

Matrix Methods in Signal Processing ...

(Lecture notes for EECS 551)

Jeff Fessler
University of Michigan

December 9, 2018

Chapter 0

EECS 551 Course introduction: F17

Contents (final version)

0.1 Course logistics	0.2
0.2 Julia language	0.9
0.3 Course topics	0.16



These lecture notes initially were based extensively on Prof. Raj Nadakuditi's hand-written notes. I am grateful to him for sharing his course materials. I also thank GSIs David Hong and Steven Whitaker and 551 students in F17 and F18 for many corrections to earlier versions.

These notes were typeset using \LaTeX . One way to learn \LaTeX is to use <http://overleaf.com>.

0.1 Course logistics

EECS 551: Matrix Methods For Signal Processing, Data Analysis & Machine Learning 4 credits

Lecture: Tue, Thu 9-10:30AM, 1500 EECS

012 Fri. 9:30-10:30 AM, 1003 EECS

Discussion: 011 Fri. 10:30-11:30 AM, 1500 EECS

Instructor: Prof. Jeff Fessler fessler@umich.edu <https://web.eecs.umich.edu/~fessler/>

Office hours: Tue, Wed 10:30-11:30AM, 4431 EECS.

Include [eecs551-f18] in email subject for possibly less slow response. Use [Canvas](#) when possible.

GSI (office hours held in 3312 EECS):

- Steven Whitaker stwhit@umich.edu <http://web.eecs.umich.edu/~stwhit>
Mon 9-10AM Wed 3:30-5PM Thu 2:30-4PM

Course materials:

Action: bookmark these links.

- Primary site is [Canvas](#): <https://umich.instructure.com/courses/248963>
(homework, solutions, lecture notes, announcements, etc.)
- Annotated versions of class notes: <https://umich.box.com/v/551f18markup>
- Secondary site (demos, back-ups): <http://web.eecs.umich.edu/~fessler/course/551>

The course goal is providing a mathematical foundation for subsequent signal processing and machine learning courses, while also introducing matrix-based SP/ML methods that are useful in their own right.

Prerequisites

DSP (*i.e.*, EECS 351, formerly 451) or graduate standing.

Prof. Nadakuditi's EECS 598 perhaps relies less on DSP background.

Exams

(other ECE exams)

Midterm Exam 1: Mon. Oct. 22, 6-8:00 PM, 1303&1311 EECS

(more feedback,
less stress per exam)

Midterm Exam 2: Mon. Nov. 19, 6-8:00 PM, 1303&1311 EECS

Final Exam: Fri. Dec 14, 1:30-3:30 PM, Room TBA

Grades

Homework and task sheets	20%
clicker and quizzes	5%
Midterm exam 1	20%
Midterm exam 2	25%
Final exam	30%

Final grade cutoffs will be 90/80/70% or lower. Exam scores may be standardized.

Honor code

The UM College of Engineering Honor Code applies. You should be familiar with it.

See <https://ossa.engin.umich.edu/honor-council/> for details.

See collaboration policies below.

Homework

Typically due on Thursday at 4PM. Typically an automatic extension to Friday at 4PM. No further.

Submit scans of solutions to <https://gradescope.com>.

(HW1 on [Canvas](#)!)

Hopefully will be graded and “returned” via gradescope within a week.

Written regrade requests via gradescope within 3 days of return date.

Actions:

- Check for your name on [gradescope](#) (should be there thanks to [Canvas](#) integration)
- Review [gradescope scan/pdf submission process](#). There are also [video instructions](#).

Collaboration policy: Homework assignments are to be completed on your own. You are allowed to consult with other students (and instructors) during the conceptualization of a solution, but all written work, whether in scrap or final form, is to be generated by you, working alone. Also, you are not allowed to use, or in any way derive advantage from, the existence of solutions prepared in prior years. Violation of this policy is an honor code violation. If you have questions about this policy, please contact me. While collaboration can sometimes be helpful to learning, if overused, it can inhibit the development of your problem solving skills.

Ethics

Sharing any materials from this class with other individuals not in the class without written instructor permission will be treated as an Honor Code violation. Posting your own solutions (including code) on public sites like [github.com](#) is also prohibited. Keep your materials private! In particular, uploading any materials from this class to web sites akin to [coursehero.com](#) will be reported to the Honor Council.

Homework grading

Homework grading is constrained by GEO union policies. See:

http://web.eecs.umich.edu/~fessler/course/551/r/grading_geo.txt

<http://web.eecs.umich.edu/~fessler/course/551/r/grader-duties.pdf>

Manually graded problems will be on a scale of 0-3:

- 0 No solution was attempted
- 1 A solution was attempted but the approach used did not recognizably conform to any in the solution set
- 2 The approach used recognizably conformed to one in the solution set, but the answer was incorrect.
- 3 A solution approach recognizably conformed to one in the solution set, and the answer was correct.

JULIA-based auto-graded problems (details on HW1) typically will be 10 points each (10 or 0).

Quizzes

Starting in the second week of the course, there will be short quizzes (typically 4-6 questions) on [Canvas](#) that are due by 9am on Tue. and Thu. The quizzes will become available 24 hours before each class. These are designed to be quick checks of your understanding of the material covered up to that point. You have 10 minutes to complete the quiz. With about 21 quizzes of about 5 points each, each quiz question counts for a minuscule $2.5\%/21/5 \approx 0.024\%$ of your final grade, so their main purpose is learning, not assessment. Thus, [Canvas](#) shows you the answers right after you take it.

Post concerns about any quiz question to a [Canvas](#) Discussion *after* the Quiz due date.

Apparently [Canvas](#) only allows a student to view a past quiz (*e.g.*, for exam review) that they attempted!

Missing class

- Classes are captured/recorded and viewable on Canvas.
- Missed clicker questions and quizzes cannot be made up.
- Annotated notes are available online.
- Daily topic list: <http://web.eecs.umich.edu/~fessler/course/551> (topics)

Books and other resources

Action: Decide whether to buy reference textbook [1]:
Laub, 2005; *Matrix analysis for scientists and engineers*.
Should be on reserve at UM Engineering Library

\$50 at <http://bookstore.siam.org/ot91> - 30% member discount.
Student membership is free: <https://siam.org/students/memberships.php>
(Select “University of Michigan” not “UM Ann Arbor” as the Academic Member Institution.”)

Books that are useful references: [2] [3] [4]. Online books: [5–7].
Khan Academy: <http://www.khanacademy.org/math/linear-algebra>

Clickers

<http://caenfaq.engin.umich.edu/10909-clickers/>

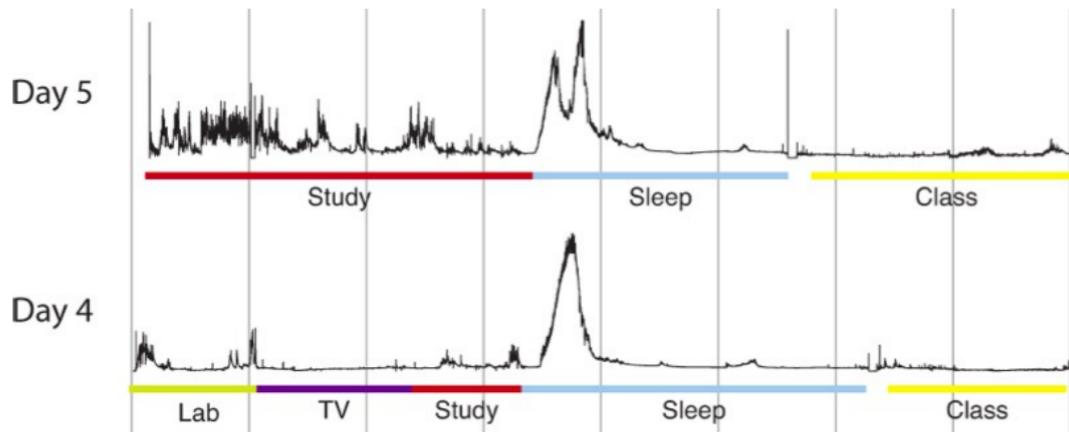
Bring batteries!

Action: Buy at <http://computershowspace.umich.edu/remotes/> (\$29 used, buy back for \$19)

Action: Register your clicker at [Canvas](#).

Clicker question scoring: 2 points for answering, 3 points for correct answer. (Learning, not assessment.)

Why? From M Poh, M Swenson, R Picard: “A wearable sensor for unobtrusive, long-term assessment of electrodermal activity.” IEEE Tr. on Biomed. Engin., 57(5):1243-52, May 2010. [8]



PDF lecture note features

These notes highlight some important **terms** in red.

Many important terms have links in the pdf documents to **Wikipedia** in violet. Some links look like: [wiki]
Those links are clickable in the pdf and should cause your browser to open at the appropriate url.

Define. Key definitions are shaded like this.

Particularly important topics are shaded like this.

JULIA code is shaded like this.

Boxes with this color need completion during class.

A road hazard or “**dangerous bend” symbol** in the margin warns of tricky material.



A double diamond symbol is “experts only” material included for reference that is likely beyond the scope of EECS 551 exams.



These notes are not a textbook; they are designed for classroom use. My goal is that every other page or so (except in this part) should have something more interactive than just text, such as a figure or a clicker question or a JULIA code snippet or some incomplete equation(s) that students must complete during class.

These notes are formatted with 16:10 aspect ratio to match the projector in the lecture room; that format is well-suited for printing two slides per paper side. If you print, please save paper by using that option.

Action: Print the next chapter (not this one) or bring pdf to class on a suitable device for annotating.

0.2 Julia language

All code examples and homework code templates will use the **Julia programming language**.

Why?

- It is free; you can download from <https://julialang.org>
- It is a real programming language, developed for numerical computing [9].
- Prof. Nadakuditi and/or I have used it in W17 (at MIT), F17, W18, F18 for 551/598...
- Interactive (like Python and MATLAB), yet fast execution because it is compiled.
- Much of its syntax is like MATLAB, so much easier for me to learn and use than python.
- DSP / data-science / machine learning are all done with many software languages...
- **Jupyter notebooks** (based on IPython) are educational, integrating math with documented code and figures.
- **JuliaBox** allows within-browser use in the cloud, mostly for limited toy experimenting. Not recommended!

Some documentation / books:

- Online docs: <https://docs.julialang.org/en/stable/manual/documentation/>
- Wikibook Intro to JULIA: https://en.wikibooks.org/wiki/Introducing_Julia
- Online “Getting started with JULIA” book: <https://search.lib.umich.edu/catalog/record/013714926>
- Cheatsheet: <https://cheatsheets.quantecon.org/julia-cheatsheet.html>
- Cheatsheet: <http://math.mit.edu/~stevenj/Julia-cheatsheet.pdf>
- YouTube intro video: <https://www.youtube.com/watch?v=puMIBYthsjQ>

News articles about business uses:

- Forbes magazine article
- InfoWorld comparison of JULIA and Python

Sponsors of Juliacon 2018:

Platinum
Sponsors



Gold
Sponsors



Microsoft

Maven

INVENIA

Julia
computing

Capital One

GORDON AND BETTY
 MOORE
FOUNDATION

Silver
Sponsors

gambitresearch

TANGENT WORKS
MACHINE LEARNING

amazon

The
Alan Turing
Institute

JEFFREY SARNOFF

EVN

CONNING[®]

A brief comparison of three interactive languages

Operation	MATLAB	JULIA	Python import numpy as np
Dot product	<code>dot(x,y)</code>	<code>dot(x,y)</code>	<code>np.dot(x,y)</code>
Matrix mult.	<code>A * B</code>	<code>A * B</code>	<code>A @ B</code>
Element-wise	<code>A .* B</code>	<code>A .* B</code>	<code>A * B</code>
Scaling	<code>3 * A</code>	<code>3A</code>	<code>3 * A</code>
Matrix power	<code>A^2</code>	<code>A^2</code>	<code>np.linalg.matrix_power(A, 2)</code>
Element-wise	<code>A.^2</code>	<code>A.^2</code>	<code>A**2</code>
Inverse	<code>inv(A)</code>	<code>inv(A)</code>	<code>np.linalg.inv(A)</code>
Inverse	<code>A^(-1)</code>	<code>A^(-1)</code>	<code>np.linalg.inv(A)</code>
Indexing	<code>A(i,j)</code>	<code>A[i,j]</code>	<code>A[i,j]</code>
Range	<code>1:9</code>	<code>1:9</code>	<code>np.arange(1,9,1)</code>
Range	<code>linspace(0,4,9)</code>	<code>LinRange(0,4,9)</code>	<code>np.arange(0,4.01,0.5)</code>
Strings	<code>'text'</code>	<code>"text"</code>	(either)
Inline func.	<code>f = @(x,y) x+y</code>	<code>f(x,y) = x+y</code>	<code>f = lambda x,y : x+y</code>
Increment	<code>A = A + B</code>	<code>A += B</code>	<code>A += B</code>
Herm. transp.	<code>A'</code>	<code>A'</code>	<code>A.transpose()</code>
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	<code>[1 2; 3 4]</code>	<code>[1 2; 3 4]</code>	<code>np.array([[1, 2], [3, 4]])</code>

The JULIA column assumes you have typed: `using LinearAlgebra` for using the `dot` function.
 See <https://cheatsheets.quantecon.org/> for more.

JULIA logistics

- Software parts of homework solutions (JULIA code) will be auto-graded, with *unlimited* tries.
- More details in Discussion section for HW
- Some tutorials (and cautionary notes for MATLAB users):
<https://web.eecs.umich.edu/~fessler/course/551/julia/tutor/>
 - `julia-tutor-vector` : vector/matrix operations and “**call by reference**” aspect of JULIA
 - `julia-tutor-slice` : matrix indexing (slicing)
 - `julia-tutor-sum SIMD.html` : acceleration using `@simd`
- Later we will do in-class activities using JULIA and you will want to bring a laptop with JULIA on it.
- Editors: Juno: JULIA IDE <http://junolab.org/>, Atom: <https://atom.io/>, vim: <https://www.vim.org/> and more, see list at JULIA web site: <https://julialang.org/>
- We will use JULIA version 0.7 / 1.0 in F18.
Beware of online Q/A for older versions of JULIA!
- **Actions:** Bring laptop to Discussion section Friday.



JULIA: getting started

- **Actions:** Download v0.7 from <https://julialang.org/downloads/>
- Install (*e.g.*, on a Mac, double click the .dmg file, copy Julia-0.7.app to Applications folder)
- Add the app to your dock and double click to launch it.
(experts: create a shell alias to `Julia-0.7.app/Contents/Resources/julia/bin/julia`)
- You should see JULIA’s **REPL** prompt: `julia>`
- Type `1+2` to verify it works.

- Press the `?` key and then type a command like `LinRange` to get documentation about it.
- Experiment with some basic commands like `x=3` and `x+2` and `x^2`
- Press the `]` key to enter the package manager **REPL** that has a prompt like `(v0.7) pkg>`
Add some useful packages using the package manager:
 - `add Plots`
 - `add Arpack`
 - `add IJulia`
 - `precompile` (this may take time to run, but will save time later)
 - Press the delete key to exit the package manager REPL and return to JULIA's main REPL.
- Windows users: currently you need the “master” branch of IJulia, *i.e.*, add `IJulia#master` but hopefully this will be fixed by the start of the term. Your GSI uses Windows and can help here if needed.
- At the JULIA prompt, try launching a Jupyter notebook:
`using IJulia; notebook()`
(Could be slow the first time as it gets compiled.)
For help with Jupyter, see <https://github.com/JuliaLang/IJulia.jl>
e.g., you might prefer `notebook(detached=true)`
or `notebook(detached=true, dir="/some/path")`
- Experiment with the Jupyter notebook, and peruse some online resources.
- If you have problems later, return to the package manager and run `update` to get the latest package versions. Then run `precompile` again.
- Not recommended: JuliaPRO: <https://juliacomputing.com/products/juliapro>



One professor's software language history

year	name	still using?
1977	BASIC	
1980	FORTRAN	
1981	Z80 assembly	
1982	APL	
1983	PASCAL	
1983	C	Y
1985	LISP	
1986	CSH	Y
1988	Matlab	Y
2000	Perl	
2004	Python	
2010	CUDA	Y
2017	Julia	Y

JULIA has a machine-learning library called [Flux](#) [10].

It also has an interface to [Tensorflow](#) if you prefer that.

There is also a CUDA interface for GPU programming [11].

MATLAB is “free” for UM students:

<http://caenfaq.engin.umich.edu/10378-Free-Software-for-Students/matlab-for-students>

Clicker survey questions

How are you feeling about JULIA? (Pick the answer that is your strongest feeling.)

- A: Anticipate it will be useful
- B: Bothered about learning something new
- C: Concerned about my software skills
- D: Indifferent (or none of the others apply)
- E: Excited to be on the cutting edge of numerical computing

Prior software experience?

- A: Not Matlab, but any of C or C++ or Python
- B: Matlab only
- C: Matlab and (Python | C | C++)
- D: JULIA and (Matlab | Python | C | C++)
- E: None of the above

Office hours (do not answer if your schedule is still uncertain)

- A: Tue 10:30-11:30 works for me, but not Wed.
- B: Wed 10:30-11:30 works for me, but not Tue.
- C: Both Tue and Wed work
- D: Neither Tue nor Wed work for me, but some GSI hour(s) work
- E: None of the Prof. or GSI office hours work for me

0.3 Course topics

Here are some likely course topics in approximate chronological order, along with sections from [1].

1. Introduction to matrices

Topic (Laub §)	Finite difference	Hermitian	invertible matrix
Julia	pentadiagonal	linear algebra	Laplace's formula
vector (1.1)	Gram matrix	dot product	eigenvalues (9.1)
matrix (1.1)	upper Hessenberg	inner product	characteristic polynomial
linear operation	eigenvalue algorithms	outer product	fundamental theorem of algebra
DFT	lower Hessenberg	rank 1 matrix	Gram matrix
system of linear equations	rectangular diagonal	matrix multiplication	covariance matrix
convolution	dense matrix	(1.2)	singular matrix
LTI	sparse matrix	orthogonal (1.3)	scatter matrix
term-document matrix	Toeplitz	orthonormal	trace (1.1)
diagonal	Circulant	norm	field (2.1)
upper triangular	block matrix	orthogonal matrix	vector space (2.1)
Gaussian elimination	block diagonal matrix	unitary matrix	linear transform (3.1)
lower triangular	transpose	Parseval's theorem	
Cholesky decomposition	Hermitian transpose	determinant (1.4)	
tridiagonal	symmetric	Gram matrix	

2. Matrix factorizations / SVD

eigenvalues (10.1)
spectral theorem
eigenvectors
orthogonal

orthonormal
eigendecomposition
diagonalizable
SVD (5.1)

spectral norm (7.4)
MIMO channels
beamforming
positive definite (10.2)

positive semi-definite

3. Subspaces and rank

dimensionality reduction
subspace (2.2)
periodic functions
span (2.3)
linear combinations
linearly independent (2.3)
linearly dependent
monomials

basis (2.3)
discrete cosine transform
wavelet transform
coordinate system
additive synthesis
basis (2.3)
subspace sum (2.4)
intersection

direct sum
orth. complement (3.4)
linear map (3.1)
range (3.4)
column space
row space
rank (3.5)
null space (3.4)

kernel
nullity
fundamental theorem of
linear algebra (3.5)
orthogonal basis

4. Linear equations and least-squares

linear equations (6.1)
linear least-squares (8.1)
residual
orthogonal polynomials
convex function
normal equations (8.1)
SVD (8.4)
over-determined system

Moore-Penrose
pseudoinverse (4.1)
left inverse (4.3)
right inverse
rectangular diagonal
SVD (5.2)
QR decomposition (8.5)
under-determined

orthogonality principle
error
linear variety
flat
idempotent matrix
minimum norm sol. (8.1)
compressed sensing
truncated SVD

floating-point precision
condition number
low-rank approx. (8.1)
Tikhonov regularization
regularization parameter
conjugate gradient

5. Norms

vector norm (7.3)
Euclidean norm
triangle inequality
Parseval's theorem
inner product spaces (7.2)
inner product

Frobenius inner product
parallelogram law
Cauchy-Schwarz ineq.
angle between subspaces
correlation coefficient
matrix norms (7.4)

sub-multiplicative
Frobenius norm
induced norm
Schatten p-norm
norm equivalence
unitarily invariant norms

Procrustes problem
polar decomposition
idempotent matrix

6. Low-rank approximation

dimensionality reduction	Unitary invariance	estimate	ISTA
low-rank approximation	PCA	weakly differentiable	proximal gradient method
factor analysis	multidimensional scaling	Divergence	proximity operation
scree plot	geodesic distances	OptShrink	
Eckart-Young-Mirsky theorem	Heaviside step function	matrix completion	
	Stein's unbiased risk	Netflix problem	

7. Optimization basics

optimization	matrix powers	Lipschitz continuity
convergence	positive semidefinite	Taylor's theorem
(matrix) square root	positive definite	logistic regression
principal square root	gradient descent	Hessian matrix

8. Special matrices

monic polynomial

companion matrix

minimum polynomial

Vandermonde matrix

Kronecker sum

circulant matrix

DFT

fast Fourier transform

power iteration

positive matrix

nonnegative matrix

primitive matrix

Geršgorin disk theorem

Perron-Frobenius

theorem

Gelfand's formula

multiplicity one

algebraic multiplicity

geometric multiplicity

irreducible matrices

Markov chain

directed graph

transition matrix

stochastic eigenvector

law of total probability

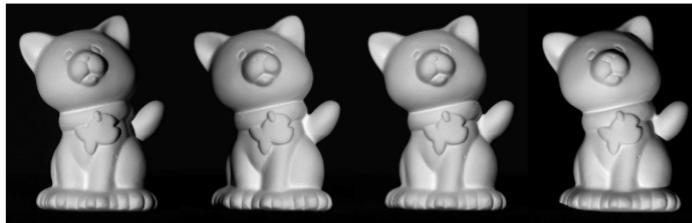
irreducible matrix

simple eigenvalue

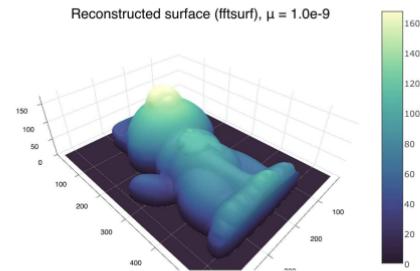
strongly connected graph

Google's PageRank

Many applications, especially in HW, e.g., photometric stereo:



Reconstructed surface (fftsurf), $\mu = 1.0e-9$



Bibliography

- [1] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005.
- [2] T. K. Moon and W. C. Stirling. *Mathematical methods and algorithms for signal processing*. Prentice-Hall, 2000.
- [3] G. H. Golub and C. F. Van Loan. *Matrix computations*. 2nd ed. Johns Hopkins Univ. Press, 1989.
- [4] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge: Cambridge Univ. Press, 1985.
- [5] L. Eldén. *Matrix methods in data mining and pattern recognition*. Errata: <http://users.mai.liu.se/larel04/matrix-methods/index.html>. Soc. Indust. Appl. Math., 2007.
- [6] K. B. Petersen and M. S. Pedersen. *The matrix cookbook*. ., 2012.
- [7] S. Boyd and L. Vandenberghe. *Introduction to applied linear algebra: Vectors, matrices, and least squares*. ., 2017.
- [8] M-Z. Poh, N. C. Swenson, and R. W. Picard. “A wearable sensor for unobtrusive, long-term assessment of electrodermal activity”. In: *IEEE Trans. Biomed. Engin.* 57.5 (May 2010), 1243–52.
- [9] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. “Julia: A fresh approach to numerical computing”. In: *SIAM Review* 59.1 (2017), 65–98.
- [10] M. Innes. “Flux: Elegant machine learning with Julia”. In: *J. of Open Source Software* 3.25 (2018), p. 602.
- [11] T. Besard, C. Foket, and B. De Sutter. “Effective extensible programming: unleashing Julia on GPUs”. In: *IEEE Trans. Parallel. Dist. Sys.* (2018).

Chapter 1

Introduction to Matrices

Contents (final version)

1.0 Introduction (<code>s,intro</code>)	1.2
1.1 Basics (<code>s,why,vector</code>)	1.3
1.2 Matrix structures (<code>s,notate</code>)	1.13
Notation	1.13
Common matrix shapes and types (<code>s,shape</code>)	1.14
Matrix transpose and symmetry (<code>s,transpose</code>)	1.19
1.3 Multiplication (<code>s,vdot</code>)	1.21
Vector-vector multiplication	1.21
Matrix-vector multiplication (<code>s,mul,mat,v</code>)	1.24
Matrix-matrix multiplication (<code>s,mult</code>)	1.29
Matrix multiplication properties (<code>s,mul,prop</code>)	1.29
Using matrix-vector operations in high-level computing languages (<code>s,plot2</code>)	1.35
Invertibility (<code>s,inv</code>)	1.43
1.4 Orthogonality (<code>s,orth,vec</code>)	1.46
Orthogonal vectors	1.46
Cauchy-Schwarz inequality (<code>s,cauchy</code>)	1.48

Orthogonal matrices (<code>s,orth,mat</code>)	1.49
1.5 Matrix determinant (<code>s,determinant</code>)	1.51
1.6 Eigenvalues (<code>s,eigl</code>)	1.58
Properties of eigenvalues (<code>s,eig,prop</code>)	1.62
1.7 Trace (<code>s,trace</code>)	1.65
1.8 Appendix: Fields, Vector Spaces, Linear Transformations (<code>s,vect</code>)	1.66

1.0 Introduction

This chapter reviews vectors and matrices and basic *properties* like shape, orthogonality, determinant, eigenvalues and trace. It also reviews some *operations* like multiplication and transpose.

Source material for this chapter includes [1, §1.1-1.4, 2.1, 3.1, 9.1].

1.1 Basics**Why vector?**

L§1.1

- Data organization
- To group into matrices (data)
or to be acted on by matrices (operations)

Example: Personal attributes - age, height, weight, eye color (?) ...

Example: Digital grayscale image (!)

- A vector in the vector space of 2D arrays.
- I rarely think of a digital image as a “matrix” !

(Read the Appendix on p. [1.66](#) about general vector spaces.)

Why matrix?

Two reasons:

- **data** array
 - **linear operation** aka **linear map** or **linear transformation**
-

Example: data matrix:

return to personal attributes

person1, person2, ...

Example: linear operation:

DFT matrix in 1D

(DFT is a prereq term)

DFT matrix in 2D

unified as matrix-vector operation:

$$\underbrace{\mathbf{X}}_{N \times 1 \text{ spectrum}} = \underbrace{\mathbf{W}}_{N \times N \text{ DFT}} \underbrace{\mathbf{x}}_{\hookrightarrow N \times 1 \text{ signal}}$$

Example (classic): solve **system of linear equations** with N equations in N unknowns:

$$\begin{aligned} ax_1 + bx_2 + cx_3 &= u \\ dx_1 + ex_2 + fx_3 &= v \implies \mathbf{Ax} = \mathbf{b}, \text{ with } \mathbf{A} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \mathbf{b} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \end{aligned}$$

Certainly the matrix-vector notation $\mathbf{Ax} = \mathbf{b}$ is more concise (and general).

A traditional linear algebra course would focus extensively on solving $\mathbf{Ax} = \mathbf{b}$.

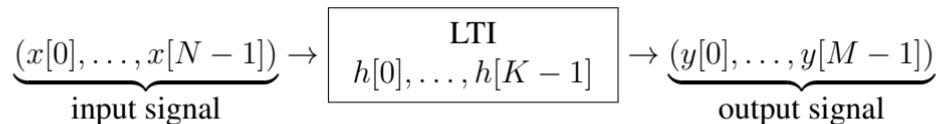
This topic will not be a major focus of 551!

Example: linear operation:

convolution in DSP:

(convolution is a prereq term)

(**LTI** is a prereq term)



Matrix-vector representation of convolution:

$$\mathbf{y} = \mathbf{H}\mathbf{x}$$

where \mathbf{x} is length- N vector, \mathbf{y} is length- M vector and \mathbf{H} is a $M \times N$ matrix with elements

$$H_{mn} = h[m - n]. \quad (1.1)$$

What is M in terms of N and K ? (Choose best answer.)

(DSP prerequisite!)

A: $\max(N, K) - 1$

B: $\max(N, K)$

C: $N + K$

D: $N + K + 1$

E: None of these.

??

The convolution operation is the component of any **convolutional neural network** (**CNN**) [2] [3, p. 514].

Other matrix operators for other important linear operations: wavelet transform, DCT, 2D DFT, ...

Example: data matrix

(Read)

A **term-document matrix** is used in **information retrieval**.

Document1: EECS551 meets on Tuesdays and Thursdays

Document2: Tuesday is the most exciting day of the week

Document3: Let us meet next Thursday.

Keywords (terms): EECS551 meet Tuesday Thursday exciting day week next

(Note: use stem, ignore generic words "the" "and")

Term-document (binary) matrix:

Term	Doc1	Doc2	Doc3
EECS551	1	0	0
meet	1	0	1
Tuesday	1	1	0
Thursday	1	0	1
exciting	0	1	0
day	0	1	0
week	0	1	0
next	0	0	1

The entries in a (mostly) numerical table like this are naturally represented by a 8×3 **matrix** T (because each column is a vector in \mathbb{R}^8 and there are three columns) as follows:

Why mostly? Column and row labels...

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Query as a matrix vector operation: find all documents relevant to the query “exciting days of the week.” See example query vector to the right:

In matrix terms, find columns of T that are “close” (will be defined precisely later) to query vector q .

$$q = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

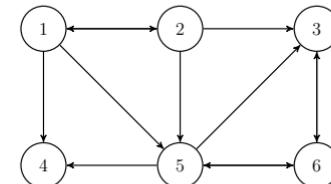
This is an information retrieval application expressed using matrix/vector operations.

Example: Networks, Graphs, and Adjacency Matrices

(Read)

Consider a set of six web pages that are related via web links (outlinks and inlinks) according to the following **directed graph** [4, Ex. 1.3, p. 7]:

The 6×6 **adjacency matrix** A for this graph has a nonzero value (unity) in element A_{ij} if there is a link from node (web page) j to i :



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 1/3 & 0 \end{bmatrix}.$$

The **link graph matrix** L is found from the adjacency matrix by normalizing (each column) with respect to the number of outlinks so each column sums to unity,

We will see later that Google's **PageRank** algorithm [5] for quantifying importance of web pages is related to an **eigenvector** of L . (There is a **left eigenvector** with $\lambda = 1$ so there must also be a **right eigenvector** with that **eigenvalue** and that is the one of interest.) So evidently representing web page relationships using a **matrix** and computing properties of that matrix is useful in many domains, even some that might seem quite unexpected at first.

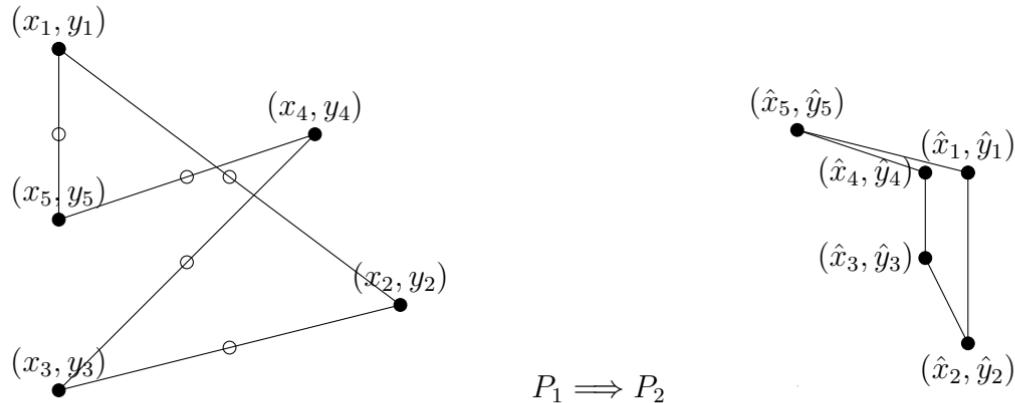
Example: a simple linear operation

(Read)

In the preceding two examples (viewing a 2D function and computing volume under a 2D function), the matrix was a 2D array of *data*. Recall that there were two “whys” for matrices: data and *linear operations*. Now we turn to an example where the matrix is associated with an **linear operation** that we apply to vectors.

Consider a polygon P_1 defined by N vertex points $(x_1, y_1), \dots, (x_N, y_N)$, connected in that order, and where the last point is also connected to the first point so there N edges.

Define a new polygon P_2 where each vertex is the *average* of the two points along an edge of polygon P_1 .



Mathematically, the first new vertex point is linearly related to the old points as: $\hat{x}_1 = \frac{x_1 + x_2}{2}$, $\hat{y}_1 = \frac{y_1 + y_2}{2}$.

In general, the n th new vertex point is related to the old vertex points by the following formula:

$$\hat{x}_n = \frac{x_n + x_{(n+1) \bmod N}}{2}, \quad \hat{y}_n = \frac{y_n + y_{(n+1) \bmod N}}{2}, \quad n = 1, \dots, N.$$

It is convenient to collect all of these relationships into length- N vectors as follows:

$$\begin{bmatrix} \hat{x}_1 \\ \vdots \\ \hat{x}_{N-1} \\ \hat{x}_N \end{bmatrix} = \frac{1}{2} \begin{bmatrix} x_1 + x_2 \\ \vdots \\ x_{N-1} + x_N \\ x_N + x_1 \end{bmatrix}, \quad \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_{N-1} \\ \hat{y}_N \end{bmatrix} = \frac{1}{2} \begin{bmatrix} y_1 + y_2 \\ \vdots \\ y_{N-1} + y_N \\ y_N + y_1 \end{bmatrix}. \quad (1.2)$$

This is a **linear operation** so we can write it concisely using **matrix-vector multiplication**:

$$\hat{\mathbf{x}} = \mathbf{A}\mathbf{x}, \quad \hat{\mathbf{y}} = \mathbf{A}\mathbf{y}, \quad \mathbf{A} \triangleq \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & & \\ 0 & 0 & \dots & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & \dots & 1 \end{bmatrix}. \quad (1.3)$$

The matrix form might not seem more illuminating than the expressions in (1.2) at first. But now suppose we ask the following question: what happens to the polygon shape if we perform the process repeatedly

(assuming we do some appropriate normalization)? The answer is not obvious from (1.2) but in matrix form the answer is related to the **power iteration** that we will discuss later for computing the **principle eigenvector** of A . See [6] and <https://www.jasondavies.com/random-polygon-ellipse/> and Apr. 2018 SIAM News and Sep. 2018 SIAM News.

1.2 Matrix structures

Notation

L§1.1

- \mathbb{R} : real numbers
- \mathbb{C} : complex numbers
- \mathbb{F} : field (see Appendix on p. 1.67), which here will always be \mathbb{R} or \mathbb{C} .

We will use this symbol frequently to discuss properties that hold for both real and complex cases.

- \mathbb{R}^N : set of N -tuples of real numbers
- \mathbb{C}^N : set of N -tuples of complex numbers
- \mathbb{F}^N : either \mathbb{R}^N or \mathbb{C}^N
- $\mathbb{R}^{M \times N}$: set of real $M \times N$ matrices
- $\mathbb{C}^{M \times N}$: set of complex $M \times N$ matrices
- $\mathbb{F}^{M \times N}$: either $\mathbb{R}^{M \times N}$ or $\mathbb{C}^{M \times N}$

Because $\mathbb{R}^{M \times N} \subset \mathbb{C}^{M \times N}$, whenever you see the symbol $\mathbb{F}^{M \times N}$ you can just think of it as $\mathbb{C}^{M \times N}$, but it means the properties being discussed also hold for $\mathbb{R}^{M \times N}$.

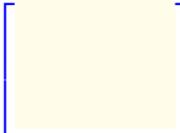
- vectors are typically column vectors here (but see Appendix for more detail)
- row vectors are written \mathbf{x}^\top or \mathbf{x}' , where $\mathbf{x} \in \mathbb{R}^n$ or $\mathbf{x} \in \mathbb{C}^n$.



Common matrix shapes and types

For each matrix shape, this table gives one *of many* examples of why that shape is useful in practice.

These shapes are defined by where non-zero values may be. A matrix of all zeros is in all of these categories.

diagonal		Covariance matrix of a random vector with uncorrelated elements. Easiest to invert! Definition: $a_{ij} = 0$ if $i \neq j$
upper triangular		Arises in Gaussian elimination for solving systems of equations. Quite easy to invert. Definition: $a_{ij} = 0$ if $j < i$
lower triangular		Used in the Cholesky decomposition .
tridiagonal		Finite difference approximation of a 2nd derivative. Arises in numerical solutions to differential equations.

pentadiagonal

$$\begin{bmatrix} & & \\ & \text{yellow} & \\ & & \end{bmatrix}$$

Gram matrix for discrete approximation to 2nd derivative.**upper Hessenberg**

$$\begin{bmatrix} & & \\ & \text{yellow} & \\ & & \end{bmatrix}$$

Arises in **eigenvalue algorithms**. Fairly easy to invert.**lower Hessenberg**

$$\begin{bmatrix} & & \\ & \text{yellow} & \\ & & \end{bmatrix}$$

Ditto.

The above are all **square matrix** shapes!There is also a **rectangular diagonal matrix** shape that will be useful later for the **SVD**:

$$\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \Bigg| \begin{bmatrix} 0 & & \\ & \ddots & \\ & & 0 \end{bmatrix}.$$

Often these are just called “diagonal matrices” too.

Important matrix classes

- A **full matrix** aka **dense matrix**: most entries are nonzero.
- A **sparse matrix**: many entries are zero or few entries are nonzero.
Arises in many fields including medical imaging (tomography) [7]
(Obviously terms like “most” “many” “few” are qualitative.)
- **Toeplitz**: constant along each diagonal (not necessarily square)
Arises in any application that has **time invariance** or **shift invariance**
- **Circulant**: each row is a shifted (circularly to the right by one column) version of the previous row.
Arises when considering periodic boundary conditions, *e.g.*, with the DFT

Which statement is correct?

- A: All Toeplitz matrices are circulant.
- B: All circulant matrices are Toeplitz.
- C: Both are true.
- D: Neither statement is true.

??

Which statement is correct?

- A: All Toeplitz matrices are full.
- B: All sparse matrices are Toeplitz.
- C: There are no sparse Toeplitz matrices.
- D: None of these statements is true.

??

What kind of matrix is A in (1.3)? Choose most specific correct answer.

- A: Diagonal
- B: Toeplitz
- C: Circulant
- D: Upper Hessenberg
- E: None

??

What kind of matrix is the convolution matrix H (1.1) on p. 1.6? Choose most specific correct answer.

A: Diagonal

B: Toeplitz

C: Circulant

D: Upper Hessenberg

E: None

??

??

Block matrix classes

The above definitions of shapes and classes were defined in terms of the scalar entries of a matrix. We generalize these definitions by replacing scalars with matrices, leading to **block matrix** shapes. We again use square brackets to denote the construction of a block matrix.

Example: **block diagonal matrix**

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 \end{bmatrix}$$

where \mathbf{A}_1 and \mathbf{A}_2 are arbitrary (possibly full) matrices and each $\mathbf{0}$ denotes an all-zero matrix of the appropriate size.

If \mathbf{A}_1 is 6×8 and \mathbf{A}_2 is 7×9 what is the size of the $\mathbf{0}$ matrix in the upper right?

A: 6×7

B: 6×9

C: 7×8

D: 8×7

E: 8×9

??

Example: **block circulant matrix**

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{C} \\ \mathbf{C} & \mathbf{A} & \mathbf{B} \\ \mathbf{B} & \mathbf{C} & \mathbf{A} \end{bmatrix}$$

Combinations of block shapes and the shape of each block are also important.

Example: a matrix form of the 2D DFT is **block circulant with circulant blocks** (BCCB). (See EECS 556.)

Matrix transpose and symmetry

Define. The **transpose** of a $M \times N$ matrix \mathbf{A} is denoted \mathbf{A}^\top and is the $N \times M$ matrix whose (i, j) th entry is the (j, i) th entry of \mathbf{A} .

If $\mathbf{A} \in \mathbb{C}^{M \times N}$ then its **Hermitian transpose** (or **conjugate transpose**) is the $N \times M$ matrix whose (i, j) th entry is the complex conjugate of the (j, i) th entry of \mathbf{A} .

Common notations for Hermitian transpose are: \mathbf{A}' , \mathbf{A}^H and \mathbf{A}^* .

These notes will mostly use \mathbf{A}' because that notation matches JULIA.

If \mathbf{A} is real, then $\mathbf{A}' = \mathbf{A}^\top$ so these notes will mostly use \mathbf{A}' regardless of whether \mathbf{A} is real or complex for simplicity of notation.

Define. A (square) matrix \mathbf{A} is called **symmetric** iff $\mathbf{A} = \mathbf{A}^\top$.

Define. A (square) matrix \mathbf{A} is called **Hermitian symmetric** or just **Hermitian** iff $\mathbf{A} = \mathbf{A}'$.

See [1, Example 1.2].

Properties of transpose

- $(\mathbf{A}')' = \mathbf{A}$
- $(\mathbf{A} + \mathbf{B})' = \mathbf{A}' + \mathbf{B}'$ (if \mathbf{A} and \mathbf{B} have same size so that matrix addition is well defined)
- $(\mathbf{AB})' = \mathbf{B}'\mathbf{A}'$ if $\mathbf{A} \in \mathbb{F}^{M \times N}$ and $\mathbf{B} \in \mathbb{F}^{N \times K}$, i.e., if inner dimensions match (see matrix multiplication on p. 1.29)

- And more properties of **transpose** and **conjugate transpose** ...

Practical implementation

Caution: in JULIA: `x'` denotes the **Hermitian transpose** of a (possibly complex) vector or matrix `x`, whereas `x.'` in MATLAB or `transpose(x)` in JULIA denotes the **transpose**.



When performing mathematical operations with complex data, we usually need the Hermitian transpose. However, when simply rearranging how data is stored, sometimes we need only the transpose. Many hours of MATLAB software debugging time are spent on missing or extra periods (*i.e.*, `x'` vs `x.'`) for a transpose! To help avoid this problem, JULIA 0.7 and beyond has deprecated `x.'` so use `transpose(x)` in the rare cases where we have complex data but need a transpose rather than a Hermitian transpose.

Practical considerations:

- The transpose or Hermitian transpose of a **vector** takes negligible time in a modern language like JULIA because the array elements stay in the same order in memory; one just needs to modify the variable type from Array to Adjoint Array. See [julia-tutor-vector](#)
- However, the transpose or Hermitian transpose of a (large) **matrix** requires considerable shuffling of values in memory and should be avoided when possible to save compute time.

Example. To compute $\mathbf{x}' \mathbf{A}' \mathbf{y}$ it may be faster (for large data) to use `(A * x)' * y` instead of `x' * A' * y` because the latter may require a transpose operation (unless the compiler is smart enough to avoid it).

For complex data, `conj(y' * A * x)` is another alternative.

1.3 Multiplication

The defining two operations in **linear algebra** are addition and multiplication of vectors and matrices. Addition is trivial so these notes focus on multiplication.

Vector-vector multiplication

If $\mathbf{x} \in \mathbb{C}^N$ and $\mathbf{y} \in \mathbb{C}^N$ are two vectors (of the same length) then their **dot product** or **inner product** is:

$$\langle \mathbf{y}, \mathbf{x} \rangle = \mathbf{x}' \mathbf{y} = \underbrace{\begin{bmatrix} x_1^* & \dots & x_N^* \end{bmatrix}}_{1 \times N} \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}}_{N \times 1} = \sum_{i=1}^N x_i^* y_i \quad (\text{scalar})$$

Most books use $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{y}' \mathbf{x}$ whereas [1] uses $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}' \mathbf{y}$. These notes will use the common convention.

The dot product is central to **linear discriminant analysis (LDA)**, a topic in pattern recognition and machine learning, for two-class classification. In SP terms, it is at the heart of the **matched filter** for signal detection. The dot product is central to convolution and to neural networks [8], e.g., the **perceptron** [9].

In JULIA: `x' * y` and `*(x', y)` and `x' y` and `dot(x, y)` and `·(x, y)` all perform $\mathbf{x}' \mathbf{y}$. The form `x' y` is remarkably similar to the math. Caution: `x' y` (extra space) does not work.

In contrast, the **outer product** of vector $\mathbf{x} \in \mathbb{C}^M$ with vector of possibly different length $\mathbf{y} \in \mathbb{C}^N$ is the following $M \times N$ matrix:

$$\mathbf{x}\mathbf{y}' = \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}}_{M \times 1} \underbrace{\begin{bmatrix} y_1^* & \cdots & y_N^* \end{bmatrix}}_{1 \times N} = \underbrace{\begin{bmatrix} x_1y_1^* & x_1y_2^* & \cdots & x_1y_N^* \\ \vdots & & & \\ x_My_1^* & x_My_2^* & \cdots & x_My_N^* \end{bmatrix}}_{M \times N}.$$

Later we will see that this outer product is a **rank 1 matrix** (unless either \mathbf{x} or \mathbf{y} are 0).

Outer products are central to matrix decompositions like the SVD discussed soon.

In JULIA: `x * y'` is the most clear syntax, although `*(x, y')` also works.

Colon notation

Next we consider multiplication with a matrix.

First we need some notation because matrix multiplication uses rows and/or columns of a matrix.

For $\mathbf{A} \in \mathbb{F}^{M \times N}$:

- $\mathbf{A}_{:,1}$ denotes the first column of \mathbf{A} (a vector of length M)
- $\mathbf{A}_{1,:}$ denotes the first row of \mathbf{A} (a row vector of length N)

In JULIA: `A[:, 1]` and `A[1, :]` essentially do the same.

Using this colon notation we have the following two ways to think about a matrix.

- Column partition of a $M \times N$ matrix:

$$\mathbf{A} = [\mathbf{A}_{:,1} \ \dots \ \mathbf{A}_{:,N}]$$

- Row partition of a $M \times N$ matrix:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,:} \\ \vdots \\ \mathbf{A}_{M,:} \end{bmatrix}$$

Of course we also have the element-wise way of writing out a $M \times N$ matrix:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \dots & a_{1,N} \\ \vdots & \dots & \vdots \\ a_{M,1} & \dots & a_{M,N} \end{bmatrix},$$

but for multiplication operations we often think about the column or row partitions above instead.

Matrix-vector multiplication

If $A \in \mathbb{F}^{M \times N}$ and $x \in \mathbb{F}^N$, then the **matrix-vector product** $y = Ax$ is a vector of length M defined by

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} a_{1,1}x_1 + \cdots + a_{1,N}x_N \\ a_{2,1}x_1 + \cdots + a_{2,N}x_N \\ \vdots \\ a_{M,1}x_1 + \cdots + a_{M,N}x_N \end{bmatrix}. \quad (1.4)$$

A matrix-vector product requires MN floating-point multiplies and $M(N - 1)$ floating-point additions. In short, it needs $O(MN)$ **floating-point operations (FLOPs)**.

In JULIA (or MATLAB) we simply write `y=A*x`
to perform matrix-vector multiplication $y = Ax$.

The terrific similarity between this syntax and the mathematical expression is a key benefit of such high-level languages.

In contrast, for low-level languages (like C and Fortran), one must either call a function in a numerical linear algebra library or write a double loop as shown to the right, looking nothing like the math.

```
(M, N) = size(A)
y = similar(x, M) # preallocate!
for m=1:M
    inprod = 0 # accumulator
    for n=1:N
        inprod += A[m, n] * x[n]
    end
    y[m] = inprod
end
```

The **matrix-vector product** (1.4) has two mathematically (but not practically) equivalent vector forms:

$$\mathbf{A} \in \mathbb{F}^{M \times N}, \mathbf{x} \in \mathbb{F}^N \implies \mathbf{Ax} = \underbrace{\begin{bmatrix} \mathbf{A}_{1,:} \mathbf{x} \\ \vdots \\ \mathbf{A}_{M,:} \mathbf{x} \end{bmatrix}}_{M \text{ "inner products"}} = \underbrace{\sum_{n=1}^N \mathbf{A}_{:,n} x_n}_{\text{linear combination of columns}} = \mathbf{A}_{:,1}x_1 + \cdots + \mathbf{A}_{:,N}x_N. \quad (1.5)$$

In JULIA, instead of using `y = A * x` we could instead implement matrix-vector multiplication using either of the following two loops, corresponding to the two vector forms in (1.5):

```
y = similar(x, M) # preallocate
for m=1:M
    y[m] = transpose(A[m, :]) * x
end
```

```
y = zeros(eltype(x), M) # init 0
for n=1:N
    y .+= A[:, n] .* x[n]
end
```

For efficiency reasons and for better readability, in JULIA, unlike in MATLAB, one must must preallocate space for the result vector in the left (inner product) form so that memory does not grow with iteration.



For a comparison of the compute efficiency of these implementations and optimized variants, see:

<https://web.eecs.umich.edu/~fessler/course/551/julia/tutor/julia-tutor-multiply-matrix-vector.html>

Matrix-vector multiplication with transpose

If \mathbf{A} is a $M \times N$ matrix and \mathbf{y} is a $M \times 1$ vector, then the (Hermitian) transposed **matrix-vector product** is

$$\mathbf{A}'\mathbf{y} = \underbrace{\begin{bmatrix} (\mathbf{A}_{:,1})' \mathbf{y} \\ \vdots \\ (\mathbf{A}_{:,N})' \mathbf{y} \end{bmatrix}}_{N \text{ inner products}} = \underbrace{\sum_{m=1}^M (\mathbf{A}_{m,:})' y_m}_{\text{linear combination}}$$

In JULIA (and MATLAB), a 2D array is stored in memory with its first index varying fastest.

Example. If $\mathbf{A} = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$ then `A[:]` or `vec(A)` both reveal the memory storage order: $\begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$.

Computing $\mathbf{y} = \mathbf{A}\mathbf{x}$ via inner products means $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 5x_1 + 7x_2 \\ 6x_1 + 8x_2 \end{bmatrix}$.

Here the memory access order is non-sequential: the top row uses 5 and 7, which are not adjacent in memory. For large arrays, non-sequential access can lead to slower execution time because of poor memory cache use.

In contrast, computing $\mathbf{x} = \mathbf{A}'\mathbf{y}$ via inner products means $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5y_1 + 6y_2 \\ 7y_1 + 8y_2 \end{bmatrix}$.

Here the elements of \mathbf{A} are accessed sequentially, improving cache use.

The following test compares matrix-vector multiplication versus transposed-matrix-vector multiplication. Both Ax and $A'y$ require $O(MN)$ FLOPs. The comments show time for experiments on a 2012 MacBook Pro, 2.6GHz 4-core CPU, OS 10.12.6, for two versions of JULIA.

```
M = 1000; N = M; A = rand(M,N); x = rand(N); y = rand(M);
A*x; A'y; transpose(A)*y; # warmup, versn 0.6.4 0.7.0
@time for k=1:1000; A*x; end          # 0.23 0.23
@time for k=1:1000; A'y; end          # 0.19 0.19
@time for k=1:1000; A'*y; end         # 0.19 0.19
@time for k=1:1000; transpose(A)*y; end # 5.69 0.24
```

- $A' * y$ is a bit faster than $A * x$ for the reasons discussed above.
- `transpose(A)` used to be a very slow operation; even though it needs no floating point computations, the memory reordering takes time.
- Clearly the compiler is smart enough that $A'y$ does not actually transpose A , but rather sets an internal flag so that it can perform $A'y$ efficiently.

Since JULIA 0.7, the same efficiency holds for `transpose(A) *y`.

