way correspond to an “extremism/conformism index”, since Senators with a low score tend to vote along the average (on the “average” bill), while those with a high score would tend to vote opposite to it.

With that choice, here is what we get:



Average vs. extreme scores on average bill: This image shows the values of the projections of the Senator's votes $x_j - \hat{x}$ (that is, with average across Senators removed) on the (normalized) “average bill” direction defined above.

The party affiliation of each Senator is shown, with names of Democrats in blue and those of Republicans in red. Based on this crude analysis, we could be led to conclude that the former tend to vote more according to the average, while those of the latter tend to vote less according to the average.

See also: [Senate voting data matrix](#).

Antenna Array Design [SOCP](#) > [Applications](#) > [Back](#) | Antenna Array Design | [Next](#)

See [here](#).