In light of these considerations, which of the following weighted inner product calculations $y'Ax$ is likely to run fastest:

A: $y' * (A*x)$

B: $(y' *A) *x$

C: Roughly same time

??

```
M = 1000; N = M; A = rand(M,N); x = rand(N); y = rand(M);
y'* (A*x); (y'*A)*x; transpose(y)*A*x; # warmup 0.6.4 1.0.0
@time for k=1:4000; y'* (A*x); end      # 0.96 0.88 sec
@time for k=1:4000; (y'*A)*x; end      # 0.83 0.75 sec
@time for k=1:4000; (y'*A*x); end      # 0.83 0.76 sec
@time for k=1:4000; transpose(y)*A*x; end # 0.83 0.75 sec
```

- Here the JULIA compiler is smart enough to do it in the appropriate order, so no parentheses are needed.
- Vector transpose takes no time because no memory shuffling is needed.

But now consider the following very similar computation of the weighted inner product $x' A' y$.

```
M = 1000; N = M; A = rand(M,N); x = rand(N); y = rand(M);
x'* (A'*y); (x'*A')*y; (x'*transpose(A))*y; # warmup 0.6.4 1.0.0
@time for k=1:1000; x'* (A'*y); end      # 0.20 0.20 sec
@time for k=1:1000; (x'*A')*y; end      # 0.24 0.22 sec
@time for k=1:1000; (x'*A'*y); end      # 5.76 0.23 sec
@time for k=1:1000; (x'*transpose(A))*y; end # 5.75 0.22 sec
```

- Here the compiler is *not* smart enough to determine the best execution order.
(Apparently it parses things left to right and performs an unnecessary transpose.)
- Here, using appropriate parentheses can (dramatically in 0.6.4, slightly in 1.0.0) accelerate the code!
- A minute of thinking and a few seconds of typing (parentheses) can save compute time (and energy).

Matrix-matrix multiplication

L§1.2

(Includes matrix-vector and vector-vector multiplication as special cases.)

Let $\mathbf{A} \in \mathbb{F}^{M \times K}$ and $\mathbf{B} \in \mathbb{F}^{K \times N}$ then the result of the **matrix-matrix product** is $\mathbf{C} = \mathbf{AB} \in \mathbb{F}^{M \times N}$.

Define. The standard definition for **matrix multiplication**: is

$$C_{ij} = \sum_{k=1}^K A_{ik} B_{kj}, \quad \text{for } i = 1, \dots, M, j = 1, \dots, N. \quad (1.6)$$

Matrix multiplication properties

- The **distributive property** of matrix multiplication is: $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$ if the sizes match appropriately.
- There is also **associative property**: $\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C}$ if the sizes match.
- There is no general **commutative property**! In general $\mathbf{AB} \neq \mathbf{BA}$ even if the sizes match.
- Multiplication by an (appropriately sized) identity matrix has no effect: $\mathbf{IA} = \mathbf{AI} = \mathbf{A}$.
If \mathbf{A} is $M \times N$, then we sometimes write: $\mathbf{I}_M \mathbf{A} = \mathbf{A} \mathbf{I}_N = \mathbf{A}$.
- Matrix **concatenation**: if $\mathbf{A} \in \mathbb{F}^{M \times N}$ and $\mathbf{B} \in \mathbb{F}^{N \times K}$ and $\mathbf{C} \in \mathbb{F}^{N \times L}$ then $\mathbf{A} [\mathbf{B} \ \mathbf{C}] = [\mathbf{AB} \ \mathbf{AC}]$.
Exercise: find the corresponding property for vertical concatenation.
- See [\[wiki\]](#) for more matrix multiplication properties.



Four views of matrix multiplication

Besides formula (1.6), there are four (!) ways of viewing (and implementing) **matrix-matrix product**s.

Why would you need four different versions?

For big data and distributed computing using **parallel processing** some versions are preferable to others [10].

Version 1 (dot or inner product form)

Using row partition of \mathbf{A} and column partition of \mathbf{B} , one can verify:

$$\mathbf{C} = \mathbf{AB} = \underbrace{\begin{bmatrix} \mathbf{A}_{1,:} \\ \vdots \\ \mathbf{A}_{M,:} \end{bmatrix}}_{M \times K} \underbrace{\begin{bmatrix} \mathbf{B}_{:,1} & \dots & \mathbf{B}_{:,N} \end{bmatrix}}_{K \times N} = \underbrace{\begin{bmatrix} \mathbf{A}_{1,:}\mathbf{B}_{:,1} & \dots & \mathbf{A}_{1,:}\mathbf{B}_{:,N} \\ \vdots \\ \mathbf{A}_{M,:}\mathbf{B}_{:,1} & \dots & \mathbf{A}_{M,:}\mathbf{B}_{:,N} \end{bmatrix}}_{M \times N}.$$

Specifically, the elements of \mathbf{C} are given by the following (conjugate-less) **inner products**:

$$C_{ij} = \underbrace{\mathbf{A}_{i,:}}_{1 \times K} \underbrace{\mathbf{B}_{:,j}}_{K \times 1}.$$

In JULIA code:

Simple but opaque: `C = A * B`

Double loop of vector “inner products:”

```
C = zeros(eltype(A), M, N)
for m=1:M
    for n=1:N
        C[m, n] = transpose(A[m, :]) * B[:, n]
    end
end
```

Triple loop completely in terms of scalars per (1.6):

```
C = zeros(eltype(A), M, N)
for m=1:M
    for n=1:N
        inprod = 0 # accumulator
        for k=1:K
            inprod += A[m, k] * B[k, n]
        end
        C[m, n] = inprod
    end
end
```

The transpose `transpose(A)` (not Hermitian transpose `A'`) is crucial for complex matrices!

Instead of: `C[m, n] = transpose(A[m, :]) * B[:, n]`

you could use: `C[m, n] = dot(conj(A[m, :]), B[:, n])`

or `C[m, n] = sum(A[m, :] .* B[:, n])`

Note: For a $K \times N$ matrix B , in JULIA `B[k, :]` returns a 1D column vector of length N , not a $N \times 1$ array nor a “row vector (a $1 \times N$ array). See [julia-tutor-vector](#).



Version 2 (column-wise accumulation)

Using column partition of \mathbf{A} and elements of \mathbf{B} :

$$\mathbf{C} = \mathbf{AB} = [\mathbf{A}_{:,1} \ \dots \ \mathbf{A}_{:,K}] \begin{bmatrix} B_{1,1} & \dots & B_{1,N} \\ \vdots & & \\ B_{K,1} & \dots & B_{K,N} \end{bmatrix}$$

$$\implies \mathbf{C}_{:,n} = [\mathbf{A}_{:,1} \ \dots \ \mathbf{A}_{:,K}] \begin{bmatrix} B_{1,n} \\ \vdots \\ B_{K,n} \end{bmatrix} = \sum_{k=1}^K \mathbf{A}_{:,k} B_{k,n}.$$

In JULIA code we would loop over columns of \mathbf{A} , performing vector-scalar multiplication:

```
C = zeros(eltype(A), M, N) # must preallocate in julia, unlike in matlab
for n=1:N
    for k=1:K
        C[:,n] += A[:,k] * B[k,n]
    end
end
```

Note the “`+ =`” accumulation operation in this code akin to C.

Version 3 (matrix-vector products)

Writing just B using column partition, one can verify:

$$\begin{aligned} C = AB &= \underbrace{A}_{M \times K} \underbrace{\begin{bmatrix} B_{:,1} & \dots & B_{:,N} \end{bmatrix}}_{K \times N} = \underbrace{\begin{bmatrix} AB_{:,1} & \dots & AB_{:,N} \end{bmatrix}}_{M \times N} \\ &\implies C_{:,j} = AB_{:,j} \end{aligned}$$

JULIA code using a single loop over columns of B of matrix-vector products:

```
C = zeros(eltype(A), M, N)
for n=1:N
    C[:,n] = A * B[:,n]
end
```

Practical tip. By default, using `C = zeros(M, N)` would make a `Float64` array. That is fine for small problems and when very good numerical precision is needed. To save memory for large problems (at the price of reduced precision) use `zeros(Float32, M, N)`. If desperate for memory savings, consider `zeros(Float16, M, N)`. Here we use `eltype(A)` so that `C` matches the precision of `A`.

This class will mostly use the above **matrix-vector** operation view.

Version 4 (sum of outer products)

Now using column partition of \mathbf{A} and row partition of \mathbf{B} :

$$\mathbf{C} = \mathbf{AB} = \underbrace{\begin{bmatrix} \mathbf{A}_{:,1} & \dots & \mathbf{A}_{:,K} \end{bmatrix}}_{M \times K} \underbrace{\begin{bmatrix} \mathbf{B}_{1,:} \\ \vdots \\ \mathbf{B}_{K,:} \end{bmatrix}}_{K \times N} = \sum_{k=1}^K \underbrace{\mathbf{A}_{:,k}}_{M \times 1} \underbrace{\mathbf{B}_{k,:}}_{1 \times N} \quad (1.7)$$

For proof, see [1, Theorem 1.3].

JULIA code using a single loop (over inner dimension) of **outer products**:

```
C = zeros(eltype(A), M, N)
for k=1:K
    C .+= A[:,k] * transpose(B[k,:]) # column times row!
end
```

Block matrix multiplication

The inner operation in (1.7) is an example of **block matrix multiplication** and the definition for such operations is basically the same as (1.6) as long as all the matrix dimensions involved match properly.

Using matrix-vector operations in high-level computing languages

(Read)

Particularly in high-level, array-oriented languages like JULIA, matrix and vector operations abound. One should be on the lookout for opportunities to “parallelize” (vectorize) using such operations.

Example. Suppose we want to visualize the 2D Gaussian bump function $f(x, y) = e^{-(x^2+3y^2)}$.

Elementary implementation with double loop:

```
using Plots; plotly()
x = LinRange(-2, 2, 101)
y = LinRange(-1.1, 1.1, 103)
M = length(x)
N = length(y)
F = zeros(M,N)
for m=1:M
    for n=1:N
        F[m,n] = exp(-(x[m]^2 + 3 * y[n]^2))
    end
end
heatmap(x, y, F, transpose=true, color=:grays, aspect_ratio=:equal)
```

How do we leverage matrix-vector concepts to write this more concisely?

(Concise code is often easier to read and maintain, and often looks more like the mathematical expressions.)

Idea: Define a matrix \mathbf{A} such that $A_{ij} = x_i^2 + 3y_j^2$.

Then use element-wise exponential operation: $\mathbf{F} = \exp . (-\mathbf{A})$

In JULIA, `exp . (A)` applies exponentiation **element-wise** to an array.

In JULIA 0.6.4, `exp (A)` gives a warning; use `expm (A)` for a **matrix exponential**. 

How do we define \mathbf{A} where $A_{ij} = x_i^2 + 3y_j^2$ without writing a double loop?

One way is to use outer products. If $\mathbf{u} \in \mathbb{R}^M$ has elements $u_i = x_i^2$ and $\mathbf{v} \in \mathbb{R}^N$ has elements $v_j = y_j^2$, then the following **sum of outer products** yields a $M \times N$ matrix:

$$\begin{aligned} \mathbf{A} &= \mathbf{u}\mathbf{1}'_N + 3\mathbf{1}_M\mathbf{v}' = \begin{bmatrix} x_1^2 \\ \vdots \\ x_M^2 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & \dots & 1 \end{bmatrix}}_{1 \times N} + 3 \underbrace{\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}}_{M \times 1} \begin{bmatrix} y_1^2 & \dots & y_N^2 \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} x_1^2 & \dots & x_1^2 \\ \vdots & \vdots & \vdots \\ x_M^2 & \dots & x_M^2 \end{bmatrix}}_{\text{repeated column}} + 3 \underbrace{\begin{bmatrix} y_1^2 & \dots & y_N^2 \\ \vdots & \vdots & \vdots \\ y_1^2 & \dots & y_N^2 \end{bmatrix}}_{\text{repeated row}}, \end{aligned} \quad (1.8)$$

where $\mathbf{1}_N$ denotes the vector of all ones, i.e., `ones (N)` in JULIA or `ones (N, 1)` in MATLAB.

Here we need `ones (1, N)` and `ones (M)`.

The key line of the following JULIA code looks reasonably close to the above outer-product form.

```
using Plots; plotly()
x = LinRange(-2, 2, 101)
y = LinRange(-1.1, 1.1, 103)
M = length(x)
N = length(y)
A = (x.^2) * ones(1,N) + 3 * ones(M,1) * (y.^2)'
F = exp.(-A)
heatmap(x, y, F, transpose=true, color=:grays, aspect_ratio=:equal)
```

This version avoids you, the user, from writing a double loop. Of course JULIA itself must do double loops internally when evaluating `exp.(-A)` and similar expressions.

The outer product approach has the benefit of avoiding writing double loops. However, this type of operation arises so frequently that JULIA provides software functions that avoid the unnecessary operations of multiplying by the `ones` vector.

In JULIA there is automatic **broadcast** of singleton dimensions by the scalar addition operator:

```
A = x.^2 .+ 3 * (y.^2)'
```

Two somewhat distinct uses of “`.`” here:

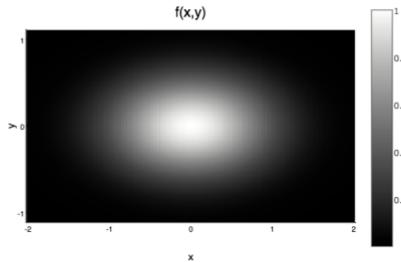
- `.^2` element-wise power
- `.+` normally element-wise addition of arrays but here with automatic broadcast like in (1.8).

This broadcast feature leads to the most concise code that looks the most like the math:

```
using Plots; plotly()
x = LinRange(-2, 2, 101)
y = LinRange(-1.1, 1.1, 103)
A = x.^2 .+ 3 * (y.^2)' # a lot is happening here!
F = exp.(-A)
heatmap(x, y, F, transpose=true, color=:grays, aspect_ratio=:equal)
```

Finally, if your goal is merely to make a picture of the function $f(x, y)$, without illustrating any matrix properties, the following is a “JULIA way” to do it using its **multiple dispatch** feature. This version is the shortest of all so I added a few labeling commands.

```
using Plots; plotly()
x = LinRange(-2, 2, 101)
y = LinRange(-1.1, 1.1, 103)
f(x,y) = exp(-(x^2 + 3y^2)) # look, no "*" !
heatmap(x, y, f, transpose=true, color=:grays, aspect_ratio=:equal)
xlabel!("x"); ylabel!("y"); title!("f(x,y)")
```



https://web.eecs.umich.edu/~fessler/course/551/julia/demo/01_gauss2d.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/01_gauss2d.ipynb

Example. One can perform 1D numerical integration using a vector operation (a dot product).

(Read)

To compute the area under a curve:

$$\text{Area} = \int_a^b f(x) dx \approx \sum_{m=1}^M \underbrace{(x_m - x_{m-1})}_{\text{base}} \underbrace{f(x_m)}_{\text{height}} = \mathbf{w}' \mathbf{f}, \quad \mathbf{w} = \begin{bmatrix} x_1 - x_0 \\ \vdots \\ x_M - x_{M-1} \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_M) \end{bmatrix}, \quad (1.9)$$

where (possibly nonuniformly spaced) sampled points satisfy: $a = x_0 < x_1 < \dots < x_M = b$.

This summation is a **dot product** between two vectors in \mathbb{R}^M and is easy in JULIA.

```
f(x) = x^2 # parabola
x = LinRange(0, 3, 2000) # sample points
w = diff(x) # "widths" of rectangles
Area = w' * f.(x[2:end])
```

What should the exact value be for the area in this example? $\int_0^3 x^2 dx = x^3/3|_{x=3} = 9$

Why `x[2:end]` instead of `x[1:end]` or simply `x`?

Because `w = diff(x)` returns a vector without the first element “ $x_1 - x_0$ ” in (1.9).

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/01_area.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/01_area.ipynb

Example. Consider the problem of computing numerically the volume under a 2D function $g(x, y)$:

$$V = \int_a^b \int_c^d g(x, y) \, dy \, dx .$$

We can also use matrix-vector products for this operation:

$$V = \int_a^b \int_c^d g(x, y) \, dy \, dx \approx S \triangleq \sum_{m=1}^M \sum_{n=1}^N (x_m - x_{m-1})(y_n - y_{n-1}) f(x_m, y_n),$$

where here $c = y_0 < y_1 < \dots < y_N = d$.

Define vector $\mathbf{w} \in \mathbb{R}^M$ as in 1D example above and $\mathbf{u} \in \mathbb{R}^N$ and $\mathbf{F} \in \mathbb{R}^{M \times N}$ by

$$\mathbf{w} = \begin{bmatrix} x_1 - x_0 \\ \vdots \\ x_M - x_{M-1} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} y_1 - y_0 \\ \vdots \\ y_N - y_{N-1} \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} f(x_1, y_1) & \dots & f(x_1, y_N) \\ \vdots & & \vdots \\ f(x_M, y_1) & \dots & f(x_M, y_N) \end{bmatrix} .$$

Then the above double sum is the following product in matrix-vector form:

$$V \approx S = \mathbf{w}' \mathbf{F} \mathbf{u}.$$

Proof.

$$\begin{aligned} \mathbf{w}' \mathbf{F} \mathbf{u} &= [w_1 \ \dots \ w_M] \begin{bmatrix} f(x_1, y_1) & \dots & f(x_1, y_N) \\ \vdots & & \vdots \\ f(x_M, y_1) & \dots & f(x_M, y_N) \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} \\ &= [w_1 \ \dots \ w_M] \begin{bmatrix} \sum_{n=1}^N u_n f(x_1, y_n) \\ \vdots \\ \sum_{n=1}^N u_n f(x_M, y_n) \end{bmatrix} = \sum_{m=1}^M w_m \sum_{n=1}^N u_n f(x_m, y_n) = S. \end{aligned}$$

Again this computation is easy to code in a high-level language like JULIA.

```
f(x, y) = exp(-(x^2 + 3*y^2)) # gaussian bump function
x = LinRange(0, 3, 2000) # sample points
y = LinRange(0, 2, 1000) # sample points
w = diff(x) # "widths" of rectangles in x
u = diff(y) # "widths" of rectangles in y
F = f.(x[2:end], y[2:end]') # automatic broadcasting again!
S = w' * F * u
```

(It would also be fine to write it as a double loop, albeit with more typing and possibly slower in some languages.)

In this case it is possible to determine the analytical value (*cf.* EECS 501) but that would probably take more time than writing and running this code.

Invertibility

(Read)

For a nonzero scalar x , we know that $x \frac{1}{x} = 1$. Matrix inversion is the generalization of this identity.

If $\mathbf{A} \in \mathbb{F}^{M \times N}$ and $\mathbf{B} \in \mathbb{F}^{N \times M}$ and $\mathbf{B}\mathbf{A} = \mathbf{I}_{N \times N}$ then we call \mathbf{B} a **left inverse** of \mathbf{A} .

If and $\mathbf{C} \in \mathbb{F}^{N \times M}$ and $\mathbf{A}\mathbf{C} = \mathbf{I}_{M \times M}$ then we call \mathbf{C} a **right inverse** of \mathbf{A} .

For non-square matrices, at most only one of a left or right inverse can exist, and is not unique.

A square matrix \mathbf{A} is called **invertible** iff there exists a matrix \mathbf{X} of the same size such that $\mathbf{AX} = \mathbf{XA} = \mathbf{I}$.

Fact. For square matrices, a left inverse exists iff only a right inverse exists and the two are identical [wiki].

Proving this fact seems to be a bit subtle. See:

<https://math.stackexchange.com/questions/216569/assuming-ab-i-prove-ba-i>

Basic matrix inversion properties

If \mathbf{A} is an **invertible matrix**:

- $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
- $c \neq 0 \implies (c\mathbf{A})^{-1} = \frac{1}{c}\mathbf{A}^{-1}$
- \mathbf{A}' is also invertible and $(\mathbf{A}')^{-1} = (\mathbf{A}^{-1})'$
- $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$ if \mathbf{B} is invertible and has same size as \mathbf{A}

Trying to “avoid” matrix inversion

(Read)

Typical methods for inverting a $N \times N$ matrix use $O(N^3)$ operations, which is very expensive for large matrices, so we often seek matrix identities to reduce computation when possible.

- The following inverse of 2×2 block matrices holds if \mathbf{A} and \mathbf{B} are invertible (and if sizes match):

$$\begin{bmatrix} \mathbf{A} & \mathbf{D} \\ \mathbf{C} & \mathbf{B} \end{bmatrix}^{-1} = \begin{bmatrix} [\mathbf{A} - \mathbf{D}\mathbf{B}^{-1}\mathbf{C}]^{-1} & -\mathbf{A}^{-1}\mathbf{D}\Delta^{-1} \\ -\Delta^{-1}\mathbf{C}\mathbf{A}^{-1} & \Delta^{-1} \end{bmatrix}, \quad (1.10)$$

where $\Delta = \mathbf{B} - \mathbf{C}\mathbf{A}^{-1}\mathbf{D}$ denotes the **Schur complement** of \mathbf{A} for this matrix.

- Using the associative property and some rearranging yields the **Sherman-Morrison-Woodbury identity** also known as the **matrix inversion lemma**:

$$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B} (\mathbf{C}^{-1} + \mathbf{D}\mathbf{A}^{-1}\mathbf{B})^{-1} \mathbf{D}\mathbf{A}^{-1}, \quad (1.11)$$

provided the matrices have compatible sizes and \mathbf{A} and \mathbf{C} are invertible.

- A special case that is useful if we have previously computed the inverse of \mathbf{A} and now need the inverse of the “rank-one update” $\mathbf{A} + \mathbf{xy}'$, if it is invertible, is the **Sherman-Morrison formula**:

$$(\mathbf{A} + \mathbf{xy}')^{-1} = \mathbf{A}^{-1} - \frac{1}{1 + \mathbf{y}'\mathbf{A}^{-1}\mathbf{x}} \mathbf{A}^{-1}\mathbf{xy}'\mathbf{A}^{-1}.$$

- Multiplying (1.11) on the right by \mathbf{B} and simplifying yields the following useful related equality, sometimes called the **push-through identity**:

$$[\mathbf{A} + \mathbf{BCD}]^{-1} \mathbf{B} = \mathbf{A}^{-1}\mathbf{B} [\mathbf{D}\mathbf{A}^{-1}\mathbf{B} + \mathbf{C}^{-1}]^{-1} \mathbf{C}^{-1}. \quad (1.12)$$

Challenge: prove (or disprove) that if \mathbf{A} and \mathbf{C} are invertible,
then $(\mathbf{A} + \mathbf{B}\mathbf{C}\mathbf{D})$ is invertible iff $(\mathbf{C}^{-1} + \mathbf{D}\mathbf{A}^{-1}\mathbf{B})$ is invertible, so that (1.11) is self consistent.

More invertible matrix properties

(Read)

The following properties of any invertible matrix \mathbf{A} relate to topics *discussed later in the chapter*:

- \mathbf{A} has linearly independent columns
- \mathbf{A} has full rank
- If \mathbf{A} is unitary then $\mathbf{A}^{-1} = \mathbf{A}'$
- The determinant of \mathbf{A} is nonzero and $\det\{\mathbf{A}^{-1}\} = 1/\det\{\mathbf{A}\}$.

1.4 Orthogonality

For ordinary scalars: $0 \cdot x = 0$.

For vectors and matrices, there are more interesting ways to multiply and end up with zero!

Orthogonal vectors

L§1.3

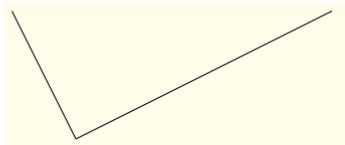
Define. We say two vectors \mathbf{x} and \mathbf{y} (of same length) are **orthogonal** (or **perpendicular**) if their inner product is zero: $\mathbf{x}'\mathbf{y} = 0$. In such cases we write $\mathbf{x} \perp \mathbf{y}$.

If, in addition, $\mathbf{x}'\mathbf{x} = \mathbf{y}'\mathbf{y} = 1$ (*i.e.*, both **unit norm**), then we call them **orthonormal** vectors.

Define. A collection of vectors that are pairwise orthogonal is called an **orthogonal set**.

A collection of **unit-norm** vectors that are pairwise orthogonal is called an **orthonormal set**.

Example. In \mathbb{R}^3 , the vectors $\mathbf{v}_1 = (4, 2, 0)$ and $\mathbf{v}_2 = (-1, 2, 0)$ are orthogonal (but not orthonormal).



Euclidean norm

Define. The (Euclidean) **norm** of a vector $\mathbf{x} \in \mathbb{F}^N$ is defined by

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\mathbf{x}' \mathbf{x}} = \sqrt{\sum_{n=1}^N |x_n|^2}. \quad (1.13)$$

Later we will write $\|\mathbf{x}\|_2$ when needed, but for now we focus on the Euclidean norm, aka **2 norm**.

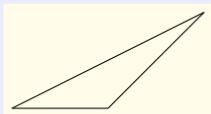
Properties of the Euclidean norm

(Read)

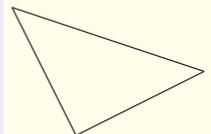
- **Triangle inequality:** $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$
- Quadratic expansion:

$$\|\mathbf{u} + \mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 + 2 \operatorname{real}\{\mathbf{u}' \mathbf{v}\} + \|\mathbf{v}\|_2^2 \quad (1.14)$$

Proof: $\|\mathbf{u} + \mathbf{v}\|_2^2 = \sum_{i=1}^n |u_i + v_i|^2 = \sum_{i=1}^n (u_i + v_i)(u_i + v_i)^* = \sum_{i=1}^n |u_i|^2 + 2 \operatorname{real}\{u_i^* v_i\} + |v_i|^2$



- **Pythagorean theorem:** $\mathbf{x} \perp \mathbf{y} \implies \|\mathbf{u} + \mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2$



Cauchy-Schwarz inequality

The **Cauchy-Schwarz inequality** (or **Schwarz** or **Cauchy-Bunyakovsky-Schwarz** inequality) relates inner products and norms as follows:

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} \sqrt{\langle \mathbf{y}, \mathbf{y} \rangle}, \quad \forall \mathbf{x}, \mathbf{y} \in \mathcal{V}, \quad (1.15)$$

where $\|\cdot\|$ denotes the Euclidean norm on \mathbb{F}^N .

For a proof, see Ch. 5.

Angle between vectors

Define. The **angle** θ between two nonzero vectors $\mathbf{x}, \mathbf{y} \in \mathcal{V}$ is defined by

$$\cos \theta = \frac{|\langle \mathbf{x}, \mathbf{y} \rangle|}{\|\mathbf{x}\| \|\mathbf{y}\|} \implies \theta \in [0, \pi/2].$$

The Cauchy-Schwarz inequality is equivalent to the statement $|\cos \theta| \leq 1$.

What is the angle between two orthogonal vectors?

A: 0

B: 1

C: $\pi/2$

D: π

E: None of these

??

Orthogonal matrices

Define. We say a (square) matrix $\mathbf{Q} \in \mathbb{R}^{N \times N}$ is an **orthogonal matrix** iff $\mathbf{Q}^\top \mathbf{Q} = \mathbf{Q} \mathbf{Q}^\top = \mathbf{I}$.

(Personally I think the term “orthonormal matrix” would be more appropriate, but alas.)

Define. We say a (usually complex) square matrix $\mathbf{Q} \in \mathbb{C}^{N \times N}$ is a **unitary matrix** iff $\mathbf{Q}' \mathbf{Q} = \mathbf{Q} \mathbf{Q}' = \mathbf{I}$.

The set of columns of a **unitary matrix** is an **orthonormal set**. (So is the set of rows.)



The set of columns of an **orthogonal matrix** is **orthonormal set**. (So is the set of rows.)

A interesting generalization is a **tight frame** where only one of the two conditions holds [11, 12].



Unitary / orthogonal matrices are a key building block in this course.

If a (possibly complex) matrix \mathbf{A} has orthonormal columns, it is unitary. (?)

A: True

B: False

??

If a square, real matrix \mathbf{A} has orthogonal columns, then it is an orthogonal matrix. (?)

A: True

B: False

??

Invertibility of unitary matrices

(Read)

A key property of orthogonal and unitary matrices is that their inverse is simply their transpose: $\mathbf{Q}^{-1} = \mathbf{Q}'$.

In other words, the easiest (non-diagonal) matrices to invert are unitary matrices.

Recall \mathbf{Q} is unitary iff

$$\mathbf{Q}'\mathbf{Q} = \mathbf{Q}\mathbf{Q}' = \mathbf{I}. \quad (1.16)$$

Thus by definition of matrix inverse we have

$$\mathbf{Q}^{-1} = \mathbf{Q}'. \quad (1.17)$$

Because of the equivalence of the left and right inverses for invertible square matrices, we really only need one of the two conditions in (1.16) to conclude the other and (1.17). So often people just write $\mathbf{Q}'\mathbf{Q} = \mathbf{I}$ when mentioning that a (square) matrix \mathbf{Q} is unitary.

Norm invariance to rotations

Orthogonal/unitary matrices act somewhat like rotation operations. An important property of $N \times N$ **orthogonal** or **unitary** matrices is that they do not change the **Euclidean norm** of a vector:

$$\mathbf{Q} \text{ orthogonal or unitary} \implies \|\mathbf{Q}\mathbf{x}\| = \|\mathbf{x}\|, \quad \forall \mathbf{x} \in \mathbb{C}^N. \quad (1.18)$$

Proof: $\|\mathbf{Q}\mathbf{x}\| = \sqrt{(\mathbf{Q}\mathbf{x})'(\mathbf{Q}\mathbf{x})} = \sqrt{\mathbf{x}'\mathbf{Q}'\mathbf{Q}\mathbf{x}} = \sqrt{\mathbf{x}'\mathbf{I}\mathbf{x}} = \sqrt{\mathbf{x}'\mathbf{x}} = \|\mathbf{x}\|$, using the orthogonality of \mathbf{Q} .

This fact is related to **Parseval's theorem**.

1.5 Matrix determinant

L§1.4

Now we begin discussing important matrix properties. We start with the matrix **determinant** [1, Sect. 1.4], a property that is defined only for square matrices. Data matrices are very rarely square, so we essentially never examine the determinant of a data matrix directly! But if \mathbf{X} is a $M \times N$ data matrix, often we will work with the $N \times N$ **Gram matrix** $\mathbf{X}'\mathbf{X}$ (e.g., when solving least-squares problems) and a Gram matrix is always square. Many operator matrices (like DFT) are also square.

There are many ways to introduce the determinant of a matrix because it has many properties¹. Here we consider the following four “axioms” expressed in terms of matrices.

Define. If $\mathbf{A} \in \mathbb{F}^{N \times N}$ then the determinant of \mathbf{A} is defined so that

- D1: If \mathbf{A} is upper triangular and $N \times N$ then $\det\{\mathbf{A}\} = a_{11} \cdot a_{22} \cdots \cdots a_{NN}$
- D2: If $\mathbf{A}, \mathbf{B} \in \mathbb{F}^{N \times N}$ then $\det\{\mathbf{AB}\} = \det\{\mathbf{A}\} \det\{\mathbf{B}\}$
- D3: $\det\{\mathbf{A}'\} = \det\{\mathbf{A}\}^*$ where z^* denotes the complex conjugate of z .
- D4: If $\mathbf{P}_{i,j} \in \mathbb{R}^{N \times N}$ denotes the matrix that swaps the i th and j th rows, $i \neq j$, then $\det\{\mathbf{P}_{i,j}\} = -1$.

¹ [wiki] lists about 13 properties and claims that a certain set of 3 of them completely characterize the determinant. For our purposes there is no need to make more work for ourselves by using a minimal set so we start with 4 axioms instead.

Fact. A matrix \mathbf{A} is **invertible** iff its determinant is nonzero.

So a linear system of N equations with N unknowns has a unique solution iff the **determinant** corresponding to the coefficients is nonzero. This is the historical reason for the term **determinant** because it “determines” uniqueness of the solution to such a set of equations.

From D2, the following **determinant commutative property** follows immediately:

$$\mathbf{A}, \mathbf{B} \in \mathbb{F}^{N \times N} \implies \det\{\mathbf{AB}\} = \det\{\mathbf{BA}\}. \quad (1.19)$$

Mathematically, a concise definition of the row-swapping matrix $\mathbf{P}_{i,j}$ (a special permutation matrix) used in D4 is

$$\mathbf{P}_{i,j} = \mathbf{I} - e_j e'_j - e_i e'_i + e_j e'_i + e_i e'_j,$$

for $i, j \in \{1, \dots, N\}$, where e_i denotes the i th unit vector.

Example. For $N = 5$: $\mathbf{P}_{1,4} = \mathbf{P}_{4,1} =$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \text{so} \quad \mathbf{P}_{1,4} \begin{bmatrix} \mathbf{A}_{1,:} \\ \mathbf{A}_{2,:} \\ \mathbf{A}_{3,:} \\ \mathbf{A}_{4,:} \\ \mathbf{A}_{5,:} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{4,:} \\ \mathbf{A}_{2,:} \\ \mathbf{A}_{3,:} \\ \mathbf{A}_{1,:} \\ \mathbf{A}_{5,:} \end{bmatrix}.$$

In words, left multiplying a matrix by $\mathbf{P}_{i,j}$ swaps the i th and j th rows of the matrix.

Row (or column) swap property

A consequence of D2 and D4 is that swapping any two rows of a matrix negates the sign of the determinant:

$$i \neq j \implies \det\{\mathbf{P}_{i,j} \mathbf{A}\} = -\det\{\mathbf{A}\}.$$

By D3, the same property holds for swapping two columns.

Column (or row) scaling property

Use the above four axioms to prove that multiplying a column of \mathbf{A} by a scalar gives a new matrix whose determinant scales by that scalar factor:

$$\mathbf{B} = [\mathbf{a}_1, \dots, \mathbf{a}_{n-1}, b\mathbf{a}_n, \mathbf{a}_{n+1}, \dots, \mathbf{a}_N] \implies \det\{\mathbf{B}\} = b \det\{\mathbf{A}\}.$$

Proof. Simply write $\mathbf{B} = \mathbf{AD}$ where $\mathbf{D} = \text{diag}\{1, \dots, 1, b, 1, \dots, 1\}$,
so $\det\{\mathbf{B}\} = \det\{\mathbf{A}\} \det\{\mathbf{D}\} = b \det\{\mathbf{A}\}$.

Note the strategy of writing (linear) operations in terms of matrix products so we can apply D2.

Matrix inversion property

Exercise. Use two of the above “axioms” to prove $\det\{\mathbf{A}^{-1}\} = 1 / \det\{\mathbf{A}\}$.

Row (or column) combination property

Multiplying a row of a matrix by a scalar and adding it to a different row does not change the determinant [1, property 8, p. 5], *i.e.*:

$$\text{if } \mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,:} \\ \vdots \\ \mathbf{A}_{M,:} \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} \mathbf{A}_{1,:} \\ \vdots \\ \mathbf{A}_{m-1,:} \\ b\mathbf{A}_{k,:} + \mathbf{A}_{m,:} \\ \mathbf{A}_{m+1,:} \\ \vdots \\ \mathbf{A}_{M,:} \end{bmatrix}, \text{ with } k \neq m, \text{ then } \det\{\mathbf{B}\} = \det\{\mathbf{A}\}.$$

Proof: $\mathbf{B} = (\mathbf{I} + b\mathbf{e}_m\mathbf{e}'_k)\mathbf{A} \implies \det\{\mathbf{B}\} = \det\{\mathbf{I} + b\mathbf{e}_m\mathbf{e}'_k\} \det\{\mathbf{A}\} = \det\{\mathbf{A}\}$.

Determinant formula

A general rule, called **Laplace's formula**, for a $N \times N$ matrix \mathbf{A} is:

$$\det\{\mathbf{A}\} = \sum_{n=1}^N (-1)^{m+n} a_{m,n} \det\{\mathbf{A}_{m,n}\},$$

for any $m \in \{1, \dots, N\}$, where here $\mathbf{A}_{m,n}$ denotes the submatrix of \mathbf{A} formed by deleting the m th row and n th column and $\det\{\mathbf{A}_{m,n}\}$ is called a **minor** of \mathbf{A} . (I have rarely needed to use this.)

Avoiding computation

Naive methods for computing the determinant of a $N \times N$ matrix would require $O(N!)$ operations, but more efficient **decomposition methods** require $O(N^3)$ operations (or less). This is still expensive for large N , so we often seek properties to use to reduce computation when possible.

- Generalizing property D2 on p. 1.51, if \mathbf{A} is **block upper triangular** (or **block lower triangular**) then its determinant is the product of the determinants of its diagonal blocks:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & \dots \\ 0 & \mathbf{A}_{22} & \mathbf{A}_{23} & \dots \\ \vdots & & \ddots & \\ 0 & \dots & 0 & \mathbf{A}_{KK} \end{bmatrix} \implies \det\{\mathbf{A}\} = \prod_{k=1}^K \det\{\mathbf{A}_{kk}\}. \quad (1.20)$$

- Using **block matrix triangularization** and (1.20), if \mathbf{A} is invertible then [13]:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & 0 \\ \mathbf{C} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{B} \\ 0 & \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} \end{bmatrix} \implies \det\left\{ \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \right\} = \det\{\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\} \det\{\mathbf{A}\}. \quad (1.21)$$

Similarly if \mathbf{D} is invertible then we can find the determinant of a large block matrix in terms of determinants of combinations of its blocks:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{B} \\ 0 & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C} & 0 \\ \mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix} \implies \det\left\{ \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \right\} = \det\{\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}\} \det\{\mathbf{D}\}. \quad (1.22)$$

- The **matrix determinant lemma** says if A is square and invertible and x and y have the appropriate size then the determinant of the “rank-one update” is: (proof)

$$\det\{A + xy'\} = (1 + y' A^{-1} x) \det\{A\}.$$

This property can be useful if the inverse and determinant of A are already known.

Example. Find the determinant of $B = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

$$B = I + 11' \implies \det\{B\} = 1 + 1'I1 = 4. \quad \text{Check: } \det([2 \ 1 \ 1; \ 1 \ 2 \ 1; \ 1 \ 1 \ 2])$$

- Sylvester's determinant identity** can be useful for rectangular matrices $A, B \in \mathbb{F}^{M \times N}$ with $N \ll M$:

$$\det\{I_M + AB'\} = \det\{I_N + B'A\}. \quad (1.23)$$

- More generally:

$$A, B \in \mathbb{F}^{M \times N}, X \in \mathbb{F}^{M \times M} \text{ invertible} \implies \det\{X + AB'\} = \det\{I_N + B'X^{-1}A\} \det\{X\}.$$

The proof follows from (1.21) and (1.22).

Practical use in JULIA:

```
using LinearAlgebra
then det(A)
```

or

```
using LinearAlgebra: det
```

2 by 2 matrix

Use the above properties to show that the determinant of a 2×2 matrix $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is $ad - bc$.

Proof. If a is nonzero, then multiplying the first row by $-c/a$ and adding to the second row yields:

$$\mathbf{B} = \begin{bmatrix} a & b \\ 0 & d - bc/a \end{bmatrix}, \text{ which is upper triangular so has determinant } a(d - bc/a) = ad - bc.$$

If a is zero, then swapping the first and second rows yields $\mathbf{C} = \begin{bmatrix} c & d \\ 0 & b \end{bmatrix}$ which again is upper triangular and the determinant is cb , so the determinant of \mathbf{A} in this case is $-cb = 0d - cb = ad - cb$. \square

Example. $\det\left\{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right\} = 0 \cdot 0 - 1 \cdot 1 = -1$, consistent with D4.

Determinant of 3×3 matrix

If $\mathbf{A} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$ then $\det\{\mathbf{A}\} = a \det\left\{\begin{bmatrix} e & f \\ h & i \end{bmatrix}\right\} - b \det\left\{\begin{bmatrix} d & f \\ g & i \end{bmatrix}\right\} + c \det\left\{\begin{bmatrix} d & e \\ g & h \end{bmatrix}\right\}$.

You should verify this yourself from the properties.

1.6 Eigenvalues

L§9.1

Define. An important property of any square matrix $\mathbf{A} \in \mathbb{F}^{N \times N}$ is its **eigenvalues**, often denoted $\lambda_1, \dots, \lambda_N$, defined as the set of solutions of the **characteristic equation**

$$\det\{\mathbf{A} - z\mathbf{I}\} = 0, \quad (1.24)$$

where \mathbf{I} denotes the identity matrix of the same size as \mathbf{A} .

Viewed as a function of z , we call $\det\{\mathbf{A} - z\mathbf{I}\}$ the **characteristic polynomial**, and the eigenvalues of \mathbf{A} are its roots.

When $\mathbf{A} \in \mathbb{F}^{N \times N}$, the characteristic polynomial has **degree** N and thus N (possibly complex) roots by the **fundamental theorem of algebra**. Thus by the definition (1.24), \mathbf{A} has N eigenvalues (but they are not necessarily distinct).

The literature can be inconsistent about how many eigenvalues a $N \times N$ matrix has. For example, the characteristic polynomial of the 2×2 matrix $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ is $\det\{\mathbf{A} - z\mathbf{I}\} = z^2 = (z - 0)(z - 0)$ which has a repeated root at zero and sometimes is said to have “only one eigenvalue” [wiki]. We will not use that terminology; we will say that matrix has two eigenvalues, both of which are zero.



As mentioned earlier, data matrices are nearly never square so basically we will never need to examine the eigenvalues of a data matrix $\mathbf{X} \in \mathbb{F}^{M \times N}$ directly. But often we will consider the eigenvalues of the square $M \times M$ **Gram matrix** $\mathbf{X}'\mathbf{X}$ or of the square $N \times N$ **outer-product matrix** $\mathbf{X}\mathbf{X}'$ that arises when estimating a **covariance matrix** or a **scatter matrix**.

Example. Determine the eigenvalues of $\mathbf{A} = \mathbf{I}_N + \mathbf{y}\mathbf{y}'$ for $\mathbf{y} \in \mathbb{F}^N$. Using (1.23):

$$\begin{aligned}\det\{\mathbf{A} - z\mathbf{I}_N\} &= \det\{(1-z)\mathbf{I}_N + \mathbf{y}\mathbf{y}'\} = (1-z)^N \det\{\mathbf{I}_N + \mathbf{y}\mathbf{y}'/(1-z)\} \\ &= (1-z)^N \det\{1 + \mathbf{y}'\mathbf{y}/(1-z)\} = (1-z)^{N-1} (1 - z + \|\mathbf{y}\|^2),\end{aligned}$$

which has $N - 1$ roots at 1 and one root at $1 + \|\mathbf{y}\|^2$.

Eigenvectors

If z is an eigenvalue of \mathbf{A} , then (1.24) implies $\mathbf{A} - z\mathbf{I}$ is a **singular matrix** (not invertible), so there exists a nonzero vector \mathbf{v} such that $(\mathbf{A} - z\mathbf{I})\mathbf{v} = 0$ or equivalently

$$\mathbf{Av} = z\mathbf{v}. \tag{1.25}$$

Conversely, if (1.25) holds for a nonzero vector \mathbf{v} , then z is an eigenvalue of \mathbf{A} .

Define. Any nonzero vector \mathbf{v} that satisfies (1.25) is called an **eigenvector** of \mathbf{A} .

Example. For the matrix $A = I_N + \mathbf{y}\mathbf{y}'$ for $\mathbf{y} \in \mathbb{F}^N$, one eigenvector is \mathbf{y} because $A\mathbf{y} = (I_N + \mathbf{y}\mathbf{y}')\mathbf{y} = (\mathbf{y} + \mathbf{y}\mathbf{y}'\mathbf{y}) = (1 + \|\mathbf{y}\|^2)\mathbf{y}$. Any vector of the form $\alpha\mathbf{y}$ for $\alpha \in \mathbb{F}$ is also an eigenvector. All other eigenvectors are orthogonal to \mathbf{y} because if $\mathbf{x} \perp \mathbf{y}$ then $A\mathbf{x} = \mathbf{x}$.

Practical implementation

To find a set of eigenvalues and eigenvectors of a square matrix A in JULIA:

```
using LinearAlgebra  
z,V = eigen(A)
```

To obtain just the eigenvalues use any of

```
eigvals(A)  
eigen(A).values  
z,_ = eigen(A)
```

The underscore `_` output is discarded.

To obtain just a set of eigenvectors use any of

```
eigvecs(A)  
eigen(A).vectors  
_,V = eigen(A)
```



The usual notation for an eigenvalue is λ and *when the eigenvalues are all real*, by convention we often choose to order them such that $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_N$. But the `eigvals` and `eigen` commands return the eigenvalues in arbitrary order, usually not decreasing.

Example. Find the eigenvalues of $\mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix}$.

(Read)

The characteristic polynomial is $\det\{\mathbf{A} - z\mathbf{I}\} = \det\left\{\begin{bmatrix} 2-z & 1 \\ -1 & 2-z \end{bmatrix}\right\} = (2-z)(2-z) + 1 = z^2 - 4z + 5$,

which has roots $z = 2 \pm i$, so the two eigenvalues of \mathbf{A} are $\lambda_1 = 2 + i$, $\lambda_2 = 2 - i$, where $i = \sqrt{-1}$.

To find the eigenvectors, note that (by inspection):

$$(\mathbf{A} - \lambda_1 \mathbf{I}) \mathbf{v}_1 = \begin{bmatrix} -i & 1 \\ -1 & -i \end{bmatrix} \mathbf{v}_1 = \mathbf{0} \text{ when } \mathbf{v}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ i \end{bmatrix}$$

$$(\mathbf{A} - \lambda_2 \mathbf{I}) \mathbf{v}_2 = \begin{bmatrix} i & 1 \\ -1 & i \end{bmatrix} \mathbf{v}_2 = \mathbf{0} \text{ when } \mathbf{v}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -i \end{bmatrix}.$$

JULIA check: `eigen([2 1; -1 2])`

Properties of eigenvalues

The defining properties of determinant on p. 1.51 lead to useful eigenvalue properties.

- D1 \implies If \mathbf{A} is upper triangular, then the eigenvalues of \mathbf{A} are its diagonal elements.
- D3 \implies The eigenvalues of \mathbf{A}' are the complex conjugates of the eigenvalues of \mathbf{A} .
- D2 \implies The eigenvalues of $\alpha\mathbf{A}$ are α times the eigenvalues of \mathbf{A} for $\alpha \in \mathbb{F}$.
- D2 \implies The eigenvalues of \mathbf{A} are invariant to similarity transforms [1, Theorem 9.19].

If \mathbf{A} is $N \times N$ and \mathbf{T} is an $N \times N$ invertible matrix then $\mathbf{T}\mathbf{A}\mathbf{T}^{-1}$ has the same eigenvalues as \mathbf{A} .

As a special case, if \mathbf{Q} is $N \times N$ is **orthogonal** (or **unitary** in complex case) matrix, then $\mathbf{Q}\mathbf{A}\mathbf{Q}'$ has the same eigenvalues as \mathbf{A} .

Proof. $\det\{\mathbf{T}\mathbf{A}\mathbf{T}^{-1} - z\mathbf{I}\} = \det\{\mathbf{T}\mathbf{A}\mathbf{T}^{-1} - z\mathbf{T}\mathbf{T}^{-1}\} = \det\{\mathbf{T}(\mathbf{A} - z\mathbf{I})\mathbf{T}^{-1}\} = \det\{(\mathbf{A} - z\mathbf{I})\mathbf{T}^{-1}\mathbf{T}\} = \det\{(\mathbf{A} - z\mathbf{I})\mathbf{I}\} = \det\{\mathbf{A} - z\mathbf{I}\}$. So $\mathbf{T}\mathbf{A}\mathbf{T}^{-1}$ and \mathbf{A} have the same characteristic equation.

Here we used the **distributive property** of matrix multiplication.

The determinant of any square matrix is the product of its eigenvalues [1, Theorem 9.25]:

- $\mathbf{A} \in \mathbb{F}^{N \times N} \implies \det\{\mathbf{A}\} = \prod_{i=1}^N \lambda_i(\mathbf{A}).$

More product properties...

If \mathbf{A} has eigenvalues $\{\lambda_1, \dots, \lambda_N\}$, then \mathbf{A}^2 has eigenvalues $\{\lambda_1^2, \dots, \lambda_N^2\}$ and \mathbf{A}^k has eigenvalues $\{\lambda_1^k, \dots, \lambda_N^k\}$ for $k \in \mathbb{N}$.

Proof: $\mathbf{AV} = \mathbf{V}\Lambda \implies \mathbf{A}^2\mathbf{V} = \mathbf{A}(\mathbf{AV}) = \mathbf{A}(\mathbf{V}\Lambda) = (\mathbf{AV})\Lambda = (\mathbf{V}\Lambda)\Lambda = \mathbf{V}\Lambda^2$.

Define. Let $\mathcal{S} - \{x\}$ denote the set \mathcal{S} with the vector x removed.

Suppose $\mathbf{A} \in \mathbb{F}^{M \times N}$ and $\mathbf{B} \in \mathbb{F}^{N \times M}$.

If z is a nonzero eigenvalue of \mathbf{AB} , then z is also a nonzero eigenvalue of \mathbf{BA} .

We summarize this concisely as the following **commutative property** for eigenvalues:

$$\text{eig}\{\mathbf{AB}\} - \{0\} = \text{eig}\{\mathbf{BA}\} - \{0\}, \quad (1.26)$$

i.e., the nonzero elements of each set of eigenvalues are the same.

Proof. $\mathbf{ABv} = z\mathbf{v}$ for some nonzero \mathbf{v} , then clearly $\mathbf{u} \triangleq \mathbf{Bv}$ is also nonzero. Multiplying by \mathbf{B} yields $\mathbf{BABv} = z\mathbf{Bv} \implies \mathbf{BAu} = zu$ where \mathbf{u} is nonzero, so z is a nonzero eigenvalue of \mathbf{BA} . \square

What is the set of nonzero eigenvalues of the outer product matrix $\mathbf{A} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$?

A: {1, 2, 3}

B: {1, 2}

C: {1}

D: {2}

E: {6}

??

Exercise. What are the eigenvalues of $\mathbf{AB} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2 & 2 & -2 & 2 \end{bmatrix}$?

$$\mathbf{BA} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2 & 2 & -2 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 0 & 4 \end{bmatrix}.$$

So the only nonzero eigenvalue of \mathbf{BA} is 4 (repeated) and thus the only nonzero eigenvalue of \mathbf{AB} is also 4. But can we be sure that $\text{eig}\{\mathbf{AB}\} = (4, 4, 0, 0)$?

The proof on the preceding page does not seem general enough to make that conclusion.

One can verify this conjecture for this specific example using:

```
A = [1 1; 1 1; 1 -1; 1 1]
B = [1 1 1 1; -2 2 -2 2]
eigvals(A*B)
```

Challenge: resolve this question either by finding a more general proof, or finding counter-example where ♦♦ some nonzero eigenvalue has different multiplicity for \mathbf{AB} than for \mathbf{BA} .

In practice, I usually use (1.26) to look for the smallest and or largest eigenvalue, for which multiplicity is unimportant (and hence I have not thought about it).

1.7 Trace

Another important matrix property is its **trace**.

Define. The **trace** of a square matrix, denoted $\text{trace}\{\mathbf{A}\}$, is the sum of its diagonal elements [1, p. 6].

Properties of matrix trace _____ (proved in HW)

- $\text{trace}\{\cdot\}$ is a linear function: $\text{trace}\{\alpha \mathbf{A} + \beta \mathbf{B}\} = \alpha \text{trace}\{\mathbf{A}\} + \beta \text{trace}\{\mathbf{B}\}$
- A **cyclic commutative property**:

$$\mathbf{A} \in \mathbb{F}^{M \times N}, \mathbf{B} \in \mathbb{F}^{N \times M} \implies \text{trace}\{\mathbf{AB}\} = \text{trace}\{\mathbf{BA}\}. \quad (1.27)$$

- Why cyclic? Because (let $\mathbf{A} = \mathbf{X}$ and $\mathbf{B} = \mathbf{YZ}$):

$$\text{trace}\{\mathbf{XYZ}\} = \text{trace}\{\mathbf{YZX}\} = \text{trace}\{\mathbf{ZXY}\} \neq \text{trace}\{\mathbf{XZY}\}$$

- $\text{trace}\{\mathbf{A}\}$ is the sum of the eigenvalues of \mathbf{A} [1, Theorem 9.25].
(The proof involves the **Jordan normal form**, a linear algebra topic of less importance in SP/ML.)

Practical use in JULIA:

```
using LinearAlgebra
then tr(A)
```

or

```
using LinearAlgebra: tr
```

1.8 Appendix: Fields, Vector Spaces, Linear Transformations



In the academic literature (in the software community) there are two different meanings of the term **vector**.

- In the numerical methods community and in MATLAB, a **vector** is simply a column of a 2D matrix.
In other words, a (column) vector is a $N \times 1$ array of numbers.
- In general mathematics, *e.g.*, linear algebra and functional analysis, a vector belongs to a **vector space**.
For a nice overview of how this distinction affected the design of the JULIA language, see [this video](#).

Although this course will mostly use the numerical methods perspective, students who want a thorough understanding should also be familiar with the more general notion of a vector space.

This Appendix reviews vector spaces and linear operators defined on vector spaces. Although the definitions in this section are quite general and thus might appear somewhat abstract, the ideas are important even for the topics of a sophomore-level signals and systems course! For example, analog systems like passive RLC networks are linear systems that are represented mathematically by linear transformations from one (infinite dimensional) vector space to another.

The definition of a vector space uses the concept of a field of scalars, so we first review that.

Field of scalars

L§2.1

A **field** or **field of scalars** \mathbb{F} is a collection of elements $\alpha, \beta, \gamma, \dots$ along with an “addition” and a “multiplication” operator [14]. For every pair of scalars α, β in \mathbb{F} , there must correspond a scalar $\alpha + \beta$ in \mathbb{F} , called the **sum** of α and β , such that

- Addition is commutative: $\alpha + \beta = \beta + \alpha$
- Addition is associative: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
- There exists a unique element $0 \in \mathbb{F}$, called **zero**, for which $\alpha + 0 = \alpha, \forall \alpha \in \mathbb{F}$
- For every $\alpha \in \mathbb{F}$, there corresponds a unique scalar $(-\alpha) \in \mathbb{F}$ for which $\alpha + (-\alpha) = 0$.

For every pair of scalars α, β in \mathbb{F} , there must correspond a scalar $\alpha\beta$ in \mathbb{F} , called the **product** of α and β , such that

- Multiplication is commutative: $\alpha\beta = \beta\alpha$
- Multiplication is associative: $\alpha(\beta\gamma) = (\alpha\beta)\gamma$
- Multiplication distributes over addition: $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$
- There exists a unique element $1 \in \mathbb{F}$, called **one**, or **unity**, or the **identity** element, for which $1\alpha = \alpha, \forall \alpha \in \mathbb{F}$
- For every nonzero $\alpha \in \mathbb{F}$, there corresponds a unique scalar $\alpha^{-1} \in \mathbb{F}$, called the **inverse** of α for which $\alpha\alpha^{-1} = 1$.

Example. The set \mathbb{Q} of rational numbers is a field (with the usual definitions of addition and multiplication).

The only fields that we will need are the field of real numbers \mathbb{R} and the field of complex numbers \mathbb{C} .

Vector spaces

A **vector space** or **linear space** consists of

- A field \mathbb{F} of scalars.
- A set \mathcal{V} of entities called **vectors**
- An operation called **vector addition** that associates a **sum** $x + y \in \mathcal{V}$ with each pair of vectors $x, y \in \mathcal{V}$ such that
 - Addition is commutative: $x + y = y + x$
 - Addition is associative: $x + (y + z) = (x + y) + z$
 - There exists a unique element $\mathbf{0} \in \mathcal{V}$, called the **zero vector**, for which $x + \mathbf{0} = x, \forall x \in \mathcal{V}$
 - For every $x \in \mathcal{V}$, there corresponds a unique vector $(-x) \in \mathcal{V}$ for which $x + (-x) = \mathbf{0}$.
- An operation called **multiplication by a scalar** that associates with each scalar $\alpha \in \mathbb{F}$ and vector $x \in \mathcal{V}$ a vector $\alpha x \in \mathcal{V}$, called the **product** of α and x , such that:
 - Associative: $\alpha(\beta x) = (\alpha\beta)x$
 - Distributive $\alpha(x + y) = \alpha x + \alpha y$
 - Distributive $(\alpha + \beta)x = \alpha x + \beta x$
 - If 1 is the identity element of \mathbb{F} , then $1x = x, \forall x \in \mathcal{V}$.
- No operations are presumed to be defined for multiplying two vectors or adding a vector and a scalar.

Examples of important vector spaces

- **Euclidean n -dimensional space or n -tuple space:** $\mathcal{V} = \mathbb{R}^n$.

If $\mathbf{x} \in \mathcal{V}$, then $\mathbf{x} = (x_1, x_2, \dots, x_n)$ where $x_i \in \mathbb{R}$.

The field of scalars is $\mathbb{F} = \mathbb{R}$. Of course $\mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$, and $\alpha\mathbf{x} = (\alpha x_1, \dots, \alpha x_n)$.



This space is very closely related to the definition of a vector as a column of a matrix. They are so close that most of the literature does not distinguish them (nor does MATLAB). However, to be rigorous, \mathbb{R}^n is not the same as a column of a matrix because strictly speaking there is no inherent definition of the **transpose** of a vector in \mathbb{R}^n , whereas transpose is well defined for any matrix including a $n \times 1$ matrix.

- **Complex Euclidean n -dimensional space:** $\mathcal{V} = \mathbb{C}^n$. If $\mathbf{x} \in \mathcal{V}$, then $\mathbf{x} = (x_1, x_2, \dots, x_n)$ where $x_i \in \mathbb{C}$.

The field of scalars is $\mathbb{F} = \mathbb{C}$ and $\mathbf{x} + \mathbf{y} = (x_1 + y_1, \dots, x_n + y_n)$ and $\alpha\mathbf{x} = (\alpha x_1, \dots, \alpha x_n)$.

- $\mathcal{V} = \mathcal{L}_2(\mathbb{R}^3)$. The set of functions $f : \mathbb{R}^3 \rightarrow \mathbb{C}$ that are **square integrable**:

$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |f(x, y, z)|^2 dx dy dz < \infty$. The field is $\mathbb{F} = \mathbb{C}$.

Addition and scalar multiplication are defined in the natural way.

To show that $f, g \in \mathcal{L}_2(\mathbb{R}^3)$ implies $f + g \in \mathcal{L}_2(\mathbb{R}^3)$, one can apply the triangle inequality:

$$\|f + g\| \leq \|f\| + \|g\| \text{ where } \|f\| = \langle f, f \rangle, \text{ and } \langle f, g \rangle = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y, z) g^*(x, y, z) dx dy dz.$$

- The set of functions on the plane \mathbb{R}^2 that are zero outside of the unit square.
- The set of solutions to a homogeneous linear system of equations $A\mathbf{x} = \mathbf{0}$.

Linear transformations and linear operators

L§3.1

Define. Let \mathcal{U} and \mathcal{V} be two vector spaces over a common field \mathbb{F} .

A function $\mathcal{A} : \mathcal{U} \rightarrow \mathcal{V}$ is called a **linear transformation** or **linear mapping** from \mathcal{U} into \mathcal{V} iff $\forall \mathbf{u}_1, \mathbf{u}_2 \in \mathcal{U}$ and all scalars $\alpha, \beta \in \mathbb{F}$:

$$\mathcal{A}(\alpha \mathbf{u}_1 + \beta \mathbf{u}_2) = \alpha \mathcal{A}(\mathbf{u}_1) + \beta \mathcal{A}(\mathbf{u}_2).$$

Example:

- Let $\mathbb{F} = \mathbb{R}$ and let \mathcal{V} be the space of continuous functions on \mathbb{R} . Define the linear transformation \mathcal{A} by: if $F = \mathcal{A}(f)$ then $F(x) = \int_0^x f(t) dt$. Thus integration (with suitable limits) is linear.

If \mathcal{A} is a linear transformation from \mathcal{V} into \mathcal{V} , then we say \mathcal{A} is a **linear operator**. However, the terminology distinguishing linear transformations from linear operators is not universal, and the two terms are often used interchangeably.

Simple fact for linear transformations:

- $\mathcal{A}[0] = 0$. Proof: $\mathcal{A}[0] = \mathcal{A}[00] = 0\mathcal{A}[0] = 0$. This is called the “zero in, zero out” property.

Caution! (From [15]) By induction it follows that $\mathcal{A}(\sum_{i=1}^n \alpha_i \mathbf{u}_i) = \sum_{i=1}^n \alpha_i \mathcal{A}(\mathbf{u}_i)$ for any finite n , but the above *does not* imply in general that linearity holds for infinite summations or integrals. Further assumptions about “smoothness” or “regularity” or “continuity” of \mathcal{A} are needed for that.



Bibliography

- [1] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005.
- [2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proc. IEEE* 86.11 (Nov. 1998), 2278–2324.
- [3] F. Rosenblatt. *Principles of neurodynamics; perceptrons and the theory of brain mechanisms*. Washington: Spartan, 1962.
- [4] L. Eldén. *Matrix methods in data mining and pattern recognition*. Errata: <http://users.mai.liu.se/larel04/matrix-methods/index.html>. Soc. Indust. Appl. Math., 2007.
- [5] K. Bryan and T. Leise. “The \$25,000,000,000 eigenvector: The linear algebra behind Google”. In: *SIAM Review* 48.3 (Sept. 2006), 569–81.
- [6] A. N. Elmachtoub and C. F. Van Loan. “From random polygon to ellipse: An eigenanalysis”. In: *SIAM Review* 52.1 (2010), 151–70.
- [7] J. A. Fessler. *ASPIRE 3.0 user’s guide: A sparse iterative reconstruction library*. Tech. rep. 293. Available from <http://web.eecs.umich.edu/~fessler>. Univ. of Michigan, Ann Arbor, MI, 48109-2122: Comm. and Sign. Proc. Lab., Dept. of EECS, July 1995.
- [8] C. Champlin, D. Bell, and C. Schocken. “AI medicine comes to Africa’s rural clinics”. In: *IEEE Spectrum* 54.5 (May 2017), 42–8.
- [9] R. P. Lippmann. “An introduction to computing with neural nets”. In: *IEEE ASSP Mag.* 4.2 (Apr. 1987), 4–22.
- [10] T. Davis. *Block matrix methods: Taking advantage of high-performance computers*. Univ. Florida TR-98-204. 1998.
- [11] J. Kovacevic and A. Chebira. “Life beyond bases: the advent of frames (Part I)”. In: *IEEE Sig. Proc. Mag.* 24.4 (July 2007), 86–104.
- [12] J. Kovacevic and A. Chebira. “Life beyond bases: the advent of frames (Part II)”. In: *IEEE Sig. Proc. Mag.* 24.5 (Sept. 2007), 115–25.
- [13] A. G. Akritas, E. K. Akritas, and G. I. Malaschonok. “Various proofs of Sylvester’s (determinant) identity”. In: *Mathematics and Computers in Simulation* 42.4 (Nov. 1996), 585–93.
- [14] B. Noble and J. W. Daniel. *Applied linear algebra*. 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1977.
- [15] T. Kailath. *Linear systems*. New Jersey: Prentice-Hall, 1980.

Chapter 2

Matrix factorizations / decompositions

Contents (final version)

2.0 Introduction	2.2
2.1 Spectral Theorem (for symmetric matrices)	2.4
Normal matrices	2.6
Square asymmetric and non-normal matrices	2.9
Geometry of matrix diagonalization	2.11
2.2 SVD	2.17
Existence	2.17
Geometry	2.18
2.3 The matrix 2-norm or spectral norm	2.23
2.4 Relating SVDs and eigendecompositions	2.27
When does $U = V$?	2.30
2.5 Positive semidefinite matrices	2.34
2.6 Summary	2.37
SVD computation using eigendecomposition	2.38

2.0 Introduction

One of the main topics in the previous chapter was matrix multiplication. This chapter focuses on matrix factorizations, which is perhaps somewhat like the reverse of matrix multiplication.

Source material for this chapter includes [1, §10.1, 5.1, 7.4, 10.2].

There are many factorizations used in linear algebra and numerical linear algebra. Here are 7 important ones. The first 5 are for square matrices only. Of these 7, only the SVD accommodates any matrix size and type.

$A = LU$	$M = N$	LU decomposition by Gaussian elimination , for some (not all) square A : L is lower triangular; U is upper triangular. For general A , use pivoting .
$A = LL'$	$M = N$	Cholesky decomposition (for any positive-definite A): L is lower triang.
$A = Q\Lambda Q'$	$M = N$	orthogonal eigendecomposition (for any symmetric or normal A) Q is unitary; Λ is diagonal (and real if $A = A'$)
$A = V\Lambda V^{-1}$	$M = N$	diagonalization (possible only for some square A): V is (linearly independent) eigenvectors; Λ is eigenvalues
$A = QRQ'$	$M = N$	Schur decomposition (for any square A): Q is unitary; R is upper triangular
$A = QR$	$M \geq N$	QR decomposition via Gram-Schmidt orthogonalization (for any “tall” A): Q has orthonormal columns; R is upper triangular
$A = U\Sigma V'$	any M, N	SVD (for any A): U and V are unitary, columns are singular vectors Σ is (rectangular) diagonal with real, nonnegative singular values

The LU, QR and Cholesky decompositions are important for solving systems of equations, but are less helpful for analysis than the other forms. This chapter focuses on the two particularly important matrix decompositions: the eigendecomposition and the SVD.

There are no applications in this chapter but the tools are the foundation for most of the applications that appear in later chapters. Sometimes we use these tools for mathematical analysis (on paper only, especially for very large problems) but often we use them numerically.

A short summary of this chapter is: an **eigendecomposition** is usually the right tool for (square) Hermitian matrices whereas an **SVD** is usually the right tool otherwise.

Square matrices

Recall that any (square) matrix $\mathbf{A} \in \mathbb{F}^{N \times N}$ has N (possibly non-distinct) **eigenvalues** $\lambda_1, \dots, \lambda_N \in \mathbb{C}$.

For each eigenvalue λ_n , the matrix $\mathbf{A} - \lambda_n \mathbf{I}$ is **singular**, so there must exist a (nonzero) vector $\mathbf{v}_n \in \mathbb{F}^N$ (an **eigenvector**) such that

$$(\mathbf{A} - \lambda_n \mathbf{I}) \mathbf{v}_n = \mathbf{0} \implies \mathbf{A}\mathbf{v}_n = \lambda_n \mathbf{v}_n, \text{ for } n = 1, \dots, N. \quad (2.1)$$

In matrix form:

$$\mathbf{AV} = \mathbf{V}\Lambda, \quad \mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_N], \quad \Lambda = \text{diag}\{\lambda_1, \dots, \lambda_N\}. \quad (2.2)$$

Because each \mathbf{v}_n is nonzero, by convention we always normalize each to have *unit norm* for decompositions. Despite this normalization, \mathbf{V} is never unique because we can always scale each \mathbf{v}_n by ± 1 or even $e^{i\phi}$.

For a general square matrix, this is all we can say about an $N \times N$ eigenvector matrix \mathbf{V} . However, for (Hermitian) symmetric matrices we can say much more, thanks to the spectral theorem discussed next.

2.1 Spectral Theorem (for symmetric matrices)

L§10.1

If $\mathbf{A} \in \mathbb{F}^{N \times N}$ is (Hermitian) symmetric, *i.e.*, $\mathbf{A} = \mathbf{A}'$, then the **spectral theorem** says the following.

- The eigenvalues of \mathbf{A} are all real.
- Remarkably, there is an **orthonormal basis** for \mathbb{F}^N consisting of **eigenvectors** of \mathbf{A} , *i.e.*, there exists \mathbf{V} in (2.2) that is an **orthogonal** (or **unitary**) matrix, *i.e.*, $\mathbf{V}'\mathbf{V} = \mathbf{V}\mathbf{V}' = \mathbf{I}$, so $\mathbf{V}^{-1} = \mathbf{V}'$.
- Multiplying (2.2) on the right by \mathbf{V}' yields a **unitary eigendecomposition** (a matrix factorization):

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'. \quad (2.3)$$

- If $\mathbf{A} \in \mathbb{R}^{N \times N}$ is symmetric, then there exists a *real* orthogonal matrix \mathbf{V} satisfying (2.3).

In words, *every symmetric/Hermitian (hence square) matrix has an orthogonal/unitary eigendecomposition*. This factorization is very useful for analysis and sometimes for computation.

As mentioned previously, a data matrix \mathbf{X} is rarely square, but the Gram matrix $\mathbf{X}'\mathbf{X}$ and the outer-product matrix $\mathbf{X}\mathbf{X}'$ are always square. Furthermore, the Gram matrix and the outer-product matrix are always (Hermitian) symmetric, so the spectral theorem applies.

In signal processing, we usually need an eigendecomposition only for matrices of the form $\mathbf{X}'\mathbf{X}$ or $\mathbf{X}\mathbf{X}'$.

Proof that eigenvalues are real. If \mathbf{v} is an eigenvector of $\mathbf{A} = \mathbf{A}'$ with eigenvalue λ , then $\mathbf{Av} = \lambda\mathbf{v} \Rightarrow \mathbf{v}'\mathbf{Av} = \lambda\mathbf{v}'\mathbf{v} \Rightarrow (\mathbf{v}'\mathbf{Av})' = \lambda'\mathbf{v}'\mathbf{v} \Rightarrow \mathbf{v}'\mathbf{A}'\mathbf{v} = \lambda'\mathbf{v}'\mathbf{v} \Rightarrow \mathbf{v}'\mathbf{Av} = \lambda'\mathbf{v}'\mathbf{v} \Rightarrow \lambda = \lambda' = \lambda^*$.

Sketch of proof of orthogonality of eigenvectors for the case of distinct eigenvalues. Suppose \mathbf{v} is an eigenvector of $\mathbf{A} = \mathbf{A}'$ with (real) eigenvalue λ , and \mathbf{u} is an eigenvector with different (real) eigenvalue $\beta \neq \lambda$.

$$\mathbf{Av} = \lambda\mathbf{v} \Rightarrow \mathbf{u}'\mathbf{Av} = \lambda\mathbf{u}'\mathbf{v} \Rightarrow (\mathbf{u}'\mathbf{Av})' = (\lambda\mathbf{u}'\mathbf{v})' \Rightarrow \mathbf{v}'\mathbf{A}'\mathbf{u} = \lambda\mathbf{v}'\mathbf{u} \Rightarrow \mathbf{v}'\mathbf{Au} = \lambda\mathbf{v}'\mathbf{u}.$$

$$\mathbf{Au} = \beta\mathbf{u} \Rightarrow \mathbf{v}'\mathbf{Au} = \beta\mathbf{v}'\mathbf{u} \Rightarrow \lambda\mathbf{v}'\mathbf{u} = \beta\mathbf{v}'\mathbf{u} \Rightarrow \mathbf{v}'\mathbf{u} = 0 \text{ because } \beta \neq \lambda.$$

(Read)

Example. Consider the symmetric matrix $\mathbf{A} = \begin{bmatrix} 4 & 6 \\ 6 & 9 \end{bmatrix}$ for which $\det\{\mathbf{A} - \lambda\mathbf{I}\} = (4 - \lambda)(9 - \lambda) - 6^2 = \lambda^2 - 13\lambda$. So the eigenvalues of \mathbf{A} are $\{13, 0\}$. $\mathbf{A} - 13\mathbf{I} = \begin{bmatrix} -9 & 6 \\ 6 & -4 \end{bmatrix} = \begin{bmatrix} 3 \\ -2 \end{bmatrix} \begin{bmatrix} -3 & 2 \end{bmatrix}$ so an eigenvector corresponding to eigenvalue 13 is $\mathbf{v}_1 = \frac{1}{\sqrt{13}} \begin{bmatrix} 2 \\ 3 \end{bmatrix}$. Similarly an eigenvector corresponding to eigenvalue 0 is $\mathbf{v}_2 = \frac{1}{\sqrt{13}} \begin{bmatrix} -3 \\ 2 \end{bmatrix}$. Thus an eigendecomposition is

$$\mathbf{A} = \underbrace{\begin{bmatrix} 4 & 6 \\ 6 & 9 \end{bmatrix}}_{\mathbf{V}} = \underbrace{\frac{1}{\sqrt{13}} \begin{bmatrix} 2 & -3 \\ 3 & 2 \end{bmatrix}}_{\mathbf{V}} \underbrace{\begin{bmatrix} 13 & 0 \\ 0 & 0 \end{bmatrix}}_{\Lambda} \underbrace{\frac{1}{\sqrt{13}} \begin{bmatrix} 2 & 3 \\ -3 & 2 \end{bmatrix}}_{\mathbf{V}'} = 13\mathbf{v}_1\mathbf{v}_1' + 0\mathbf{v}_2\mathbf{v}_2' = \underbrace{\begin{bmatrix} 2 \\ 3 \end{bmatrix}}_{\sqrt{\lambda_1}\mathbf{v}_1} \underbrace{\begin{bmatrix} 2 & 3 \end{bmatrix}}_{\sqrt{\lambda_1}\mathbf{v}_1'}$$

Normal matrices

(Read)

Hermitian symmetry is a *sufficient* but not *necessary* condition for existence of a **unitary eigendecomposition**.

Define. A square matrix A is a **normal matrix** iff $A'A = AA'$.

The **spectral theorem** says: A square matrix A is **diagonalizable** by a **unitary** matrix, *i.e.*, $A = V\Lambda V'$, iff it is a **normal** matrix.

For a normal matrix, Λ need not be real, whereas for a (Hermitian) symmetric matrix, Λ is real.

Every unitary matrix U is a normal matrix. (?)

A: True

B: False

??

Permutation matrix

Example. An important type of normal matrix is a **permutation matrix**.

Define. A $N \times N$ permutation matrix has exactly one 1 in each row and one 1 in each column and all other elements are zero.

If \mathbf{P} is a permutation matrix, then $\mathbf{P}^{-1} = \mathbf{P}'$ so \mathbf{P} is an orthogonal matrix and is **normal**. Thus \mathbf{P} has a unitary eigendecomposition and typically its eigenvalues and eigenvectors are complex [wiki].

Example. The permutation matrix that (circularly) shifts each element of a vector in \mathbb{F}^3 by one index is

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

This permutation matrix happens to be a **circulant matrix** (see Ch. 7), so its eigenvalues are given by the 3-point **DFT** of the first column: $\{e^{-i2\pi k/3} : k = 0, 1, 2\} = \{1, e^{\pm i2\pi/3}\}$. (See HW.)

Example. The following **rotation matrix** is asymmetric unless ϕ is a multiple of π : $\mathbf{R}_\phi = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix}$.

Yet this matrix must have two eigenvalues with corresponding eigenvectors. The eigenvalues satisfy

$(\cos \phi - \lambda)^2 + \sin^2 \phi = 0$ leading to $\lambda_{\pm} = e^{\pm i\phi}$. Corresponding eigenvectors are $\mathbf{v}_{\pm} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ \pm i \end{bmatrix}$ because

$$\mathbf{R}_\phi \begin{bmatrix} 1 \\ \pm i \end{bmatrix} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} 1 \\ \pm i \end{bmatrix} = \begin{bmatrix} \cos \phi \pm i \sin \phi \\ -\sin \phi \pm i \cos \phi \end{bmatrix} = (\cos \phi \pm i \sin \phi) \begin{bmatrix} 1 \\ \pm i \end{bmatrix} = e^{\pm i\phi} \begin{bmatrix} 1 \\ \pm i \end{bmatrix}.$$

This rotation matrix is **normal** because $\mathbf{R}_{-\phi} = \mathbf{R}'_\phi = \mathbf{R}_\phi^{-1}$, i.e., multiplying by \mathbf{R} and then \mathbf{R}' corresponds to rotating by ϕ and then rotating back, and $\mathbf{R}'\mathbf{R} = \mathbf{R}\mathbf{R}' = \mathbf{I}_2$. (In fact \mathbf{R} is unitary.)

\mathbf{R} is diagonalizable by the unitary matrix $\mathbf{V} = [\mathbf{v}_+ \ \mathbf{v}_-]$ and here is a **unitary eigendecomposition**:

$$\mathbf{R}_\phi = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} = \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix} \right) \begin{bmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{bmatrix} \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -i \\ 1 & i \end{bmatrix} \right).$$

\mathbf{R} is *not* diagonalizable by a real matrix \mathbf{V} because rotation leads to a vector pointing in a different direction, unless ϕ is a multiple of π . But \mathbf{R} is diagonalizable by the above unitary matrix \mathbf{V} .

What is the best way to think about the rotation matrix \mathbf{R} ?

A: A data matrix.

B: An operator matrix.

C: Neither.

??

Square asymmetric and non-normal matrices

(Read)

Some, but not all, square *asymmetric* matrices that are not **normal** matrices are **diagonalizable**.

Define. A square matrix is **diagonalizable** iff it is **similar** to a diagonal matrix, *i.e.*, iff there exists an **invertible** matrix \mathbf{V} such that $\mathbf{V}^{-1}\mathbf{A}\mathbf{V}$ is diagonal.

Specifically, if $\mathbf{A} \in \mathbb{F}^{N \times N}$ has N **linearly independent** eigenvectors $\mathbf{V} = [\mathbf{v}_1 \ \dots \ \mathbf{v}_N]$, then

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}, \quad \Lambda = \text{diag}\{\lambda_n\}.$$

- If \mathbf{A} is asymmetric and its eigenvalues are all real, then \mathbf{A} cannot have a unitary eigendecomposition.
(If it did, then we would have $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}' = \mathbf{A}'$, contradicting asymmetry.)
- If \mathbf{A} has N distinct eigenvalues (no repeated roots of characteristic equation), then \mathbf{A} is diagonalizable.
(But having distinct eigenvalues is not a necessary condition for being diagonalizable.)
- If \mathbf{A} is both invertible and diagonalizable then $\mathbf{A}^{-1} = \mathbf{V}\Lambda^{-1}\mathbf{V}^{-1}$.
- Being diagonalizable does not imply invertibility because some eigenvalues can be 0.

Some square matrices are not diagonalizable (see example below). Such matrices might arise when trying to solve a system of N equations in N unknowns so they are a major topic in a linear algebra course, but they are much less important in signal processing so we do not dwell on them further here.

Anyway, every matrix, even if non-square, has a **singular value decomposition (SVD)** so that will be the primary tool we use for data matrices.

Example. The matrix $\mathbf{A} = \begin{bmatrix} 3 & 1 \\ 0 & 3 \end{bmatrix}$ has repeated eigenvalue $\lambda = 3$, and $\mathbf{A} - 3\mathbf{I} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ so $\mathbf{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

Here $\mathbf{A}\mathbf{V} = \mathbf{V}\Lambda$, where $\mathbf{V} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ and $\Lambda = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$ contains both eigenvalues of \mathbf{A} , but the columns of \mathbf{V} are linearly dependent so \mathbf{A} is not diagonalizable.

Readers interested in general **eigendecompositions** and **diagonalizable** matrices should study **minimal polynomials** and the **Jordan normal form**.

Challenge. When \mathbf{A} is 3×3 and symmetric, it has 6 **degrees of freedom (DoF)** (the upper triangular elements). Now $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'$, where \mathbf{V} and Λ are both 3×3 matrices. Explain the DoF in terms of \mathbf{V} and Λ .

??

Every permutation matrix has a linearly independent set of eigenvectors. (?)

A: True

B: False

??

Every permutation matrix has real eigenvalues. (?)

A: True

B: False

??

Geometry of matrix diagonalization

Let $\mathbf{A} \in \mathbb{F}^{N \times N}$ be a (Hermitian) symmetric matrix, so $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'$.

Consider the linear transform $\mathbf{x} \mapsto \mathbf{y} = \mathbf{A}\mathbf{x} = \mathbf{V}\Lambda\mathbf{V}'\mathbf{x}$.

We can think of $\mathbf{A}\mathbf{x}$ as a cascade of three linear transforms:

$$\mathbf{x} \xrightarrow{\mathbf{V}'} \mathbf{w} \xrightarrow{\Lambda} \mathbf{z} \xrightarrow{\mathbf{V}} \mathbf{y}.$$

- $\mathbf{x} \mapsto \mathbf{w} = \mathbf{V}'\mathbf{x}$ is a coordinate change (a rotation in fact, possibly with a sign flip for one axis).
 \mathbf{w} denotes the coefficient vector for \mathbf{x} in the basis \mathbf{V} , because $\mathbf{x} = \mathbf{V}\mathbf{w}$.
- $\mathbf{w} \mapsto \mathbf{z} = \Lambda\mathbf{w}$ is scaling of each coordinate by the diagonal elements of Λ .
- $\mathbf{z} \mapsto \mathbf{y} = \mathbf{V}\mathbf{z}$ is going back to the original coordinate system.

It is useful to understand these three operations geometrically.

Example. We illustrate for the case of a symmetric 2×2 matrix.

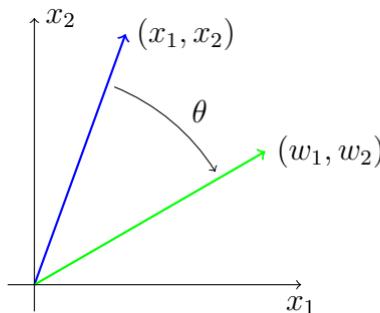
Fact. Every $\mathbf{V} \in \mathbb{R}^{2 \times 2}$ with $\mathbf{V}'\mathbf{V} = \mathbf{I}_2$ (i.e., **orthogonal**) has the following form for some rotation angle θ :

$$\mathbf{V} = \begin{bmatrix} \cos \theta & -q \sin \theta \\ \sin \theta & q \cos \theta \end{bmatrix}, \quad q \in \{\pm 1\}.$$

Exercise. Verify that $\mathbf{V}'\mathbf{V} = \mathbf{I}_2$.

For simplicity we focus on the case of rotation matrices hereafter where $q = +1$.

Consider the linear transformation $\mathbf{x} \mapsto \mathbf{w} = \mathbf{V}'\mathbf{x} = \begin{bmatrix} x_1 \cos \theta + x_2 \sin \theta \\ -x_1 \sin \theta + x_2 \cos \theta \end{bmatrix}$. Graphically:



Importantly, the length of \mathbf{x} and \mathbf{w} are the same:

$$\mathbf{w} = \mathbf{V}'\mathbf{x} \implies \|\mathbf{w}\|_2^2 = \mathbf{w}'\mathbf{w} = (\mathbf{V}'\mathbf{x})'\mathbf{V}'\mathbf{x} = \mathbf{x}'\mathbf{V}\mathbf{V}'\mathbf{x} = \mathbf{x}'\mathbf{I}\mathbf{x} = \mathbf{x}'\mathbf{x} = \|\mathbf{x}\|_2^2.$$

The next mapping is

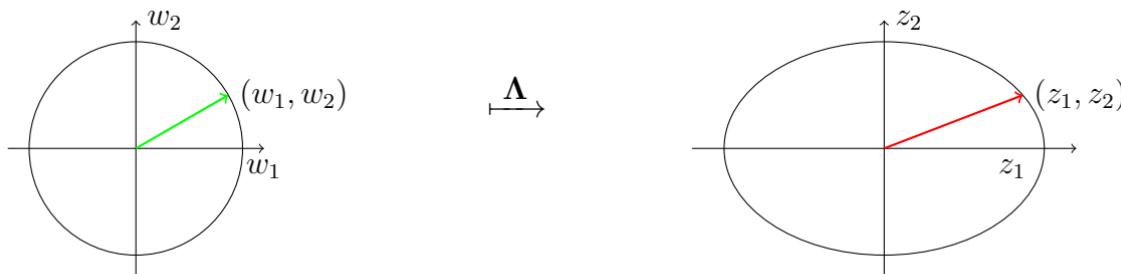
$$\mathbf{w} \mapsto \mathbf{z} = \Lambda \mathbf{w}, \text{ where } \Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \implies \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 w_1 \\ \lambda_2 w_2 \end{bmatrix}.$$

For interpretation, assume $\lambda_1, \lambda_2 \neq 0$ and suppose $\|\mathbf{x}\|_2 = 1 \implies \mathbf{x}'\mathbf{x} = 1 \implies x_1^2 + x_2^2 = 1$, which in turn implies $w_1^2 + w_2^2 = 1$, i.e., \mathbf{x} and \mathbf{w} lie on the unit circle. Because $w_1 = z_1/\lambda_1$ and $w_2 = z_2/\lambda_2$, we have

$$\left(\frac{z_1}{\lambda_1}\right)^2 + \left(\frac{z_2}{\lambda_2}\right)^2 = 1,$$

i.e., \mathbf{z} lies on an ellipse with axes governed by λ_1 and λ_2 .

Graphically:



Typically z is *not* collinear with w .

The exception is when $\lambda_1 = \lambda_2$, in which case $A = \lambda_1 I_2$, which is a trivial case.

Q. When does $y = Ax$ produce y that is collinear with x ?

A. When x is eigenvector of A , because then $Ax = \lambda x$.

For the third and final mapping, we return to geometry.

If $x \xrightarrow{V'} w$ represents counter-clockwise rotation, then $w \xrightarrow{V} x$ must represent clockwise rotation, because $VV' = I$ so $V(V'x) = (VV')x = Ix = x$.

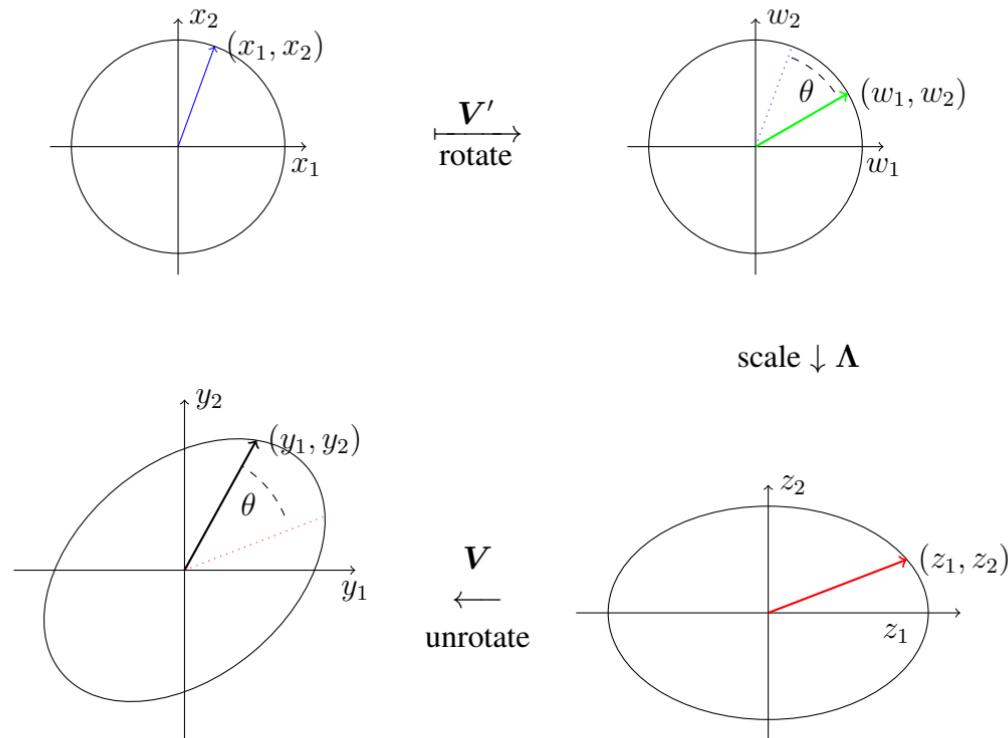
If V includes a sign flip, e.g., $V = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, then $V^{-1} = V'$ also undoes that sign flip.

- An orthogonal matrix with determinant equal to +1 is called a **rotation**.
- If the determinant is -1 then it is an **improper rotation**.

We ignore the possibility of a sign flip in the graphical illustrations, for simplicity.

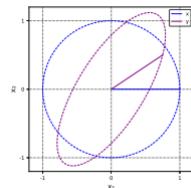
The next page gives a graphical summary in 2×2 case of $y = Ax = V \underbrace{\Lambda}_{w} \overbrace{V'x}^z$.

(The same principles apply in higher dimensions.)



Demo

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/02_eigshow1.html
https://web.eecs.umich.edu/~fessler/course/551/julia/demo/02_eigshow1.ipynb



Practical implementation

`d = eigvals(A)` returns 1D array of eigenvalues

`V = eigvecs(A)` returns matrix of eigenvectors

`obj = eigen(A)` returns an “object” (akin to a dictionary) of type `Eigen` with components:

`d = obj.values` or `d = eigen(A).values` (vector containing eigenvalues)

`V = obj.vectors` or `V = eigen(A).vectors` (matrix with eigenvectors)

Note the use of argument/index chaining: `f(arg1).arg2` `f(arg1)[1]`

To extract both parts in one line use: `d, V = eigen(A)`

If A is **diagonalizable**, then $A = V \text{diag}\{d\} V^{-1}$, i.e., $A = V * \text{Diagonal}(d) * \text{inv}(V)$

2.2 SVD

Existence

L§5.1

If $X \in \mathbb{F}^{M \times N}$ then there exists (for proof, see [1, Theorem 5.1] or [wiki]) matrices U, V, Σ such that

$$X = U\Sigma V' = \sum_{k=1}^{\min(M,N)} \sigma_k u_k v'_k. \quad (2.4)$$

This factorization is called the **singular value decomposition (SVD)**, where:

- U is $M \times M$ and unitary: $U'U = UU' = I_M$, and its columns are the **left singular vectors** of X
- V is $N \times N$ and unitary: $V'V = VV' = I_N$ and its columns are the **right singular vectors** of X
- Σ is a $M \times N$ **rectangular diagonal matrix** containing the **singular values** of X

For the history of the SVD, see [2].

- The $M \times N$ matrix Σ looks like one of:

$$\Sigma = \underbrace{\begin{bmatrix} \sigma_1 & & & 0 \\ & \ddots & & \\ 0 & & \sigma_N & \\ \hline & & \mathbf{0}_{M-N,N} & \end{bmatrix}}_{M > N \text{ (tall)}} \quad \text{or} \quad \Sigma = \underbrace{\begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_M \end{bmatrix} \left| \begin{array}{c} \\ \\ \mathbf{0}_{M,N-M} \end{array} \right.}_{N > M \text{ (wide)}} \quad \text{or} \quad \Sigma = \underbrace{\begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & & \sigma_M \end{bmatrix}}_{M = N \text{ (square)}}.$$

- These possible shapes of Σ are why the sum in (2.4) has the $\min(M, N)$ limit.
- The **singular values** $\sigma_1, \dots, \sigma_{\min\{M, N\}}$ are real and nonnegative.
- By convention we use descending order: $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min\{M, N\}}$.
- The first r singular values are positive, where $0 \leq r \leq \min\{M, N\}$ is the **rank** of the matrix (later).
- A subtle point is that the sum form (2.4) does not use all columns of U when $M > N$, nor all columns of V when $N > M$. More on this later when we discuss the **compact SVD**.

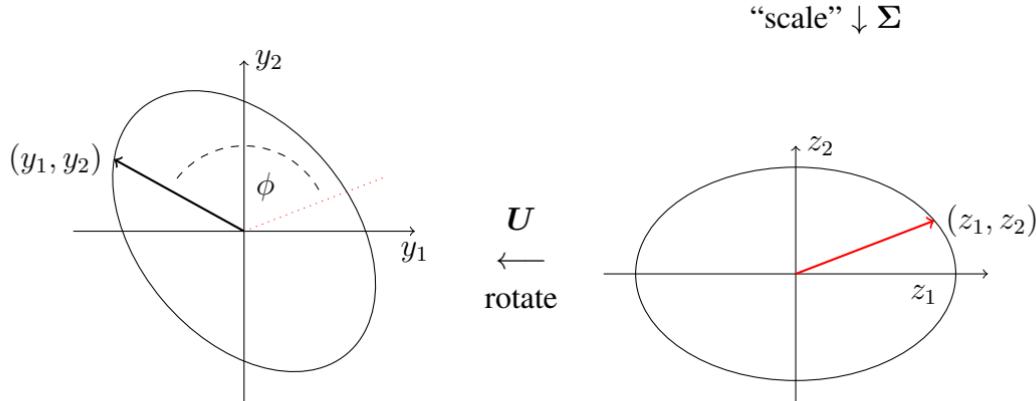
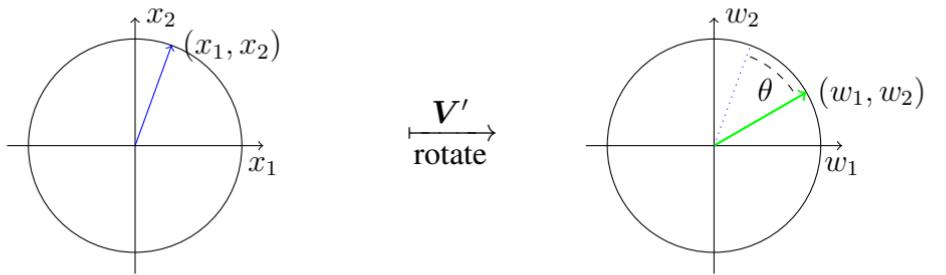
Geometry

Example. If A is any real 2×2 matrix, then its SVD looks like:

$$A = U\Sigma V' = \begin{bmatrix} \cos \phi & q_1 \sin \phi \\ -\sin \phi & q_1 \cos \phi \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} \cos \theta & -q_2 \sin \theta \\ \sin \theta & q_2 \cos \theta \end{bmatrix}, \quad q_1, q_2 \in \{\pm 1\},$$

where $\theta \neq \phi$ in general. (In fact when $M \neq N$, U and V even have different sizes!)

Consider the case $q_1 = q_2 = +1$ for simplicity. The next page illustrates the 2×2 geometry is graphically:



What are the differences here (for SVD) from before (eigendecomposition)?

- Final rotation angle differs: $\phi \neq \theta$ in general, i.e., $\mathbf{U} \neq \mathbf{V}$ in general.
- For non-square matrices, Σ is non-square, so the interpretation that it “scales” is incomplete.
- The SVD always exists, even for (asymmetric) 2×2 matrices that have no eigendecomposition.

Example. Determine the SVD of the **rotation matrix** $\mathbf{R} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix}$.

(Recall that no *real* eigendecomposition exists for this important matrix in general.)

By inspection we can choose $\mathbf{U} = \mathbf{R}$ and $\Sigma = \mathbf{I}_2$ and $\mathbf{V} = \mathbf{I}_2$.

This matrix \mathbf{R} has a unique SVD. (?)

A: True

B: False

??

More geometry

For eigendecomposition we asked “when is \mathbf{Ax} aligned with \mathbf{x} ?”

Should we ask that question for the SVD? No, **\mathbf{A} is non-square in general!**

What choice of vectors \mathbf{x} and \mathbf{z} makes \mathbf{Ax} perpendicular to \mathbf{Az} ?

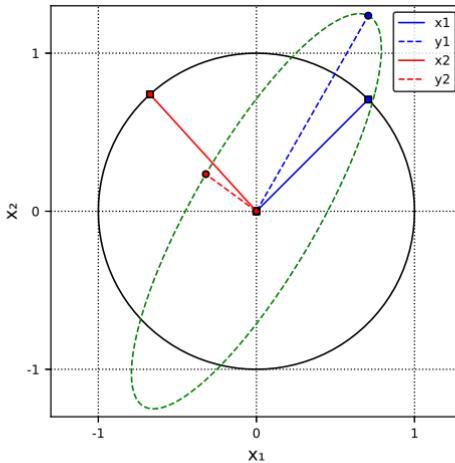
Choose the right singular vectors: $\mathbf{x} = \mathbf{v}_1$ and $\mathbf{z} = \mathbf{v}_2$.

Proof: $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}' = \sigma_1\mathbf{u}_1\mathbf{v}_1' + \sigma_2\mathbf{u}_2\mathbf{v}_2'$ (a sum of outer products) so $\mathbf{Ax} = \mathbf{Av}_1 = \sigma_1\mathbf{u}_1$ and $\mathbf{Az} = \mathbf{Av}_2 = \sigma_2\mathbf{u}_2$ but $\mathbf{u}_1 \perp \mathbf{u}_2$, so $\mathbf{Ax} \perp \mathbf{Az}$. \square

Demo

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/02_eigshow2.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/02_eigshow2.ipynb



In general, SVD maps “circles” (precisely: the set of vectors with unit norm) to “rotated ellipses.”

- Major and minor axes directions correspond to the left singular vectors in \mathbf{U} .
- Eccentricity depends on the singular values.

Practical implementation

(Read)

`obj = svd(A)` returns an “object” (akin to a dictionary) of type `SVD` with components:

`U = obj.U` has size $M \times \min(M, N)$.

Caution: `U` differs from $\mathbf{U} \in \mathbb{F}^{M \times M}$ in the typical **tall** case where $M > N$



`s = obj.S` for vector `s` of length $\min(M, N)$ such that $\text{diag}\{s\}$ is the core of Σ

`Vt = obj.Vt` for \mathbf{V}' has size $\min(M, N) \times N$

`V = obj.V` (internally this is an `Adjoint` array to avoid a transpose!)

Caution: `V` differs from $\mathbf{V} \in \mathbb{F}^{N \times N}$ in the **wide** case where $M < N$.



Another option is `U, s, V = svd(A)` or `U, s, V = (svd(A) ... ,)`

One can rebuild \mathbf{A} (to within numerical precision) using `U * Diagonal(s) * V'`

If one wants the “full” SVD where \mathbf{U} is $M \times M$ and \mathbf{V} is $N \times N$ then use `svd(A, full=true)` but we rarely need that in practice.

The default `svd(A)` is equivalent to `svd(A, full=false)` because this is the usual use case.

The non-full default in JULIA is equivalent to the “economy” option `svd(A, 'econ')` in MATLAB, also known as the **thin SVD**.

Ch. 3 describes a **compact SVD** that differs further from the “economy” and “full” SVD.

In terms of sizes: `compact` \leq `economy` = `thin` \leq `full`. See Ch. 3 for an example.

See [3] for a recent survey of methods for computing an SVD.

2.3 The matrix 2-norm or spectral norm

Moving towards one (of many) applications of the SVD, we now ask: What unit norm “input vector” \mathbf{x} produces an “output vector” \mathbf{Ax} having the “largest” output (as defined by norm, aka energy)? In other words, find unit norm \mathbf{x}_* such that $\|\mathbf{Ax}_*\| \geq \|\mathbf{Ax}\|$ for all unit norm \mathbf{x} .

Note the “systems” language here. Very often when we talk about $\mathbf{y} = \mathbf{Ax}$ we also think about a system block diagram like

$$\begin{array}{ccc} \mathbf{x} & \rightarrow & \boxed{\mathbf{A}} \\ \text{input } \mathbb{F}^M & & \text{(linear) system} \\ \end{array} \rightarrow \mathbf{y} = \mathbf{Ax} \quad \text{output } \mathbb{F}^N.$$

In this setting, we are thinking of \mathbf{A} as an operation, not as data.

Expressing the question mathematically:

L§7.4

$$\mathbf{x}_* = \underset{\mathbf{x}: \|\mathbf{x}\|=1}{\arg \max} \|\mathbf{Ax}\|, \quad \text{where } \|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\mathbf{x}' \mathbf{x}}.$$

Claim: $\mathbf{x}_* = \mathbf{v}_1$, the first right singular vector (having the largest singular value σ_1).

Another maximizer is $\mathbf{x}_* = e^{i\phi} \mathbf{v}_1$.

Proof. Because \mathbf{V} is orthogonal (or unitary), we can write \mathbf{x} in terms of the \mathbf{V} basis as $\mathbf{x} = \mathbf{V}\mathbf{z}$ where $\mathbf{z} = \mathbf{V}'\mathbf{x}$ are the coefficients. Note that $\|\mathbf{x}\| = 1 \iff \|\mathbf{z}\| = 1$.

Thus $\mathbf{A}\mathbf{x} = \mathbf{U}\Sigma\mathbf{V}'\mathbf{x} = \mathbf{U}\Sigma\mathbf{V}'\mathbf{V}\mathbf{z} = \mathbf{U}\Sigma\mathbf{z}$ so $\|\mathbf{A}\mathbf{x}\| = \sqrt{(\mathbf{A}\mathbf{x})'(\mathbf{A}\mathbf{x})} = \sqrt{\mathbf{z}'\Sigma'\mathbf{U}'\mathbf{U}\Sigma\mathbf{z}} = \sqrt{\mathbf{z}'\Sigma'\Sigma\mathbf{z}}$
 $= \sqrt{\sum_{k=1}^r \sigma_k^2 |z_k|^2} \leq \sqrt{\sigma_1^2 \sum_{k=1}^r |z_k|^2} = \sigma_1 \|\mathbf{z}\| = \sigma_1$, where $r = \min(M, N)$. So $\|\mathbf{x}\| = 1 \implies \|\mathbf{A}\mathbf{x}\| \leq \sigma_1$. This gives an upper bound on the norm of the output. But we can achieve that upper bound by choosing $\mathbf{x} = \mathbf{v}_1$ because then $\mathbf{z} = (1, 0, 0, \dots, 0)$ and $\mathbf{A}\mathbf{x} = \sigma_1 \mathbf{u}_1$ so $\|\mathbf{A}\mathbf{x}\| = \|\sigma_1 \mathbf{u}_1\| = \sigma_1$. \square

Fact. The solution $\mathbf{x}_* = \mathbf{v}_1$ is unique up to within a phase factor $e^{i\phi}$ iff $\sigma_1 > \sigma_2$.

Define. This property of a matrix is very important; it is called the **matrix 2-norm** or **spectral norm**:

$$\|\mathbf{A}\|_2 \triangleq \max_{\mathbf{x}: \|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2 = \max_{\mathbf{x} \neq 0} \frac{\|\mathbf{A}\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sigma_1. \quad (2.5)$$

By definition the 2-norm of a matrix \mathbf{A} gives a **tight upper bound** on how much the 2-norm of a vector can be amplified when multiplying by \mathbf{A} :

$$\|\mathbf{A}\mathbf{x}\|_2 \leq \|\mathbf{A}\|_2 \|\mathbf{x}\|_2 = \sigma_1 \|\mathbf{x}\|_2.$$

It is tight because the upper bound is achieved when $\mathbf{x} = \mathbf{v}_1$.

This important SVD property and has practical applications for power maximization, shown next.

Example. **Multi-input multi-output (MIMO)** communications with multi-antenna systems

Consider a system with N transmit antennas and M receiving antennas, such as in **802.11n wifi**.

Let a_{ij} denote the (complex) gain between the j th transmit antenna and the i th receive antenna.

(Matrix \mathbf{A} depends on amplifier and receiver properties and the multi-path wave propagation between them.)

	M Receive	N Transmit
1	$-<$	$>-$ 1
\vdots	$-<$	$>-$ 2
M	$-<$	$>-$ \vdots
		$>-$ N

Goal: Design transmit amplitudes so that the received signal has largest possible **signal to noise ratio (SNR)**.

For some signal we want to transmit, we use amplitudes x_1, \dots, x_N for the N transmit antennas.

There are transmit power limits (amplifier hardware and allowable interference) so we constrain the input amplitudes: $\|\mathbf{x}\| \leq 1$.

After transmission, the signals received by the M antennas will have amplitudes $\mathbf{y} = \mathbf{Ax}$. To maximize SNR, we want to design \mathbf{x} to make the received signal energy $\|\mathbf{y}\|$ as large as possible. (The background noise power is independent of \mathbf{x} .) This problem has the form $\arg \max_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|\mathbf{Ax}\|$ so a solution is

$$\mathbf{x} = \mathbf{v}_1, \text{ the first right singular vector.}$$

Exercise. The reader should verify the second equality in (2.5) and that

$$\arg \max_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \|\mathbf{Ax}\| = \arg \max_{\mathbf{x}: \|\mathbf{x}\|=1} \|\mathbf{Ax}\|.$$

In practice, the designer of a wifi system does not know the \mathbf{A} for your house, and in fact \mathbf{A} changes if you rearrange the furniture or relocate your computer. So the wifi router must determine \mathbf{A} “on the fly” and this process is called **channel estimation**. The basic idea is that the transmitter first sends N orthonormal training waveforms $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{C}^N$, called **pilot signals**, to the receiver. The receiver records the corresponding outputs $\mathbf{y}_1, \dots, \mathbf{y}_N \in \mathbb{C}^M$, where $\mathbf{y}_n = \mathbf{A}\mathbf{x}_n$. Writing as a matrix:

$$[\mathbf{y}_1 \ \dots \ \mathbf{y}_N] = \mathbf{A}[\mathbf{x}_1 \ \dots \ \mathbf{x}_N] \implies \mathbf{Y} = \mathbf{AX}.$$

Because \mathbf{X} is unitary by design, we can estimate \mathbf{A} as

$$\mathbf{A} = \mathbf{Y}\mathbf{X}^{-1} = \mathbf{Y}\mathbf{X}'.$$

The receiver must know what pilot signals \mathbf{X} are transmitted; this specification is part of the protocol.

Once the router has determined \mathbf{A} , it can compute its SVD and use \mathbf{v}_1 as the best transmit amplitude vector.

This process is related to the topic known as **beamforming**.

See [this illustration of an improved constellation using SVD-based transmit beamforming](#).

One might ask: why estimate *all* of \mathbf{A} when we need only \mathbf{v}_1 ?

Later we will see how to estimate \mathbf{v}_1 with fewer measurements (at least when N is large).

2.4 Relating SVDs and eigendecompositions

The two big topics of this chapter are SVDs and eigendecompositions. Are they related? Every matrix, even if rectangular, has an SVD, whereas only some square matrices have an eigendecomposition. Nevertheless, we can find some relationships.

- A **right singular vector** of \mathbf{A} is an eigenvector of $\mathbf{A}'\mathbf{A}$.
- A **left singular vector** of \mathbf{A} is an eigenvector of $\mathbf{A}\mathbf{A}'$.
- The $\min(M, N)$ singular values of a matrix $\mathbf{A} \in \mathbb{F}^{M \times N}$ are the square roots of the eigenvalues of $\mathbf{A}'\mathbf{A}$ or $\mathbf{A}\mathbf{A}'$, whichever is the smaller matrix.

Example. For the case where \mathbf{A} is tall and $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}'$, then

$$\mathbf{A}'\mathbf{A} = \mathbf{V} \underbrace{\Sigma' \Sigma}_{\Lambda} \mathbf{V}' = \mathbf{V} \underbrace{\begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_N^2 \end{bmatrix}}_{\Lambda} \mathbf{V}',$$

which has eigenvalues $\{\sigma_k^2\}$. So the square roots of those eigenvalues of $\mathbf{A}'\mathbf{A}$ (properly sorted) are the singular values of \mathbf{A} .

This is about all we can say to relate SVDs and eigendecompositions for general (rectangular) matrices, so next we turn to square matrices.

In general, if \mathbf{A} is an arbitrary diagonalizable square matrix, then its eigendecomposition $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$ is *unrelated* to any SVD of \mathbf{A} .

However, for **normal** matrices, we can relate any eigendecomposition of \mathbf{A} to an SVD of \mathbf{A} .

If \mathbf{A} is a **normal** $N \times N$ matrix (*e.g.*, Hermitian symmetric), then it has a unitary eigendecomposition

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}' = \sum_{n=1}^N \lambda_n \mathbf{v}_n \mathbf{v}'_n.$$

Without loss of generality, we can order the eigenvalues with decreasing magnitudes, *i.e.*, $|\lambda_1| \geq \dots \geq |\lambda_N|$. However, even after doing so, $\mathbf{V}\Lambda\mathbf{V}'$ is still not an SVD of \mathbf{A} in general because some of the eigenvalues λ_n can be negative or even complex, so $\Sigma \neq \Lambda$. To construct an SVD of \mathbf{A} in terms of \mathbf{V} and Λ we use the fact that $\lambda_n = \text{sign}(\lambda_n) |\lambda_n|$, where $\text{sign}(|z| e^{i\angle z}) = e^{i\angle z}$ for $z \in \mathbb{C}$, as follows:

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}' = \sum_{n=1}^N \lambda_n \mathbf{v}_n \mathbf{v}'_n = \underbrace{\sum_{n=1}^N \underbrace{|\lambda_n|}_{\sigma_n} \underbrace{\text{sign}(\lambda_n)}_{\mathbf{u}_n} \mathbf{v}_n \mathbf{v}'_n}_{\boxed{\mathbf{U}\Sigma\mathbf{V}'}}. \quad (2.6)$$

So when \mathbf{A} is normal with eigenvectors \mathbf{V} and eigenvalues Λ with descending magnitudes, an SVD of \mathbf{A} is:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}', \quad \mathbf{U} = \mathbf{V}\mathbf{S}, \quad \mathbf{S} \triangleq \text{diag}\{\text{sign}(\lambda_n)\}, \quad \Sigma = \text{diag}\{|\lambda_n|\}. \quad (2.7)$$

Of course this SVD is not unique; we could have associated some or all of the sign values with \mathbf{V} , for example.

The construction (2.6) shows that if \mathbf{A} is **normal**, then for any unitary eigendecomposition of \mathbf{A} of the form $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'$, we can construct an SVD of the form (2.7) where the matrix \mathbf{V} of right singular vectors consists entirely of eigenvectors of \mathbf{A} , and where the matrix \mathbf{U} of left singular vectors also consists of entirely of eigenvectors of \mathbf{A} , because in (2.6) we have $\mathbf{u}_n = \text{sign}(\lambda_n)\mathbf{v}_n$ so \mathbf{u}_n is also an eigenvector of \mathbf{A} .

In short, if \mathbf{A} is **normal** than we can construct an SVD of \mathbf{A} using any orthonormal set of eigenvectors of \mathbf{A} . However, it does *not* follow that all singular vectors of a **normal** matrix \mathbf{A} are eigenvectors!

Example. Consider $\mathbf{A} = \begin{bmatrix} 3 & 0 \\ 0 & -3 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. This \mathbf{x} is a right singular vector because $\mathbf{A}'\mathbf{A}\mathbf{x} = 9\mathbf{x}$. However, this \mathbf{x} is not an eigenvector of \mathbf{A} . To elaborate, here are two different SVDs of \mathbf{A} , one that is constructed from a set of eigenvectors of \mathbf{A} , and one that is not:

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} 3 & 0 \\ 0 & -3 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{V}} \underbrace{\begin{bmatrix} 3 & 0 \\ 0 & -3 \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{V}'} && \text{(an elementary eigendecomposition)} \\ &= \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}}_{\tilde{\mathbf{U}}} \underbrace{\begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{V}'} && \text{(SVD vers. 1, using } \mathbf{V} \text{ and } \mathbf{u}_n = \text{sign}(\lambda_n)\mathbf{v}_n\text{)} \\ &= \underbrace{\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}}_{\hat{\mathbf{U}}} \underbrace{\begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}}_{\Sigma} \underbrace{\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}}_{\hat{\mathbf{V}}'} && \text{(SVD vers. 2, unrelated to eigenvectors of } \mathbf{A}\text{).} \end{aligned}$$

Fact. If \mathbf{A} is **normal** and has eigenvalues with distinct magnitudes, or if eigenvalues with equal magnitudes have equal values, then for every SVD of \mathbf{A} , the left and right singular vectors are all eigenvectors of \mathbf{A} . (Proof in a HW problem.)

When does $\mathbf{U} = \mathbf{V}$?

A question about the SVD $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}'$ that sometimes arises is: When does $\mathbf{U} = \mathbf{V}$?

This is an imprecise question because \mathbf{U} and \mathbf{V} are not unique. A more precise version is this:

For what class of matrices does there exist an SVD in which $\mathbf{U} = \mathbf{V}$?

First, if $N \neq M$ then \mathbf{U} and \mathbf{V} have different sizes. So focus here on the square case where $N = M$.

If $\mathbf{A} = \mathbf{V}\Sigma\mathbf{V}'$ then clearly \mathbf{A} is Hermitian symmetric.

But is symmetry a sufficient condition for possibly having $\mathbf{U} = \mathbf{V}$? **No.**

Recall from (2.3) that we can write any (Hermitian) symmetric matrix as $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'$ where Λ is the diagonal of the eigenvalues. This looks a lot like an SVD with $\mathbf{U} = \mathbf{V}$. However, for the SVD we always have $\sigma_k \geq 0$ whereas the eigenvalues of (symmetric) \mathbf{A} are real but not necessarily nonnegative. 

So to have $\mathbf{U} = \mathbf{V}$ we need \mathbf{A} to be Hermitian symmetric *and* to have nonnegative eigenvalues.

An SVD of \mathbf{A} can have $\mathbf{U} = \mathbf{V}$ iff \mathbf{A} is square, $\mathbf{A} = \mathbf{A}'$, and all eigenvalues of \mathbf{A} are nonnegative.

Refer to (2.6) for the key idea.

Example. Consider the eigendecomposition of the following matrix and the following three (not unique!) SVD forms:

$$\begin{aligned}
 \mathbf{A} &= \underbrace{\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}}_{\mathbf{V}} = \underbrace{\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}}_{\Sigma} \underbrace{\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}}_{\mathbf{V}'}
 \quad (\text{eigendecomposition}) \\
 &= \underbrace{\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}}_{\Sigma} \underbrace{\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}}_{\mathbf{V}'}
 \quad (\text{SVD version 1}) \\
 &= \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}}_{\tilde{\mathbf{U}}} \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\tilde{\mathbf{V}}'}
 \quad (\text{SVD version 2}) \\
 &= \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\tilde{\mathbf{U}}} \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}}_{\tilde{\mathbf{V}}'}
 \quad (\text{SVD version 3}).
 \end{aligned}$$

Even though \mathbf{A} is symmetric, and we have exhibited three different SVDs for it, we cannot find an SVD here where \mathbf{U} and \mathbf{V} are the same because \mathbf{A} has a negative eigenvalue:

$$\det\{\mathbf{A} - z\mathbf{I}\} = z^2 - 4 \implies z = \pm 2.$$

Which of the above forms corresponds to (2.6)? SVD version 1, where $\mathbf{U} = \mathbf{V} \operatorname{diag}\{\operatorname{sign}(\lambda_i)\}$.

SVD clicker questions

Every permutation matrix has an SVD. (?)

A: True

B: False

??

If $A = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 2 \end{bmatrix}$, then $\max_{x \in \mathbb{R}^3 : \|x\|=1} \|Ax\| = 5$. (?)

A: True

B: False

??

The singular values of a permutation matrix all equal to 1. (?)

A: True

B: False

??

(A uniqueness question.) If $A = U\Sigma V'$ and $A = \tilde{U}\tilde{\Sigma}\tilde{V}'$ are both SVDs of A , then $\Sigma = \tilde{\Sigma}$. (?)

A: True

B: False

??

??

By definition, to determine if x is an **eigenvector** of a square matrix A , simply compute $y = Ax$ and see if $y = \alpha x$ for some $\alpha \in \mathbb{F}$.

If x is an **eigenvector** of a **normal** matrix A having unitary eigendecomposition $A = V\Lambda V'$, then $x = \alpha v_j$ for some $\alpha \in \mathbb{F}$, where v_j is one of the columns of V . (?)

A: True

B: False

??

For a $M \times N$ matrix A , how do we test if $v \in \mathbb{F}^N$ is a right singular vector or $u \in \mathbb{F}^M$ is a left singular vector?

- A **right singular vector** of A is an eigenvector of $A'A$.
- A **left singular vector** of A is an eigenvector of AA' .

If x is a **right singular vector** of a matrix A having SVD $A = U\Sigma V'$, then $x = \alpha v_j$ for some $\alpha \in \mathbb{F}$, where v_j is one of the columns of V . (?)

A: True

B: False

??

2.5 Positive semidefinite matrices

L§10.2

The preceding example is the exception, not the rule. Most of the time we use the SVD for non-square matrices anyway. As mentioned previously, when we do consider a square matrix, it is usually a Gram matrix $\mathbf{X}'\mathbf{X}$ or an outer-product matrix $\mathbf{X}\mathbf{X}'$. These matrices are not only symmetric, they are **positive semi-definite**.

Define. A $N \times N$ (square) Hermitian matrix \mathbf{A} is **positive semi-definite** iff $\mathbf{x}'\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{C}^N$.

Define. A (square) Hermitian matrix \mathbf{A} is **positive definite** iff $\mathbf{x}'\mathbf{A}\mathbf{x} > 0$ for all $\mathbf{x} \neq 0$.

Lemma. If $\mathbf{A} = \mathbf{B}\mathbf{B}'$ for any matrix \mathbf{B} , then \mathbf{A} is positive semi-definite.

Proof. $\mathbf{x}'\mathbf{A}\mathbf{x} = \mathbf{x}'\mathbf{B}\mathbf{B}'\mathbf{x} = \|\mathbf{B}'\mathbf{x}\|_2^2 \geq 0$.

We write $\mathbf{A} \succ 0$ to denote positive definite and $\mathbf{A} \succeq 0$ to denote positive semi-definite.

Which of the following statements is true?

- A: All positive-semidefinite matrices are positive definite.
- B: All positive-definite matrices are positive semi-definite.
- C: Neither statement is always true.

??

In words, any Gram matrix $\mathbf{X}'\mathbf{X}$ or outer-product matrix $\mathbf{X}\mathbf{X}'$ is positive semi-definite, i.e., $\mathbf{X}'\mathbf{X} \succeq 0$.

Theorem. If $\mathbf{A} = \mathbf{B}\mathbf{B}'$ for any matrix \mathbf{B} , then $\mathbf{A} = \mathbf{U}\Sigma\mathbf{U}'$ with $\Sigma_{ii} \geq 0$.

In words, for such matrices an eigendecomposition with (real, nonnegative) eigenvalues in descending order is also an SVD.

Proof. Let $\mathbf{B} = \mathbf{U}\Sigma_B\mathbf{V}'$ denote an SVD of \mathbf{B} . Recall Σ_B is real and nonnegative.

Then $\mathbf{A} = \mathbf{B}\mathbf{B}' = \mathbf{U}\Sigma_B\mathbf{V}'\mathbf{V}\Sigma'_B\mathbf{U}' = \mathbf{U}\Sigma_B\Sigma'_B\mathbf{U}' = \mathbf{U}\Sigma\mathbf{U}'$, where $\Sigma = \Sigma_B\Sigma'_B$ is diagonal with entries σ_k^2 (and some zeros if \mathbf{B} is tall).

So when $\mathbf{A} = \mathbf{B}\mathbf{B}'$ the *eigenvalues* of \mathbf{A} are the *square of the singular values* of \mathbf{B} (and some zeros if \mathbf{B} is tall), and hence real and nonnegative.

Example. Consider $\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 3 \\ 1 & 0 \end{bmatrix} \implies \mathbf{A} = \mathbf{B}\mathbf{B}' = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 9 & 0 \\ 1 & 0 & 1 \end{bmatrix}$

`eigvals(A)` returns $(0, 2, 9)$

`svdvals(B)` returns $(3, \sqrt{2})$

Note that `eigen` and `eigvals` do not return eigenvalues in descending magnitude order in general, whereas `svd` and `svdvals` always return singular values in descending order.



Relating positive (semi)definiteness to eigenvalues

Let \mathbf{A} be any $N \times N$ Hermitian matrix, then \mathbf{A} has an eigendecomposition of the form $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'$.

Now $\mathbf{x}'\mathbf{A}\mathbf{x} = \mathbf{x}'\mathbf{V}\Lambda\mathbf{V}'\mathbf{x} = \mathbf{z}'\Lambda\mathbf{z} = \sum_i \lambda_i |z_i|^2$ where $\mathbf{z} = \mathbf{V}'\mathbf{x}$.

Thus if (Hermitian) \mathbf{A} has all nonnegative eigenvalues, then $\mathbf{x}'\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{C}^N$, so $\mathbf{A} \succeq 0$.

The converse is also true: If \mathbf{A} is symmetric positive semi-definite, then \mathbf{A} has nonnegative eigenvalues.

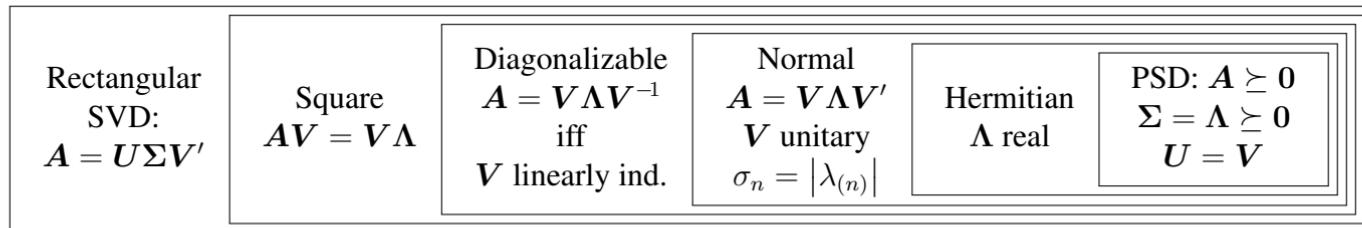
Combining, a Hermitian matrix \mathbf{A} is **positive semi-definite** iff all of its eigenvalues are nonnegative.

Similarly, a Hermitian matrix \mathbf{A} is **positive definite** iff all of its eigenvalues are positive.

To summarize, any positive semi-definite matrix has real and nonnegative eigenvalues,

and it has an SVD that matches its eigendecomposition (with descending eigenvalue order), with $\mathbf{U} = \mathbf{V}$ and $\Sigma = \Lambda$, i.e., $\sigma_n(\mathbf{A}) = \lambda_n(\mathbf{A}) \geq 0$.

Venn diagram of matrices and decompositions



where $|\lambda_{(n)}|$ denotes the n th largest magnitude eigenvalue.

2.6 Summary

In practice, we usually end up using:

- an eigendecomposition, $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'$, when working with positive semi-definite matrices (like a Gram matrix or outer-product matrix),
- the SVD, $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}'$, for most other matrices.

A curious note about terminology:

- columns of \mathbf{V} are called the **right eigenvectors** for the eigendecomposition, because $\mathbf{AV} = \mathbf{V}\Lambda$ even though $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'$.
- columns of \mathbf{U} are called the **left singular vectors** for the SVD, because $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}'$.



SVD computation using eigendecomposition

(Read)

There is a concise overview of practical computation of the SVD here: [\[wiki\]](#).

To perform an SVD by hand using an eigendecomposition, or if you were stuck on a desert island with a computer that had an `eigen` command but no `svd` command, here is how you could do it for the case of a tall $M \times N$ matrix with $M \geq N$ and **rank** N .

First use the eigendecomposition of $\mathbf{A}\mathbf{A}'$ to find \mathbf{U} and Σ :

$$\mathbf{A}\mathbf{A}' = \mathbf{U}\Lambda\mathbf{U}' = \mathbf{U} \underbrace{\Sigma\Sigma'}_{M \times M} \mathbf{U}'.$$

Here \mathbf{U} will be the left singular vectors of \mathbf{A} and the $N \leq M$ singular values will be $\sigma_n = \sqrt{\lambda_n}$, $n = 1, \dots, N$. Now obtain \mathbf{V}' by multiplying \mathbf{A} on the left by $\text{diag}\{1/\sigma_n\} \mathbf{U}[:, 1:N]'$ as follows:

$$\begin{aligned} \text{diag}\{1/\sigma_n\} \mathbf{U}[:, 1:N]' \mathbf{A} &= \text{diag}\{1/\sigma_n\} \mathbf{U}[:, 1:N]' (\mathbf{U}\Sigma\mathbf{V}') = \text{diag}\{1/\sigma_n\} [\mathbf{I} \quad \mathbf{0}] \Sigma \mathbf{V}' \\ &= \text{diag}\{1/\sigma_n\} \text{diag}\{\sigma_n\} \mathbf{V}' = \mathbf{V}'. \end{aligned}$$

Unfortunately this process does not work when \mathbf{A} is not full rank, and it requires an SVD of the “large” size $M \times M$ so it is impractical.

Example.

(Read)

Exercise. Find an **SVD** of $\mathbf{A} = \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix}$.

Recall that this (square but asymmetric) matrix does not have an orthogonal eigendecomposition.

$$\mathbf{A}'\mathbf{A} = \begin{bmatrix} 0 & 0 \\ 3 & 0 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 9 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 9 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \mathbf{V}\Sigma^2\mathbf{V}' \implies \mathbf{V} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ and } \Sigma = \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix}.$$

$$\mathbf{A}\mathbf{A}' = \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 0 \end{bmatrix} = \mathbf{I}_2 \begin{bmatrix} 9 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{I}_2 = \mathbf{U}\Sigma^2\mathbf{U}' \implies \mathbf{U} = \mathbf{I}_2.$$

Note that to find \mathbf{V} properly, we applied a permutation to have the eigenvalues of $\mathbf{A}'\mathbf{A}$ in descending order. And in general one would need to do more work to properly match the \mathbf{U} and \mathbf{V} ordering.

Thus an SVD is $\mathbf{A} = \underbrace{\mathbf{U}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}}_{\mathbf{V}'}$.

Bibliography

- [1] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005.
- [2] G. W. Stewart. “On the early history of the singular value decomposition”. In: *SIAM Review* 35.4 (1993), 551–66.

- [3] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki. “The singular value decomposition: anatomy of optimizing an algorithm for extreme scale”. In: *SIAM Review* 60.4 (2018), 808–65.

Chapter 3

Subspaces and rank

Contents (final version)

3.0 Introduction	3.3
3.1 Subspaces	3.4
Span	3.7
Linear independence	3.9
Basis	3.11
Dimension	3.14
Sums and intersections of subspaces	3.15
Direct sum of subspaces	3.17
Dimensions of sums of subspaces	3.18
Orthogonal complement of a subspace	3.19
Linear transformations	3.20
Range of a matrix	3.21
3.2 Rank of a matrix	3.23
Rank of a matrix product	3.26
Unitary invariance of rank / eigenvalues / singular values	3.28

3.3 Nullspace and the SVD	3.29
Nullspace or kernel	3.29
The four fundamental spaces	3.33
Anatomy of the SVD	3.34
SVD of finite differences (discrete derivative)	3.39
Synthesis view of matrix decomposition	3.42
3.4 Orthogonal bases	3.43
3.5 Spotting eigenvectors	3.46
3.6 Summary	3.50

3.0 Introduction

An important operation in signal processing and machine learning is **dimensionality reduction**. There are many such methods, but the starting point is usually *linear* methods that map data to a lower-dimensional set called a **subspace**. The notion of *dimension* is quantified by **rank**. This chapter reviews subspaces, span, dimension, rank and null space. These linear algebra concepts may seem to lack signal processing context initially, but they are crucial to thoroughly understanding the SVD, a primary tool for the rest of the course (and beyond).

Source material for this chapter includes [1, §2.2-2.4, 3.4, 3.1, 3.5, 5.1].

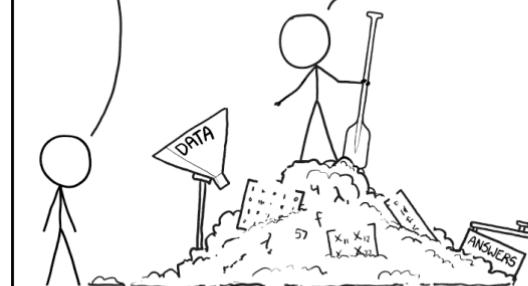
From <https://xkcd.com/1838>

THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG
PILE OF LINEAR ALGEBRA, THEN COLLECT
THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.



3.1 Subspaces

L§2.2

Define [1, Def. 2.6]. For a **vector space** \mathcal{V} defined on a field \mathbb{F} , a nonempty subset $\mathcal{S} \subseteq \mathcal{V}$ is called a **subspace** or **linear subspace** of \mathcal{V} iff

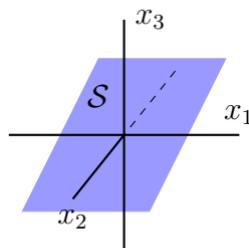
- \mathcal{S} is closed under vector addition: $\mathbf{u}, \mathbf{v} \in \mathcal{S} \implies \mathbf{u} + \mathbf{v} \in \mathcal{S}$
- \mathcal{S} is closed under scalar multiplication: $\mathbf{v} \in \mathcal{S}$ and $\alpha \in \mathbb{F} \implies \alpha\mathbf{v} \in \mathcal{S}$

Fact. A subspace \mathcal{S} always includes the zero vector $\mathbf{0}$.

Proof. Because a field \mathbb{F} always includes the scalar 0 , and because \mathcal{S} is nonempty, it contains some vector \mathbf{v} . Because \mathcal{S} is closed under scalar multiplication, \mathcal{S} contains the vector $0\mathbf{v}$ which is the zero vector. \square

Laub [1, p. 9] uses the notation $\mathcal{S} \subseteq \mathcal{V}$ to indicate that \mathcal{S} is a subspace of \mathcal{V} , although the more usual meaning of the symbol \subseteq is to denote a subset.

When visualizing subspaces, think of lines or planes or hyperplanes going through the origin $\mathbf{0}$.



Example. $\mathcal{S} = \{\mathbf{0}\}$ where $\mathbf{0} \in \mathcal{V}$ for some vector space \mathcal{V} .

This is the most minimalist and uninteresting subspace.

Example. The subspace of **symmetric** matrices $\mathcal{S} = \{\mathbf{A} \in \mathbb{R}^{N \times N} : \mathbf{A} \text{ is symmetric}\}$.

Here $\mathcal{V} = \mathbb{R}^{N \times N}$ and $\mathcal{S} \subseteq \mathcal{V}$.

It is easy to verify that \mathcal{S} is closed under vector addition and scalar multiplication.

Example. The subset of **orthogonal** matrices $\mathcal{S} = \{\mathbf{A} \in \mathbb{R}^{N \times N} : \mathbf{A}'\mathbf{A} = \mathbf{I}\}$.

Is this a subspace of $\mathbb{R}^{N \times N}$?

A: Y

B: N

C: Insufficient information

??

Example. $\mathcal{S} = \{\alpha \mathbf{1}_N : \alpha \in \mathbb{C}\} \subseteq \mathbb{C}^N$. We will see shortly that here $\mathcal{S} = \text{span}(\mathbf{1}_N)$.

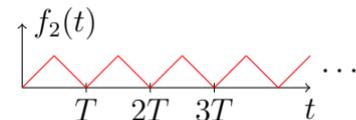
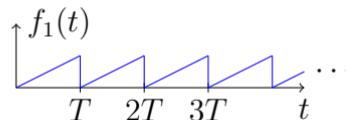
Periodic functions as a subspace

(A signal processing example.)

Let \mathcal{V} denote the vector space consisting of all 1D **periodic functions** having period T for some $T \neq 0$.

In other words, if $f \in \mathcal{V}$, then $f(t + T) = f(t)$, $\forall t \in \mathbb{R}$.

One can verify that \mathcal{V} is indeed a vector space, *i.e.*, it is closed under vector addition and scalar multiplication.



Now consider the set \mathcal{S} of 1D functions having period $T > 0$ that are band-limited to maximum frequency KT for some $K \in \mathbb{N}$.

Exercise. Verify that \mathcal{S} is a **subspace** in \mathcal{V} , *i.e.*, $\mathcal{S} \subseteq \mathcal{V}$.

Is the set of all 1D periodic functions a vector space?

No, it is not closed under summation because the sum of two periodic functions with different periods whose ratio is irrational is aperiodic.

The vector space \mathcal{V} above is **infinite dimensional** because $\forall k \in \mathbb{Z}$, $e^{i2\pi kt/T} \in \mathcal{V}$, and that set is **linearly independent**. In fact it is a basis, and that fact is the foundation of **Fourier series**.



Span

L§2.3

Define. Given a set of vectors $\{\mathbf{u}_1, \dots, \mathbf{u}_N\}$ in a vector space \mathcal{V} over field \mathbb{F} , the **span** of those vectors is

$$\text{span}(\{\mathbf{u}_1, \dots, \mathbf{u}_N\}) \triangleq \left\{ \sum_{n=1}^N \alpha_n \mathbf{u}_n : \alpha_n \in \mathbb{F} \right\}. \quad (3.1)$$

This set is also called the **linear span** or **hull**. (Caution: it differs from the **convex hull** of a set.)

Often we will collect the vectors into a matrix $\mathbf{U} = [\mathbf{u}_1 \dots \mathbf{u}_N]$ and define $\text{span}(\mathbf{U}) \triangleq \text{span}(\{\mathbf{u}_1, \dots, \mathbf{u}_N\})$, although the usual mathematical definition is that the argument of $\text{span}(\cdot)$ is a set of vectors, not a matrix.

Exercise. Verify that $\text{span}(\{\mathbf{u}_1, \dots, \mathbf{u}_N\})$ is a **subspace** of the vector space \mathcal{V} .

Example. For $\mathcal{V} = \mathbb{R}^3$, if $\mathbf{u}_1 = (1, 0, 1)$ and $\mathbf{u}_2 = (1, 0, -1)$ then $\text{span}(\{\mathbf{u}_1, \mathbf{u}_2\})$ is the entire (x, z) plane.

Example. For $\mathcal{V} = \mathbb{R}^{N \times N}$, the vector space of $N \times N$ matrices, $\text{span}(\{\mathbf{e}_1 \mathbf{e}'_1, \dots, \mathbf{e}_N \mathbf{e}'_N\})$ is the subspace of all $N \times N$ diagonal matrices because

$$\begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_N \end{bmatrix} = d_1 \begin{bmatrix} 1 & & \\ & 0 & \\ & & \ddots \\ & & & 0 \end{bmatrix} + \cdots + d_N \begin{bmatrix} 0 & & \\ & \ddots & \\ & & 1 \end{bmatrix} = d_1 \mathbf{e}_1 \mathbf{e}'_1 + \cdots + d_N \mathbf{e}_N \mathbf{e}'_N.$$



Span of infinite collection of vectors

The definition of **span** in (3.1) is for a finite set of vectors. Some infinite-dimensional vector spaces are also important, such as the space of T -periodic functions above. Another example is the vector space of all polynomials. To work with such vector spaces, we use the following more general definition of **span**.

Define. If \mathcal{S} is a (possibly uncountably infinite) subset of a vector space \mathcal{V} over field \mathbb{F} , the span of \mathcal{S} consists of *all* (finite by definition) **linear combinations** of elements in \mathcal{S} :

$$\text{span}(\mathcal{S}) \triangleq \left\{ \mathbf{x} \in \mathcal{V} : \mathbf{x} = \sum_{n=1}^N \alpha_n \mathbf{u}_n, \mathbf{u}_n \in \mathcal{S}, \alpha_n \in \mathbb{F}, N \in \mathbb{N} \right\}. \quad (3.2)$$

Define. The span of the empty set is the zero vector: $\text{span}(\emptyset) = \mathbf{0}$.

These definitions ensure that the **span** of *any* set (empty, finite, or infinite) is a subspace.

Example. Let $\mathcal{V} = \mathbb{R}^3$ and consider the following (uncountably infinite) set:

$$\mathcal{S} = \{\alpha(1, 1, 1) : \alpha \in \mathbb{R}\} \cup \{(1, 0, 0)\}.$$

In words: \mathcal{S} is a line (through the origin) and another point not on that line.

What is $\text{span}(\mathcal{S})$?

A: a line

B: a plane

C: all of \mathbb{R}^3

D: None of these

??

Linear independence

Define. A set of vectors $\mathbf{u}_1, \dots, \mathbf{u}_N$ is **linearly dependent** iff there exists a tuple of coefficients $\alpha_1, \dots, \alpha_N \in \mathbb{F}$, not all of which are zero, where

$$\sum_{n=1}^N \alpha_n \mathbf{u}_n = \mathbf{0}.$$

In words, a set of vectors is **linearly dependent** iff any one of the vectors is in the **span** of the other vectors. This latter interpretation generalizes to infinite dimensional vector spaces.

A set of vectors that is not linearly dependent is called a linearly independent set.

L§2.3

Define. A set of vectors $\mathbf{u}_1, \dots, \mathbf{u}_N$ in a vector space is **linearly independent** iff for any scalars $\alpha_1, \dots, \alpha_N \in \mathbb{F}$:

$$\sum_{n=1}^N \alpha_n \mathbf{u}_n = \mathbf{0} \implies \alpha_1 = \dots = \alpha_N = 0.$$

In other words, no **linear combination** of the vectors is zero, except when all the coefficients are zero.

Example. In $\mathbb{R}^{N \times N}$, the set of matrices $\{\mathbf{e}_1\mathbf{e}'_1, \dots, \mathbf{e}_N\mathbf{e}'_N\}$ is linearly independent.

Generalization to infinite sets



Recall that $\mathcal{S} - \{\mathbf{x}\}$ denotes the set \mathcal{S} with the vector \mathbf{x} removed.

Define. A (possibly uncountably infinite) set of vectors \mathcal{S} in a vector space is **linearly dependent** iff

$$\exists \mathbf{x} \in \mathcal{S} \text{ s.t. } \mathbf{x} \in \text{span}(\mathcal{S} - \{\mathbf{x}\}),$$

otherwise the set is called **linearly independent**.

Example. Consider the vector space of all polynomials. Let \mathcal{S} denote the (countably infinite) set of all **monomials**, i.e., $\mathcal{S} = \{f(x) = x^n : n = 0, 1, \dots\}$. One can show by elementary algebra [wiki] that \mathcal{S} is linearly independent. (One cannot write a monomial like x^5 as a linear combination of other monomials.)

Basis

Now we use the concepts of linear independence and span to define a particularly important concept: basis. L§2.3

Define [1, Def. 2.14]. A set of vectors $\{\mathbf{b}_1, \dots, \mathbf{b}_N\}$ in (a vector space or subspace) \mathcal{V} is a **basis** for \mathcal{V} iff

- $\{\mathbf{b}_1, \dots, \mathbf{b}_N\}$ is a **linearly independent** set,
- $\text{span}(\{\mathbf{b}_1, \dots, \mathbf{b}_N\}) = \mathcal{V}$.

In general bases are not unique; nearly all vector spaces have multiple bases; the only exception is the trivial vector space $\mathcal{V} = \{\mathbf{0}\}$. In other words, generally a basis is *not unique*.

Example. If $\{\mathbf{b}_1, \dots, \mathbf{b}_N\}$ is a basis for \mathcal{V} , then $\{-\mathbf{b}_1, \dots, -\mathbf{b}_N\}$ is also a basis for \mathcal{V} ,

Example. Euclidean space \mathbb{R}^N has many interesting bases used in signal processing, including those based on the DFT, the **discrete cosine transform (DCT)**, and **orthogonal wavelet transform (OWT)**, among others.

Fact. If $\{\mathbf{b}_1, \dots, \mathbf{b}_N\}$ is a basis for \mathcal{V} , then every $\mathbf{v} \in \mathcal{V}$ has a *unique* representation of the form

$$\mathbf{v} = \sum_{n=1}^N \alpha_n \mathbf{b}_n = \mathbf{B} \boldsymbol{\alpha}, \quad \boldsymbol{\alpha} = [\alpha_1 \ \dots \ \alpha_N]^\top, \quad \mathbf{B} = [\mathbf{b}_1 \ \dots \ \mathbf{b}_N]. \quad (3.3)$$

See [1, p. 12]. The coefficients $\{\alpha_n \in \mathbb{F}\}$ are the **coordinates** of \mathbf{v} with respect to the basis \mathbf{B} .

A basis is a generalization of the usual concept of a **coordinate system**.

(Read)

Sketch of proof of uniqueness of (3.3).

For any $\mathbf{v} \in \mathcal{V}$, there *exists* some coordinates $(\alpha_1, \dots, \alpha_N)$ by definition of **span**.

To prove uniqueness, use contradiction. Suppose the representation were not unique, *i.e.*, suppose there exists $(\beta_1, \dots, \beta_N)$ such that $\mathbf{v} = \sum_{n=1}^N \beta_n \mathbf{b}_n$, where at least one β_n differs from α_n . Then $\mathbf{0} = \mathbf{v} - \mathbf{v} = \sum_{n=1}^N \beta_n \mathbf{b}_n - \sum_{n=1}^N \alpha_n \mathbf{b}_n = \sum_{n=1}^N (\beta_n - \alpha_n) \mathbf{b}_n$. But because at least one coefficient in that sum is nonzero, that would imply that the set $\{\mathbf{b}_n\}$ is linearly dependent, contradicting the definition of a basis.

Basis vectors are always nonzero, because of the linear independence condition in the definition.

Example. The set of complex exponential signals $\{e^{i2\pi kt/T}, t \in \mathbb{R} : -K \leq k \leq K\}$ is a **basis** for all T -periodic signals that are band-limited with maximum frequency K/T .

This fact about sinusoids (not proved here) is the foundation for **additive synthesis** musical sound generation.

The definition of **basis** above also generalizes to infinite dimensional spaces. Simply replace $\{b_1, \dots, b_N\}$ with $\{b_1, b_2, \dots\}$ and allow an infinite sum in (3.3). Dealing rigorously with infinite sums requires care ♦♦ beyond the scope of EECS 551; see EECS 600!

Example. The set of monomials is a basis for the vector space of all polynomials.

Example. The following vector space is important in signal processing.

The vector space of sinusoids of frequency ν is: $\mathcal{V} = \{A \cos(2\pi\nu t + \phi), t \in \mathbb{R} : A, \phi \in \mathbb{R}\}$.

Is $\mathcal{S} = \{3 \cos(2\pi\nu t), 5 \sin(2\pi\nu t), 7 \cos(2\pi\nu t - \pi/4)\}$ a basis for \mathcal{V} ?

Hint: [\[wiki\]](#)

- A: Yes
- B: No, because \mathcal{S} has linear dependence
- C: No, because \mathcal{S} does not span \mathcal{V}
- D: Both B & C.

??

Warm-up questions

Let $\mathcal{V} = \mathbb{R}^3$ and $\mathcal{S} \triangleq \{(1, 1, 1), (1, 0, 0)\}$. What is $\text{span}(\mathcal{S})$?

- A: 0 B: a line C: a plane D: all of \mathbb{R}^3 E: None of these

??

Upper triangular matrices are a subspace of $\mathbb{F}^{N \times N}$. (?)

- A: True B: False

??

A basis for such upper triangular matrices in vector space $\mathbb{F}^{N \times N}$ contains how many vectors?

- A: N B: N^2 C: $N^2/2$ D: $N(N - 1)$ E: None of these

??

Dimension

L§2.3

Define. The **dimension** of a subspace \mathcal{S} is the number of elements in any **basis** for \mathcal{S} .

This definition is well defined because every subspace has a **basis**, and, even though that basis is not unique in general, every basis has the same number of elements (possibly infinite).

Example. $\dim(\mathbb{R}^N) = N$. Use the canonical or standard basis: $\{e_n : n = 1, \dots, N\}$.

See [1, Ex. 2.20] for further examples.

What is the dimension of the subspace of $N \times N$ diagonal matrices?

- A: 1 B: N C: $2N$ D: N^2 E: ∞

??

Define. There are two types of subspaces (and vector spaces).

- A **finite-dimensional** (sub)space has a basis with $\dim \in \mathbb{N}$.
- Otherwise we call the (sub)space **infinite dimensional**.

What is the dimension of the trivial vector space $\mathcal{V} = \{0\}$? $\text{span}(\emptyset) = \mathbf{0}$ and $\dim(\emptyset) = 0$. See p. 3.8.

What is the dimension of the vector space of all polynomials?

- A: 0 B: 1 C: infinite D: undefined

??

Dimension of a span

Fact. If $\mathcal{S} = \text{span}(\{\mathbf{u}_1, \dots, \mathbf{u}_N\})$ then $\dim(\mathcal{S}) \leq N$.

Exercise. Prove using the definition of dimension and basis.

To further discuss dimension, we first need to discuss subspace sums.

Sums and intersections of subspaces

L§2.4

Define. If $\mathcal{S}, \mathcal{T} \subseteq \mathcal{V}$ then

- the **sum** of two subspaces is defined as

$$\mathcal{S} + \mathcal{T} = \{s + t : s \in \mathcal{S}, t \in \mathcal{T}\}$$

- the **intersection** of two subspaces is defined as

$$\mathcal{S} \cap \mathcal{T} = \{v \in \mathcal{V} : v \in \mathcal{S}, v \in \mathcal{T}\}.$$

Exercise. Verify that $\mathcal{S} + \mathcal{T}$ and $\mathcal{S} \cap \mathcal{T}$ are both subspaces of \mathcal{V} . (See [1, Theorem 2.22].)

Example. If \mathcal{S} is the subspace of upper Hessenberg matrices in $\mathbb{R}^{N \times N}$ and \mathcal{T} is the subspace of lower Hessenberg matrices in $\mathbb{R}^{N \times N}$ then $\mathcal{S} + \mathcal{T} = \mathbb{R}^{N \times N}$.

Considering the same \mathcal{S} and \mathcal{T} , what is $\mathcal{S} \cap \mathcal{T}$?

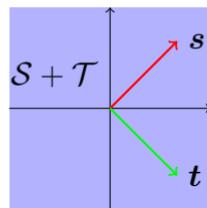
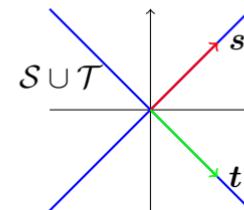
- A: \emptyset B: diagonal matrices C: tridiagonal matrices D: $\mathbb{R}^{N \times N}$ E: None of these

??

Caution: $\mathcal{S} + \mathcal{T}$ is *not* the same as the **union** of subspaces $\mathcal{S} \cup \mathcal{T}$.

Example. Consider $\mathcal{V} = \mathbb{R}^2$ and $\mathcal{S} = \text{span}(s)$, $s = (1, 1)$ and $\mathcal{T} = \text{span}(t)$, $t = (1, -1)$.

Then the **subspace sum** $\mathcal{S} + \mathcal{T} = \mathbb{R}^2$ because $\{s, t\}$ spans \mathbb{R}^2 whereas the **subspace union** $\mathcal{S} \cup \mathcal{T}$ is just two lines:

 \neq 

We may discuss **union of subspaces** [2–4] later in the course.

A union of subspaces is a subspace. (?)

- A: True

- B: False

??

Direct sum of subspaces

Now we define a particularly useful version of subspace sum.

L§2.4

Define. We write subspace \mathcal{U} as a **direct sum**

$$\mathcal{U} = \mathcal{S} \oplus \mathcal{T}$$

of subspaces \mathcal{S} and \mathcal{T} , all in a common vector space \mathcal{V} , iff

- $\mathcal{S} \cap \mathcal{T} = \mathbf{0}$
- $\mathcal{S} + \mathcal{T} = \mathcal{U}$.

In which case we say \mathcal{S} and \mathcal{T} are **complements** of each other in \mathcal{U} .

Example. $\mathcal{V} = \mathbb{R}^2$ and $\mathcal{S} = \text{span}(s)$, $s = (1, 1)$ and $\mathcal{T} = \text{span}(t)$, $t = (1, -1)$.

Then $\mathcal{V} = \mathcal{S} \oplus \mathcal{T}$.

For $s, t \neq \mathbf{0}$ and $t \neq \alpha s$ for all α , let $\mathcal{S} = \text{span}(s)$ and $\mathcal{T} = \text{span}(t)$. Is $\text{span}(\{s, t\}) = \mathcal{S} \oplus \mathcal{T}$?

- A: Always
- B: If and only if s and t are orthogonal
- C: If and only if s and t are orthonormal
- D: Never
- E: None of the above.

??

Dimensions of sums of subspaces

Yet another definition that is crucial for understanding the SVD.

[1, Theorem 2.26] If $\mathcal{U} = \mathcal{S} \oplus \mathcal{T}$ then:

- every $\mathbf{u} \in \mathcal{U}$ can be written uniquely in the form $\mathbf{u} = \mathbf{s} + \mathbf{t}$ for some $\mathbf{s} \in \mathcal{S}$ and $\mathbf{t} \in \mathcal{T}$,
- $\dim(\mathcal{U}) = \dim(\mathcal{S}) + \dim(\mathcal{T})$.

To prove uniqueness, suppose $\mathbf{u} = \mathbf{s}_1 + \mathbf{t}_1 = \mathbf{s}_2 + \mathbf{t}_2$.

Then $\underbrace{\mathbf{s}_1 - \mathbf{s}_2}_{\in \mathcal{S}} = \underbrace{\mathbf{t}_2 - \mathbf{t}_1}_{\in \mathcal{T}}$. But $\mathcal{S} \cap \mathcal{T} = \mathbf{0} \implies \mathbf{s}_1 - \mathbf{s}_2 = \mathbf{0}$ and $\mathbf{t}_2 - \mathbf{t}_1 = \mathbf{0} \implies$ the representation is unique.

This property gives us a tool to help quantify dimension.

Example. In the previous example, $\dim(\mathcal{S}) = \dim(\mathcal{T}) = 1$ and $\dim(\mathcal{V}) = 2$.

(Read)

More generally, if we have any two subspaces in a vector space \mathcal{V} , then [1, Thm. 2.27]:

$$\dim(\mathcal{S} + \mathcal{T}) = \dim(\mathcal{S}) + \dim(\mathcal{T}) - \dim(\mathcal{S} \cap \mathcal{T}). \quad (3.4)$$

So the direct sum equation above is a special case of this equality.

(We use this fact later in a proof about low-rank decomposition.)

Example. In $\mathcal{V} = \mathbb{R}^3$, if \mathcal{S} is the $x-y$ plane and \mathcal{T} is the $y-z$ plane, then $\dim(\mathcal{S} + \mathcal{T}) = 3$, $\dim(\mathcal{S}) = \dim(\mathcal{T}) = 2$, and $\dim(\mathcal{S} \cap \mathcal{T}) = 1$.

Orthogonal complement of a subspace

L§3.4

Define. For a subspace \mathcal{S} of a vector space \mathcal{V} , the **orthogonal complement** of \mathcal{S} is the subset of vectors in \mathcal{V} that are orthogonal to every vector in \mathcal{S} :

$$\mathcal{S}^\perp = \{\mathbf{v} \in \mathcal{V} : \langle \mathbf{s}, \mathbf{v} \rangle = \mathbf{v}' \mathbf{s} = 0, \forall \mathbf{s} \in \mathcal{S}\}.$$

Key properties of orthogonal complements when \mathcal{V} is finite dimensional (like \mathbb{R}^N or \mathbb{C}^N) [1, Theorem 3.11]:

\mathcal{S}^\perp is itself a **subspace** of \mathcal{V}

$$(\mathcal{S}^\perp)^\perp = \mathcal{S} \tag{3.5}$$

$$\mathcal{S} \oplus \mathcal{S}^\perp = \mathcal{V} \tag{3.6}$$

$$\dim(\mathcal{S}) + \dim(\mathcal{S}^\perp) = \dim(\mathcal{V}) \tag{3.7}$$

Example. For $\mathcal{V} = \mathbb{R}^3$, if $\mathcal{S} = \text{span}((1, 1, 0), (1, -1, 0))$ then $\mathcal{S}^\perp = \text{span}((0, 0, 1))$.

In \mathbb{R}^3 , if \mathcal{S} is a line through the origin, then what geometric shape is \mathcal{S}^\perp ?

A: empty set

B: point

C: line

D: plane

E: \mathbb{R}^3

??

Linear transformations

(Read) L§3.1

Define. Let \mathcal{V} and \mathcal{W} be **vector spaces** on a common field \mathbb{F} . A function $L : \mathcal{V} \mapsto \mathcal{W}$ is a **linear transformation** or **linear map** or **linear transform** [1, Def. 3.1] iff

$$L(\alpha\mathbf{u} + \beta\mathbf{v}) = \alpha L(\mathbf{u}) + \beta L(\mathbf{v}), \quad \forall \alpha, \beta \in \mathbb{F} \text{ and } \forall \mathbf{u}, \mathbf{v} \in \mathcal{V}.$$

Example. Consider $\mathcal{V} = \mathbb{R}^2$ and let \mathcal{W} denote the space of T -periodic functions on \mathbb{R} .

Construct $L(\cdot)$ by $s = L([a \ b]^\top) \iff s(t) = a \cos(2\pi t/T) + b \sin(2\pi 5t/T + \pi/4)$.

Exercise. Verify that $L(\cdot)$ is a linear transformation.

Matrix-vector multiplication as a linear transformation

Example. Let $\mathcal{V} = \mathbb{C}^N$ and $\mathcal{W} = \mathbb{C}^M$ and $\mathbf{A} \in \mathbb{C}^{M \times N}$.

Consider the transformation defined by $\mathbf{y} = L(\mathbf{x}) \iff \mathbf{y} = \mathbf{Ax}$, i.e., $\mathbf{x} \mapsto \mathbf{y} = \mathbf{Ax}$.

This transformation is **linear**.

Proof. $L(\alpha\mathbf{x} + \beta\mathbf{z}) = \mathbf{A}(\alpha\mathbf{x} + \beta\mathbf{z}) = \alpha\mathbf{Ax} + \beta\mathbf{Az} = \alpha L(\mathbf{x}) + \beta L(\mathbf{z})$, where $\alpha, \beta \in \mathbb{C}$ are arbitrary as are $\mathbf{x}, \mathbf{z} \in \mathbb{C}^N$. □

For more examples of linear transformations, see [1, Ex. 3.2].

Range of a matrix

Define. The **range** of a matrix $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_N]$, also known as its **column space**, is the span of its columns:

$$\mathcal{R}(\mathbf{A}) \triangleq \text{span}(\{\mathbf{a}_1, \dots, \mathbf{a}_N\}).$$

Equivalently:

$$\mathcal{R}(\mathbf{A}) = \{ \mathbf{Ax} : \mathbf{x} \in \mathbb{F}^N \}.$$

The range of a matrix in $\mathbb{F}^{M \times N}$ is a subspace of \mathbb{F}^M .

Define. The **row space** of a matrix \mathbf{A} is the span of its rows: $\mathcal{R}(\mathbf{A}')$.

If \mathbf{D} is a diagonal matrix in $\mathbb{R}^{N \times N}$, what is $\mathcal{R}(\mathbf{D})$?

- A: \emptyset B: Usually \mathbb{R}^N . C: Always \mathbb{R}^N . D: Usually $\mathbb{R}^{N \times N}$. E: Always $\mathbb{R}^{N \times N}$.

??

Range and matrix multiplication

(Read)

Often we are interested in the range of matrix products.

Clearly if $\mathbf{A} \in \mathbb{F}^{M \times N}$ and $\mathbf{B} \in \mathbb{F}^{N \times K}$ then

$$\mathcal{R}(\mathbf{AB}) \subset \mathcal{R}(\mathbf{A}), \quad (3.8)$$

because $\mathcal{R}(\mathbf{AB}) = \{\mathbf{Ax} : \mathbf{x} = \mathbf{Bz}, \mathbf{z} \in \mathbb{F}^K\} \subset \{\mathbf{Ax} : \mathbf{x} \in \mathbb{F}^N\}.$

If \mathbf{A} is a $M \times N$ matrix and \mathbf{B} is a $N \times N$ **invertible matrix** then $\mathcal{R}(\mathbf{AB}) = \mathcal{R}(\mathbf{A}).$

Proof. Using (3.8) it suffices to show that $\mathcal{R}(\mathbf{A}) \subset \mathcal{R}(\mathbf{AB}).$

If $\mathbf{y} \in \mathcal{R}(\mathbf{A})$ then there is some $\mathbf{x} \in \mathbb{F}^N$ such that $\mathbf{y} = \mathbf{Ax}.$ Because \mathbf{B} is invertible, we can define $\mathbf{z} = \mathbf{B}^{-1}\mathbf{x}$ for which $\mathbf{ABz} = \mathbf{AB}(\mathbf{B}^{-1}\mathbf{x}) = \mathbf{Ax}$ so $\mathbf{y} \in \mathcal{R}(\mathbf{AB}).$ \square

However, invertibility of \mathbf{B} is not a necessary condition.

Using concepts introduced later, one can show that a more general sufficient condition is for \mathbf{B} to have full **row rank**.

But full row rank is also not a necessary condition.

Example. If $\mathbf{A} = \mathbf{0}$, then $\mathcal{R}(\mathbf{A}) = \mathcal{R}(\mathbf{AB})$ for any suitably sized matrix $\mathbf{B}.$

3.2 Rank of a matrix

The preceding material about subspaces applied to vectors in general vector spaces. Now we specialize to a concept that is specific to matrices: the **rank** of a matrix.

L§3.5

Define. For any $M \times N$ matrix \mathbf{A} :

- **column rank** of $\mathbf{A} \triangleq \dim(\mathcal{R}(\mathbf{A})) = \#$ of linearly independent columns of \mathbf{A}
- **row rank** of $\mathbf{A} \triangleq \dim(\mathcal{R}(\mathbf{A}')) = \#$ of linearly independent rows of \mathbf{A}

Theorem. For any $M \times N$ matrix \mathbf{A} , its **row rank** = its **column rank**.

Proof.

(Read)

Let r denote the column rank of $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_N]$. Because r is the column rank, there exists a basis $\mathbf{V} = [\mathbf{v}_1 \dots \mathbf{v}_r]$ such that one can write every vector in $\mathcal{R}(\mathbf{A})$ as a linear combination of the columns of \mathbf{V} . Each column of \mathbf{A} is in $\mathcal{R}(\mathbf{A})$, thus we can express each column of \mathbf{A} as a linear combination of the columns of \mathbf{V} , i.e.,

$$\mathbf{a}_1 = c_{11}\mathbf{v}_1 + \cdots + c_{r1}\mathbf{v}_r$$

$$\vdots$$

$$\mathbf{a}_N = c_{1N}\mathbf{v}_1 + \cdots + c_{rN}\mathbf{v}_r,$$

where the c_{ij} values denote the coordinates or coefficients w.r.t. the basis \mathbf{V} .

In matrix form we get a sum-of-outer-products form of matrix multiplication:

$$\underbrace{\mathbf{A}}_{M \times N} = \underbrace{\begin{bmatrix} \mathbf{v}_1 & \dots & \mathbf{v}_r \end{bmatrix}}_{M \times r} \underbrace{\begin{bmatrix} c_{11} & \dots & c_{1N} \\ \vdots & & \\ c_{r1} & \dots & c_{rN} \end{bmatrix}}_{r \times N} = \begin{bmatrix} \mathbf{v}_1 & \dots & \mathbf{v}_r \end{bmatrix} \begin{bmatrix} - & \mathbf{c}_1^\top & - \\ - & \vdots & - \\ - & \mathbf{c}_r^\top & - \end{bmatrix} = \mathbf{V}\mathbf{C} = \sum_{k=1}^r \mathbf{v}_k \mathbf{c}_k^\top.$$

Now the m th row of \mathbf{A} is a linear combination of the rows of \mathbf{C} :

$$\mathbf{A}_{[m,:]} = \mathbf{e}'_m \mathbf{A} = \sum_{k=1}^r (\mathbf{e}'_m \mathbf{v}_k) \mathbf{c}_k^\top = \sum_{k=1}^r v_{mk} \mathbf{c}_k^\top.$$

This construction holds for any row of \mathbf{A} (or linear combinations thereof)

$$\begin{aligned} \Rightarrow \mathcal{R}(\mathbf{A}') &= \text{row space of } \mathbf{A} \subseteq \mathcal{R}(\mathbf{C}^\top) \\ \Rightarrow \text{row rank of } \mathbf{A} &\leq r = \text{column rank of } \mathbf{A}. \end{aligned}$$

Because \mathbf{A} was arbitrary, the same argument applies to its transpose, so:

$$\begin{aligned} \text{row rank of } \mathbf{A}' &\leq \text{column rank of } \mathbf{A}' \\ \Rightarrow \text{column rank of } \mathbf{A} &\leq \text{row rank of } \mathbf{A} \\ \Rightarrow \text{column rank of } \mathbf{A} &= \text{row rank of } \mathbf{A}. \end{aligned}$$

□

For an alternate proof, see [1, Theorem. 3.17].

Rank definition summary

Because the row rank and column rank of a matrix are always identical, we generally simply speak of the **rank** of a matrix without the “row” or “column” qualifier. And it suffices to define:

$$\text{rank}(\mathbf{A}) \triangleq \dim(\mathcal{R}(\mathbf{A})).$$

Corollary:

$$\mathbf{A} \in \mathbb{F}^{M \times N} \implies \text{rank}(\mathbf{A}) \leq \min(M, N). \quad (3.9)$$

Proof.

(Read)

$$\text{rank}(\mathbf{A}) = \text{row rank of } \mathbf{A} \leq \# \text{ rows} = M$$

$$\text{rank}(\mathbf{A}) = \text{col rank of } \mathbf{A} \leq \# \text{ cols} = N$$

$$\implies \text{rank}(\mathbf{A}) \leq \min(M, N) \quad \square$$

What is the rank of $\begin{bmatrix} 5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 7 \end{bmatrix}$?

A: 0

B: 1

C: 2

D: 3

E: 4

??

Rank of a matrix product

L§3.5

Theorem. Multiplying matrices never increases rank:

$$\text{rank}(\mathbf{AB}) \leq \min(\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B})). \quad (3.10)$$

Proof.

(Read)

$$\mathbf{AB} = \begin{bmatrix} | & & | \\ \mathbf{a}_1 & \dots & \mathbf{a}_N \\ | & & | \end{bmatrix} \begin{bmatrix} - & \mathbf{b}_1^\top & - \\ & \vdots & \\ - & \mathbf{b}_N^\top & - \end{bmatrix} = \mathbf{a}_1\mathbf{b}_1^\top + \dots + \mathbf{a}_N\mathbf{b}_N^\top$$

\implies every column of \mathbf{AB} is a linear combination of columns of \mathbf{A}

$\implies \mathcal{R}(\mathbf{AB}) \subseteq \mathcal{R}(\mathbf{A})$

$\implies \text{rank}(\mathbf{AB}) \leq \text{rank}(\mathbf{A})$

Similarly, because $(\mathbf{AB})' = \mathbf{B}'\mathbf{A}'$, applying the same logic we have

$\text{rank}(\mathbf{AB}) = \text{rank}((\mathbf{AB})') = \text{rank}(\mathbf{B}'\mathbf{A}') \leq \text{rank}(\mathbf{B}') = \text{rank}(\mathbf{B})$.

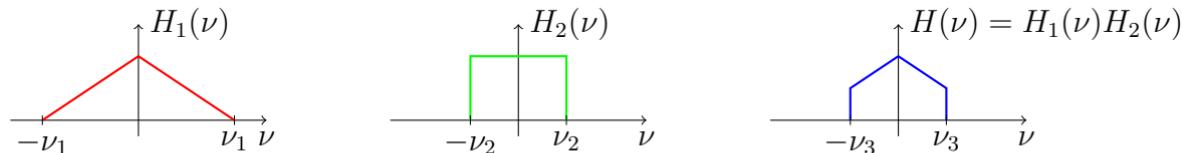
Combining both inequalities yields (3.10). □

\mathbf{AB} is a composition of two linear transforms.

\implies Composition cannot enlarge subspace dimension.

Caution: in general: $\text{rank}(\mathbf{AB}) \neq \text{rank}(\mathbf{BA})$.

Example. DSP analogy: cascade of two filters with band-limited frequency responses.



where $[\nu_3, \nu_3] = [\nu_1, \nu_1] \cap [\nu_2, \nu_2]$. Composing filters cannot recover lost frequencies.

For $\mathbf{u} \in \mathbb{C}^M$ and $\mathbf{v} \in \mathbb{C}^N$, what is the minimum and maximum possible rank of **outer product** $\mathbf{u}\mathbf{v}'$?

- A: 0,1 B: 1,1 C: 0,min(M,N) D: 1,min(M,N) E: None of these

??

Other properties of rank

There is also a lower bound for the rank of a product, called **Sylvester's rank inequality**: if \mathbf{A} is an $m \times n$ matrix and \mathbf{B} is $n \times k$, then

$$\text{rank}(\mathbf{A}) + \text{rank}(\mathbf{B}) - n \leq \text{rank}(\mathbf{AB}).$$

Furthermore, rank is **subadditive**. If $\mathbf{A}, \mathbf{B} \in \mathbb{F}^{M \times N}$, then

$$\text{rank}(\mathbf{A} + \mathbf{B}) \leq \text{rank}(\mathbf{A}) + \text{rank}(\mathbf{B}).$$

Warm-up question(s)

For $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{C}^M$ and $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{C}^N$, what is the minimum and maximum possible rank of the $M \times N$ matrix $3\mathbf{u}_1\mathbf{v}_1' + 5\mathbf{u}_2\mathbf{v}_2'$?

- A: 0,1 B: 1,2 C: 0,min(M,N) D: 1,min(M,N) E: None of these ??

If $\mathcal{V} = \mathbb{F}^6$ and \mathbf{x} and \mathbf{y} are two linearly independent vectors in \mathcal{V} , then what is the dimension of $(\text{span}(\{\mathbf{x}, \mathbf{y}\}))^\perp$?

- A: 1 or less B: 2 C: 3 D: 4 E: 5 or more ??

Unitary invariance of rank / eigenvalues / singular values

Theorem. If $\mathbf{A} \in \mathbb{F}^{M \times N}$ then multiplying by a **unitary** matrix on the left or right does not change the rank:

- $\mathbf{Q} \in \mathbb{F}^{M \times M}$ and \mathbf{Q} unitary $\implies \text{rank}(\mathbf{Q}\mathbf{A}) = \text{rank}(\mathbf{A})$
- $\mathbf{Q} \in \mathbb{F}^{N \times N}$ and \mathbf{Q} unitary $\implies \text{rank}(\mathbf{A}\mathbf{Q}) = \text{rank}(\mathbf{A})$

Rank and eigenvalues

Corollary. If \mathbf{A} is Hermitian (or **normal**) with eigendecomposition $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}'$ then because \mathbf{V} is unitary:

$$\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{V}\Lambda\mathbf{V}') = \text{rank}(\Lambda) = \text{number of nonzero eigenvalues of } \mathbf{A}.$$

Rank and SVD

- By unitary invariance, if \mathbf{A} has SVD $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}'$, then $\text{rank}(\mathbf{A}) = \text{rank}(\Sigma)$.
- $\text{rank}(\Sigma) = r = \# \text{ of nonzero singular values of } \mathbf{A}$, because $\mathcal{R}(\Sigma) = \text{span}(\{\mathbf{e}_1, \dots, \mathbf{e}_r\})$

Thus the SVD expression for a matrix \mathbf{A} having rank r where $r \leq \min(M, N)$ simplifies:

$$\mathbf{A} \in \mathbb{F}^{M \times N} \implies \mathbf{A} = \mathbf{U}\Sigma\mathbf{V}' = \sum_{k=1}^{\min(M,N)} \sigma_k \mathbf{u}_k \mathbf{v}'_k = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k.$$

When we write $\sum_{k=1}^{\min(M,N)}$ then some of the σ_k values may be zero, namely $\sigma_{r+1}, \dots, \sigma_{\min(M,N)}$. When we write $\sum_{k=1}^r$ where r is the rank of \mathbf{A} , then all the σ_k values used in the sum are nonzero.

3.3 Nullspace and the SVD

To fully understand an SVD of a matrix, we need both **range** and **nullspace** (and orthogonal complements thereof).

Nullspace or kernel

L§3.4

The set of vectors that yield zero when multiplied by a matrix is often important.

Define. The **null space** or **kernel** of a $M \times N$ matrix \mathbf{A} is

$$\mathcal{N}(\mathbf{A}) = \ker(\mathbf{A}) \triangleq \{\mathbf{x} \in \mathbb{F}^N : \mathbf{Ax} = \mathbf{0}\}.$$

Clearly we always have $\mathbf{0} \in \mathcal{N}(\mathbf{A})$.

Exercise. Verify that $\mathcal{N}(\mathbf{A})$ is indeed a **subspace**.

Thus using the subspace font “ \mathcal{N} ” is appropriate.

If $\mathbf{A} \in \mathbb{C}^{M \times N}$, then $\mathcal{N}(\mathbf{A})$ is a subspace of what vector space?

A: \mathbb{C}^M B: \mathbb{C}^N C: \mathbb{R}^N D: \mathbb{R}^M

E: None of these.

??

Decomposition theorem for matrices

If $\mathbf{A} \in \mathbb{F}^{M \times N}$ then the input and output spaces of \mathbf{A} satisfy [1, Theorem 3.14]:

$$\mathcal{N}(\mathbf{A}) \oplus \mathcal{N}^\perp(\mathbf{A}) = \mathbb{F}^N$$

$$\mathcal{R}(\mathbf{A}) \oplus \mathcal{R}^\perp(\mathbf{A}) = \mathbb{F}^M.$$

In other words, every “input” vector $\mathbf{x} \in \mathbb{F}^N$ can be decomposed uniquely as $\mathbf{x} = \mathbf{x}_0 + \mathbf{x}_1$, where $\mathbf{x}_0 \in \mathcal{N}(\mathbf{A})$ so $\mathbf{A}\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{x}_1 \in \mathcal{N}^\perp(\mathbf{A})$.

Likewise, every vector $\mathbf{y} \in \mathbb{F}^M$ can be decomposed uniquely as $\mathbf{y} = \mathbf{y}_1 + \mathbf{y}_0$, where $\mathbf{y}_1 \in \mathcal{R}(\mathbf{A})$, so $\mathbf{A}\mathbf{x}_1 = \mathbf{y}_1$ for some $\mathbf{x}_1 \in \mathbb{F}^N$, and $\mathbf{y}_0 \in \mathcal{R}^\perp(\mathbf{A})$.

The above statement was for an arbitrary $\mathbf{y} \in \mathbb{F}^M$.

If we have an “output vector” $\mathbf{y} = \mathbf{A}\mathbf{x}$, then $\mathbf{y} \in \mathcal{R}(\mathbf{A})$ by definition and $\mathbf{y}_0 = \mathbf{0}$.

Relationships between null space and range of a matrix

[1, Theorem 3.12]. For any matrix \mathbf{A} , its null space and range are related by:

$$\mathcal{N}^\perp(\mathbf{A}) = \mathcal{R}(\mathbf{A}')$$

$$\mathcal{R}^\perp(\mathbf{A}) = \mathcal{N}(\mathbf{A}').$$

Proof: If $\mathbf{x} \in \mathcal{N}(\mathbf{A})$ then $\mathbf{Ax} = \mathbf{0}$, so $\forall \mathbf{y}$ we have $\mathbf{y}'(\mathbf{Ax}) = 0 \implies \mathbf{x}'(\mathbf{A}'\mathbf{y}) = 0 \implies \mathbf{x} \in \mathcal{R}^\perp(\mathbf{A}')$.

Conversely, if $\mathbf{x} \in \mathcal{R}^\perp(\mathbf{A}')$ then $\forall \mathbf{y}, \mathbf{0} = \mathbf{x}'(\mathbf{A}'\mathbf{y})$.

Take $\mathbf{y} = \mathbf{Ax}$ and we have $\|\mathbf{Ax}\| = 0$ which implies $\mathbf{Ax} = \mathbf{0}$, so $\mathbf{x} \in \mathcal{N}(\mathbf{A})$.

Thus using (3.5): $\mathcal{N}(\mathbf{A}) = \mathcal{R}^\perp(\mathbf{A}') \implies \mathcal{N}^\perp(\mathbf{A}) = \mathcal{R}(\mathbf{A}')$.

The proof of the second equality is left to HW (cf. [1, Problem 3.6]). □

Corollary:

$$\begin{aligned} \dim(\mathcal{N}^\perp(\mathbf{A})) &= \dim(\mathcal{R}(\mathbf{A}')) = \dim(\mathcal{R}(\mathbf{A})) = \text{rank}(\mathbf{A}) \\ \dim(\mathcal{R}^\perp(\mathbf{A})) &= \dim(\mathcal{N}(\mathbf{A}')). \end{aligned} \tag{3.11}$$

Nullity

Define. The **nullity** of a matrix is the dimension of its null space:

$$\text{nullity}(\mathbf{A}) \triangleq \dim(\mathcal{N}(\mathbf{A})).$$

Rank plus **nullity** property [1, Corollary 3.18]. If $\mathbf{A} \in \mathbb{F}^{M \times N}$ then

$$\underbrace{\dim(\mathcal{N}(\mathbf{A}))}_{\text{nullity}(\mathbf{A})} + \underbrace{\dim(\mathcal{R}(\mathbf{A}))}_{\text{rank}(\mathbf{A})} = N.$$

Proof. Because $\mathbb{F}^N = \mathcal{N}(\mathbf{A}) \oplus \mathcal{N}^\perp(\mathbf{A})$ we have from (3.6), (3.7) and (3.11):

$$N = \dim(\mathcal{N}(\mathbf{A})) + \dim(\mathcal{N}^\perp(\mathbf{A})) = \dim(\mathcal{N}(\mathbf{A})) + \text{rank}(\mathbf{A}).$$

□

If $\mathbf{u} \in \mathbb{C}^M - \{0\}$ and $\mathbf{v} \in \mathbb{C}^N - \{0\}$, what is the nullity of their **outer product**, i.e., $\text{nullity}(\mathbf{u}\mathbf{v}')$?

A: 0

B: 1

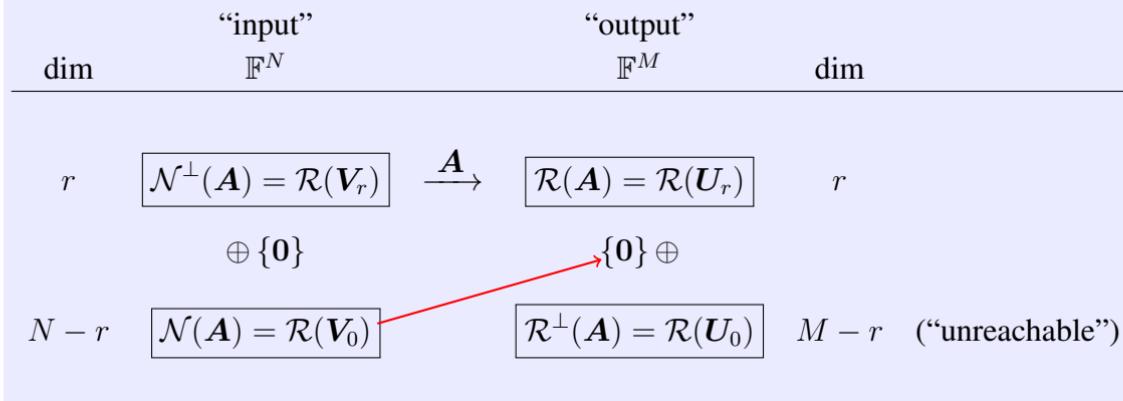
C: $N - 1$ D: N E: M

??

The four fundamental spaces related to a matrix \mathbf{A}

Now we unify the null space and range space (and their orthogonal complements) for a general matrix \mathbf{A} . L§3.5

The following diagram summarizes the **fundamental theorem of linear algebra** for a matrix $\mathbf{A} \in \mathbb{F}^{M \times N}$ where we think of \mathbf{A} as a mapping from \mathbb{F}^N to \mathbb{F}^M . See [1, Section 3.5] and [1, Theorem 5.1].



(The next page defines the matrices \mathbf{U}_r , \mathbf{U}_0 , \mathbf{V}_r , \mathbf{V}_0 .)

$\mathbf{A} \in \mathbb{F}^{M \times N}$ has rank $r \leq \min(M, N)$, so we can partition the SVD components as follows:

$$\mathbf{A} = \sum_{k=1}^{\min(M,N)} \sigma_k \mathbf{u}_k \mathbf{v}'_k = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k = \mathbf{U} \Sigma \mathbf{V}' = \underbrace{\left[\begin{array}{c|c} \mathbf{U}_r & \mathbf{U}_0 \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right]}_{\substack{M \times r \mid M \times (M-r)}} \underbrace{\left[\begin{array}{c|c} \Sigma_r & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right]}_{\substack{r \times r \\ (N-r) \times (N-r)}} \underbrace{\left[\begin{array}{c|c} \mathbf{V}'_r \\ \hline \mathbf{V}'_0 \end{array} \right]}_{\substack{r \times N \\ (N-r) \times N}}$$

where Σ_r is $r \times r$ and contains the *nonzero* singular values of \mathbf{A} along its diagonal.

What is the size of the lower right 0 above?

- A: $M \times N$ B: $M \times (N - r)$ C: $(M - r) \times N$ D: $(M - r) \times (N - r)$ E: None of these

??

If $\mathbf{x} = \mathbf{V}_0 z$ for some $z \in \mathbb{F}^{N-r}$, then what is \mathbf{Ax} ?

- A: 0 B: $\mathbf{U}_r \Sigma_r \mathbf{z}$ C: $\mathbf{U}_r \mathbf{V}_r \mathbf{z}$ D: $\mathbf{U}_0 \mathbf{V}_r \mathbf{z}$ E: None of these

??

Anatomy of the SVD

There are two cases of the above partitioning to consider in more detail: when \mathbf{A} is **tall** or **wide**.

SVD of a tall matrix

When \mathbf{A} is “**tall**” or “thin,” i.e., $M > N \implies r \leq N < M$, then we can simplify:

$$\underbrace{\mathbf{A}}_{M \times N} = \mathbf{U}\Sigma\mathbf{V}' = \left[\begin{array}{c|c} \mathbf{U}_r & \mathbf{U}_0 \\ \hline M \times r & M \times (M-r) \end{array} \right] \left[\begin{array}{c} \Sigma_r \\ \hline \mathbf{0} \end{array} \right] \underbrace{\mathbf{V}'_r}_{r \times N} = \underbrace{\mathbf{U}_r}_{M \times r} \underbrace{\Sigma_r}_{r \times r} \underbrace{\mathbf{V}'_r}_{r \times N}. \quad (3.12)$$

What is the size of the lower $\mathbf{0}$ above?

- A: $M \times N$ B: $M \times (N-r)$ C: $(M-r) \times N$ D: $(M-r) \times (N-r)$ E: None of these

??

Caution. When we write $\mathbf{U}\Sigma\mathbf{V}' = \mathbf{U}_r\Sigma_r\mathbf{V}'_r$, we do not mean that the individual terms match (they do not!); we mean that the overall product matches.



If $r = N$ then $\mathcal{N}(\mathbf{A}) = \mathbf{0}$ and $\mathbf{V}_r = \mathbf{V}_N = \mathbf{V}$ and there is no \mathbf{V}_0 .

SVD of a wide matrix

When \mathbf{A} is **wide**, i.e., $M < N \implies r \leq M < N$, then we can simplify:

$$\underbrace{\mathbf{A}}_{M \times N} = \mathbf{U} \Sigma \mathbf{V}' = \underbrace{\mathbf{U}_r}_{M \times r} \underbrace{\left[\begin{array}{c|c} \Sigma_r & \mathbf{0} \end{array} \right]}_{r \times N} \underbrace{\begin{bmatrix} \mathbf{V}'_r \\ \mathbf{V}'_0 \end{bmatrix}}_{N \times N} \quad \} \quad \begin{matrix} r \times N \\ (N-r) \times N \end{matrix} = \underbrace{\mathbf{U}_r}_{M \times r} \underbrace{\Sigma_r}_{r \times r} \underbrace{\mathbf{V}'_r}_{r \times N}.$$

If $r = M$ then $\dim(\mathcal{R}(\mathbf{A})) = M$ and $\mathbf{U}_r = \mathbf{U}_M = \mathbf{U}$ and there is no \mathbf{U}_0 and all of \mathbb{F}^M is reachable, i.e., $\mathcal{R}(\mathbf{A}) = \mathbb{F}^M$.

However, note that $\dim(\mathcal{N}(\mathbf{A})) = N - r > M - r \geq 0$, so a wide \mathbf{A} has a nontrivial null space.

Note the symmetry between the above two “compact” SVD representations for the tall and wide cases, as there must be because if \mathbf{A} is tall then \mathbf{A}' is wide.

Practical use of SVD

(Read)

The above cases are somewhat (but not exactly) related to the `(U, s, V) = svd(A, full=false)` command in JULIA.

- `(U, s, V) = svd(A, full=true)` returns the “full” SVD, where U is $M \times M$ and V is $N \times N$ and s is a vector of length $\min(M, N)$.
- `(U, s, V) = svd(A, full=false)` or just `(U, s, V) = svd(A)` returns the “economy” SVD, where U is $M \times \min(M, N)$ and $U * \text{Diagonal}(s) * V'$ equals A to within numerical precision.

In class, when we speak of the **compact SVD**, we mean $A = U_r \Sigma_r V_r'$ as described in these pages.

Computing the compact SVD requires multiple JULIA commands, such as:

```
r = rank(A)
```

```
U, s, V = svd(A)
```

```
Ur = U[:, 1:r]; Vr = V[:, 1:r]; sr = s[1:r]
```

after which we can recover A to within numerical precision by

```
Ur * Diagonal(sr) * Vr'
```

Exercise. Make a small tall matrix like `A = [4 2; 2 1; 0 0]` and experiment with the SVD options above and look at numerical effects by evaluating the recovery error `A - U * Diagonal(s) * V'`

For a **tall** matrix \mathbf{A} , when we use the (default) `full=false` option of JULIA's SVD, *i.e.*,

`(U, s, V) = svd(A)`

or

`(U, s, V) = svd(A, full=false)`

then the output components correspond to:

$$\underbrace{\begin{bmatrix} \mathbf{U}_{:,1} & \dots & \mathbf{U}_{:,N} \end{bmatrix}}_{\substack{M \times N \\ \mathbf{U}}} \underbrace{\begin{bmatrix} \Sigma_r & \mathbf{0}_{(N-r) \times r} \\ \mathbf{0}_{(N-r) \times (N-r)} & \mathbf{0}_{(N-r) \times (N-r)} \end{bmatrix}}_{\substack{N \times N \\ \text{Diagonal}(s)}} \underbrace{\begin{bmatrix} \mathbf{V}' \\ \mathbf{V}' \end{bmatrix}}_{\substack{N \times N \\ \mathbf{V}'}} ,$$

where $\mathbf{s} = (\sigma_1, \dots, \sigma_r, 0, \dots, 0)$ has length $N \leq M$ in the tall case.

In practice, the last $N - r$ elements of the vector \mathbf{s} will not be exactly zero, because of finite numerical precision.

Caution: for a tall \mathbf{A} , \mathbf{U} is $M \times N$ whereas \mathbf{U} is $M \times M$, and \mathbf{s} is a N vector whereas Σ is $M \times N$.

Example. For a concrete example, see the rank-1 case on p. 3.48.

SVD of finite differences (discrete derivative)

(Read)

Example. The **derivative** of $f(t) = \sin(\omega t)$ is $\dot{f}(t) = \omega \cos(\omega t)$ and the second derivative is $\ddot{f}(t) = -\omega^2 \sin(\omega t)$. Consider an analog system whose inputs are twice differentiable signals, and whose output is the second derivative of the input:

$$f(t) \rightarrow \boxed{\text{second derivative}} \rightarrow \ddot{f}(t).$$

The **eigenfunctions** of this system include any signal of the form $A \cos(\omega t + \phi)$ or $A \sin(\omega t + \phi)$ or $A e^{\omega t + \phi}$, all of which have eigenvalue $-\omega^2$. We now explore the matrix-vector analog of this property.

Consider the $(N - 1) \times N$ matrix \mathbf{C} that performs a **finite difference** operation when multiplying a vector:

$$\mathbf{C} \triangleq \begin{bmatrix} -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 \\ & \ddots & \ddots & & & \\ 0 & \dots & 0 & -1 & 1 & 0 \\ 0 & \dots & 0 & 0 & -1 & 1 \end{bmatrix}, \text{ so } \mathbf{C}\mathbf{x} = \begin{bmatrix} x_2 - x_1 \\ \vdots \\ x_N - x_{N-1} \end{bmatrix} \text{ for } \mathbf{x} \in \mathbb{C}^N. \quad (3.13)$$

The $N \times N$ Gram matrix $\mathbf{C}'\mathbf{C}$ is almost Toeplitz and the related $N - 1 \times N - 1$ 2nd-difference matrix \mathbf{CC}'

is exactly Toeplitz:

$$\mathbf{C}'\mathbf{C} = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & & \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & 0 & \dots & 0 & -1 & 1 \end{bmatrix} \quad \mathbf{C}\mathbf{C}' = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & & \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & 0 & \dots & 0 & -1 & 2 \end{bmatrix}$$

These three important matrices arise in many signal and image processing applications (see HW02 and EECS 556) and in every field of engineering that uses a **finite-element method (FEM)** based on finite differences to approximate differential equations arising from physics.

One can show using **trigonometric identities** that an economy SVD of \mathbf{C} involves the **discrete cosine transform (DCT)** and **discrete sine transform (DST)** as follows [5]:

$$\mathbf{C} = \mathbf{U}\Sigma\mathbf{V}' = \text{DST } \Sigma \text{ DCT}'. \quad (3.14)$$

So for this important matrix, the left and right singular vectors turn out to form matrices that are themselves important in signal processing. The facts that $\mathbf{C}'\mathbf{C} = \text{DCT } \Sigma^2 \text{ DCT}'$ and $\mathbf{C}\mathbf{C}' = \text{DST } \Sigma^2 \text{ DST}'$ are discrete analogues of the fact that the second derivative of a sinusoid is a sinusoid. The following code verifies (3.14).

```
using LinearAlgebra
N = 6
C = diagm(0 => -ones(Int,N-1), 1 => ones(Int,N-1))[1:(N-1),:] # diff

h = pi / N # from strang:06:bts
U = [sin(j*k*h) for j=1:(N-1), k=1:(N-1)] # DST
for i=1:(N-1); U[:,i] ./= norm(U[:,i]); end
V = [cos((2*j-1)*k*h/2) for j=1:N, k=1:(N-1)] # DCT
for i=1:(N-1); V[:,i] ./= norm(V[:,i]); end
@show maximum(abs.(U' * C * V .* (1 .- Matrix(I, N-1, N-1)))) # verify
```

Synthesis view of matrix decomposition

(Read)

- Eigen-decomposition of square matrix (when it exists, *e.g.*, \mathbf{A} Hermitian):

$$\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}' = \sum_i \lambda_i \mathbf{q}_i \mathbf{q}_i'$$

with $\mathbf{Q}'\mathbf{Q} = \mathbf{I}$

- SVD of a $M \times N$ matrix:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}' = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}_k'$$

where $\sigma_k > 0$ for $k = 1, \dots, r$ = rank, and $\mathbf{U}'\mathbf{U} = \mathbf{I}_M$ and $\mathbf{V}'\mathbf{V} = \mathbf{I}_N$.

In both cases we can “synthesize” the matrix using a sum of rank-1, outer-product terms.

3.4 Orthogonal bases

In signal processing and beyond, bases consisting of orthogonal vectors are particularly important.

Define. A collection of vectors $\{\mathbf{b}_1, \mathbf{b}_2, \dots\}$ in a vector space \mathcal{V} is called a **orthogonal basis** for \mathcal{V} iff

- $\{\mathbf{b}_1, \mathbf{b}_2, \dots\}$ is a **basis** for \mathcal{V} , i.e.,
 - $\{\mathbf{b}_1, \mathbf{b}_2, \dots\}$ is **linearly independent**
 - $\text{span}(\{\mathbf{b}_1, \mathbf{b}_2, \dots\}) = \mathcal{V}$
- The basis vectors are **orthogonal**, i.e., $\langle \mathbf{b}_n, \mathbf{b}_m \rangle = 0$ for $n \neq m$.

The following theorem shows that the above definition *almost* has a redundancy.

(Read)

Theorem (orthogonality implies linear independence for nonzero vectors).

If $\{\mathbf{v}_1, \mathbf{v}_2, \dots\}$ are nonzero orthogonal vectors, then they are also linearly independent.

Proof (by contradiction). Suppose $\{\mathbf{v}_1, \mathbf{v}_2, \dots\}$ are orthogonal, nonzero, and also linearly dependent. Then there exists $N \in \mathbb{N}$ and $c_1, \dots, c_N \in \mathbb{F}$, not all equal to zero, such that $\mathbf{0} = \sum_{n=1}^N \mathbf{v}_n c_n$. Pick any $m \in \{1, \dots, N\}$ and we have $0 = \mathbf{v}'_m \mathbf{0} = \mathbf{v}'_m \sum_{n=1}^N \mathbf{v}_n c_n = \mathbf{v}'_m \mathbf{v}_m c_m = \|\mathbf{v}_m\|_2^2 c_m \implies c_m = 0$ because \mathbf{v}_m is nonzero. This holds for every m , contradicting the assumption that not all c_n are zero. \square

Thus the linear independence condition in the definition above is implied by the orthogonality condition, at least for nonzero vectors. So an equivalent definition of an orthogonal basis of \mathcal{V} is a set of nonzero vectors $\{\mathbf{b}_1, \mathbf{b}_2, \dots\}$ that are orthogonal and that span \mathcal{V} .

The above definition is general enough to accommodate infinite-dimensional vector spaces.
In finite dimensions the situation is simpler:

The N columns of any orthogonal matrix $\mathbf{V} \in \mathbb{R}^{N \times N}$ are an **orthonormal basis** for \mathbb{R}^N .

The N columns of any unitary matrix $\mathbf{V} \in \mathbb{C}^{N \times N}$ are an **orthonormal basis** for \mathbb{C}^N .

Proof.

- The orthogonality condition ensures linear independence by the above theorem.
- For any vector $\mathbf{x} \in \mathbb{F}^N$ we have $\mathbf{x} = \mathbf{I}\mathbf{x} = (\mathbf{V}\mathbf{V}')\mathbf{x} = \underbrace{\mathbf{V}}_{\text{basis}} \underbrace{(\mathbf{V}'\mathbf{x})}_{\text{coef.}} \in \text{span}(\mathbf{V}) \implies \text{span}(\mathbf{V}) = \mathbb{F}^N$.

Finding coordinates in an orthogonal basis

For $\mathbf{x} \in \mathbb{F}^N$, its elements are coordinates in the **standard basis** aka **canonical basis**:

$$\mathbf{x} = \mathbf{I}\mathbf{x} = \sum_{n=1}^N e_n x_n = \underbrace{\left[\begin{matrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{matrix} \right], \left[\begin{matrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{matrix} \right], \dots, \left[\begin{matrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{matrix} \right], \left[\begin{matrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{matrix} \right]}_{\text{standard basis}}, \underbrace{\left[\begin{matrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{matrix} \right]}_{\hookrightarrow \text{coordinates}}.$$

This is the most trivial orthogonal basis.

Now suppose $\mathbf{Q} \in \mathbb{F}^{N \times N}$ is an **orthogonal matrix** (or **unitary matrix**) and hence a basis for \mathbb{F}^N .

By definition of an **orthogonal matrix** (or **unitary matrix**): $\mathbf{Q}\mathbf{Q}' = \mathbf{I}$, hence

$$\begin{aligned} \mathbf{x} = \mathbf{I}\mathbf{x} &= (\mathbf{Q}\mathbf{Q}')\mathbf{x} = \underbrace{\mathbf{Q}}_{\substack{\text{basis coord.}}} \underbrace{(\mathbf{Q}'\mathbf{x})}_{\substack{\text{basis} \\ \hookrightarrow \text{coordinates}}} = \underbrace{\mathbf{Q}}_{\substack{\text{basis}}} \underbrace{\boldsymbol{\alpha}}_{\substack{\hookrightarrow \text{coordinates}}}, \text{ where } \boldsymbol{\alpha} = \mathbf{Q}'\mathbf{x} \in \mathbb{F}^N \\ &= \sum_{n=1}^N \underbrace{\mathbf{q}_n}_{\substack{\text{basis} \\ \hookrightarrow \text{coordinate!} \\ \text{vector}}} \underbrace{\alpha_n}_{\substack{\hookrightarrow \text{coordinate!}}}, \quad \text{where } \mathbf{Q} = [\mathbf{q}_1 \dots \mathbf{q}_N]. \end{aligned}$$

Alternate perspective: to write \mathbf{x} in the basis \mathbf{Q} , we want $\mathbf{x} = \mathbf{Q}\boldsymbol{\alpha}$, so we need the coordinate vector to be $\boldsymbol{\alpha} = \mathbf{Q}^{-1}\mathbf{x} = \mathbf{Q}'\mathbf{x}$.

It is important to appreciate the convenience of an orthogonal basis for finding coefficients (coordinates).

- For a basis in general we need $\boldsymbol{\alpha} = \mathbf{Q}^{-1}\mathbf{x}$, requiring matrix inversion that needs $O(N^3)$ computation.
- For an orthogonal basis, we need $\boldsymbol{\alpha} = \mathbf{Q}'\mathbf{x}$, which is simply matrix multiplication that needs $O(N^2)$ computation in general. For some bases (like DFT, DCT, OWT) it can be just $O(N \log N)$.

How do we know that the inverse \mathbf{Q}^{-1} exists for a general basis? ??

Note that the definition of an **orthogonal basis** is in terms of a set of vectors, but often we collect those vectors into a matrix and call the matrix an orthogonal basis. One must be careful though about the terminology. If $\{b_1, \dots, b_N\}$ is an **orthogonal basis** for \mathbb{F}^N , the matrix $B = [b_1 \ \dots \ b_N]$ is not necessarily an **orthogonal matrix**. 

Example. The vectors $\left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 3 \end{bmatrix} \right\}$ form an orthogonal basis for \mathbb{R}^2 , but $\begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$ is not an orthogonal matrix.

3.5 Spotting eigenvectors

For some matrices, one can find eigenvectors “by inspection.” Recognizing these cases is a useful skill.

Example. Consider the symmetric **outer product**: $A = zz'$.

Observe that

$$Az = (zz')z = z(z'z) = \underbrace{(z'z)}_{\text{scalar}} z.$$

Thus z is an eigenvector of A with eigenvalue that is the norm squared of z : $\lambda = z'z = \|z\|_2^2$.

This symmetric matrix has rank 1 and it has one nonzero eigenvalue.

Example. Consider the $N \times N$ matrix with $N = 2n$ where

$$\mathbf{A} = \left[\begin{array}{ccc|cc} 1 & \dots & 1 & & \\ \vdots & & \vdots & 0 & \\ 1 & \dots & 1 & & \\ \hline & & & 9 & \dots & 9 \\ 0 & & & \vdots & & \vdots \\ & & & 9 & \dots & 9 \end{array} \right] = \left[\begin{array}{c} \mathbf{1}_n \\ \mathbf{0}_n \end{array} \right] \left[\begin{array}{c|c} \mathbf{1}'_n & \mathbf{0}'_n \end{array} \right] + \left[\begin{array}{c} \mathbf{0}_n \\ 3\mathbf{1}_n \end{array} \right] \left[\begin{array}{c|c} \mathbf{0}'_n & 3\mathbf{1}'_n \end{array} \right].$$

By inspection one can build on the previous example to see that two of the eigenvectors are:

$$\mathbf{u}_1 = \begin{bmatrix} \frac{1}{\sqrt{n}} \mathbf{1}_n \\ \mathbf{0}_n \end{bmatrix}, \quad \mathbf{u}_2 = \begin{bmatrix} \mathbf{0}_n \\ \frac{1}{\sqrt{n}} \mathbf{1}_n \end{bmatrix}.$$

What is the corresponding eigenvalue λ_1 ?

A: 1

B: \sqrt{n}

C: n

D: $1/n$

E: $1/\sqrt{n}$

??

The second non-zero eigenvalue is 9 times larger. All the other eigenvalues are zero.

This symmetric matrix has rank 2 and it has two nonzero eigenvalues.

As noted previously, in general for symmetric matrices, rank = number of (possibly non-distinct) nonzero eigenvalues.

Example. What about asymmetric matrices? $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \mathbf{e}_1 \mathbf{e}_2'$ has $\lambda_1 = \lambda_2 = 0$ but $r = 1$.

SVD by inspection

For some matrices, one can find an SVD by inspection.

Example. Consider the $M \times N$ rank-1 outer-product matrix $\mathbf{A} = \mathbf{b}\mathbf{c}'$ with $\mathbf{b} \neq \mathbf{0}_M$ and $\mathbf{c} \neq \mathbf{0}_N$. Clearly:

$$\mathbf{A} = \mathbf{b}\mathbf{c}' = \underbrace{\begin{array}{ccc} \frac{\mathbf{b}}{\|\mathbf{b}\|} & \underbrace{\|\mathbf{b}\| \|\mathbf{c}\|}_{\sigma_1} & \underbrace{\left(\frac{\mathbf{c}}{\|\mathbf{c}\|}\right)'}_{\mathbf{v}'_1} \\ \mathbf{u}_1 & 1 \times 1 & \mathbf{v}'_1 \\ M \times 1 & 1 \times 1 & 1 \times N \end{array}}_{\text{compact SVD}} = \underbrace{\begin{bmatrix} \frac{\mathbf{b}}{\|\mathbf{b}\|} & \mathbf{U}_0 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} \|\mathbf{b}\| \|\mathbf{c}\| & \mathbf{0}_{1 \times N-1} \\ \mathbf{0}_{M-1 \times 1} & \mathbf{0}_{M-1 \times N-1} \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} \frac{\mathbf{c}}{\|\mathbf{c}\|} & \mathbf{V}_0 \end{bmatrix}'}_{\mathbf{V}},$$

where \mathbf{U}_0 is a $M \times (M - 1)$ matrix with orthonormal columns that span the subspace $\text{span}^\perp(\mathbf{b})$ and \mathbf{V}_0 is a $N \times (N - 1)$ matrix with orthonormal columns that span the subspace $\text{span}^\perp(\mathbf{c})$.

Is it unique? No, because we could use $-\mathbf{b}$ and $-\mathbf{c}$, and there are many choices for \mathbf{U}_0 and \mathbf{V}_0 .

If \mathbf{A} is **tall** ($M > N$), then the economy SVD would be similar to the full SVD except \mathbf{U} involves only the first $N - 1$ columns of \mathbf{U}_0 and the inner matrix would be $N \times N$ and all zeros except for σ_1 in the upper left.

Matrix-vector products and the SVD

(Read)

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{U}\Sigma\mathbf{V}'\mathbf{x} \\ &= \mathbf{U}\Sigma(\underbrace{\mathbf{V}'\mathbf{x}}_{\hookrightarrow \text{coordinates of } \mathbf{x} \text{ in term of basis } \mathbf{V}}) \end{aligned}$$

Define the coordinates (coefficients) to be $\boldsymbol{\alpha} = \mathbf{V}'\mathbf{x}$. Then expanding the matrix product:

$$\mathbf{A}\mathbf{x} = \sum_{k=1}^r \underbrace{\sigma_k}_{\text{gain coor.}} \underbrace{\alpha_k}_{\hookrightarrow \text{left singular vector (basis vector)}} \underbrace{\mathbf{u}_k}_{\cdot}.$$

In particular, if $\mathbf{A} \in \mathbb{F}^{M \times N}$ and $\mathbf{x} = \mathbf{v}_n$, the n th right singular vector, for some $n \in \{1, \dots, N\}$, then

$$\boldsymbol{\alpha} = \mathbf{V}'\mathbf{x} = \mathbf{e}_n \implies \mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{v}_n = \sigma_n \mathbf{u}_n.$$

3.6 Summary

This chapter has a lot of general concepts in it (subspace, basis, dimension, null space, rank). All of the definitions lead to the key result: “four fundamental subspaces” portion that relates the null space and range of a matrix (and the orthogonal complements thereof) to components of its SVD like V_0 and U_r .

A concept related to **rank** that arises in compressed sensing theory is the **spark** of a matrix.



For a futuristic (?) demonstration of the importance of subspaces, see:

<https://www.youtube.com/watch?v=H4qkodI6rSM>

Bibliography

- [1] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005.
- [2] Y. M. Lu and M. N. Do. “Sampling signals from a union of subspaces [A new perspective for the extension of this theory]”. In: *IEEE Sig. Proc. Mag.* 25.2 (Mar. 2008), 41–7.
- [3] T. Blumensath. “Sampling and reconstructing signals from a union of linear subspaces”. In: *IEEE Trans. Info. Theory* 57.7 (July 2011), 4660–71.
- [4] R. Vidal. “Subspace clustering”. In: *IEEE Sig. Proc. Mag.* 28.2 (Mar. 2011), 52–68.
- [5] G. Strang. “Bringing the SVD to life”. In: *SIAM News* 39.1 (Jan. 2006).

Chapter 4

Linear equations and least-squares

Contents (final version)

4.0 Introduction to linear equations	4.2
Linear regression and machine learning	4.4
4.1 Linear least-squares estimation	4.6
Solving LLS using the normal equations	4.10
Solving LLS problems using the compact SVD	4.12
Uniqueness of LLS solution	4.17
Moore-Penrose pseudoinverse	4.19
4.2 Linear least-squares estimation: Under-determined case	4.26
Orthogonality principle	4.28
Minimum-norm LS solution via pseudo-inverse	4.31
4.3 Truncated SVD solution	4.35
Low-rank approximation interpretation of truncated SVD solution	4.38
Noise effects	4.39
Tikhonov regularization aka ridge regression	4.41
4.4 Frames and tight frames	4.43

4.5 Projection and orthogonal projection	4.49
Binary classifier design using least-squares	4.61
4.6 Summary of LLS solution methods in terms of SVD	4.62

4.0 Introduction to linear equations

Source material for this chapter includes [1, §6.1, 8.1–8.4, 4.1, 5.2].

L§6.1

Solving a **system of linear equations** arises in numerous applications. Mathematically, a system of M equations in N unknowns is usually written

$$\mathbf{A}\mathbf{x} = \mathbf{y}, \quad \mathbf{A} \in \mathbb{F}^{M \times N}, \quad \mathbf{x} \in \mathbb{F}^N, \quad \mathbf{y} \in \mathbb{F}^M. \quad (4.1)$$

Typically \mathbf{A} is known from some type of (linear) modeling, \mathbf{y} corresponds to some type of measurements, and the goal is to solve for \mathbf{x} .

However, despite the equals sign in the classic expression (4.1), in practice often there does not *exist* any \mathbf{x} that yields equality, particularly when \mathbf{A} is tall, so the notation “ $\mathbf{A}\mathbf{x} \approx \mathbf{y}$ ” would be more realistic. Some other times there is not a *unique* \mathbf{x} that satisfies (4.1), in particular whenever \mathbf{A} is wide. Linear algebra texts focus on examining when a solution \mathbf{x} exists and is unique for (4.1).

Solving $\mathbf{A}\mathbf{x} = \mathbf{y}$

(Read)

[1, Theorem 6.1] reviews the principal existence and uniqueness results for (4.1).

1. There *exists* a solution \mathbf{x} iff $\mathbf{y} \in \mathcal{R}(\mathbf{A})$.
2. There *exists* a solution \mathbf{x} for all $\mathbf{y} \in \mathbb{R}^M$ iff $\mathcal{R}(\mathbf{A}) = \mathbb{F}^M$.
3. A solution \mathbf{x} is *unique* iff $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$.
4. There exists a unique solution for all $\mathbf{y} \in \mathbb{R}^M$ iff \mathbf{A} is $M \times M$ and non-singular (invertible),
i.e., none of the eigenvalues or singular values of \mathbf{A} are zero.
5. There is at most one solution for all $\mathbf{y} \in \mathbb{R}^M$ iff \mathbf{A} has linearly independent columns,
i.e., $\mathcal{N}(\mathbf{A}) = \emptyset$, and this is possible only if $M \geq N$.
6. The homogeneous system $\mathbf{A}\mathbf{x} = \mathbf{0}$ has a nontrivial (*i.e.*, nonzero) solution iff $\text{rank}(\mathbf{A}) < N$.

(You should read and verify these points.)

To understand #3, suppose $\mathbf{x}_0 \in \mathcal{N}(\mathbf{A})$ and $\mathbf{x}_0 \neq \mathbf{0}$ and suppose we have a solution $\mathbf{A}\mathbf{x}_1 = \mathbf{y}$. Then $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{x}_0$ is also a solution because $\mathbf{A}\mathbf{x}_2 = \mathbf{A}\mathbf{x}_1 + \mathbf{A}\mathbf{x}_0 = \mathbf{y} + \mathbf{0}$.

When \mathbf{A} is wide ($M < N$), $\mathcal{N}(\mathbf{A})$ is nonempty and there are (infinitely) many solutions to (4.1). One must design some way to choose among all those solutions.

When \mathbf{A} is square ($M = N$) and full rank (and hence invertible), then there is a unique solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$. We say that “the number of equations” equals “the number of unknowns.”

Linear regression and machine learning

In **linear regression**, we are given a set of training data consisting of M input (feature) vectors $\mathbf{a}_1, \dots, \mathbf{a}_M \in \mathbb{F}^N$ and corresponding responses $y_1, \dots, y_M \in \mathbb{F}$ and we want to find coefficients / weights $\mathbf{x} \in \mathbb{F}^N$ such that $\mathbf{a}_m^T \mathbf{x} \approx y_m$. Stacking up the input vectors into a matrix \mathbf{A} and the response variables into a vector \mathbf{y} yields the following matrix-vector form:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} \approx \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_M^T \end{bmatrix} \mathbf{x}, \quad \text{i.e.,} \quad \mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon} \in \mathbb{F}^M$ denotes noise or other errors.

We want to determine \mathbf{x} so that given some future feature vector \mathbf{a} , we can **predict** the corresponding (unknown) response by simply computing $\mathbf{a}^T \mathbf{x}$.

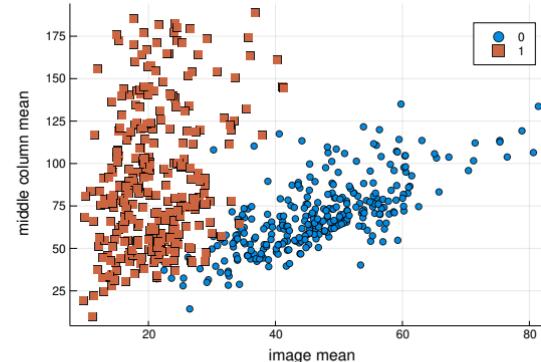
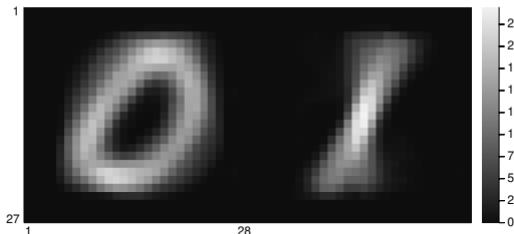
This is a classical problem in statistics and it is key “machine learning” method that one usually should consider (for estimation/regression problems) before considering more complicated methods. In statistics the common notation is $\mathbf{y} \approx \mathbf{X}\boldsymbol{\beta}$ whereas in linear algebra the common notation is $\mathbf{y} \approx \mathbf{A}\mathbf{x}$.

The variables here have many different names:

- y : response, labels, regressand, endogenous variable, measured variable, criterion variable, dependent variable, predicted variable, ...
- a (rows of A) : features, regressors, exogenous variables, explanatory variables, covariates, input variables, predictor variables, independent variables, ...
- x : parameter vector, regression coefficients, unknown, ...
- ϵ : noise, disturbance, error, ...

Example. Predict child's height from parent's height. [\[wiki\]](#)

Example. Hand-written digit classification using hand-crafted features.



4.1 Linear least-squares estimation

L§8.1

In a typical situation where $M > N$, one can show that $\mathcal{R}(\mathbf{A}) \neq \mathbb{F}^M$. So there will be (infinitely) many \mathbf{y} (those not in $\mathcal{R}(\mathbf{A})$) for which no solution \mathbf{x} exists.

Thus, when $M > N$, instead of insisting on *exactly* solving $\mathbf{Ax} = \mathbf{y}$, usually we look for *approximate* solutions where $\mathbf{Ax} \approx \mathbf{y}$. To do this, we must quantify “approximately equal.”

The most important and common approximate solution is to use **linear least-squares (LLS)** estimation or fitting, where we find an estimate $\hat{\mathbf{x}}$ that “best fits” the data \mathbf{y} using a Euclidean norm distance as follows:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathbb{F}^N} \|\mathbf{Ax} - \mathbf{y}\|_2^2. \quad (4.2)$$

- $\hat{\mathbf{x}}$ is called the **linear least-squares estimate** (or solution)
(The “hat” or caret above \mathbf{x} is often used to denote an estimate.)
- $\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}$ is called the **residual**
- $\|\mathbf{Ax} - \mathbf{y}\|_2$ is the Euclidean norm of the residuals for a candidate solution \mathbf{x} and is a measure of the “error” of the fit. It is sometimes called the “goodness of fit” even though a larger value means a worse fit!
- $\arg \min_{\mathbf{x}}$ means that we seek the argument $\hat{\mathbf{x}}$ that minimizes the fitting error; the minimum value of the fitting error: $\min_{\mathbf{x} \in \mathbb{F}^N} \|\mathbf{Ax} - \mathbf{y}\|_2^2$ usually is of less interest.

In words we often say “we minimized the squared error” but we really mean “we found the solution \mathbf{x} that

minimized the squared error.”

Although this technique is ancient, it remains one of the most important linear algebra methods for signal processing and data analysis in general.

Example. Suppose we observe noisy samples of signal:

L§8.3

$$y_m = s(t_m) + \epsilon_m, \quad m = 1, \dots, M$$

and we believe that the signal is a cubic polynomial:

$$s(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3$$

with unknown coefficients. In matrix-vector form:

$$\mathbf{y} \approx \mathbf{A}\mathbf{x}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 1 & t_1 & t_1^2 & t_1^3 \\ 1 & t_2 & t_2^2 & t_2^3 \\ \vdots & & & \vdots \\ 1 & t_M & t_M^2 & t_M^3 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}.$$

A figure on p. 4.9 illustrates this problem. Here the “features” are the time points t_m raised to different powers, and we call it **polynomial regression**.

To find the coefficient vector \mathbf{x} , one option would be to take just $M = 4$ samples. As long as we pick 4 distinct t_m points then one can show (using the fact that monomials are linearly independent functions) that the 4×4 matrix \mathbf{A}_4 has **linearly independent** columns. Thus \mathbf{A}_4 is **invertible** and we could use $\hat{\mathbf{x}} = \mathbf{A}_4^{-1}\mathbf{y}$ as an estimate of the coefficients.

However, in the presence of noise in the data, this approach would give very noisy and unreliable estimates of the coefficients.

Instead, it is preferable to use all $M \gg 4$ samples and estimate $\hat{\mathbf{x}}$ using linear least-squares.

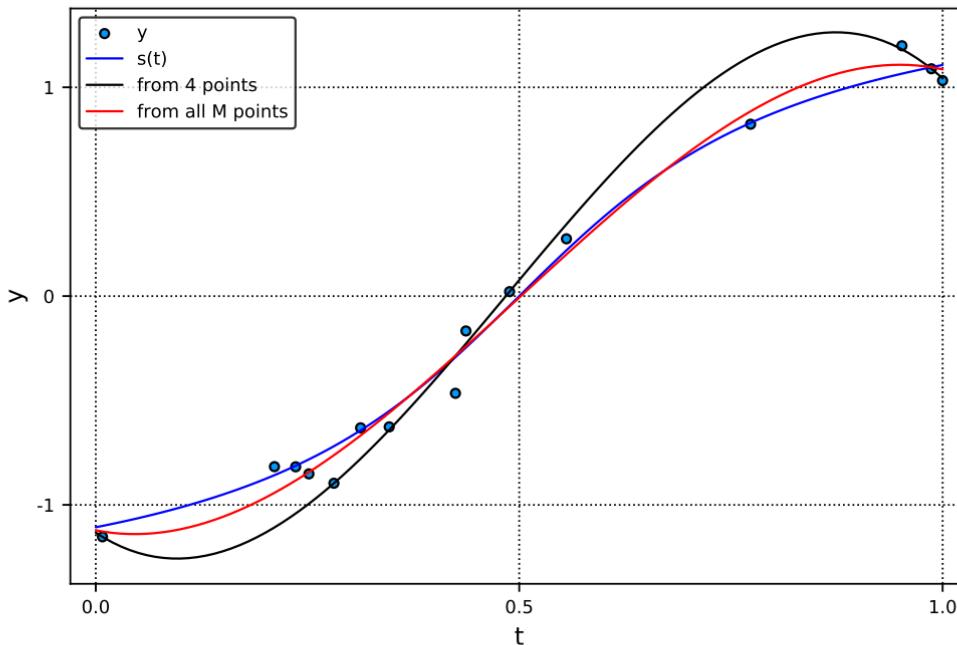
For higher polynomial orders, it is more stable to use **orthogonal polynomials** as the basis, instead of monomials. We use monomials here for simplicity in this example.

For a demo, see

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/04_ls_fit1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/04_ls_fit1.ipynb

The following curve suggests that fitting a cubic polynomial using all $M \gg 4$ points via (4.2) is better than using just 4 points with A_4^{-1} . (One can prove that statement statistically; see EECS 564.)



A key line in the demo code is $xh = A \setminus y$ and next we explain in detail the very important mathematical foundation behind that computation.

Solving LLS using the normal equations

Review of 1D minimization by example

(Read)

To find the minimizer of a function $f : \mathbb{R} \mapsto \mathbb{R}$ such as

$$f(x) = x(x - 1)^3,$$

you simply take the **derivative** and set it equal to zero:

$$0 = \dot{f}(x) = (x - 1)^2(4x - 1).$$

This case has two roots at $x = 1$ and $x = 1/4$. Because $f(1) = 0$ and $f(1/4) < 0$ it looks like $x = 1/4$ is the minimizer. To be sure we also take the second derivative:

$$\ddot{f}(x) = 6(2x^2 - 3x + 1) \implies \ddot{f}(1/4) = 9/4 > 0,$$

so $x = 1/4$ is at least a local minimizer. Checking $\pm\infty$ verifies. This example was extra work because f is non-convex.

Solving the LLS problem

L§8.2

Now consider a LLS cost function corresponding to (4.2):

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{y}\|_2^2, \quad (4.3)$$

where now $f : \mathbb{F}^N \mapsto \mathbb{R}$. One can show that this $f(\mathbf{x})$ is a **convex function**, so to find a minimizer of f it suffices to set the **gradient** to zero. The gradient of f w.r.t. \mathbf{x} , arranged as a column vector here, is:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f \\ \vdots \\ \frac{\partial}{\partial x_N} f \end{bmatrix} = \mathbf{A}'(\mathbf{Ax} - \mathbf{y}).$$

Setting this gradient to zero, i.e., $\nabla f(\mathbf{x})|_{\mathbf{x}=\hat{\mathbf{x}}} = \mathbf{0}$, and rearranging yields the **normal equations**:

$$\underbrace{\mathbf{A}'\mathbf{A}}_{N \times N \text{ Gram matrix}} \hat{\mathbf{x}} = \underbrace{\mathbf{A}'\mathbf{y}}_{N \times 1}. \quad (4.4)$$

Note that we started with a $M \times N$ matrix \mathbf{A} in (4.1) and (4.2), but the normal equations *always* have a square $N \times N$ **Gram matrix**. We could apply any classical method for solving this system of N equations in N unknowns, such as **Gaussian elimination**, but instead we next apply the SVD directly to (4.2).

Solving LLS problems using the compact SVD

L§8.4

The **compact SVD** provides an insightful way to analyze the LLS problem (4.2). It is also a reasonable way to solve LLS problems where M and N are not too large. (Large problems require iterative methods, discussed later.) Using a compact SVD $\mathbf{A} = \mathbf{U}_r \boldsymbol{\Sigma}_r \mathbf{V}_r'$ from (3.12), the LLS cost function (4.3) becomes:

$$\begin{aligned}
 \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 &= \|\mathbf{U}_r \boldsymbol{\Sigma}_r \mathbf{V}_r' \mathbf{x} - \mathbf{y}\|_2^2, \\
 &= \|\mathbf{U}_r \boldsymbol{\Sigma}_r \mathbf{V}_r' \mathbf{x}\|_2^2 - 2 \operatorname{real}\{(\mathbf{U}_r \boldsymbol{\Sigma}_r \mathbf{V}_r' \mathbf{x})' \mathbf{y}\} + \|\mathbf{y}\|_2^2, \\
 &= \|\boldsymbol{\Sigma}_r \mathbf{V}_r' \mathbf{x}\|_2^2 - 2 \operatorname{real}\{(\boldsymbol{\Sigma}_r \mathbf{V}_r' \mathbf{x})' (\mathbf{U}_r' \mathbf{y})\} + \|\mathbf{U}_r' \mathbf{y}\|_2^2 + \|\mathbf{y}\|_2^2 - \|\mathbf{U}_r' \mathbf{y}\|_2^2, \\
 &= \underbrace{\|\boldsymbol{\Sigma}_r \mathbf{V}_r' \mathbf{x} - \mathbf{U}_r' \mathbf{y}\|_2^2}_{\text{Term 1}} + \underbrace{\|\mathbf{y}\|_2^2 - \|\mathbf{U}_r' \mathbf{y}\|_2^2}_{\text{Term 2}},
 \end{aligned}$$

where we **completed the square** and used the fact that $\|\mathbf{U}_r \mathbf{z}_r\|_2 = \|\mathbf{z}_r\|_2$ because \mathbf{U} is unitary.

Term 2 is independent of \mathbf{x} , so to minimize the LLS cost function we want must minimize Term 1, *i.e.*,

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \|\boldsymbol{\Sigma}_r \mathbf{V}_r' \mathbf{x} - \mathbf{U}_r' \mathbf{y}\|_2^2. \quad (4.5)$$

If $r = 0$ (*i.e.*, if $\mathbf{A} = \mathbf{0}$) then Term 1 vanishes, so we focus on the usual case where $1 \leq r \leq N$ hereafter.

Although (4.5) initially might look more complicated than the original LLS problem, it is actually simpler because Σ_r is invertible and V_r has orthonormal columns. By inspection (without taking any gradients!) one possible solution to (4.5) is:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \|\Sigma_r V_r' \mathbf{x} - U_r' \mathbf{y}\|_2^2 = V_r \Sigma_r^{-1} U_r' \mathbf{y}, \quad (4.6)$$

because this solution makes Term 1 identically zero.

Verification: $\Sigma_r V_r' \hat{\mathbf{x}} = \Sigma_r \underbrace{V_r' (V_r \Sigma_r^{-1} U_r' \mathbf{y})}_I = \underbrace{\Sigma_r \Sigma_r^{-1}}_I U_r' \mathbf{y} = U_r' \mathbf{y}.$

Recall that Σ_r is $r \times r$ and contains the r nonzero singular values of A along its diagonal, so it is invertible.

If $r = N$, i.e., if A has full column rank, then $V_r = V$ and the solution (4.6) is the *unique* minimizer.

However, if $r < N$, then there are multiple minimizers. All minimizers are given by

$$\hat{\mathbf{x}} = V_r \Sigma_r^{-1} U_r' \mathbf{y} + V_0 z_0, \quad (4.7)$$

where $V = [V_r \ V_0]$ and z_0 is *any* vector of the appropriate length!

Any such solution makes Term 1 identically zero because:

$$\Sigma_r V_r' \hat{\mathbf{x}} = \Sigma_r V_r' (V_r \Sigma_r^{-1} U_r' \mathbf{y} + V_0 z_0) = U_r' \mathbf{y} + \Sigma_r \underbrace{V_r' V_0}_0 z_0 = U_r' \mathbf{y}.$$

When A is $M \times N$, what is the length of the vector z_0 in (4.7)?

A: r

B: $N - r$

C: $M - r$

D: N

E: M

??

Letting $\mathbf{x} = \mathbf{V}\mathbf{z} = [\mathbf{V}_r \quad \mathbf{V}_0] \begin{bmatrix} \mathbf{z}_r \\ \mathbf{z}_0 \end{bmatrix}$, so \mathbf{z} denotes the coordinates of \mathbf{x} in the \mathbf{V} coordinate system, another way of writing the general solution in (4.7) is

$$\hat{\mathbf{x}} = [\mathbf{V}_r \quad \mathbf{V}_0] \begin{bmatrix} \Sigma_r^{-1} \mathbf{U}'_r \mathbf{y} \\ z_0 \end{bmatrix} = \mathbf{V} \hat{\mathbf{z}}, \quad \hat{\mathbf{z}} = \begin{bmatrix} \Sigma_r^{-1} \mathbf{U}'_r \mathbf{y} \\ z_0 \end{bmatrix} = \begin{bmatrix} [\mathbf{U}'_r \mathbf{y}]_1 / \sigma_1 \\ \vdots \\ [\mathbf{U}'_r \mathbf{y}]_r / \sigma_r \\ \hline z_0 \end{bmatrix}. \quad (4.8)$$

This is a general expression for “the” LLS solution in terms of a compact SVD of \mathbf{A} and the data \mathbf{y} .

The arbitrary choice of z_0 may seem unsettling. We will address that concern shortly.

There are two ways that \mathbf{A} can have rank $r < N$, leading to a non-unique solution:

- If \mathbf{A} is square or tall but has linearly dependent columns.
- If \mathbf{A} is wide, because then $r \leq \min(M, N) = M < N$.

When \mathbf{A} is tall with full rank $r = N$, the (unique) LLS solution is $\hat{\mathbf{x}} = \mathbf{V}_r \Sigma_r^{-1} \mathbf{U}'_r \mathbf{y} = \mathbf{V} \Sigma_N^{-1} \mathbf{U}'_N \mathbf{y}$.

(Read)

Example. Return to the example of fitting a cubic polynomial on p. 4.7. In this case $M \gg N = 4$. As long as at least 4 of the $\{t_m\}$ values are distinct, the fact that monomials are linearly independent functions implies that \mathbf{A} has full rank, *i.e.*, $r = N = 4$. In this (typical) case, the SVD simplifies to

$$\underbrace{\mathbf{A}}_{M \times 4} = \underbrace{\mathbf{U}}_{M \times M} \underbrace{\Sigma}_{M \times 4} \underbrace{\mathbf{V}'}_{4 \times 4} = \underbrace{\mathbf{U}_4}_{M \times 4} \underbrace{\Sigma_4}_{4 \times 4} \underbrace{\mathbf{V}'_4}_{4 \times 4}, \quad \Sigma = \begin{bmatrix} \Sigma_4 \\ \mathbf{0}_{(M-4) \times 4} \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \\ 0 & 0 & 0 & \sigma_4 \\ \hline & & & \mathbf{0}_{(M-4) \times 4} \end{bmatrix}.$$

The solution $\hat{\mathbf{x}}$ in (4.7) simplifies to

$$\hat{\mathbf{x}} = \mathbf{V} \Sigma_4^{-1} \mathbf{U}'_4 \mathbf{y}. \quad (4.9)$$

Because $r = N = 4$ here, this is a final and unique LLS solution; there is no arbitrary vector \mathbf{z}_0 to select.

Practical implementation

(Read)

Returning to the JULIA demo for this example, we implement this SVD-based solution using
 $\text{U}, \text{s}, \text{V} = \text{svd}(\text{A})$

Recall that for this **economy SVD** version:

- U is $M \times N$ instead of the usual $M \times M$, i.e., is $\text{U}_r = \text{U}_N$ in (3.12)
- s is a vector of the $(\sigma_1, \dots, \sigma_N)$ values, so $\Sigma_r = \Sigma_N = \text{Diagonal}(\text{s})$ because $r = N$ here.

Using this economy SVD, two mathematically equivalent forms of the LS solution are:

- $\text{xh} = \text{V} * \text{Diagonal}(1 ./ \text{s}) * (\text{U}' * \text{y})$ which looks like $\hat{\mathbf{x}} = \mathbf{V}\Sigma_4^{-1}\mathbf{U}'_4\mathbf{y}$ in (4.9)
- $\text{xh} = \text{V} * ((1 ./ \text{s}) .* (\text{U}' * \text{y}))$ which looks like (4.8).

Why the parentheses $(\text{U}' * \text{y})$? Saves computation because \mathbf{U}'_N is wide.

When \mathbf{A} is tall with full rank ($r = N$), the compact SVD (used in our mathematical analysis) and the economy SVD (returned by `svd`) are identical, i.e., $\mathbf{U}_r = \mathbf{U}_N$, $\Sigma_r = \Sigma_N$, and $\mathbf{V}_r = \mathbf{V}_N = \mathbf{V}$.

Uniqueness of LLS solution (reprise)

(Read)

If \mathbf{A} is $M \times N$, then by the definition of rank, $\text{rank}(\mathbf{A}) = N$ iff \mathbf{A} has linearly independent columns. Having $M \geq N$ is a necessary condition for linear independence of the columns of \mathbf{A} .

Conversely, if $M < N$ then $\text{rank}(\mathbf{A}) \neq N$ because $\text{rank}(\mathbf{A}) \leq \min(M, N) = M < N$.

Over-determined (tall) case

In the frequent case where $M > N$ we have (graphically):

$$\underbrace{\mathbf{A}}_{M \times N} \underbrace{\mathbf{x}}_{N \times 1} = \underbrace{\mathbf{y}}_{M \times 1}.$$

In words, there are more equations than unknowns, called an **over-determined system**. Rarely is there an exact solution in this case due to noise in the data \mathbf{y} , so “best fit” solutions like LLS (4.2) are used instead.

When $M \geq N$ and \mathbf{A} has rank $r = N$, the solution (4.7), (4.8) has no arbitrary terms and simplifies to:

$$\hat{z}_n = \underbrace{\frac{[\mathbf{U}'\mathbf{y}]_n}{\sigma_n}}_{n = 1, \dots, N} \implies \hat{\mathbf{z}} = \Sigma_N^{-1} \mathbf{U}'_N \mathbf{y}, \quad \hat{\mathbf{x}} = \mathbf{V} \hat{\mathbf{z}} \implies \hat{\mathbf{x}} = \mathbf{V} \Sigma_N^{-1} \mathbf{U}'_N \mathbf{y}. \quad (4.10)$$

This is the *unique* solution to (4.2) when $\text{rank}(\mathbf{A}) = N$.

Over-determined full-rank case using SVD

The expression (4.10) uses the **compact SVD** and it is useful to also write it in terms of the **full SVD**:

$$\hat{\mathbf{x}} = \mathbf{V} \Sigma_N^{-1} \mathbf{U}'_N \mathbf{y} = \mathbf{V} \left[\begin{array}{c|c} \Sigma_N^{-1} & \mathbf{0}_{N \times (M-N)} \end{array} \right] \begin{bmatrix} \mathbf{U}'_N \\ \mathbf{U}'_0 \end{bmatrix} \mathbf{y} = \mathbf{V} \left[\begin{array}{c|c} \Sigma_N^{-1} & \mathbf{0}_{N \times (M-N)} \end{array} \right] \mathbf{U}' \mathbf{y}.$$

This form is suboptimal for *computation* because the term $\mathbf{U}'_0 \mathbf{y}$ is computed but then multiplied by 0.

An even more concise expression is

$$M \geq N \text{ and } \text{rank}(\mathbf{A}) = N \implies \hat{\mathbf{x}} = \mathbf{V} \Sigma^+ \mathbf{U}' \mathbf{y}, \quad (4.11)$$

where $\Sigma^+ \triangleq \left[\begin{array}{c|c} \Sigma_N^{-1} & \mathbf{0}_{N \times (M-N)} \end{array} \right]$ denotes the **Moore-Penrose pseudoinverse** of $\Sigma = \begin{bmatrix} \Sigma_N \\ \mathbf{0}_{(M-N) \times N} \end{bmatrix}$.

The next page discusses this new term in detail.

Moore-Penrose pseudoinverse

L§4.1

Define. In general, for any matrix $\mathbf{A} \in \mathbb{F}^{M \times N}$, the **Moore-Penrose pseudoinverse** of \mathbf{A} , denoted $\mathbf{A}^+ \in \mathbb{F}^{N \times M}$, is a generalization of the usual notation of matrix inverse that satisfies the following four properties:

- $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$ (weaker than $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$)
- $\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+$
- $(\mathbf{A}^+\mathbf{A})' = \mathbf{A}^+\mathbf{A}$ (symmetry)
- $(\mathbf{A}\mathbf{A}^+)' = \mathbf{A}\mathbf{A}^+$

Properties. (For proofs see [\[wiki\]](#).)

(Another notation is \mathbf{A}^\dagger .) L§4.3

- \mathbf{A}^+ is unique and $(\mathbf{A}^+)^+ = \mathbf{A}$
- If \mathbf{A} is invertible, then $\mathbf{A}^+ = \mathbf{A}^{-1}$
- $\mathbf{A}^+ = (\mathbf{A}'\mathbf{A})^+\mathbf{A}'$
- If \mathbf{A} has full column rank (linearly independent columns), then $\mathbf{A}^+ = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}'$.
In this case, \mathbf{A}^+ is a **left inverse** because $\mathbf{A}^+\mathbf{A} = \mathbf{I}_N$.
- $\mathbf{A}^+ = \mathbf{A}'(\mathbf{A}\mathbf{A}')^+$
- If \mathbf{A} has full row rank, (linearly independent rows), then $\mathbf{A}^+ = \mathbf{A}'(\mathbf{A}\mathbf{A}')^{-1}$.
In this case, \mathbf{A}^+ is a **right inverse** because $\mathbf{A}\mathbf{A}^+ = \mathbf{I}_M$.
- $\mathbf{0}_{M \times N}^+ = \mathbf{0}_{N \times M}$ so in particular $\mathbf{0}^+ = \mathbf{0}$
- $(\mathbf{A}')^+ = (\mathbf{A}^+)'$

Pseudo-inverse and matrix products

Caution: in general $(AB)^+ \neq B^+A^+$, so be careful with matrix products. However, for $B \in \mathbb{F}^{K \times L}$:

- (P1) If Q is a $M \times K$ matrix with **orthonormal columns**, i.e., $Q'Q = I_{K \times K}$, then $(QB)^+ = B^+Q'$. 
 - $(QB)(QB)^+(QB) = QB(B^+Q')QB = QBB^+B = QB$
 - $(QB)^+QB(QB)^+ = (B^+Q')QB(B^+Q') = B^+BB^+Q' = B^+Q' = (QB)^+$, etc.
- (P2) If Q is a $L \times N$ matrix with **orthonormal rows**, i.e., $QQ' = I_{L \times L}$, then $(BQ)^+ = Q'B^+$.
 - $(BQ)(BQ)^+(BQ) = BQ(Q'B^+)BQ = BB^+BQ = BQ$
 - $(BQ)^+(BQ)(BQ)^+ = (Q'B^+)BQ(Q'B^+) = Q'B^+BB^+ = Q'B^+ = (BQ)^+$, etc.
- (P3) If $A \in \mathbb{F}^{M \times N}$ has full column rank (linearly independent columns)
and $B \in \mathbb{F}^{N \times K}$ has full row rank (linearly independent rows), then $(AB)^+ = B^+A^+$.

Fortunately, these product properties serve our needs.

A special case of the first two preceding results is when $B = I$, leading to:

- (P4) If Q is a matrix with **orthonormal columns**, then $Q^+ = Q'$.
- (P5) If Q is a matrix with **orthonormal rows**, then $Q^+ = Q$.

In the context of the **compact SVD**, (P4) implies that $U_r^+ = U_r'$ and $V_r^+ = V_r'$.

Because U_r has orthonormal columns and V_r' has orthonormal rows and Σ_r is invertible, it follows that:

$$A = U_r \Sigma_r V_r' \implies A^+ = (U_r \Sigma_r V_r')^+ = (U_r (\Sigma_r V_r'))^+ \stackrel{P1}{=} (\Sigma_r V_r')^+ U_r' \stackrel{P2}{=} V_r \Sigma_r^+ U_r' = V_r \Sigma_r^{-1} U_r'.$$

Example. A pseudo-inverse example of particular interest when working with the **SVD** is that of a **rectangular diagonal matrix**:

$$\Sigma = \underbrace{\left[\begin{array}{c|c} \Sigma_r & \mathbf{0}_{r \times (N-r)} \\ \hline \mathbf{0}_{(M-r) \times r} & \mathbf{0}_{(M-r) \times (N-r)} \end{array} \right]}_{M \times N} \implies \Sigma^+ = \underbrace{\left[\begin{array}{c|c} \Sigma_r^{-1} & \mathbf{0}_{r \times (M-r)} \\ \hline \mathbf{0}_{(N-r) \times r} & \mathbf{0}_{(N-r) \times (M-r)} \end{array} \right]}_{N \times M}.$$

Exercise. Verify that this Σ^+ satisfies the four conditions for a pseudo-inverse on p. 4.19.

An important special case is when $r = N$, i.e., \mathbf{A} has full column rank, in which case:

$$\Sigma = \underbrace{\left[\begin{array}{c} \Sigma_N \\ \hline \mathbf{0}_{(M-N) \times N} \end{array} \right]}_{M \times N} \implies \Sigma^+ = \underbrace{\left[\begin{array}{c|c} \Sigma_N^{-1} & \mathbf{0}_{N \times (M-N)} \end{array} \right]}_{N \times M}.$$

Noting that $\Sigma^+ \Sigma = \mathbf{I}_N$ in the tall full-rank case, it is trivial to verify the four defining properties for this special case.

Caution: in general $\mathbf{A}^+ \mathbf{A} \neq \mathbf{I}_N$ and $\mathbf{A} \mathbf{A}^+ \neq \mathbf{I}_M$.



Pseudo-inverse and SVD

Using the orthogonal matrix product properties of the pseudo-inverse yields the following **SVD** property:

$$\underbrace{\mathbf{A}}_{M \times N} = \underbrace{\mathbf{U}}_{M \times M} \underbrace{\Sigma}_{M \times N} \underbrace{\mathbf{V}'}_{N \times N} \implies \underbrace{\mathbf{A}^+}_{N \times M} = \underbrace{\mathbf{V}}_{N \times N} \underbrace{\Sigma^+}_{N \times M} \underbrace{\mathbf{U}'}_{M \times M}. \quad (4.12)$$

The **compact SVD** version of pseudo-inverse is also useful:

$$\mathbf{A}^+ = \mathbf{V} \Sigma^+ \mathbf{U}' = [\mathbf{V}_r \mid \mathbf{V}_0] \Sigma^+ \begin{bmatrix} \mathbf{U}'_r \\ \mathbf{U}'_0 \end{bmatrix} \implies \mathbf{A}^+ = \underbrace{\mathbf{V}_r}_{N \times r} \underbrace{\Sigma_r^{-1}}_{r \times r} \underbrace{\mathbf{U}'_r}_{r \times M}. \quad (4.13)$$

Here is one sanity check for the pseudo-inverse expression (4.12):

$$\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}'\mathbf{V}\Sigma^+\mathbf{U}'\mathbf{U}\Sigma\mathbf{V}' = \mathbf{U}\underbrace{\Sigma\Sigma^+\Sigma}_{\text{Identity}}\mathbf{V}' = \mathbf{U}\Sigma\mathbf{V}' = \mathbf{A}.$$

The key expression (4.12) simplifies to the usual matrix inverse in the rare case where \mathbf{A} is square and invertible: $\mathbf{A}^{-1} = \mathbf{V}\Sigma^{-1}\mathbf{U}'$. (However, for invertible matrices often we do not need to use the SVD.)

LLS solution $\hat{\mathbf{x}} = \mathbf{A}^+\mathbf{y}$ lies in which of the four fundamental spaces of \mathbf{A} ?

- A: $\mathcal{N}(\mathbf{A})$ B: $\mathcal{N}^\perp(\mathbf{A})$ C: $\mathcal{R}(\mathbf{A})$ D: $\mathcal{R}^\perp(\mathbf{A})$ E: None of these.

??

Warm-up questions

If A has linearly independent columns, then $\|A^+Ax\| = \|x\|$.

A: True

B: False

??

Let A have **full SVD** $A = U\Sigma V'$ and **compact SVD** $A = U_r\Sigma_r V'_r$, where we partition unitary matrix V as usual as $V = [V_r \quad V_0]$. Which of the following is the tuple $(\mathcal{N}(V'), \mathcal{N}(V'_r))$?

A: $(0, 0)$ B: $(0, \mathcal{R}(V_0))$ C: $(\mathcal{R}(V_0), 0)$ D: $(\mathcal{R}(V_0), \mathcal{R}(V_0))$ E: None of these.

??

If matrix A has **SVD** $A = U\Sigma V' = U_r\Sigma_r V'_r$, then two expressions for its **pseudo-inverse** are:

$$A^+ = V\Sigma^+U' = V_r\Sigma_r^{-1}U'_r.$$

Which of the following is the **null space** of A^+ , i.e., $\mathcal{N}(A^+)$?

A: $\mathcal{N}(A)$ B: $\mathcal{N}^\perp(A)$ C: $\mathcal{R}(A)$ D: $\mathcal{R}^\perp(A)$ E: None of these.

??

Projector / idempotent preview

(Read)

The following matrix is called an **orthogonal projector** (see p. 4.49):

$$\mathbf{P}_{\mathcal{R}(\mathbf{A}') \perp} = \mathbf{P}_{\mathcal{N}^\perp(\mathbf{A})} \triangleq \mathbf{A}^+ \mathbf{A} = \mathbf{V} \Sigma^+ \mathbf{U}' \mathbf{U} \Sigma \mathbf{V}' = \mathbf{V} \Sigma^+ \Sigma \mathbf{V}' = \mathbf{V}_r \mathbf{V}'_r.$$

Because $\mathbf{P}_{\mathcal{R}(\mathbf{A}') \perp} \mathbf{P}_{\mathcal{R}(\mathbf{A}') \perp} = \mathbf{P}_{\mathcal{R}(\mathbf{A}') \perp}$, it is called **idempotent**.

Likewise the following matrix is also a projector (and also idempotent):

$$\mathbf{P}_{\mathcal{R}(\mathbf{A})} = \mathbf{P}_{\mathcal{N}^\perp(\mathbf{A}') \perp} \triangleq \mathbf{A} \mathbf{A}^+ = \mathbf{U} \Sigma \mathbf{V}' \mathbf{V} \Sigma^+ \mathbf{U}' = \mathbf{U} \Sigma \Sigma^+ \mathbf{U}' = \mathbf{U}_r \mathbf{U}'_r.$$

Relating pseudo-inverse and normal equations

(Read)

The following equalities (properties of the pseudo-inverse) follow:

$$\begin{aligned}\mathbf{A}' \mathbf{A} \mathbf{A}^+ &= \mathbf{A}' \text{ because } \mathbf{A}' \mathbf{A} \mathbf{A}^+ = \mathbf{V}_r \Sigma_r \mathbf{U}'_r (\mathbf{U}_r \mathbf{U}'_r) = \mathbf{V}_r \Sigma_r \mathbf{U}'_r = \mathbf{A}' \\ \mathbf{A}^+ \mathbf{A} \mathbf{A}' &= \mathbf{A}' \text{ because } \mathbf{A}^+ \mathbf{A} \mathbf{A}' = (\mathbf{V}_r \mathbf{V}'_r) \mathbf{V}_r \Sigma_r \mathbf{U}'_r = \mathbf{V}_r \Sigma_r \mathbf{U}'_r = \mathbf{A}'.\end{aligned}$$

Multiplying the first of these two equalities by \mathbf{y} yields:

$$\mathbf{A}' \mathbf{A} \mathbf{A}^+ \mathbf{y} = \mathbf{A}' \mathbf{y} \implies \mathbf{A}' \mathbf{A} \hat{\mathbf{x}} = \mathbf{A}' \mathbf{y},$$

so the pseudo-inverse solution $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y}$ satisfies the **normal equations** always, regardless of the rank of \mathbf{A} .

LLS solution using pseudo-inverse

Using (4.12), we can rewrite the LLS estimate (4.11) particularly concisely in the full-rank case as follows:

$$\mathbf{A} \in \mathbb{F}^{M \times N} \text{ and } \underbrace{\text{rank}(\mathbf{A}) = N}_{\implies M \geq N} \implies \hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathbb{F}^N} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 = \mathbf{A}^+ \mathbf{y} = (\mathbf{A}' \mathbf{A})^{-1} \mathbf{A}' \mathbf{y}. \quad (4.14)$$

This is a wonderfully elegant form on paper, but it is not the best computational approach!

We could implement (4.14) in JULIA using the pseudo-inverse function: `xh = pinv(A) * y;`

or (in the full-rank case) we could use `xh = inv(A' * A) * (A' * y);`

However, these approaches would be computationally inefficient!

We rarely use `pinv` or even `inv` in practice (at least for large problem sizes) because `xh = A \ y` uses a more efficient **QR decomposition**. Nevertheless, the SVD is most helpful conceptually.

An exception is applications where we must solve LLS problems for many \mathbf{y} with the same \mathbf{A} (as in HW).

Practical question: which of the following is likely to require less computation.

- A: `xh = inv(A' * A) * (A' * y)`
- B: `xh = inv(A' * A) * A' * y`
- C: Both always use the same computation.

??

4.2 Linear least-squares estimation: Under-determined case

We focused previously on cases where $M \geq N$ and \mathbf{A} has full rank, and derived the concise LLS solution based on the pseudo-inverse (4.14). Now we examine cases where $M < N$, called **under-determined**, as well as cases where \mathbf{A} is tall but rank deficient *i.e.*, $\text{rank}(\mathbf{A}) < N$. In these cases there is not a unique minimizer to the LLS cost function $\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$. Using the **compact SVD** of \mathbf{A} , we showed in (4.7) that any minimizer has the form

$$\hat{\mathbf{x}} = \underbrace{\mathbf{V}_r \Sigma_r^{-1} \mathbf{U}_r' \mathbf{y}}_{\hat{\mathbf{x}}_{\mathcal{R}} \in \mathcal{R}(\mathbf{V}_r) = \mathcal{N}^\perp(\mathbf{A})} + \underbrace{\mathbf{V}_0 \hat{\mathbf{z}}_0}_{\hat{\mathbf{x}}_{\mathcal{N}} \in \mathcal{N}(\mathbf{A})} = \mathbf{A}^+ \mathbf{y} + \hat{\mathbf{x}}_{\mathcal{N}}. \quad (4.15)$$

The choice of $\hat{\mathbf{z}}_0$ is arbitrary because the residual is invariant to the nullspace component:

$$\mathbf{A}\hat{\mathbf{x}} - \mathbf{y} = \mathbf{A}(\mathbf{V}_r \hat{\mathbf{z}}_r + \mathbf{V}_0 \hat{\mathbf{z}}_0) - \mathbf{y} = \mathbf{A}(\hat{\mathbf{x}}_{\mathcal{R}} + \hat{\mathbf{x}}_{\mathcal{N}}) - \mathbf{y} = \mathbf{A}\hat{\mathbf{x}}_{\mathcal{R}} + \mathbf{0} - \mathbf{y} = \mathbf{A}\hat{\mathbf{x}}_{\mathcal{R}} - \mathbf{y}.$$

In other words, for any $\hat{\mathbf{x}} = \hat{\mathbf{x}}_{\mathcal{R}} + \hat{\mathbf{x}}_{\mathcal{N}}$ where $\hat{\mathbf{x}}_{\mathcal{N}} \in \mathcal{N}(\mathbf{A})$, the error $\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}\|_2^2$ is identical.

When $r < N$, the LLS solution is *not* unique because we can choose $\hat{\mathbf{z}}_0$ arbitrarily, or equivalently we can choose any $\hat{\mathbf{x}}_{\mathcal{N}} \in \mathcal{N}(\mathbf{A})$. The LLS criterion by itself is insufficient to identify a unique best $\hat{\mathbf{x}}$ in under-determined (or rank-deficient) problems.

Conversely, in the full rank case where $\text{rank}(\mathbf{A}) = r = N$, the matrix \mathbf{V}_0 does not exist and $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$ so the LLS solution is unique.

If \mathbf{A} is wide, i.e., $M < N$, can we have $\text{rank}(\mathbf{A}) = N$?

No, because $\text{rank}(\mathbf{A}) \leq \min(M, N) = M < N$ per (3.9). So wide cases are always under-determined.

To summarize:

- When $M \geq N$ and \mathbf{A} has full rank N , the LLS solution is unique and is $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y}$.
- Otherwise (i.e., if $M < N$ or if \mathbf{A} has rank less than N), the LLS solution is not unique and any $\hat{\mathbf{x}}$ of the form $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y} + \hat{\mathbf{x}}_{\mathcal{N}}$ where $\hat{\mathbf{x}}_{\mathcal{N}} \in \mathcal{N}(\mathbf{A})$ is “equally good” from the point of view of the LLS cost function $\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$.

Dealing with non-uniqueness

To pin down a unique solution in the under-determined case, we must introduce some additional criterion to select on $\hat{\mathbf{x}}$ from the (infinitely) many candidates of the form $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y} + \hat{\mathbf{x}}_{\mathcal{N}}$.

A modern way to do this is to use a **sparsity** model and seek a sparse solution, e.g.:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \text{nonsparsity}(\mathbf{x}) \text{ s.t. } \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 \leq \epsilon.$$

But this is an advanced topic for later in the course!

Instead we focus for now on the “classical” approach of choosing the **minimum norm** solution, i.e., the LLS minimizer where $\|\hat{\mathbf{x}}\|_2^2$ is the smallest. This is a “double minimization” problem, because (conceptually) we first find all the minimizers of $\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$, and then among those minimizers we pick the one where $\|\mathbf{x}\|_2^2$ is the smallest.

But first we give a geometric interpretation of the LLS solution.

Orthogonality principle

To examine LLS geometrically, recall from the **normal equations** (4.4) that any LLS solution must satisfy

$$\mathbf{A}'\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}'\mathbf{y} \implies \mathbf{A}'(\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}) = \mathbf{0} \implies \mathbf{A}'\mathbf{r} = \mathbf{0} \implies \mathbf{z}'\mathbf{A}'\mathbf{r} = 0 \implies \mathbf{A}\mathbf{z} \perp \mathbf{r}, \forall \mathbf{z} \in \mathbb{F}^N,$$

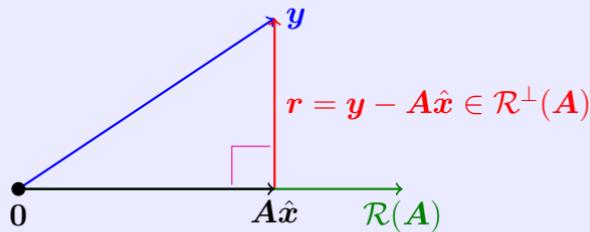
where the **residual** after fitting is denoted $\mathbf{r} = \mathbf{y} - \mathbf{A}\hat{\mathbf{x}}$.

(The normal equations are a **necessary condition** for $\hat{\mathbf{x}}$ to be a minimizer, regardless of the rank of \mathbf{A} .)

The following **orthogonality principle** of LLS estimation is a direct consequence of the above:

$$\langle \mathbf{A}\mathbf{z}, \mathbf{y} - \mathbf{A}\hat{\mathbf{x}} \rangle = (\mathbf{y} - \mathbf{A}\hat{\mathbf{x}})' \mathbf{A}\mathbf{z} = 0, \text{ i.e., } \mathbf{A}\mathbf{z} \perp (\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}), \forall \mathbf{z} \in \mathbb{F}^N. \quad (4.16)$$

In words, the **residual** is **perpendicular** to $\mathcal{R}(\mathbf{A})$. There is an important geometric interpretation:



Terminology that (unfortunately) is often used inter-changeably:

- $\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}$ is called the **residual** (from fitting)
- $\mathbf{y} - \mathbf{A}\mathbf{x}_{\text{true}}$ is called the **error** in the data
- $\hat{\mathbf{x}} - \mathbf{x}_{\text{true}}$ is called the **error** in the parameter estimate.

Often one must determine from context which meaning of “error” is meant.

If $\mathbf{y} \in \mathcal{R}(\mathbf{A})$ (which rarely happens for noisy data when $M > N$), then there is a solution with zero residual $\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}$. However, the parameter error $\hat{\mathbf{x}} - \mathbf{x}_{\text{true}}$ may still be nonzero!

Alternate proof of optimality

(Read)

Here is an alternate derivation that uses the orthogonality principle (4.16) to confirm that $\hat{\mathbf{x}}$ in (4.15) is optimal for LLS estimation (for \mathbf{A} of any size or rank).

Suppose $\mathbf{x}_? = \mathbf{A}^+ \mathbf{y} + \hat{\mathbf{x}}_{\mathcal{N}} + \mathbf{z} = \hat{\mathbf{x}} + \mathbf{z}$ for some arbitrary vector $\mathbf{z} \in \mathbb{F}^N$.

Could this $\mathbf{x}_?$ be a better estimator?

Using the norm of a sum in (1.14) and the orthogonality principle (4.16), the squared error criterion for such an $\mathbf{x}_?$ has the following lower bound:

$$\begin{aligned}\|\mathbf{A}\mathbf{x}_? - \mathbf{y}\|_2^2 &= \|\mathbf{A}(\hat{\mathbf{x}} + \mathbf{z}) - \mathbf{y}\|_2^2 = \|(\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}) + \mathbf{A}\mathbf{z}\|_2^2 \\ &= \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}\|_2^2 + \underbrace{2 \operatorname{real}\{(\mathbf{A}\hat{\mathbf{x}} - \mathbf{y})' \mathbf{A}\mathbf{z}\}}_0 + \underbrace{\|\mathbf{A}\mathbf{z}\|_2^2}_{\geq 0} \geq \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}\|_2^2,\end{aligned}$$

where the lower bound is achieved by $\mathbf{z} = \mathbf{0}$.

In words, the LLS fit is best when $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y} + \hat{\mathbf{x}}_{\mathcal{N}}$ for any vector $\hat{\mathbf{x}}_{\mathcal{N}} \in \mathcal{N}(\mathbf{A})$.

Mathematically, the set of LLS solutions is the sum of a vector and a subspace (the nullspace of \mathbf{A}):

$$\begin{aligned}\{\tilde{\mathbf{x}} \in \mathbb{F}^N : \|\mathbf{A}\tilde{\mathbf{x}} - \mathbf{y}\|_2^2 \leq \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2, \forall \mathbf{x} \in \mathbb{F}^N\} &= \boxed{\mathbf{A}^+ \mathbf{y} + \mathcal{N}(\mathbf{A})} \\ &\triangleq \{\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y} + \hat{\mathbf{x}}_{\mathcal{N}} : \hat{\mathbf{x}}_{\mathcal{N}} \in \mathcal{N}(\mathbf{A})\}.\end{aligned}$$

The sum of a vector plus a subspace is called a **linear variety** or a **flat**.

Minimum-norm LS solution via pseudo-inverse

L§8.1

We have seen that the set of optimal LLS estimates is $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y} + \mathcal{N}(\mathbf{A})$.

- If \mathbf{A} has full column rank, then $\mathcal{N}(\mathbf{A}) = \mathbf{0}$ and we have a unique solution $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y}$
- If \mathbf{A} does not have full column rank, then we want to pick one choice from the set of LLS estimates.

The classical way to pick one of the many possible estimates in the under-determined case is to choose the one with minimum norm.

Fact. The **minimum norm LLS solution** is:

$$\hat{\mathbf{x}} \triangleq \arg \min_{\mathbf{x} \in \{\mathbf{A}^+ \mathbf{y} + \mathcal{N}(\mathbf{A})\}} \|\mathbf{x}\|_2^2 = \mathbf{A}^+ \mathbf{y}. \quad (4.17)$$

This is a kind of “double minimization” because first we found a set of candidate solutions by finding minimizers of $\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$, and then we solve a different minimization problem involving $\|\mathbf{x}\|_2^2$ to select one final solution from that set of candidates.

Proof. If $\mathbf{x} = \mathbf{A}^+ \mathbf{y} + \hat{\mathbf{x}}_{\mathcal{N}}$ where $\hat{\mathbf{x}}_{\mathcal{N}} \in \mathcal{N}(\mathbf{A})$, then $\hat{\mathbf{x}}_{\mathcal{N}} \in \text{span}(\mathbf{V}_0)$, where $\mathbf{V} = [\mathbf{V}_r \mid \mathbf{V}_0]$. (Read)

Using (4.13): $\mathbf{A}^+ \mathbf{y} = \mathbf{V}_r \Sigma_r^{-1} \mathbf{U}'_r \mathbf{y} \in \mathcal{R}(\mathbf{V}_r)$.

Because \mathbf{V} is unitary, the columns of \mathbf{V}_r and \mathbf{V}_0 are orthogonal. Thus $\mathbf{A}^+ \mathbf{y} \perp \hat{\mathbf{x}}_{\mathcal{N}}$. Using (1.14):

$$\|\mathbf{x}\|_2^2 = \|\mathbf{A}^+ \mathbf{y} + \hat{\mathbf{x}}_{\mathcal{N}}\|_2^2 = \|\mathbf{A}^+ \mathbf{y}\|_2^2 + 2 \text{real}\{\hat{\mathbf{x}}'_{\mathcal{N}} \mathbf{A}^+ \mathbf{y}\} + \|\hat{\mathbf{x}}_{\mathcal{N}}\|_2^2 = \|\mathbf{A}^+ \mathbf{y}\|_2^2 + \|\hat{\mathbf{x}}_{\mathcal{N}}\|_2^2 \geq \|\mathbf{A}^+ \mathbf{y}\|_2^2,$$

where the minimum is achieved when $\hat{\mathbf{x}}_{\mathcal{N}} = \mathbf{0}$. Thus the minimum norm solution is $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y}$. □

The set over which we minimize in (4.17) is $\{\mathbf{A}^+\mathbf{y} + \mathcal{N}(\mathbf{A})\}$.

What is the cardinality of this set when $\text{rank}(\mathbf{A}) = N$?

A: 0

B: 1

C: r D: N E: ∞

??

What is the cardinality of this set when $M < N$?

A: 0

B: 1

C: r D: N E: ∞

??

In summary we have the following fortuitous situation.

- If \mathbf{A} has full column rank, then $\mathcal{N}(\mathbf{A}) = \mathbf{0}$ and we have a unique solution $\hat{\mathbf{x}} = \mathbf{A}^+\mathbf{y}$.
- If \mathbf{A} does not have full column rank, then there are multiple LLS estimates and the one with smallest Euclidean norm is $\hat{\mathbf{x}} = \mathbf{A}^+\mathbf{y}$.

Hence the **pseudo-inverse** solution has been very popular historically, and remains important today, except in many highly under-determined problems of the kind known as **compressed sensing**.

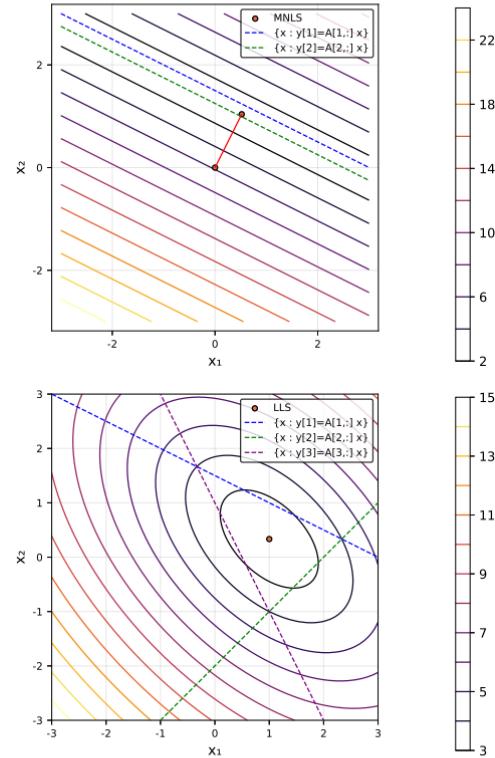
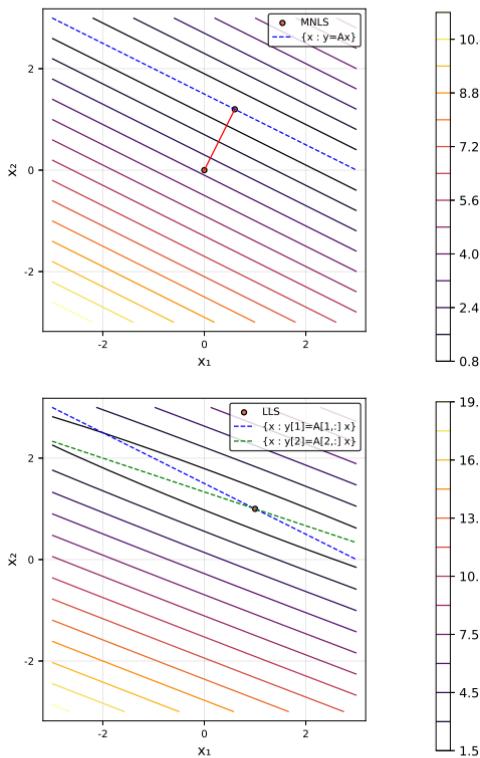
There are **quantum-computing methods** for the pseudo-inverse solution.

Example. To visualize the cost function (4.3) and solution $\hat{\mathbf{x}}$ see:

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/04_ls_cost1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/04_ls_cost1.ipynb

and plots on next page.



Interpreting the pseudo-inverse solution

(Read)

Using (4.12), we can interpret the pseudo-inverse solution

$$\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y} = \mathbf{V} \Sigma^+ \mathbf{U}' \mathbf{y}$$

as a cascade of three separate transformations:

$$\mathbf{y} \xrightarrow{\mathbf{U}'} \tilde{\mathbf{y}} \xrightarrow{\Sigma^+} \mathbf{z} \xrightarrow{\mathbf{V}} \hat{\mathbf{x}}.$$

The geometric interpretation is similar to what we did for the SVD except “in reverse” and with Σ^+ .

To interpret the residual, note that

$$\begin{aligned} \mathbf{r} &= \mathbf{y} - \mathbf{A}\hat{\mathbf{x}} = \mathbf{y} - \mathbf{A}\mathbf{A}^+\mathbf{y} = (\mathbf{I} - \mathbf{A}\mathbf{A}^+)\mathbf{y} = (\mathbf{I} - \mathbf{U}\Sigma\Sigma^+\mathbf{U}')\mathbf{y} = \mathbf{U}(\mathbf{I} - \Sigma\Sigma^+)\mathbf{U}'\mathbf{y} \\ &= [\mathbf{U}_r \mid \mathbf{U}_0] \left(\mathbf{I} - \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \right) [\mathbf{U}_r \mid \mathbf{U}_0]' \mathbf{y} = [\mathbf{U}_r \mid \mathbf{U}_0] \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} [\mathbf{U}_r \mid \mathbf{U}_0]' \mathbf{y} = \mathbf{U}_0 \mathbf{U}_0' \mathbf{y} \\ &\implies \|\mathbf{r}\|_2^2 = \|\mathbf{U}_0 \mathbf{U}_0' \mathbf{y}\|_2^2 = \mathbf{y}' \mathbf{U}_0 \mathbf{U}_0' (\mathbf{U}_0 \mathbf{U}_0' \mathbf{y}) = \mathbf{y}' \mathbf{U}_0 \mathbf{U}_0' \mathbf{y} = \|\mathbf{U}_0' \mathbf{y}\|_2^2, \end{aligned}$$

where from p. 3.33, $\mathcal{R}^\perp(\mathbf{A}) = \text{span}(\mathbf{U}_0)$. Thus the residual norm squared comes from the portion of \mathbf{y} in $\mathcal{R}^\perp(\mathbf{A})$, consistent with our earlier picture about the orthogonality principle.

4.3 Truncated SVD solution

We have seen that the minimum-norm LLS minimizer of $\|Ax - y\|_2^2$ is the pseudo-inverse solution:

$$\hat{x} = A^+y = V\Sigma^+U'y = \sum_{k=1}^r \frac{1}{\sigma_k} v_k(u'_k y).$$

This solution is mathematically elegant, but in practice sometimes it works very poorly.
This section provides one remedy based on the **truncated SVD**.

Example. Consider an application where $r = \text{rank}(A) = 2$ with singular values $\sigma_1 = 1$, $\sigma_2 = 10^{-8}$.
Then the pseudo-inverse solution here is:

$$\hat{x} = A^+y = \sum_{k=1}^2 \frac{1}{\sigma_k} v_k(u'_k y) = \frac{1}{\sigma_1} v_1(u'_1 y) + \frac{1}{\sigma_2} v_2(u'_2 y) = (1) v_1(u'_1 y) + (10^8) v_2(u'_2 y).$$

The pseudo-inverse solution “blows up” when any of the σ_k values are “too small” relative to the others, *i.e.*, if σ_k/σ_1 is near the **floating-point precision** limits.

Condition number

Such problems are called **poorly conditioned** because the \mathbf{A} has an undesirably large **condition number**, defined as

$$\kappa(\mathbf{A}) \triangleq \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})} = \frac{\sigma_1}{\sigma_{\min(N,M)}}. \quad (4.18)$$

Caution: an alternate definition of condition number is σ_1/σ_r , e.g., [2, p. 69], but we will use (4.18). 

This problem motivates the **truncated SVD** solution where we discard any singular values that are “too small” and write:

$$\hat{\mathbf{x}}_K = \sum_{k=1}^K \frac{1}{\sigma_k} \mathbf{v}_k (\mathbf{u}'_k \mathbf{y}), \quad (4.19)$$

where we choose fewer terms, $K < r$, such that $\sigma_K > \delta > 0$ for some tolerance δ .

Practical implementation of truncated SVD solution

(Read)

The tolerance δ may depend on factors such as the noise level in the data, the size of A , and whether one is using half, single, double or extended precision variables.

In JULIA, these variable types are `Float16`, `Float32`, `Float64`, and `BigFloat`.

JULIA's `pinv` has an optional second argument for specifying the tolerance.

The default value (see HW) is `eps(real(float(one(eltype(M)))))*maximum(size(A))`

The backslash function in JULIA `xh = A \ y` does not have any tolerance parameter, so one must use `pinv` instead of backslash to control the tolerance for poorly conditioned problems.

Alternatively, one can use some other method to improve the condition number, such as **Tikhonov regularization**, also known as **ridge regression**, described later on p. 4.41.

Low-rank approximation interpretation of truncated SVD solution

L§8.2

One way to interpret the truncated SVD solution (4.19) is as follows.

- First we form a **low-rank approximation** \mathbf{A}_K of \mathbf{A} , defined as

$$\mathbf{A}_K = \mathbf{U}_K \Sigma_K \mathbf{V}'_K = \sum_{k=1}^K \frac{1}{\sigma_k} \mathbf{v}_k \mathbf{u}'_k,$$

with $K < r \leq \min(M, N)$. (See Ch. 6 for more details about such approximations.)

- Then we express the pseudo-inverse LLS solution using that approximation:

$$\hat{\mathbf{x}} = \mathbf{A}_K^+ \mathbf{y} = \sum_{k=1}^K \frac{1}{\sigma_k} \mathbf{v}_k (\mathbf{u}'_k \mathbf{y}),$$

which is the same expression as (4.19).

We visualize a low-rank approximation as follows.

$$\underbrace{\mathbf{A}}_{M \times N} \approx \underbrace{\mathbf{A}_K}_{M \times N} = \underbrace{\mathbf{U}_K}_{M \times K} \underbrace{\Sigma_K}_{K \times K} \underbrace{\mathbf{V}'_K}_{K \times N}$$

Noise effects

(Read)

There are two sources of perturbations in LLS problems that can degrade the estimate $\hat{\mathbf{x}} = \mathbf{A}^+ \mathbf{y}$.

- Additive noise: $\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\varepsilon}$
- Errors in the model \mathbf{A} .

The \mathbf{A} we use for estimation might differ from the “true model” \mathbf{A}_{true} . Define $\Delta\mathbf{A} \triangleq \mathbf{A}_{\text{true}} - \mathbf{A}$.

[2, Theorem 6.12, p. 69]. (See [3].) Let $\mathbf{A} \in \mathbb{F}^{M \times N}$ with $M \geq N$ and $\text{rank}(\mathbf{A}) = N$.

Let $\kappa \triangleq \sigma_1/\sigma_N = \|\mathbf{A}\|_2/\sigma_N$ denote the condition number of \mathbf{A} , using the matrix 2-norm defined in (2.5). Assume that the model perturbations are not “too large” as follows:

$$\eta \triangleq \frac{\|\Delta\mathbf{A}\|_2}{\sigma_N} = \kappa \epsilon_A < 1, \quad \epsilon_A \triangleq \frac{\|\Delta\mathbf{A}\|_2}{\|\mathbf{A}\|_2}.$$

If the perturbed matrix $\mathbf{A} + \Delta\mathbf{A}$ has full rank, then the solution perturbation $\delta\hat{\mathbf{x}} = \hat{\mathbf{x}} - \mathbf{x}_{\text{true}}$ has the following bound in terms of the residual $\mathbf{r} = \mathbf{A}\hat{\mathbf{x}} - \mathbf{y}$:

$$\|\delta\hat{\mathbf{x}}\|_2 \leq \frac{\kappa}{1 - \eta} \left(\epsilon_A \|\mathbf{x}\|_2 + \frac{\|\boldsymbol{\varepsilon}\|_2}{\|\mathbf{A}\|_2} + \epsilon_A \kappa \frac{\|\mathbf{r}\|_2}{\|\mathbf{A}\|_2} \right).$$

See also [3–7].

-
- In the (full-rank) square case where $M = N$, the residual is $\mathbf{r} = \mathbf{0}$, so the solution error $\|\delta\hat{\mathbf{x}}\|_2$ depends on the condition number κ .
 - In the usual over-determined case where $M \geq N$ and $\mathbf{y} \notin \mathcal{R}(\mathbf{A})$, then the solution error is proportional to κ^2 so it is particularly important to try to keep κ small.

This bound is a motivation for using the truncated SVD where σ_1/σ_K is better (lower) than σ_1/σ_N .

Challenge. Find an error bound that depends on the truncated SVD where $\text{rank}(\mathbf{A}) = K$, instead of the full SVD where $\text{rank}(\mathbf{A}) = N$ was assumed above.

Tikhonov regularization aka ridge regression

A drawback of the **truncated SVD** solution to LLS problems is that it requires one to compute an SVD of \mathbf{A} , which can be impractical for large problems. An alternate approach to address ill-conditioned problems is to use **Tikhonov regularization**, also known as **ridge regression**.

We saw that the pseudo-inverse solution to LLS problems involves $1/\sigma_k$ terms that can “blow up” for small singular values, leading to very large values of $\hat{\mathbf{x}}$. Instead of directly modifying the singular values, **Tikhonov regularization** modifies the LS cost function to include a term that discourages the estimate from having excessively high values:

$$\hat{\mathbf{x}}_\beta = \arg \min_{\mathbf{x} \in \mathbb{R}^N} \underbrace{\|\mathbf{Ax} - \mathbf{y}\|_2^2}_{\text{data-fit term}} + \underbrace{\beta \|\mathbf{x}\|_2^2}_{\rightarrow \text{regularization (energy)}}, \quad (4.20)$$

where $\beta > 0$ is a **regularization parameter** that one must **tune** to trade-off between how well $\hat{\mathbf{x}}_\beta$ fits the data and how high is the energy of $\hat{\mathbf{x}}_\beta$.

By combining terms, we can rewrite the Tikhonov estimate (4.20) as:

$$\hat{\mathbf{x}}_\beta = \arg \min_{\mathbf{x} \in \mathbb{R}^N} \left\| \begin{bmatrix} \mathbf{A} \\ \sqrt{\beta} \mathbf{I} \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} \right\|_2^2 = \arg \min_{\mathbf{x} \in \mathbb{R}^N} \left\| \tilde{\mathbf{A}} \mathbf{x} - \tilde{\mathbf{y}} \right\|_2^2, \quad \tilde{\mathbf{A}} \triangleq \begin{bmatrix} \mathbf{A} \\ \sqrt{\beta} \mathbf{I} \end{bmatrix}, \quad \tilde{\mathbf{y}} \triangleq \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}.$$

If \mathbf{A} is $M \times N$, how many rows does $\tilde{\mathbf{A}}$ have?

A: M B: N C: $M + N$ D: $2M$ E: $2N$

??

What is the rank of $\tilde{\mathbf{A}}$?

A: r B: M C: N D: $M + N$

E: None of these.

??

In this simplified form, we know that the (unique!) LLS solution to (4.20) is

$$\hat{\mathbf{x}}_\beta = \tilde{\mathbf{A}}^+ \tilde{\mathbf{y}} = (\tilde{\mathbf{A}}' \tilde{\mathbf{A}})^{-1} \tilde{\mathbf{A}}' \tilde{\mathbf{y}} = (\mathbf{A}' \mathbf{A} + \beta \mathbf{I})^{-1} \mathbf{A}' \mathbf{y}.$$

If N is large, this solution requires inverting a $N \times N$ matrix (actually, solving a $N \times N$ system of equations) which is impractical. Instead we usually apply an optimization approach (such as a **conjugate gradient method**) directly to (4.20). Nevertheless, the closed-form expression for the solution is useful for analysis.

In particular it is insightful to examine the solution in terms of an **SVD** $\mathbf{A} = \mathbf{U} \Sigma \mathbf{V}'$. Here,

$$\mathbf{A}' \mathbf{A} + \beta \mathbf{I} = \mathbf{V} \Sigma' \mathbf{U}' \mathbf{U} \Sigma \mathbf{V}' + \beta \mathbf{I} = \mathbf{V} (\Sigma' \Sigma + \beta \mathbf{I}) \mathbf{V}',$$

so

$$\begin{aligned} \hat{\mathbf{x}}_\beta &= (\mathbf{A}' \mathbf{A} + \beta \mathbf{I})^{-1} \mathbf{A}' \mathbf{y} = \mathbf{V} (\Sigma' \Sigma + \beta \mathbf{I})^{-1} \mathbf{V}' \mathbf{V} \Sigma' \mathbf{U}' \mathbf{y} = \mathbf{V} (\Sigma' \Sigma + \beta \mathbf{I})^{-1} \Sigma' \mathbf{U}' \mathbf{y} \\ &= \mathbf{V}_r (\Sigma'_r \Sigma_r + \beta \mathbf{I})^{-1} \Sigma'_r \mathbf{U}'_r \mathbf{y} = \sum_{k=1}^r \mathbf{v}_k \underbrace{\left(\frac{\sigma_k}{\sigma_k^2 + \beta} \right)}_{\hookrightarrow \text{"shrinkage" or "Wiener filter" form}} (\mathbf{u}'_k \mathbf{y}). \end{aligned}$$

If $\beta \rightarrow 0$ then the ratio $\frac{\sigma_k}{\sigma_k^2 + \beta} \rightarrow \frac{1}{\sigma_k}$, unless $\sigma_k = 0$ (which never happens for $k = 1, \dots, r$).

So $\hat{\mathbf{x}}_\beta \rightarrow \mathbf{A}^+ \mathbf{y}$ as $\beta \rightarrow 0$.

4.4 Frames and tight frames

A LS minimization problem $\arg \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2$ is the easiest to solve when \mathbf{A} is unitary because in that case $\hat{\mathbf{x}} = \mathbf{A}'\mathbf{y}$. We now discuss a generalization of unitary matrices that leads to equally easy solutions.

For simplicity we focus on the case of a finite number of vectors in a finite-dimensional vector space. The concepts generalize to a countable collection of vectors in a general Hilbert space.

Define. A collection of M vectors $\{\phi_1, \dots, \phi_M\}$ in \mathbb{F}^N is called a **frame** in \mathbb{F}^N iff there exist real numbers $0 < \alpha \leq \beta < \infty$, called the **frame bounds**, such that

$$\alpha \|\mathbf{x}\|_2^2 \leq \sum_{m=1}^M |\langle \phi_m, \mathbf{x} \rangle|^2 \leq \beta \|\mathbf{x}\|_2^2, \quad \forall \mathbf{x} \in \mathbb{F}^N. \quad (4.21)$$

In other words, if we arrange those vectors into a $N \times M$ matrix $\Phi \triangleq [\phi_1 \ \dots \ \phi_M]$, then the collection of vectors is a **frame** iff there exist real numbers $0 < \alpha \leq \beta < \infty$ such that

$$\alpha \|\mathbf{x}\|_2^2 \leq \|\Phi' \mathbf{x}\|_2^2 \leq \beta \|\mathbf{x}\|_2^2, \quad \forall \mathbf{x} \in \mathbb{F}^N. \quad (4.22)$$

For brevity, we call such a Φ a **frame**.

The upper bound is important in infinite-dimensional inner product spaces, but is not very informative in \mathbb{F}^N .

If $\alpha > 0$, then what is β here in terms of the **singular values** of Φ ?

A: σ_1 B: σ_1^2 C: σ_r^2 D: σ_M

E: None of these

??

When is the upper frame bound β positive? **Always unless $\Phi = 0$.**

So in \mathbb{F}^N , the key to whether Φ is a frame or not depends on existence of $\alpha > 0$.

Properties of a frame

- Fact. $\alpha > 0$ in (4.22) iff Φ has full **row rank** (i.e., $\text{rank}(\Phi) = N$).
- Thus Φ must be wide (usually) or square, i.e., $M \geq N$.
- Thus $\alpha > 0 \implies \Phi = U\Sigma_N V'_N$ (**compact SVD** is same as **economy SVD**) where $\sigma_N > 0$, because $U_r = U_N = U$ and $V_r = V_N$.
- So $\Phi^+ = V_N \Sigma_N^{-1} U' = \Phi'(\Phi\Phi')^{-1}$ and $\Phi\Phi^+ = I_N$.

Now suppose we have a vector $x \in \mathbb{F}^N$ that we would like to express as a linear combination of the frame vectors $\{\phi_1, \dots, \phi_M\}$, i.e., we want to write

$$x = \Phi c$$

for some coefficient vector $c \in \mathbb{F}^M$. In the usual case where Φ is wide, there will be numerous possible coefficient vectors c for which $\|x - \Phi c\|_2 = 0$. The unique such c having minimum norm is given by the Moore-Penrose pseudo-inverse:

$$c = \Phi^+ x = \Phi'(\Phi\Phi')^{-1} x.$$

This is not too exciting yet because of the expensive matrix inverse. So we impose more conditions on Φ .

Tight frame

Define. A matrix Φ is a **tight frame** [8, 9] iff Φ is a **frame** with $\alpha = \beta$, i.e.,

$$\alpha \|\mathbf{x}\|_2^2 = \|\Phi' \mathbf{x}\|_2^2 = \sum_{m=1}^M |\langle \phi_m, \mathbf{x} \rangle|^2, \quad \forall \mathbf{x} \in \mathbb{F}^N. \quad (4.23)$$

If Φ is a **tight frame**, then

- $\Phi\Phi' = \alpha I_N$, so $\alpha = \sigma_1^2 = \dots = \sigma_N^2$, where $\{\sigma_k\}$ denotes the singular values of Φ ,
- its pseudo-inverse is simply $\Phi^+ = \frac{1}{\alpha} \Phi' = \frac{1}{\sigma_1^2} \Phi'$.

Proof. Using (4.23):

$$\alpha \|\mathbf{x}\|_2^2 = \|\Phi' \mathbf{x}\|_2^2 \implies \mathbf{x}'(\alpha I - \Phi\Phi')\mathbf{x} = 0, \quad \forall \mathbf{x} \in \mathbb{F}^N \implies \alpha I - \Phi\Phi' = \mathbf{0} \implies \alpha = \text{eig}\{\Phi\Phi'\} = \{\sigma_k^2\},$$

because the singular values are the square roots of those eigenvalues.

Normally, finding σ_1 requires an SVD, which is expensive for large problems. But for a tight frame,

$$\|\Phi' e_1\|_2 = \sqrt{\alpha} = \sigma_1$$

so here we can find σ_1 by a simple matrix-vector multiplication and a norm.

Parseval tight frame

Define. A matrix Φ is a **Parseval tight frame** [8, 9] iff Φ is a **tight frame** with $\alpha = \beta = 1$, i.e.,

$$\|\mathbf{x}\|_2^2 = \|\Phi' \mathbf{x}\|_2^2 = \sum_{m=1}^M |\langle \phi_m, \mathbf{x} \rangle|^2, \quad \forall \mathbf{x} \in \mathbb{F}^N. \quad (4.24)$$

What are the singular values σ_1 and σ_N of Φ in this case? $\sigma_1 = \dots = \sigma_N = 1$

Properties of Parseval tight frames

If Φ is a **Parseval tight frame**, then

- $\Phi \Phi' = I_N$.
- its pseudo-inverse is simply $\Phi^+ = \Phi'$.

Proof. Using (4.24):

$$\|\mathbf{x}\|_2^2 = \|\Phi' \mathbf{x}\|_2^2 \implies \mathbf{x}' (\mathbf{I} - \Phi \Phi') \mathbf{x} = 0, \quad \forall \mathbf{x} \in \mathbb{F}^N \implies \mathbf{I} - \Phi \Phi' = \mathbf{0}.$$

The converse also holds, i.e., if $\Phi \Phi' = I_N$, then Φ is a **Parseval tight frame**. (?)

A: True

B: False

??

Every **unitary matrix** is a **tight frame**. (?)

A: True

B: False

??

Example. A simple **Parseval tight frame** is the “Mercedes Benz” frame: $\Phi = \sqrt{\frac{2}{3}} \begin{bmatrix} 0 & -\sqrt{3}/2 & \sqrt{3}/2 \\ 1 & -1/2 & -1/2 \end{bmatrix}$.

One can verify that $\Phi\Phi' = I_2$.

Every **Parseval tight frame** is a **unitary matrix**. (?)

A: True

B: False

??

Venn diagram of frames

Rectangular
 $N \times M$ wide
 $N \leq M$

Frame
 $0 < \alpha = \sigma_N \leq \sigma_1$
 $\Phi\Phi'$ invertible

Tight frame
 $0 < \alpha = \sigma_N = \sigma_1$
 $\Phi^+ = \frac{1}{\sigma_1^2} \Phi'$

Parseval
tight frame
 $\sigma_N = \sigma_1 = 1$
 $\Phi^+ = \Phi'$

Unitary
 $M = N$
 $\mathbf{U}^+ = \mathbf{U}^{-1} = \mathbf{U}'$

Another way to construct a **tight frame** is to combine multiple unitary matrices:

$$\Phi = [U_1 \ \dots \ U_K].$$

This construction is fairly common in signal processing, *e.g.*, combining orthogonal wavelet transforms with other transforms.

If U_1 and U_2 are $N \times N$ unitary matrices, what is the frame bound of $\Phi \triangleq [U_1 \ U_2]$?

A: 1

B: 2

C: N D: $2N$

E: None

??

- Frames have numerous uses in signal processing [8–10].
- In the usual case where Φ is wide, they are considered “robust redundant signal representations” [11].
- Frame theory underpins recent advances in image denoising and attempts to provide insights into CNNs by relating them to perfect reconstruction filter banks [12] [13] [14].
- Solving LLS problems for a **Parseval tight frame** Φ is as easy as for a unitary matrix, because $\Phi^+ = \Phi'$. So given a vector x in \mathbb{F}^N , the representation of x using the (typically linearly dependent!) set of “atoms” $\{\phi_1, \dots, \phi_M\}$ for which the coefficient vector has minimum 2-norm is

$$x = \Phi c, \quad c = \Phi' x.$$

In this sense, Parseval tight frames are generalizations of orthonormal bases and unitary matrices.

- Other minimum-norm representations are also of interest, such as [15]

$$\arg \min_{c: x=\Phi c} \|c\|_p.$$

4.5 Projection and orthogonal projection

Idempotent matrix

L§7.1

Define. A (square) matrix P is called a **projection matrix** iff $P^2 = PP = P$.

Such a (square) matrix is also called an **idempotent matrix**.

Example. $P = \begin{bmatrix} 1 & a \\ 0 & 0 \end{bmatrix} = V\Lambda V^{-1}$, $V = \begin{bmatrix} 1 & -a/\sqrt{1+a^2} \\ 0 & 1/\sqrt{1+a^2} \end{bmatrix}$, $\Lambda = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ is idempotent for any a .

All eigenvalues of an idempotent matrix are either 0 or 1.

Proof. Suppose x is an eigenvector of an idempotent matrix P , i.e., $Px = \lambda x$. Multiplying both sides by P yields $P(Px) = \lambda(Px) \implies Px = \lambda^2 x$. Combining we have $\lambda^2 = \lambda$, so $\lambda = 0$ or $\lambda = 1$. \square

Every projection matrix is diagonalizable [\[wiki\]](#).

The converse of that property also holds, i.e., if A is a (square) diagonalizable matrix with eigenvalues that are all either 0 or 1, then A is always idempotent. (?)

A: True

B: False

??

??

Example. For any matrix A , the matrix $P = AA^+$ is **idempotent**, because $P^2 = (AA^+)(AA^+) = A(A^+AA^+) = AA^+ = P$.

Orthogonal projection matrix

Our primary interest will be the subset of projection matrices that are (Hermitian) symmetric matrices.

Define. A (square) matrix P is called an **orthogonal projector** or **orthogonal projection matrix** iff P is **idempotent** and P is **Hermitian**.

Caution: An **orthogonal projection matrix** typically is not an **orthogonal matrix**!

If $P = P' = P^2$ is an **orthogonal projection matrix** and if P is also an **orthogonal matrix** then $I = P'P = P^2 = P$. So the only matrix that is both is the identity matrix I .

Example. $P = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \right).$



Every **orthogonal projection matrix** has a **unitary eigendecomposition**. (?)

A: True

B: False

??

If x has unit norm, then xx' is an orthogonal projection matrix. (?)

A: True

B: False

??

Every orthogonal projection matrix P is positive semidefinite. (?)

A: True

B: False

??

Alternative explanation: if P is an orthogonal projection matrix, *i.e.*, both idempotent and Hermitian, then:

$$P = PP = P'P \succeq 0.$$

More generally, if Q is any (possibly non-square) matrix with orthonormal columns, then $P = QQ'$ is an orthogonal projection matrix, because

- $P = QQ'$ is Hermitian
- $P^2 = (QQ')(QQ') = QQ' = P$, because $Q'Q = I$.

Is the converse true? Can a $N \times N$ **orthogonal projection matrix** P be written as QQ' for some matrix Q with orthonormal columns?

A: Yes, always.

B: Yes, if $\text{rank}(P) \geq 1$.

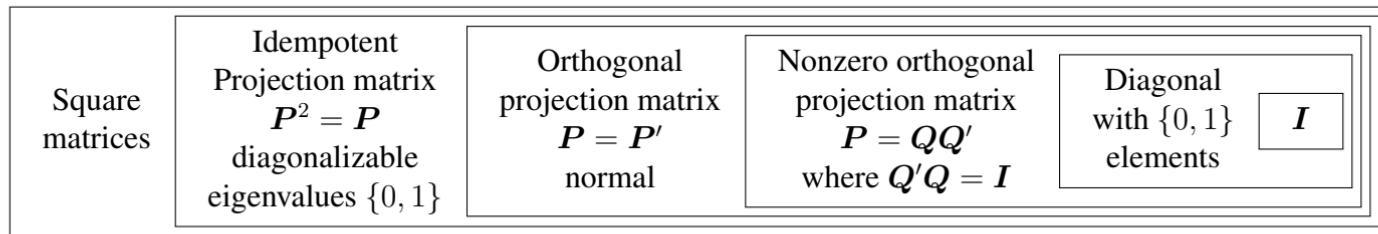
C: Yes, if $\text{rank}(P) \leq N - 1$.

D: Only if P is nonsingular.

E: No.

??

The following **Venn diagram** summarizes relationships related to **projection** operations.



Projection onto a subspace

Now we move towards using projection as a tool for solving signal processing problems.

Let $\mathbf{a}_1, \dots, \mathbf{a}_N$ denote any collection of N vectors, each in \mathbb{F}^M . The span of this set defines a subspace:

$$\mathcal{S} \triangleq \text{span}(\{\mathbf{a}_1, \dots, \mathbf{a}_N\}).$$

Equivalently (because we are working in \mathbb{F}^M here) we can group the vectors into a $M \times N$ matrix and then describe the subspace as the **range** of this matrix:

$$\mathcal{S} = \mathcal{R}(\mathbf{A}) \triangleq \{\mathbf{Ax} : \mathbf{x} \in \mathbb{F}^N\}, \quad \mathbf{A} \triangleq [\mathbf{a}_1 \ \dots \ \mathbf{a}_N].$$

In many applications (including handwritten digit recognition via nearest subspace) we will be given some vector \mathbf{y} and we want to find the vector in the subspace \mathcal{S} that is closest to \mathbf{y} :

$$\hat{\mathbf{y}} = \underset{\mathbf{s} \in \mathcal{S} = \mathcal{R}(\mathbf{A})}{\arg \min} \|\mathbf{y} - \mathbf{s}\|_2.$$

The resulting vector $\hat{\mathbf{y}}$ is called the **projection** of the point \mathbf{y} onto the subspace \mathcal{S} , and the process of finding that closest point is called **projecting** the point \mathbf{y} onto the subspace \mathcal{S} .

The key to solving this problem is to use the fact that every point in the subspace \mathcal{S} has the form $\mathbf{A}\mathbf{x}$ for some $\mathbf{x} \in \mathbb{F}^N$. Thus $\hat{\mathbf{y}} = \mathbf{A}\hat{\mathbf{x}}$ for some $\hat{\mathbf{x}} \in \mathbb{F}^N$, and we just have to find that best $\hat{\mathbf{x}}$ to get $\hat{\mathbf{y}}$. In other words, the closest point or projection problem is equivalent to:

$$\hat{\mathbf{y}} = \mathbf{A}\hat{\mathbf{x}}, \quad \hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathbb{F}^N} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2,$$

because every $\mathbf{s} \in \mathcal{S}$ is $\mathbf{s} = \mathbf{A}\mathbf{x}$ for some $\mathbf{x} \in \mathbb{F}^N$.

This form involves solving a LS problem, and a solution to that part is $\hat{\mathbf{x}} = \mathbf{A}^+\mathbf{y}$, so

$$\hat{\mathbf{y}} = \arg \min_{\mathbf{s} \in \mathcal{S}=\mathcal{R}(\mathbf{A})} \|\mathbf{y} - \mathbf{s}\|_2 = \mathbf{A}\mathbf{A}^+\mathbf{y}.$$

If \mathbf{A} does not have full rank, then there will be other LS solutions, i.e., $\hat{\mathbf{x}}$ is not unique.
Thus, $\hat{\mathbf{y}}$ is not unique. (?)

A: True

B: False

??

Practical implementation

Reiterating, the point on the subspace $\mathcal{R}(\mathbf{A})$ that is closest to the point \mathbf{y} is $\hat{\mathbf{y}} = \mathbf{A}\mathbf{A}^+\mathbf{y}$

If we need to compute this just once, for one \mathbf{A} and one \mathbf{y} , then using code like `A * (A \ y)` is fine if \mathbf{A} is small enough for backslash to work. If \mathbf{A} is large, then we use an iterative method to compute the LS coefficients $\hat{\mathbf{x}}$ first, then compute $\hat{\mathbf{y}} = \mathbf{A}\hat{\mathbf{x}}$.

But in most applications \mathbf{A} is fixed (after some training or modeling process) and we will need to perform projection for many different “test” \mathbf{y} vectors. In such cases, it is more efficient to use an SVD at the beginning (as part of the training process) to save computation at the test stage.

Specifically:

$$\mathbf{P}_{\mathcal{R}(\mathbf{A})} \triangleq \mathbf{A}\mathbf{A}^+ = \mathbf{U}_r \boldsymbol{\Sigma}_r \mathbf{V}_r' \mathbf{V}_r \boldsymbol{\Sigma}_r^{-1} \mathbf{U}_r' = \mathbf{U}_r \mathbf{U}_r'$$

So a practical implementation is something like

```
(U, s, V) = svd(A)
r = sum(s > threshold)
Ur = U[:, 1:r]
projector = (y) -> Ur * (Ur' * y)
```

We do the SVD once, then after that we need only use simple matrix-vector multiplies to perform projection.

Note that \mathbf{U}_r is an orthonormal basis for $\mathcal{R}(\mathbf{A})$ and $\mathbf{P}_{\mathcal{R}(\mathbf{A})} = \mathbf{U}_r \mathbf{U}_r'$. This is not a coincidence!

Orthonormal vs non-orthonormal bases for a subspace

(Read)

More generally, if \mathbf{Q} is a matrix whose columns are orthonormal, then those columns form an orthonormal basis for a subspace, namely $\mathcal{R}(\mathbf{Q})$, and the projection of a vector \mathbf{y} onto that subspace is simply

$$\mathbf{P}_{\mathcal{R}(\mathbf{Q})}\mathbf{y} = \mathbf{Q}(\mathbf{Q}'\mathbf{y}).$$

So **orthonormal bases** for a subspace are extremely convenient for computation, because subspace projection using such bases requires mere matrix-vector multiplication.

If a basis \mathbf{B} is not orthonormal, then to compute the projection we would need

$$\mathbf{P}_{\mathcal{R}(\mathbf{B})}\mathbf{y} = \mathbf{B}(\mathbf{B}^+\mathbf{y}),$$

as derived above, which is much more expensive in general because \mathbf{B}^+ usually requires an SVD.

Orthogonality principle revisited

(Read)

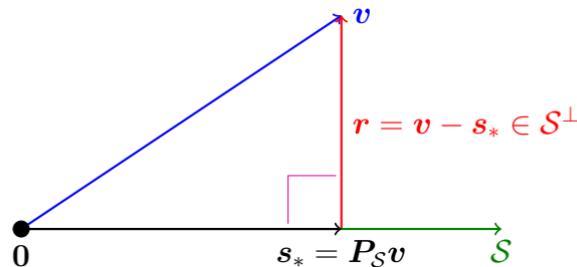
Another version of the **orthogonality principle** is the following. Let \mathcal{S} denote any subspace of a vector space \mathcal{V} , and $v \in \mathcal{V}$ be any point in that vector space. If the closest point in \mathcal{S} to v is

$$s_* = \arg \min_{s \in \mathcal{S}} \|v - s\|_2 = P_{\mathcal{S}} v,$$

then the **orthogonality principle** says that the residual vector $r = v - s_*$ is perpendicular to the entire subspace \mathcal{S} , i.e.,

$$\langle v - s_*, s \rangle = 0, \quad \forall s \in \mathcal{S}.$$

The following figure illustrates this principle.



Note that $r = v - s_* = v - P_S v = (I - P_S)v = P_S^\perp v$, $P_S^\perp \triangleq I - P_S$.

Proof sketch from [wiki]: For $s \in \mathcal{S}$ and any $\alpha \in \mathbb{F}$: $0 \leq \|(s_* + \alpha s) - v\|_2^2 - \|s_* - v\|_2^2 = 2 \text{real}\{\alpha \langle s_* - v, s \rangle\} + |\alpha|^2 \|s\|_2^2 \implies \langle s_* - v, s \rangle = 0$.

Projection onto a subspace's orthogonal complement

Recall that in a finite-dimensional vector space \mathcal{V} , if \mathcal{S} is a subspace of \mathcal{V} , then

$$\mathcal{V} = \mathcal{S} \oplus \mathcal{S}^\perp.$$

Clearly $P_{\mathcal{V}} = \mathbf{I}$, so it follows that

$$\mathbf{I} = P_{\mathcal{S}} + P_{\mathcal{S}^\perp}.$$

Thus the projection onto the subspace's **orthogonal complement** \mathcal{S}^\perp is simply:

$$P_{\mathcal{S}^\perp} = \mathbf{I} - P_{\mathcal{S}}.$$

If P is any projection matrix, then as a notation convention we define

$$P^\perp \triangleq \mathbf{I} - P.$$

It then follows that

$$P_{\mathcal{S}^\perp} = P_{\mathcal{S}}^\perp = \mathbf{I} - P_{\mathcal{S}}.$$

In words, $P_{\mathcal{S}}^\perp$ is the orthogonal projector onto \mathcal{S}^\perp , the orthogonal complement of the subspace \mathcal{S} .

Example. If $\mathcal{S} = \text{span}(\{\mathbf{1}_n\})$ then $P_{\mathcal{S}}^\perp = \mathbf{I} - \mathbf{1}_n \mathbf{1}_n^\perp = \mathbf{I} - \frac{1}{n} \mathbf{1}_n \mathbf{1}'_n$.

The matrix-vector product $\mathbf{y} = P_{\mathcal{S}}^\perp \mathbf{x}$ returns a vector \mathbf{y} having zero mean, so \mathbf{y} is orthogonal to $\mathbf{1}_n$.

Projectors and the four fundamental subspaces

Recall the SVD anatomy of a $M \times N$ matrix with rank $r \leq \min(M, N)$:

L§7.1.1

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}' = \sum_{k=1}^{\min(M,N)} \sigma_k \mathbf{u}_k \mathbf{v}'_k = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k = \left[\begin{array}{c|c} \mathbf{U}_r & \mathbf{U}_0 \\ \hline \end{array} \right] \left[\begin{array}{c|c} \Sigma_r & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array} \right] \left[\begin{array}{c|c} \mathbf{V}_r & \mathbf{V}_0 \\ \hline \end{array} \right]'.$$

The four fundamental subspaces have the following **orthonormal bases** and **orthogonal projectors**.

space in terms of \mathbf{A}	subspace in terms of \mathbf{A}'	equivalent subspace matrix	orthonormal basis	orthogonal projector
\mathbb{F}^N	$\mathcal{N}(\mathbf{A})$	$\mathcal{R}^\perp(\mathbf{A}')$	\mathbf{V}_0	$\mathbf{V}_0\mathbf{V}'_0 = \mathbf{I} - \mathbf{V}_r\mathbf{V}'_r$
\mathbb{F}^N	$\mathcal{N}^\perp(\mathbf{A})$	$\mathcal{R}(\mathbf{A}')$	\mathbf{V}_r	$\mathbf{V}_r\mathbf{V}'_r$
\mathbb{F}^M	$\mathcal{R}(\mathbf{A})$	$\mathcal{N}^\perp(\mathbf{A}')$	\mathbf{U}_r	$\mathbf{U}_r\mathbf{U}'_r$
\mathbb{F}^M	$\mathcal{R}^\perp(\mathbf{A})$	$\mathcal{N}(\mathbf{A}')$	\mathbf{U}_0	$\mathbf{U}_0\mathbf{U}'_0 = \mathbf{I} - \mathbf{U}_r\mathbf{U}'_r$

I list two forms for the projectors involving \mathbf{U}_0 and \mathbf{V}_0 because if we have stored “only” the **compact SVD**, then we will need to use \mathbf{U}_r and \mathbf{V}_r .

$$\mathcal{N}^\perp(\mathbf{A}') \oplus \mathcal{R}^\perp(\mathbf{A}) = \mathbb{F}^M. \text{ (?)}$$

A: True

B: False

??

Warm-up questions: projections

If Q is a matrix with orthonormal columns and D is diagonal matrix whose elements are all 0 or 1, then $P = QDQ'$ is an **orthogonal projection matrix**. (?)

A: True

B: False

??

Let rank r matrix A have **SVD** $A = U\Sigma V'$ and **compact SVD** $A = U_r \Sigma_r V_r'$, where we partition unitary matrix V as usual as $V = [V_r \quad V_0]$. Which of the following matrices is $P_{\mathcal{N}(A'A)}$?

A: $V_0 V_0'$ B: $V_r V_r'$ C: $V_0 V_r'$ D: $V_r V_0'$

E: None of these.

??

Continuing the previous problem, and assuming the rank r is much smaller than the matrix dimensions, which of the following JULIA snippets is the most efficient way to implement $P_{\mathcal{R}^\perp(A^+)x}$ after the line `U,s,V = svd(A)` ?

A: `V[:,1:r] * (V[:,1:r]' * x)`B: `V[:,1:r] * V[:,1:r]' * x`C: `x - V[:,1:r] * V[:,1:r]' * x`D: `x - V[:,1:r] * (V[:,1:r]' * x)`E: `V[:,(r+1):end] * (V[:,(r+1):end]' * x)`

??

Binary classifier design using least-squares

(Read)

Linear LS can be used as a simple tool for designing a binary classifier via supervised learning.

Given M feature vectors $\mathbf{v}_i \in \mathbb{F}^N$ and corresponding binary class labels $y_i = \pm 1$. We want to find a weight vector \mathbf{x} such that $\langle \mathbf{x}, \mathbf{v}_i \rangle$ has the same sign as y_i .

Form a $M \times N$ data matrix from the training feature vectors and a corresponding label vector:

$$\mathbf{A} = \begin{bmatrix} \mathbf{v}'_1 \\ \vdots \\ \mathbf{v}'_M \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix}.$$

Then solve a least-squares problem (or regularized variant thereof):

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathbb{R}^N} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2.$$

After “learning” the regression weights, the classifier for a subsequent test data point $\mathbf{v} \in \mathbb{R}^N$ is simply

$$\text{sign}(\mathbf{v}' \mathbf{x}).$$

It is somewhat unconventional to use LS for a data vector \mathbf{y} that is binary (± 1 values), whereas **logistic regression** (see Ch. 8) is really designed for such data. Nevertheless, the LS approach is simple and fast and can work adequately in some cases. This approach will be explored in Discussion.

4.6 Summary of LLS solution methods in terms of SVD

This chapter has discussed three different ways of solving the linear least-squares (LLS) problem (4.2), each of which have SVD expressions as follows.

- “Optimal” solution (minimum-norm LLS):

$$\hat{\mathbf{x}} = \sum_{k=1}^r \frac{1}{\sigma_k} \mathbf{v}_k (\mathbf{u}'_k \mathbf{y})$$

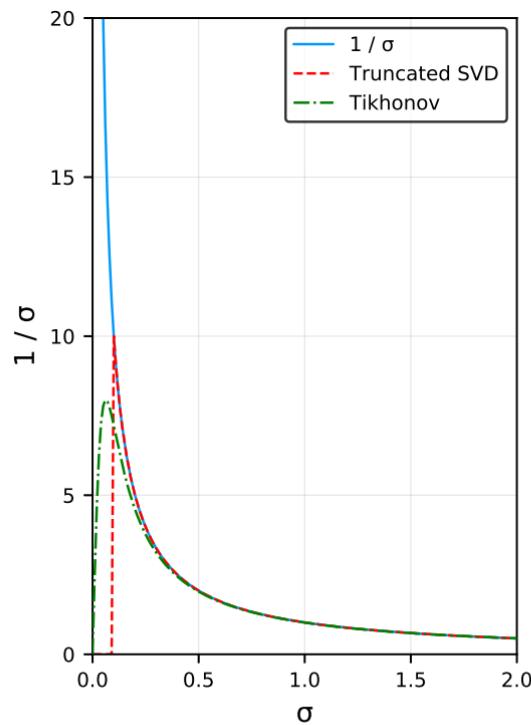
- Truncated SVD:

$$\hat{\mathbf{x}} = \sum_{k=1}^{K < r} \frac{1}{\sigma_k} \mathbf{v}_k (\mathbf{u}'_k \mathbf{y})$$

- Tikhonov regularized:

$$\hat{\mathbf{x}} = \sum_{k=1}^r \left(\frac{\sigma_k}{\sigma_k^2 + \beta} \right) \mathbf{v}_k (\mathbf{u}'_k \mathbf{y})$$

The SVD is a key tool for understanding LLS problems.



Bibliography

- [1] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005.
- [2] L. Eldén. *Matrix methods in data mining and pattern recognition*. Errata: <http://users.mai.liu.se/larel04/matrix-methods/index.html>. Soc. Indust. Appl. Math., 2007.
- [3] Per-AAke Wedin. “Perturbation theory for pseudo-inverses”. In: *BIT Numerical Mathematics* 13.2 (June 1973), 217–32.
- [4] G. W. Stewart. “On the perturbation of pseudo-inverses, projections and linear least squares problems”. In: *SIAM Review* 19.4 (1977), 634–62.
- [5] G. W. Stewart. “Stochastic perturbation theory”. In: *SIAM Review* 32.4 (1990), 579–610.
- [6] L. Meng and B. Zheng. “The optimal perturbation bounds of the Moore-Penrose inverse under the Frobenius norm”. In: *Linear Algebra and its Applications* 432.4 (Feb. 2010), 956–63.
- [7] L. Meng and B. Zheng. “New multiplicative perturbation bounds of the Moore-Penrose inverse”. In: *Linear and Multilinear Algebra* 63.5 (2015), 1037–48.
- [8] J. Kovacevic and A. Chebira. “Life beyond bases: the advent of frames (Part I)”. In: *IEEE Sig. Proc. Mag.* 24.4 (July 2007), 86–104.
- [9] J. Kovacevic and A. Chebira. “Life beyond bases: the advent of frames (Part II)”. In: *IEEE Sig. Proc. Mag.* 24.5 (Sept. 2007), 115–25.
- [10] J. Kovacevic and A. Chebira. “An introduction to frames”. In: *Found. & Trends in Sig. Pro.* 2.1 (2008), 1–94.

- [11] A. Rahimi, G. Zandi, and B. Daraby. “Redundancy of fusion frames in Hilbert spaces”. In: *Complex Analysis and Operator Theory* 10.3 (Mar. 2016), 545–65.
- [12] R. Yin, T. Gao, Y. M. Lu, and I. Daubechies. “A tale of two bases: local-nonlocal regularization on image patches with convolution framelets”. In: *SIAM J. Imaging Sci.* 10.2 (Jan. 2017), 711–50.
- [13] E. Kang and J. C. Ye. “Framelet denoising for low-dose CT using deep learning”. In: *Proc. IEEE Intl. Symp. Biomed. Imag.* 2018, 311–4.
- [14] J. C. Ye, Y. Han, and E. Cha. “Deep convolutional framelets: A general deep learning framework for inverse problems”. In: *SIAM J. Imaging Sci.* 11.2 (Jan. 2018), 991–1048.
- [15] M. Akcakaya and V. Tarokh. “A frame construction and a universal distortion bound for sparse representations”. In: *IEEE Trans. Sig. Proc.* 56.6 (June 2008), 2443–50.

Chapter 5

Norms

Contents (final version)

5.0 Vector norms	5.2
Properties of norms	5.6
Norm notation	5.6
Unitarily invariant norms	5.7
Inner products	5.8
5.1 Matrix norms and operator norms	5.13
Induced matrix norms	5.17
Norms defined in terms of singular values	5.20
Properties of matrix norms	5.23
Spectral radius	5.25
5.2 Convergence of sequences of vectors and matrices	5.28
5.3 Generalized inverse of a matrix	5.30
5.4 Procrustes analysis	5.32
Generalizations: non-square, complex, with translation	5.39
5.5 Summary	5.44

Introduction

This chapter discusses **vector norms** and **matrix norms** (also known as **operator norms**) in more generality and applies them to the **Procrustes problem**.

Source material for this chapter includes [1, §7.2-7.4].

The Tikhonov regularized least-squares problem in Ch. 4 illustrates the two primary uses of norms:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \underbrace{\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2}_{\text{distance: how far}} + \underbrace{\beta \|\mathbf{x}\|_2^2}_{\text{size: how big}} .$$

5.0 Vector norms

L§7.3

So far the only **vector norm** discussed in these notes has been the common **Euclidean norm**.

Many other vector norms are also important in signal processing.

Define. [1, p. 57] A norm on a vector space \mathcal{V} defined over a field \mathbb{F} is a function $\|\cdot\|$ from \mathcal{V} to $[0, \infty)$ that satisfies the following properties $\forall \mathbf{x}, \mathbf{y} \in \mathcal{V}$:

- $\|\mathbf{x}\| \geq 0$ (nonnegative)
- $\|\mathbf{x}\| = 0$ iff $\mathbf{x} = \mathbf{0}$ (positive)
- $\|\alpha\mathbf{x}\| = |\alpha| \|\mathbf{x}\|$ for all scalars $\alpha \in \mathbb{F}$ in the field (homogeneous)
- $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (**triangle inequality**)

Examples of vector norms

- For $1 \leq p < \infty$, the ℓ_p norm is

$$\|\mathbf{x}\|_p \triangleq \left(\sum_i |x_i|^p \right)^{1/p}. \quad (5.1)$$

- The **vector 2-norm** or **Euclidian norm** is the case $p = 2$: $\|\mathbf{x}\|_2 \triangleq \sqrt{\sum_i |x_i|^2}$.
- The 1-norm or “Manhattan norm” is the case $p = 1$: $\|\mathbf{x}\|_1 \triangleq \sum_i |x_i|$.
- The **max norm** or **infinity norm** or ℓ_∞ norm is

$$\|\mathbf{x}\|_\infty \triangleq \sup \{|x_1|, |x_2|, \dots\}, \quad (5.2)$$

where \sup denotes the **supremum** (least upper bound) of a set. One can show [2, Prob. 2.12] that

$$\|\mathbf{x}\|_\infty = \lim_{p \rightarrow \infty} \|\mathbf{x}\|_p. \quad (5.3)$$

For the vector space \mathbb{F}^N , the supremum is simply a maximum:

$$\|\mathbf{x}\|_\infty \triangleq \max \{|x_1|, \dots, |x_N|\}, \quad (5.4)$$

- For quantifying sparsity, it is useful to note that

$$\lim_{p \rightarrow 0} \|\mathbf{x}\|_p^p = \sum_i \mathbb{I}_{\{\mathbf{x}_i \neq 0\}} \triangleq \|\mathbf{x}\|_0, \quad (5.5)$$

where $\mathbb{I}_{\{\cdot\}}$ denotes the **indicator function** that is unity if the argument is true and zero if false.

However, the “0-norm” $\|\mathbf{x}\|_0$ is *not* a vector norm because it does not satisfy the at least one of the conditions of the norm definition above. The proper name for $\|\mathbf{x}\|_0$ is **counting measure**.

- Sometimes we want a **weighted** norm, e.g., the **weighted 2-norm** is

$$\|\mathbf{x}\|_{\mathbf{W}} = \sqrt{(\mathbf{x}' \mathbf{W} \mathbf{x})}.$$

Exercise. Show that the weighted Euclidean norm $\|\mathbf{x}\|_{\mathbf{W}}$ is a norm iff \mathbf{W} is a positive definite matrix.

In particular, if \mathbf{W} is a $N \times N$ diagonal matrix with positive diagonal elements w_i , then

$$\|\mathbf{x}\|_{\mathbf{W}} = \left(\sum_{i=1}^N w_i |x_i|^2 \right)^{1/2}.$$

Which of the four properties of a vector norm does the counting measure $\|\cdot\|_0$ satisfy?

A: 1,2

B: 1,3

C: 1,2,3

D: 1,2,4

E: 1,3,4

??

Practical implementation

For the preceding examples, in JULIA, first invoke

```
using LinearAlgebra
```

then use:

```
 $\|v\|_p$  norm(v, p)
```

```
 $\|v\|_2$  norm(v, 2)
```

```
 $\|v\|_1$  norm(v, 1)
```

```
 $\|v\|_\infty$  norm(v, Inf)
```

```
 $\|v\|_0$  norm(v, 0)
```

Caution. For $p < 1$, $\|\cdot\|_p$ is not a proper vector **norm**, though it is sometimes used in practical problems and `norm(v, p)` will evaluate (5.1) for any $-\infty \leq p \leq \infty$.



If W is `Diagonal(w)`, then which of these commands computes the weighted norm $\|x\|_W$?

- A: `norm(x .* w)`
- B: `norm(x .* w, 2)`
- C: `norm(x .* w.^2)`
- D: `norm(x .* sqrt.(w))`
- E: None

??

Properties of norms

- Let $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ be any two vector norms on a *finite-dimensional* space. Then there exist finite positive constants C_m and C_M (that depend on α and β) such that:

$$C_m \|\cdot\|_\alpha \leq \|\cdot\|_\beta \leq C_M \|\cdot\|_\alpha. \quad (5.6)$$

In a sense then “all norms are equivalent” to within constant factors.

- For any vector norm, the **reverse triangle inequality** is:

$$|\|\mathbf{x}\| - \|\mathbf{y}\|| \leq \|\mathbf{x} - \mathbf{y}\|, \quad \forall \mathbf{x}, \mathbf{y} \in \mathcal{V}.$$

- Any vector norm $\|\cdot\|$ on a vector space \mathcal{V} is a **convex function**:

$$\|\alpha \mathbf{x} + (1 - \alpha) \mathbf{z}\| \leq \alpha \|\mathbf{x}\| + (1 - \alpha) \|\mathbf{z}\|, \quad \forall \alpha \in [0, 1], \quad \forall \mathbf{x}, \mathbf{z} \in \mathcal{V}.$$

This fact is easy to prove using the **triangle inequality** and the homogeneity property (HW).

- For $p > 1$, $f(\mathbf{x}) \triangleq \|\mathbf{x}\|_p^p$ is **strictly convex**.

Norm notation

Some math literature uses $|\mathbf{x}|$ instead of $\|\mathbf{x}\|$ to denote a vector norm.

That notation should be avoided for matrices where $|\mathbf{A}|$ often denotes the determinant of \mathbf{A} .

Sometimes one must determine from context what $|\cdot|$ means in such literature.

Unitarily invariant norms

Some vector norms have the following useful property.

Define. A vector norm $\|\cdot\|$ on \mathbb{F}^N is **unitarily invariant** iff for every unitary matrix $\mathbf{U} \in \mathbb{F}^{N \times N}$:

$$\|\mathbf{U}\mathbf{x}\| = \|\mathbf{x}\|, \quad \forall \mathbf{x} \in \mathbb{F}^N.$$

Example. The Euclidean norm $\|\cdot\|_2$ on \mathbb{F}^N is unitarily invariant, because for any unitary \mathbf{U} (see p. 1.50):

$$\|\mathbf{U}\mathbf{x}\|_2 = \sqrt{(\mathbf{U}\mathbf{x})'(\mathbf{U}\mathbf{x})} = \sqrt{\mathbf{x}'\mathbf{U}'\mathbf{U}\mathbf{x}} = \sqrt{\mathbf{x}'\mathbf{x}} = \|\mathbf{x}\|_2, \quad \forall \mathbf{x} \in \mathbb{F}^N.$$

As noted previously, this property is related to **Parseval's theorem**.

Example. $\|\cdot\|_1$ is not unitarily invariant.

If $\mathbf{U} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$ and $\mathbf{x} = \mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ then $\|\mathbf{x}\|_1 = 1$ but $\|\mathbf{U}\mathbf{x}\|_1 = \sqrt{2}$.

Another unitary invariant norm on \mathbb{F}^N is $\|\cdot\|_{(\alpha)} \triangleq \alpha \|\cdot\|_2$ for any $\alpha > 0$.

Challenge. Find another unitarily invariant norm on \mathbb{F}^N or prove that no others exist.



Inner products

L§7.2

Most of the vector spaces used in this course are **inner product spaces**, meaning a vector space with an associated **inner product** operation.

Define. For a vector space \mathcal{V} over the field \mathbb{F} , an **inner product** operation is a function $\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{F}$ that must satisfy the following axioms $\forall \mathbf{x}, \mathbf{y} \in \mathcal{V}, \alpha \in \mathbb{F}$.

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle^* \quad (\text{Hermitian symmetry})$$

$$\langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle \quad (\text{additivity})$$

$$\langle \alpha \mathbf{x}, \mathbf{y} \rangle = \alpha \langle \mathbf{x}, \mathbf{y} \rangle \quad (\text{scaling})$$

$$\langle \mathbf{x}, \mathbf{x} \rangle \geq 0 \text{ and } \langle \mathbf{x}, \mathbf{x} \rangle = 0 \text{ iff } \mathbf{x} = \mathbf{0}. \quad (\text{positive definite})$$

Examples of inner products

Example. For vectors in \mathbb{F}^N , the usual inner product is

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{n=1}^N x_n y_n^*.$$

Example. For the (infinite dimensional) vector space of square integrable functions on the interval $[a, b]$, the ♦♦ following integral is a valid inner product:

$$\langle f, g \rangle = \int_a^b f(t)g^*(t) dt.$$

Example. For two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{F}^{M \times N}$ (a **vector space!**), the **Frobenius inner product**, also called the **Hilbert–Schmidt inner product**, is defined as:

$$\langle \mathbf{A}, \mathbf{B} \rangle \triangleq \text{trace}\{\mathbf{AB}'\} = \sum_{m=1}^M [\mathbf{AB}']_{mm} = \sum_{m=1}^M \sum_{n=1}^N a_{mn} b_{mn}^* = \langle \text{vec}(\mathbf{A}), \text{vec}(\mathbf{B}) \rangle. \quad (5.7)$$

Exercise. Verify the four properties above for these inner product examples.

Properties of inner products

- Bilinearity:

$$\left\langle \sum_i \alpha_i \mathbf{x}_i, \sum_j \beta_j \mathbf{y}_j \right\rangle = \sum_i \sum_j \alpha_i \beta_j^* \langle \mathbf{x}_i, \mathbf{y}_j \rangle, \quad \forall \{\mathbf{x}_i\}, \{\mathbf{y}_j\} \in \mathcal{V}.$$

- Any valid vector **inner product** induces a valid vector **norm**:

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}. \quad (5.8)$$

Exercise. Verify that such an **induced norm** satisfies the four conditions for a norm on p. 5.2.

- A vector norm satisfies the **parallelogram law**:

$$\frac{1}{2} (\|\mathbf{x} + \mathbf{y}\|^2 + \|\mathbf{x} - \mathbf{y}\|^2) = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2, \quad \forall \mathbf{x}, \mathbf{y} \in \mathcal{V},$$

iff it is induced by an inner product via (5.8). The required inner product is

$$\begin{aligned} \langle \mathbf{x}, \mathbf{y} \rangle &\triangleq \frac{1}{4} (\|\mathbf{x} + \mathbf{y}\|^2 - \|\mathbf{x} - \mathbf{y}\|^2 + i \|\mathbf{x} + i\mathbf{y}\|^2 - i \|\mathbf{x} - i\mathbf{y}\|^2) \\ &= \frac{\|\mathbf{x} + \mathbf{y}\|^2 - \|\mathbf{x}\|^2 - \|\mathbf{y}\|^2}{2} + i \frac{\|\mathbf{x} + i\mathbf{y}\|^2 - \|\mathbf{x}\|^2 - \|\mathbf{y}\|^2}{2}. \end{aligned}$$



- The **Cauchy-Schwarz inequality** (or **Schwarz** or **Cauchy-Bunyakovsky-Schwarz** inequality) states:

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} \sqrt{\langle \mathbf{y}, \mathbf{y} \rangle}, \quad \forall \mathbf{x}, \mathbf{y} \in \mathcal{V}, \quad (5.9)$$

for a norm $\|\cdot\|$ induced by an inner product $\langle \cdot, \cdot \rangle$ via (5.8), with equality iff \mathbf{x} and \mathbf{y} are linearly dependent.

In an inner product space on \mathbb{R} , is $\langle \mathbf{x}, \mathbf{y} \rangle \leq \|\mathbf{x}\| \|\mathbf{y}\|$?

A: Yes, always

B: Not always

C: Never

??

Proof of Cauchy-Schwarz inequality for \mathbb{F}^N

(Read)

For any $\mathbf{x}, \mathbf{y} \in \mathbb{F}^N$ let $\mathbf{A} = [\mathbf{x} \ \mathbf{y}]$, so $\mathbf{A}'\mathbf{A} = \begin{bmatrix} \mathbf{x}'\mathbf{x} & \mathbf{x}'\mathbf{y} \\ \mathbf{y}'\mathbf{x} & \mathbf{y}'\mathbf{y} \end{bmatrix}$.

$\mathbf{A}'\mathbf{A}$ is Hermitian symmetric \Rightarrow its eigenvalues are all real and nonnegative.

$\Rightarrow \det\{\mathbf{A}'\mathbf{A}\} \geq 0 \Rightarrow (\mathbf{x}'\mathbf{x})(\mathbf{y}'\mathbf{y}) - (\mathbf{y}'\mathbf{x})(\mathbf{x}'\mathbf{y}) \geq 0 \Rightarrow |\mathbf{x}'\mathbf{y}|^2 \leq (\mathbf{x}'\mathbf{x})(\mathbf{y}'\mathbf{y}) = \|\mathbf{x}\|_2^2 \|\mathbf{y}\|_2^2$.

Taking the square root of both sides yields the inequality. \square

We used the fact that $(\mathbf{y}'\mathbf{x})(\mathbf{x}'\mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle \langle \mathbf{y}, \mathbf{x} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle (\langle \mathbf{x}, \mathbf{y} \rangle)^* = |\langle \mathbf{x}, \mathbf{y} \rangle|^2$.

Angle between vectors

Define. The **angle** θ between two nonzero vectors $\mathbf{x}, \mathbf{y} \in \mathcal{V}$ is defined by

$$\cos \theta = \frac{|\langle \mathbf{x}, \mathbf{y} \rangle|}{\|\mathbf{x}\| \|\mathbf{y}\|} \implies \theta \in [0, \pi/2].$$

For real vectors, we can omit the absolute value and obtain $\theta \in [0, \pi]$.

The Cauchy-Schwarz inequality is equivalent to the statement $|\cos \theta| \leq 1$.

An inner product for random variables

(Read) ♦♦

For two real, zero-mean random variables X, Y defined on a joint probability space, a natural inner product is $E[XY]$. (Keep in mind that random variables are functions.) With this definition, the corresponding norm is $\|X\| = \sqrt{\langle X, X \rangle} = \sqrt{E[X^2]} = \sigma_X$, the standard deviation of X . Here, the Cauchy-Schwarz inequality is equivalent to usual bound on the **correlation coefficient**: $\rho_{X,Y} \triangleq \frac{E[XY]}{\sigma_X \sigma_Y} \implies |\rho_{X,Y}| \leq 1$.

With this definition of inner product, what types of random variables are “orthogonal”?

Pairs of random variables that are **uncorrelated**, i.e., where $E[XY] = 0$.

More inner product inequalities

(Read)

For the usual **inner product** on \mathbb{F}^N :

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\|_1 \|\mathbf{y}\|_\infty. \quad (5.10)$$

Proof: $|\langle \mathbf{x}, \mathbf{y} \rangle| = |\sum_i x_i y_i^*| \leq \sum_i |x_i| |y_i| \leq \sum_i |x_i| \|\mathbf{y}\|_\infty = \|\mathbf{x}\|_1 \|\mathbf{y}\|_\infty$.

More generally, if $1/p + 1/q = 1$ and $1 < p, q < \infty$, then **Hölder's inequality** states that

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q, \quad (5.11)$$

again for the usual inner product on \mathbb{F}^N .

Using (5.10), the **Frobenius inner product** (5.7) for matrices in $\mathbb{F}^{M \times N}$ satisfies:

$$\text{real}\{\langle \mathbf{A}, \mathbf{B} \rangle\} \leq |\langle \mathbf{A}, \mathbf{B} \rangle| = |\langle \text{vec}(\mathbf{A}), \text{vec}(\mathbf{B}) \rangle| \leq \|\text{vec}(\mathbf{A})\|_1 \|\text{vec}(\mathbf{B})\|_\infty. \quad (5.12)$$

Caution: in general, $\|\text{vec}(\mathbf{A})\|_1 \neq \|\mathbf{A}\|_1$ and $\|\text{vec}(\mathbf{B})\|_\infty \neq \|\mathbf{B}\|_\infty$.

5.1 Matrix norms and operator norms

L§7.4

Also important are **matrix norms** and **operator norms**; roughly speaking these functions quantify “how large” are the elements of a matrix, in different ways.

Define. [1, p. 59] A **matrix norm** on the vector space of matrices $\mathbb{F}^{M \times N}$ is a function $\|\cdot\|$ from $\mathbb{F}^{M \times N}$ to $[0, \infty)$ that satisfies the following properties $\forall \mathbf{A}, \mathbf{B} \in \mathbb{F}^{M \times N}$:

- | | |
|--|-----------------------|
| $\ \mathbf{A}\ \geq 0$ | (nonnegative) |
| $\ \mathbf{A}\ = 0$ iff $\mathbf{A} = \mathbf{0}_{M \times N}$ | (positive) |
| $\ \alpha \mathbf{A}\ = \alpha \ \mathbf{A}\ $ for all scalars $\alpha \in \mathbb{F}$ in the field | (homogeneous) |
| $\ \mathbf{A} + \mathbf{B}\ \leq \ \mathbf{A}\ + \ \mathbf{B}\ $ | (triangle inequality) |

Because the set of all $M \times N$ matrices $\mathbb{F}^{M \times N}$ is itself a **vector space**, matrix norms are simply vector norms for that space. So at first having a new definition might seem to have modest utility. However, many, *but not all*, matrix norms are **sub-multiplicative**, also called **consistent** [1, p. 61], meaning that they satisfy the following inequality:

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|, \quad \forall \mathbf{A}, \mathbf{B}. \tag{5.13}$$

These notes use the notation $\|\cdot\|$ to distinguish such **matrix norms** from the ordinary matrix norms $\|\cdot\|$ on the vector space $\mathbb{F}^{M \times N}$ that need not satisfy this extra condition.

Examples of matrix norms

- The **max norm** on $\mathbb{F}^{M \times N}$ is the element-wise maximum: $\|A\|_{\max} \triangleq \max_{i,j} |a_{ij}|$. This norm is somewhat like the infinity norm for vectors of length MN . One can compute it in JULIA using `norm(A, Inf)` after invoking `using LinearAlgebra`. Equivalently one may use `norm(A[:, :], Inf)` because the matrix shape is unimportant for this norm. (However, this differs completely from `opnorm(A, Inf)` that computes $\|A\|_\infty$ described below.) The max norm is a matrix norm on the vector space $\mathbb{F}^{M \times N}$ but it does not satisfy the **sub-multiplicative** condition (5.13) so it is of limited use. Most of the norms of interest in signal processing are sub-multiplicative, so such matrix norms are our primary focus hereafter.
- The **Frobenius norm** (aka **Hilbert-Schmidt norm** and **trace norm**) is defined on $\mathbb{F}^{M \times N}$ by

$$\|A\|_F \triangleq \sqrt{\sum_{m=1}^M \sum_{n=1}^N |a_{mn}|^2} = \sqrt{\text{trace}\{A'A\}} = \sqrt{\text{trace}\{AA'\}} = \|\text{vec}(A)\|_2, \quad (5.14)$$

and is also called the **Schur norm** and **Schatten 2-norm**. It is a very easy norm to compute.

The equalities related to trace are a HW problem.

Practical implementation: `norm(A, 2)` or `norm(A)` or `norm(A[:, :], 2)` or `norm(A[:, :])`

Again, shape of A is unimportant for this norm.



To relate the Frobenius norm of a matrix to its singular values:

$$\begin{aligned}\|A\|_F &= \sqrt{\text{trace}\{AA'\}} = \sqrt{\text{trace}\{U_r\Sigma_rV'_rV_r\Sigma_rU'_r\}} = \sqrt{\text{trace}\{\Sigma_r\Sigma_rU'_rU_r\}} \\ &= \sqrt{\text{trace}\{\Sigma_r^2\}} = \sqrt{\sum_{k=1}^r \sigma_k^2} = \|\Sigma_r\|_F = \|\Sigma\|_F.\end{aligned}$$

This norm is invariant to unitary transformations [3, p. 442], because of the trace property (1.27).

This norm is induced by the **Frobenius inner product**. It is not induced by any vector norm on \mathbb{F}^N (see ♦♦ next page) [4], but nevertheless it is **compatible** with the Euclidean vector norm because

$$\|Ax\|_2 \leq \|A\|_F \|x\|_2. \quad (5.15)$$

However, this upper bound is not tight in general. (It is tight for rank-1 matrices only.)

By combining (5.15) with the definition of matrix multiplication, one can show easily that the Frobenius norm is **sub-multiplicative** [5, p. 291].

What is the Frobenius norm of the **outer product** $\|uv'\|_F$ for $u \in \mathbb{F}^M$, $v \in \mathbb{F}^N$?

- A: $\|u\|_2 \|v\|_2$ B: $\sqrt{\|u\|_2 \|v\|_2}$ C: $|u'v|^2$ D: $|u'v|$ E: None of these.

??

$\ell_{p,q}$ norms

(Read)

For certain signal processing problems involving **group sparsity** [6, 7], the following family of $\ell_{p,q}$ **matrix norms** is useful:

$$\|\mathbf{A}\|_{p,q} \triangleq \left(\sum_{n=1}^N \left(\|\mathbf{A}_{:,n}\|_p \right)^q \right)^{1/q} = \left(\sum_{n=1}^N \left(\sum_{m=1}^M |a_{m,n}|^p \right)^{q/p} \right)^{1/q}.$$



This family considers a $M \times N$ matrix \mathbf{A} as a collection of N columns of length M .

A particularly popular special case for group sparsity problems is

$$\|\mathbf{A}\|_{1,2} = \left(\sum_{n=1}^N \left(\|\mathbf{A}_{:,n}\|_1 \right)^2 \right)^{1/2}.$$

Note that in general $\|\mathbf{A}\|_{p,p} = \|\text{vec}(\mathbf{A})\|_p$ and specifically $\|\mathbf{A}\|_{2,2} = \|\mathbf{A}\|_{\text{F}}$.

Challenge. Determine which of the $\ell_{p,q}$ norms are **sub-multiplicative**.

Induced matrix norms

If $\|\cdot\|$ is any vector norm that is suitable for both \mathbb{F}^N and \mathbb{F}^M , then a matrix norm for $\mathbb{F}^{M \times N}$ is:

$$\|A\| \triangleq \max_{x : \|x\|=1} \|Ax\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}, \quad (5.16)$$

which is called an **operator norm** (because now A acts as an operation). By construction:

$$\|Ax\| \leq \|A\| \|x\|, \quad \forall x \in \mathbb{F}^N. \quad (5.17)$$

We say such a matrix norm $\|\cdot\|$ is **induced** by the vector norm $\|\cdot\|$.

Importantly, the **sub-multiplicative** property (5.13) holds for any **induced norm** provided the number of columns of A matches the number of rows of B . This fact follows readily from the definition (5.16) and the property (5.17) because

$$\|AB\| = \max_{x : \|x\|=1} \|ABx\| \leq \max_{x : \|x\|=1} \|A\| \|Bx\| = \|A\| \|B\|.$$

Example. The most important matrix norms (**operator norms**) are induced by the vector norm $\|\cdot\|_p$, i.e.,

$$\|\mathbf{A}\|_p = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{Ax}\|_p}{\|\mathbf{x}\|_p}. \quad (5.18)$$

- The **spectral norm** $\|\cdot\|_2$, often denoted simply $\|\cdot\|$, is defined on $\mathbb{F}^{M \times N}$ by (5.18) with $p = 2$. This is the matrix norm induced by the Euclidean vector norm. As shown on p. 2.23:

$$\|\mathbf{A}\|_2 = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{Ax}\|_2}{\|\mathbf{x}\|_2} = \max \left\{ \sqrt{\lambda} : \lambda \in \text{eig}\{\mathbf{A}'\mathbf{A}\} \right\} = \sigma_1(\mathbf{A}).$$

- The **maximum row sum matrix norm** is defined on $\mathbb{F}^{M \times N}$ by

$$\|\mathbf{A}\|_\infty \triangleq \max_{1 \leq i \leq M} \sum_{j=1}^N |a_{ij}|. \quad (5.19)$$

It is induced by the ℓ_∞ vector norm. It differs from the max norm defined above! Here the shape matters!

Proof:

$$\begin{aligned} \|\mathbf{A}\|_\infty &= \max_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{Ax}\|_\infty}{\|\mathbf{x}\|_\infty} = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\max_{m=1,\dots,M} |[\mathbf{Ax}]_m|}{\|\mathbf{x}\|_\infty} = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\max_{m=1,\dots,M} \left| \sum_{n=1}^N a_{mn} x_n \right|}{\|\mathbf{x}\|_\infty} \\ &\leq \max_{\mathbf{x} \neq \mathbf{0}} \frac{\max_{m=1,\dots,M} \sum_{n=1}^N |a_{mn}| |x_n|}{\|\mathbf{x}\|_\infty} \leq \max_{\mathbf{x} \neq \mathbf{0}} \frac{\max_{m=1,\dots,M} \sum_{n=1}^N |a_{mn}| \|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} = \max_{1 \leq m \leq M} \sum_{n=1}^N |a_{mn}|. \end{aligned}$$

- The **maximum column sum matrix norm** is defined on $\mathbb{F}^{M \times N}$ by

$$\|A\|_1 \triangleq \max_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{1 \leq j \leq N} \sum_{i=1}^M |a_{ij}|. \quad (5.20)$$

It is induced by the ℓ_1 vector norm. Note that $\|A\|_1 = \|A'\|_\infty$.

Norms defined in terms of singular values

Here are three important norms used in modern signal processing problems.

- The **nuclear norm**, sometimes called the **trace norm** [1, p. 60], is the sum of the singular values:

$$\|\mathbf{A}\|_* \triangleq \sum_{k=1}^{\min(M,N)} \sigma_k.$$

- For $1 \leq p \leq \infty$, the **Schatten p-norm** of a $M \times N$ matrix is defined using the ℓ_p norm of its singular values:

$$\|\mathbf{A}\|_{S,p} = \left(\sum_{k=1}^{\min(M,N)} \sigma_k^p \right)^{1/p} = \left(\sum_{k=1}^{\text{rank}(\mathbf{A})} \sigma_k^p \right)^{1/p}.$$

- The **Ky-Fan K-norm** is the sum of the first $1 \leq K \leq \min(M, N)$ singular values of a matrix:

$$\|\mathbf{A}\|_{\text{Ky-Fan},K} = \sum_{k=1}^K \sigma_k(\mathbf{A}).$$

For a PCA generalization that uses this norm see [8].

- For (complicated!) proofs that these are in fact norms (*i.e.*, satisfy the triangle inequality), see [9, p.91].
- All of these three norms (nuclear, Schatten, and Ky-Fan) are **sub-multiplicative** [9, p.94].

Relationships between these norms:

- Nuclear norm:

$$\|A\|_* = \|A\|_{S,1} = \|A\|_{\text{Ky-Fan}, \min(M,N)}$$

- Spectral norm:

$$\|A\|_2 = \sigma_1(A) = \|A\|_{S,\infty} = \|A\|_{\text{Ky-Fan},1}$$

- Frobenius norm:

$$\|A\|_F = \|A\|_{S,2}$$

Exercise. Relate $\|A\|_F$ to a Ky-Fan norm and a nuclear norm involving A .

??

Challenge. Prove whether the **Schatten p-norm** $\|\cdot\|_{S,p}$ is or is not an **induced norm** for $1 \leq p < \infty$. ♦♦

Exercise. Define a matrix norm that unifies all of the matrix norms defined here in terms of singular values.

For $1 \leq p \leq \infty$ and $1 \leq K \leq \min(M, N)$, use the ℓ_p norm of its first K singular values:

$$\|A\|_{K,p} = \left(\sum_{k=1}^K \sigma_k^p \right)^{1/p}.$$

Practical implementation

JULIA commands (after invoking `using LinearAlgebra`) for some of these norms are as follows:

- $\|A\|_1$ opnorm(A, 1)
 - $\|A\|_2$ opnorm(A, 2) or just opnorm(A)
 - $\|A\|_\infty$ opnorm(A, Inf) ($\|A\|_{\max}$ is norm(A, Inf))
 - $\|A\|_*$ sum(svdvals(A)) or sum(svd(A).S)



Examples

For $A = [1 \ -3]' [1 \ 1 \ 1]$, what is $\text{norm}(A, \ \text{Inf})$?

- A; 2 B; 3 C; 6 D; 9 E; None of these

??

For $A = [1 \ -3]'[1 \ 1 \ 1]$, what is $\text{opnorm}(A, \ \text{Inf})$?

- A: 2 B: 3 C: 6 D: 9 E: None of these

??

For $A = [1 \ -3]' [1 \ 1 \ 1]$, what is $\text{opnorm}(A, 2)$?

- A: 2 B: 3 C: 6 D: 9 E: None of these

??

??

Properties of matrix norms

All matrix norms are also **equivalent** (to within constants that depend on the matrix dimensions). See [1, p. 61] for inequalities relating various matrix norms.

Example. (See HW):

$$\mathbf{A} \in \mathbb{F}^{M \times N} \implies \|\mathbf{A}\|_1 \leq \sqrt{M} \|\mathbf{A}\|_2.$$

Example. To relate the **spectral norm** and **nuclear norm** for a matrix \mathbf{A} having rank r :

$$\|\mathbf{A}\|_* = \sum_{k=1}^r \sigma_k \leq \sum_{k=1}^r \sigma_1 = r\sigma_1 = r \|\mathbf{A}\|_2$$

$$\|\mathbf{A}\|_2 = \sigma_1 \leq \sum_{k=1}^r \sigma_k = \|\mathbf{A}\|_*.$$

Combining:

$$\frac{1}{r} \|\mathbf{A}\|_* \leq \|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_* \leq r \|\mathbf{A}\|_2.$$

To express it in way that depends on the norm only (not r , which is a property of a specific matrix):

$$\frac{1}{\min(M, N)} \|\mathbf{A}\|_* \leq \|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_* \leq \min(M, N) \|\mathbf{A}\|_2.$$

Unitarily invariant matrix norms

Define. A matrix norm $\|\cdot\|$ on $\mathbb{F}^{M \times N}$ is called **unitarily invariant** iff for all unitary matrices $\mathbf{U} \in \mathbb{F}^{M \times M}$ and $\mathbf{V} \in \mathbb{F}^{N \times N}$:

$$\|\mathbf{U}\mathbf{A}\mathbf{V}\| = \|\mathbf{A}\|, \quad \forall \mathbf{A} \in \mathbb{F}^{M \times N}.$$

Theorem.

- The **spectral norm** $\|\mathbf{A}\|_2$ is unitarily invariant
- Any **Schatten p-norm** $\|\mathbf{A}\|_{S,p}$ is unitarily invariant
Proof sketch: unitary matrix rotations do not change singular values.
- The **Frobenius norm** $\|\mathbf{A}\|_F$ is unitarily invariant

Proof for Frobenius case:

$$\begin{aligned} \|\mathbf{U}\mathbf{A}\mathbf{V}\|_F^2 &= \text{trace}\{\mathbf{V}'\mathbf{A}'\mathbf{U}'\mathbf{U}\mathbf{A}\mathbf{V}\} = \text{trace}\{\mathbf{V}'\mathbf{A}'\mathbf{A}\mathbf{V}\} \\ &= \text{trace}\{\mathbf{V}\mathbf{V}'\mathbf{A}'\mathbf{A}\} = \text{trace}\{\mathbf{A}\mathbf{A}'\} && = \|\mathbf{A}\|_F^2. \end{aligned}$$

(We could also prove it using singular values.)

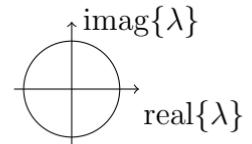
The Frobenius norm has an even more general invariance. If \mathbf{U} has M orthonormal columns and \mathbf{Q} has N orthonormal rows, then by the same proof $\|\mathbf{U}\mathbf{A}\mathbf{Q}\|_F = \|\mathbf{A}\|_F$ for any $\mathbf{A} \in \mathbb{F}^{M \times N}$.

Fact. Every **unitarily invariant** norm is **sub-multiplicative**. (See [9, p.94] for complicated proof.)

Spectral radius

Define. For any square matrix, the **spectral radius** is the maximum absolute eigenvalue:

$$\mathbf{A} \in \mathbb{F}^{N \times N} \implies \rho(\mathbf{A}) \triangleq \max_i |\lambda_i(\mathbf{A})|.$$



- By construction, $|\lambda_i(\mathbf{A})| \leq \rho(\mathbf{A})$ so all eigenvalues lie within a disk in the complex plane of radius $\rho(\mathbf{A})$, hence the name.
- In general, $\rho(\mathbf{A})$ is *not* a **matrix norm** and $\|\mathbf{A}\mathbf{x}\| \not\leq \rho(\mathbf{A}) \|\mathbf{x}\|$.
- However, if \mathbf{A} is **normal**, then recall from (2.6) that we if order its eigenvalues in decreasing order of their absolute values, then we can relate its unitary eigendecomposition to an SVD as follows:

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}' = \sum_{n=1}^N \lambda_n \mathbf{v}_n \mathbf{v}'_n = \sum_{n=1}^N \underbrace{|\lambda_n|}_{\sigma_n} \underbrace{\text{sign}(\lambda_n)}_{\mathbf{u}_n} \mathbf{v}_n \mathbf{v}'_n.$$

- Thus if \mathbf{A} is **normal** (e.g., $\mathbf{A} = \mathbf{A}'$) then $\rho(\mathbf{A}) = \sigma_1(\mathbf{A}) = \|\mathbf{A}\|_2$, so $\|\mathbf{A}\mathbf{x}\|_2 \leq \rho(\mathbf{A}) \|\mathbf{x}\|_2$.
- Furthermore, \mathbf{A} **normal** $\implies |\mathbf{x}'\mathbf{A}\mathbf{x}| \leq \rho(\mathbf{A}) \|\mathbf{x}\|_2^2$ because

$$\max_{\mathbf{x} \neq \mathbf{0}} \frac{|\mathbf{x}'\mathbf{A}\mathbf{x}|}{\|\mathbf{x}\|_2^2} = \max_{\mathbf{z} \neq \mathbf{0}} \frac{|(\mathbf{V}\mathbf{z})'\mathbf{A}(\mathbf{V}\mathbf{z})|}{\|\mathbf{V}\mathbf{z}\|_2^2} = \max_{\mathbf{z} \neq \mathbf{0}} \frac{|\mathbf{z}'\Lambda\mathbf{z}|}{\|\mathbf{z}\|_2^2} = \max_n |\lambda_n(\mathbf{A})| = \rho(\mathbf{A}) = \sigma_1(\mathbf{A}) = \|\mathbf{A}\|_2.$$

- If $\|\cdot\|$ is any **induced matrix norm** on $\mathbb{F}^{N \times N}$ and if $\mathbf{A} \in \mathbb{F}^{N \times N}$, then

$$\rho(\mathbf{A}) \leq \|\mathbf{A}\|. \quad (5.21)$$

Proof. If $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$, then $|\lambda| \|\mathbf{v}\| = \|\lambda\mathbf{v}\| = \|\mathbf{A}\mathbf{v}\| \leq \|\mathbf{A}\| \|\mathbf{v}\|$. Dividing by $\|\mathbf{v}\|$, which is fine because $\mathbf{v} \neq \mathbf{0}$, yields $|\lambda| \leq \|\mathbf{A}\|$. This inequality holds for all eigenvalues, including the one with maximum magnitude. \square

- If $\mathbf{A} \in \mathbb{F}^{N \times N}$, then $\lim_{k \rightarrow \infty} \mathbf{A}^k = \mathbf{0}$ if and only if $\rho(\mathbf{A}) < 1$.

This property is particularly important for analyzing the convergence of iterative algorithms, including training **recurrent neural networks** [10]. (cf. HW)

- For any $\mathbf{A} \in \mathbb{F}^{N \times N}$, the **spectral radius** is an infimum of all **induced matrix norms**: ◆◆

$$\rho(\mathbf{A}) = \inf \{\|\mathbf{A}\| : \|\cdot\| \text{ is an induced matrix norm}\}.$$

- **Gelfand's formula** for any **induced matrix norm** $\|\cdot\|$ for a square matrix \mathbf{A} is: ◆◆

$$\rho(\mathbf{A}) = \lim_{k \rightarrow \infty} \|\mathbf{A}^k\|^{1/k}. \quad (5.22)$$

Which equality (if any) correctly relates a singular value and a spectral radius for any general matrix $\mathbf{A} \in \mathbb{F}^{M \times N}$?

- A: $\sigma_1(\mathbf{A}) = |\rho(\mathbf{A})|$
- B: $\sigma_1(\mathbf{A}) = \rho^2(\mathbf{A})$
- C: $\sigma_1(\mathbf{A}) = \rho(\mathbf{A}'\mathbf{A})$
- D: $\sigma_1(\mathbf{A}) = \sqrt{\rho(\mathbf{A}'\mathbf{A})}$
- E: None of these.

??

Practical step size for gradient descent

The **gradient descent (GD)** method $\mathbf{x}_{k+1} = \mathbf{x}_k - \mu \mathbf{A}'(\mathbf{A}\mathbf{x}_k - \mathbf{y})$ for solving a linear least-squares problem $\arg \min_{\mathbf{x}} \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2$ **converges** (for any \mathbf{x}_0 ; see Ch. 8) iff $\rho(\mathbf{I} - \mu \mathbf{A}'\mathbf{A}) < 1$, i.e., iff the step size μ satisfies

$$0 < \mu < \frac{2}{\sigma_1^2(\mathbf{A})},$$

where, using the inequality (5.21):

$$\sigma_1^2(\mathbf{A}) = \sigma_1(\mathbf{A}'\mathbf{A}) = \|\mathbf{A}'\mathbf{A}\|_2 = \rho(\mathbf{A}'\mathbf{A}) \leq \|\mathbf{A}'\mathbf{A}\|_1 \leq \|\mathbf{A}'\|_1 \|\mathbf{A}\|_1 = \|\mathbf{A}\|_\infty \|\mathbf{A}\|_1.$$

Thus, choosing $\mu = \frac{1}{\|\mathbf{A}\|_\infty \|\mathbf{A}\|_1}$ is a valid step size that ensures GD converges (cf. `lsgd` and `lsngd`).

It is much easier to compute $\|\mathbf{A}\|_\infty$ and $\|\mathbf{A}\|_1$ than $\|\mathbf{A}\|_2$.

5.2 Convergence of sequences of vectors and matrices

In later chapters we will be discussing iterative optimization algorithms and analyzing when such algorithms converge. This is another topic involving **vector norms** and **matrix norms**.

Convergence of a sequence of numbers

Define. We say a sequence of (possibly complex) numbers $\{x_k\}$ **converges** to a limit x_* iff $|x_k - x_*| \rightarrow 0$ as $k \rightarrow \infty$, where $|\cdot|$ denotes absolute value (or complex magnitude more generally). Specifically,

$$\forall \epsilon > 0, \quad \exists N_\epsilon \in \mathbb{N} \text{ s.t. } |x_k - x_*| < \epsilon \quad \forall k \geq N_\epsilon$$

We now define convergence of a sequence of vectors or matrices by using a **norm** to quantify distance, relating to convergence of a sequence of scalars.

Define. We say a sequence of vectors $\{x_k\}$ in a vector space \mathcal{V} **converges** to a limit $x_* \in \mathcal{V}$ iff $\|x_k - x_*\| \rightarrow 0$ for some norm $\|\cdot\|$ as $k \rightarrow \infty$. Specifically,

$$\forall \epsilon > 0, \quad \exists N_\epsilon \in \mathbb{N} \text{ s.t. } \|x_k - x_*\| < \epsilon \quad \forall k \geq N_\epsilon$$

A matrix is simply a point in a vector space of matrices so we use essentially the same definition of convergence of a sequence of matrices:

Define. We say a sequence of matrices $\{\mathbf{X}_k\}$ (in a vector space \mathcal{V} of matrices) **converges** to a limit $\mathbf{X}_* \in \mathcal{V}$ iff $\|\mathbf{X}_k - \mathbf{X}_*\| \rightarrow 0$ for some (matrix) norm $\|\cdot\|$ as $k \rightarrow \infty$. Specifically,

$$\forall \epsilon > 0, \quad \exists N_\epsilon \in \mathbb{N} \text{ s.t. } \|\mathbf{X}_k - \mathbf{X}_*\| < \epsilon \quad \forall k \geq N_\epsilon$$

Example. Consider (for simplicity) the sequence of **diagonal** matrices $\{\mathbf{D}_k\}$ defined by

$$\mathbf{D}_k = \begin{bmatrix} 3 + 2^{-k} & 0 \\ 0 & (-1)^k/k^2 \end{bmatrix}.$$

This sequence converges to the limit $\mathbf{D}_* = \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix}$ because

$$\|\mathbf{D}_k - \mathbf{D}_*\|_{\text{F}} = \left\| \begin{bmatrix} 2^{-k} & 0 \\ 0 & (-1)^k/k^2 \end{bmatrix} \right\|_{\text{F}} = \sqrt{4^{-k} + 1/k^4} \rightarrow 0.$$

5.3 Generalized inverse of a matrix

The **Moore-Penrose pseudoinverse** defined on p. 4.19 is just one (particularly important) type of **generalized inverse** of a matrix. This section uses the Frobenius norm to characterize the **Moore-Penrose pseudoinverse**.

Define. A matrix $\mathbf{G} \in \mathbb{F}^{N \times M}$ is a **generalized inverse** of a matrix $\mathbf{A} \in \mathbb{F}^{M \times N}$ iff $\mathbf{AGA} = \mathbf{A}$.

- If \mathbf{A} has full column rank, then $\mathbf{A}'\mathbf{A}$ is invertible, so multiplying both sides of $\mathbf{AGA} = \mathbf{A}$ on the left by $\mathbf{A}^+ = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}'$ yields that \mathbf{G} is a generalized inverse of such an \mathbf{A} iff $\mathbf{GA} = \mathbf{I}_N$, i.e., iff \mathbf{G} is a **left inverse** of \mathbf{A} .
- Conversely, if \mathbf{A} has full row rank, then \mathbf{AA}' is invertible and \mathbf{G} is a generalized inverse of such an \mathbf{A} iff $\mathbf{AG} = \mathbf{I}_M$, i.e., iff \mathbf{G} is a **right inverse** of \mathbf{A} .

Considering an **SVD** $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}'$, one can verify from the definition that every generalized inverse of \mathbf{A} has the form

$$\mathbf{G} = \mathbf{V} \begin{bmatrix} \Sigma_r^{-1} & \mathbf{S}_2 \\ \mathbf{S}_3 & \mathbf{S}_4 \end{bmatrix} \mathbf{U}',$$

where the matrices $\mathbf{S}_2 \in \mathbb{F}^{r \times ?}$, $\mathbf{S}_3 \in \mathbb{F}^{? \times r}$, and $\mathbf{S}_4 \in \mathbb{F}^{? \times ?}$ have certain sizes (left as an Exercise for the reader) but otherwise have completely arbitrary values. In other words, the (very general!) set of generalized inverses $\mathcal{G}_{\mathbf{A}}$ of a $M \times N$ \mathbf{A} is a **linear variety** in the vector space of $N \times M$ matrices.

One can devise many ways to choose a specific generalized inverse from the set $\mathcal{G}_{\mathbf{A}}$.

Minimum Frobenius norm generalized inverse

A simple way is to choose the generalized inverse having the smallest **Frobenius norm**. This solution turns out to be simply the pseudo-inverse of \mathbf{A} :

$$\arg \min_{\mathbf{G} \in \mathcal{G}_{\mathbf{A}}} \|\mathbf{G}\|_{\text{F}} = \mathbf{A}^+.$$

Proof: $\mathbf{G} \in \mathcal{G}_{\mathbf{A}} \implies \mathbf{G} = \mathbf{V} \begin{bmatrix} \Sigma_r^{-1} & \mathbf{S}_2 \\ \mathbf{S}_3 & \mathbf{S}_4 \end{bmatrix} \mathbf{U}' \implies \|\mathbf{G}\|_{\text{F}} = \left\| \begin{bmatrix} \Sigma_r^{-1} & \mathbf{S}_2 \\ \mathbf{S}_3 & \mathbf{S}_4 \end{bmatrix} \right\|_{\text{F}}$ because the Frobenius norm is **unitarily invariant**. Because $\left\| \begin{bmatrix} \Sigma_r^{-1} & \mathbf{S}_2 \\ \mathbf{S}_3 & \mathbf{S}_4 \end{bmatrix} \right\|_{\text{F}}^2 = \|\Sigma_r^{-1}\|_{\text{F}}^2 + \|\mathbf{S}_2\|_{\text{F}}^2 + \|\mathbf{S}_3\|_{\text{F}}^2 + \|\mathbf{S}_4\|_{\text{F}}^2$, the minimum Frobenius norm solution is when each \mathbf{S}_i is all zeros.

Thus that solution has the form $\mathbf{G} = \mathbf{V} \begin{bmatrix} \Sigma_r^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{U}' = \mathbf{V} \Sigma^+ \mathbf{U}' = \mathbf{A}^+$.

In words, the Moore-Penrose pseudo-inverse of \mathbf{A} is the unique generalized inverse of \mathbf{A} with minimal Frobenius norm.

See [11] for other choices.

5.4 Procrustes analysis

(A practical application of the **SVD** and the **Frobenius matrix norm**)

One use of matrix norms is quantifying the dissimilarity of two matrices by using a norm of their difference. We illustrate that use by solving the **orthogonal Procrustes problem** [12].

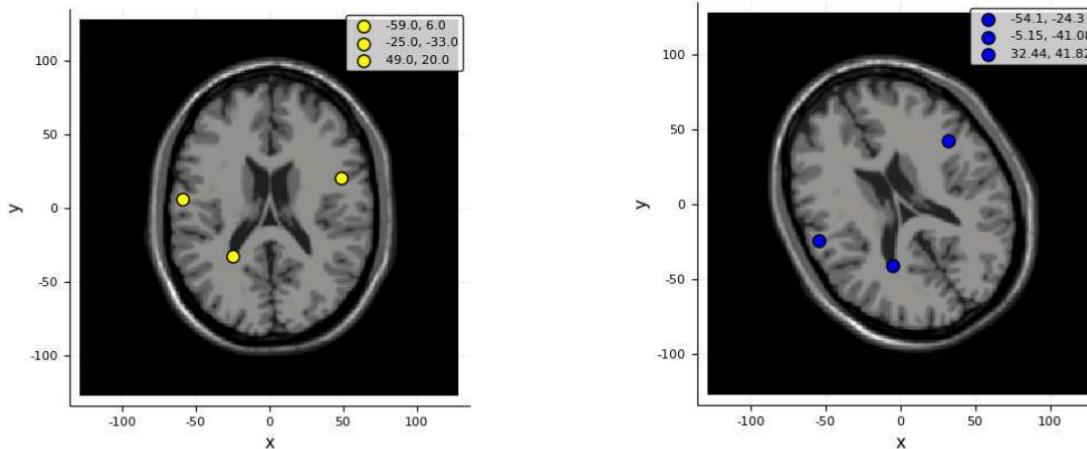
The goal of the **Procrustes problem** is to find an orthogonal matrix \mathbf{Q} in $\mathbb{R}^{M \times M}$ that makes two other matrices \mathbf{B} and \mathbf{A} in $\mathbb{R}^{M \times N}$ as similar as possible by “rotating” the columns of \mathbf{A} :

$$\hat{\mathbf{Q}} = \arg \min_{\mathbf{Q}: \mathbf{Q}'\mathbf{Q} = \mathbf{I}_M} f(\mathbf{Q}), \quad \underbrace{f(\mathbf{Q})}_{\hookrightarrow \text{cost function}} \triangleq \|\mathbf{B} - \mathbf{Q}\mathbf{A}\|_F^2. \quad (5.23)$$

- One could use some other norm but the Frobenius is simple and natural here. (Think about why!)
- I put “rotating” in quotes because the condition $\mathbf{Q}'\mathbf{Q} = \mathbf{I}$ ensures that \mathbf{Q} has orthonormal columns, but the class of matrices for which $\mathbf{Q}'\mathbf{Q} = \mathbf{I}$ also includes examples like $\mathbf{Q} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ that are not rotations.
- There are extensions that require $\det\{\mathbf{Q}\} = 1$ to ensure \mathbf{Q} corresponds to a rotation [13].
- See p. 5.39 for several generalizations (non-square, complex, translation).



One of many motivating applications is performing **image registration** of two pictures of the same scene acquired with different camera orientations, using a technique called **landmark registration**.



Example. Here the goal is to match (by rotation) two sets of landmark coordinates:

$$\mathbf{A} = \begin{bmatrix} -59 & -25 & 49 \\ 6 & -33 & 20 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} -54.1 & -5.15 & 32.44 \\ -24.3 & -41.08 & 41.82 \end{bmatrix} \approx \mathbf{Q}\mathbf{A} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \mathbf{A}.$$

Here $M = 2$ and $N = 3$ and typically $M < N$ in such problems.

Here we found the landmarks manually, but there are also automatic methods [14].

Analyze the cost function:

$$\begin{aligned}
 f(\mathbf{Q}) &= \| \mathbf{B} - \mathbf{Q}\mathbf{A} \|_{\text{F}}^2 = \text{trace}\{(\mathbf{B} - \mathbf{Q}\mathbf{A})'(\mathbf{B} - \mathbf{Q}\mathbf{A})\} \\
 &= \text{trace}\{\mathbf{B}'\mathbf{B} - \mathbf{B}'\mathbf{Q}\mathbf{A} - \mathbf{A}'\mathbf{Q}'\mathbf{B} + \mathbf{A}'\mathbf{Q}'\mathbf{Q}\mathbf{A}\} \\
 &= \text{trace}\{\mathbf{B}'\mathbf{B} - \mathbf{B}'\mathbf{Q}\mathbf{A} - \mathbf{A}'\mathbf{Q}'\mathbf{B} + \mathbf{A}'\mathbf{A}\} \\
 &= \text{trace}\{\mathbf{B}'\mathbf{B}\} + \text{trace}\{\mathbf{A}'\mathbf{A}\} - \text{trace}\{\mathbf{A}'\mathbf{Q}'\mathbf{B}\} - \text{trace}\{\mathbf{B}'\mathbf{Q}\mathbf{A}\} \\
 &= \text{trace}\{\mathbf{B}'\mathbf{B}\} + \text{trace}\{\mathbf{A}'\mathbf{A}\} - \text{trace}\{\mathbf{A}'\mathbf{Q}'\mathbf{B}\} - \text{trace}\{(\mathbf{A}'\mathbf{Q}'\mathbf{B})'\} \\
 &= \text{trace}\{\mathbf{B}'\mathbf{B}\} + \text{trace}\{\mathbf{A}'\mathbf{A}\} - 2\text{trace}\{\mathbf{A}'\mathbf{Q}'\mathbf{B}\} \\
 &= \text{trace}\{\mathbf{B}'\mathbf{B}\} + \text{trace}\{\mathbf{A}'\mathbf{A}\} - 2\text{trace}\{\mathbf{Q}'\mathbf{B}\mathbf{A}'\}
 \end{aligned}$$

expanding via **FOIL**
 $\mathbf{Q}'\mathbf{Q} = \mathbf{I}$
linearity
transpose
transpose inv.
comm. prop. of trace.

So *minimizing* $f(\mathbf{Q})$ is equivalent to *maximizing*

$$g(\mathbf{Q}) \triangleq \text{trace}\{\mathbf{Q}'\mathbf{B}\mathbf{A}'\}.$$

Use an **SVD** (of course!) of the $M \times M$ matrix $\mathbf{C} \triangleq \mathbf{B}\mathbf{A}' = \mathbf{U}\Sigma\mathbf{V}'$ so

$$\begin{aligned}
 g(\mathbf{Q}) &= \text{trace}\{\mathbf{Q}'\mathbf{B}\mathbf{A}'\} = \text{trace}\{\mathbf{Q}'\mathbf{U}\Sigma\mathbf{V}'\} = \text{trace}\{\mathbf{V}'\mathbf{Q}'\mathbf{U}\Sigma\} \\
 &= \text{trace}\{\mathbf{W}\Sigma\}, \quad \mathbf{W} = \mathbf{W}(\mathbf{Q}) \triangleq \mathbf{V}'\mathbf{Q}'\mathbf{U}.
 \end{aligned}$$

Using the orthogonality of \mathbf{U} , \mathbf{V} and \mathbf{Q} , it is clear that the $M \times M$ matrix \mathbf{W} is orthogonal (*cf.* HW):

$$\mathbf{W}'\mathbf{W} = \mathbf{U}'\mathbf{Q}\mathbf{V}\mathbf{V}'\mathbf{Q}'\mathbf{U} = \mathbf{U}'\mathbf{Q}\mathbf{I}_M\mathbf{Q}'\mathbf{U} = \mathbf{U}'\mathbf{U} = \mathbf{I}_M.$$

We must maximize $\text{trace}\{\mathbf{W}\Sigma\}$ over \mathbf{Q} orthogonal, where \mathbf{W} depends on \mathbf{Q} but Σ does not. Observe:

$$[\mathbf{W}\Sigma]_{mm} = w_{mm}\sigma_m \implies \text{trace}\{\mathbf{W}\Sigma\} = \sum_{m=1}^M w_{mm}\sigma_m.$$

To proceed, we look for an upper bound for this sum. Because \mathbf{W} is an orthogonal matrix, each of its columns have unit norm, *i.e.*, $\sum_{m=1}^N |w_{mn}|^2 = 1$ for all m , so $w_{mn} \leq 1$ for all m, n . This inequality yields the following upper bound:

$$\text{trace}\{\mathbf{W}\Sigma\} \leq \sum_{m=1}^M \sigma_m = \text{trace}\{\mathbf{I}\Sigma\}.$$

This upper bound is achieved when $\mathbf{W} = \mathbf{I}$. Now solve for \mathbf{Q} :

$$\mathbf{W} = \mathbf{V}'\mathbf{Q}'\mathbf{U} = \mathbf{I} \implies \mathbf{V}\mathbf{V}'\mathbf{Q}'\mathbf{U}\mathbf{U}' = \mathbf{V}\mathbf{U}' \implies \mathbf{Q}' = \mathbf{V}\mathbf{U}' \implies \hat{\mathbf{Q}} = \mathbf{U}\mathbf{V}'.$$

In summary, the solution to the **orthogonal Procrustes problem** is:

$$\hat{\mathbf{Q}} = \arg \min_{\mathbf{Q}: \mathbf{Q}'\mathbf{Q}=\mathbf{I}} \|\mathbf{B} - \mathbf{Q}\mathbf{A}\|_F^2 = \mathbf{U}\mathbf{V}', \text{ where } \mathbf{C} = \mathbf{B}\mathbf{A}' = \mathbf{U}\Sigma\mathbf{V}'. \quad (5.24)$$

A homework problem will express $\mathbf{C} = \mathbf{Q}\mathbf{P}$ where \mathbf{P} is positive semi-definite, using a **polar decomposition** or **polar factorization** of the square matrix $\mathbf{B}\mathbf{A}'$ [1, p. 41].

The solution to the **orthogonal Procrustes problem** is unique. (?)

A: True

B: False

??

?? See [15] for further discussion.

Sanity check (self consistency and scale invariance)

Suppose \mathbf{B} is exactly a rotated version of the columns of \mathbf{A} , along with an additional scale factor *i.e.*, $\mathbf{B} = \alpha \tilde{\mathbf{Q}} \mathbf{A}$ for some orthogonal matrix $\tilde{\mathbf{Q}}$; equivalently $\mathbf{A} = \frac{1}{\alpha} \tilde{\mathbf{Q}}' \mathbf{B}$. We now verify that the Procrustes method finds the correct rotation, *i.e.*, $\hat{\mathbf{Q}} = \tilde{\mathbf{Q}}$.

Let $\mathbf{B} = \tilde{\mathbf{U}} \tilde{\Sigma} \tilde{\mathbf{V}}'$ denote an SVD of \mathbf{B} . Then an SVD of \mathbf{C} is evident by inspection:

$$\mathbf{C} = \mathbf{B} \mathbf{A}' = \frac{1}{\alpha} \mathbf{B} \mathbf{B}' \tilde{\mathbf{Q}} = \frac{1}{\alpha} \underbrace{\tilde{\mathbf{U}} \tilde{\Sigma} \tilde{\mathbf{V}}'}_{\mathbf{B}} \underbrace{\tilde{\mathbf{V}} \tilde{\Sigma}' \tilde{\mathbf{U}}'}_{\mathbf{B}'} \tilde{\mathbf{Q}} = \underbrace{\tilde{\mathbf{U}}}_{\mathbf{U}} \underbrace{\frac{1}{\alpha} \tilde{\Sigma} \tilde{\Sigma}'}_{\tilde{\Sigma}} \underbrace{\tilde{\mathbf{U}}' \tilde{\mathbf{Q}}}_{\mathbf{V}'}$$

The Procrustes solution is indeed correct (self consistent), and **invariant** to the scale parameter α :

$$\hat{\mathbf{Q}} = \mathbf{U} \mathbf{V}' = (\tilde{\mathbf{U}})(\tilde{\mathbf{U}}' \tilde{\mathbf{Q}}) = \tilde{\mathbf{Q}}.$$

After finding $\hat{\mathbf{Q}}$, if we also want to estimate the scale, then we can solve a **linear least-squares** problem:

$$\arg \min_{\alpha} \left\| \mathbf{B} - \alpha \hat{\mathbf{Q}} \mathbf{A} \right\|_{\text{F}} = \frac{\text{trace}\left\{ \mathbf{B} \mathbf{A}' \hat{\mathbf{Q}}' \right\}}{\text{trace}\left\{ \mathbf{A} \mathbf{A}' \right\}} = \frac{\text{trace}\left\{ \mathbf{U} \Sigma \mathbf{V}' \mathbf{V} \mathbf{U}' \right\}}{\text{trace}\left\{ \mathbf{A} \mathbf{A}' \right\}} = \frac{\sum_{k=1}^r \sigma_k}{\left\| \mathbf{A} \right\|_{\text{F}}^2}, \quad (5.25)$$

where $\{\sigma_k\}$ are the singular values of $\mathbf{C} = \mathbf{B} \mathbf{A}'$

A HW problem will explore a real-world image registration example.

The next page provides a small concrete example.

Example. For determining 2D image rotation, even a single nonzero point in each image will suffice! For example, suppose the first point is at $(1, 0)$ and the second point is at (x, y) where $x = 5 \cos \phi$ and $y = 5 \sin \phi$. (This example includes scaling by a factor of 5 just to illustrate the generality.) Then $\mathbf{A} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} x \\ y \end{bmatrix}$ so

$$\mathbf{C} = \mathbf{BA}' = \begin{bmatrix} 5 \cos \phi \\ 5 \sin \phi \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \underbrace{\begin{bmatrix} \cos \phi & -q_1 \sin \phi \\ \sin \phi & q_1 \cos \phi \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & q_2 \end{bmatrix}}_{\mathbf{V}'}, \quad q_1, q_2 \in \{\pm 1\}.$$

Here \mathbf{C} is a simple outer product so finding a (full!) SVD by hand was easy.

In fact we found four SVDs, corresponding to different signs for \mathbf{u}_2 and \mathbf{v}_2 .

For each of these SVDs, the optimal rotation matrix per (5.24) is

$$\hat{\mathbf{Q}} = \mathbf{UV}' = \begin{bmatrix} \cos \phi & -q_1 \sin \phi \\ \sin \phi & q_1 \cos \phi \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & q_2 \end{bmatrix} = \begin{bmatrix} \cos \phi & -q \sin \phi \\ \sin \phi & q \cos \phi \end{bmatrix},$$

where $q \triangleq q_1 q_2 \in \{\pm 1\}$. The two Procrustes solutions here (for $q = \pm 1$) both have the correct $\cos \phi$ in the upper left and both exactly satisfy $\mathbf{B} = 5\mathbf{QA}$.

So there are two Procrustes solutions that fit the data exactly, one of which (for $q = 1$) corresponds to a rotation matrix, and the other of which (for $q = -1$) has sign flip for the second coordinate.

In 2D, any rotation matrix is a unitary matrix, but the converse is not true!

Exercise. Explore what happens with two points: colinear, symmetric around zero, non-colinear.

Generalizations: non-square, complex, with translation

(Read)

This section generalizes the Procrustes problem (5.23) in three ways: we consider complex data, we account for a possible translation, and we allow \mathbf{Q} to be non-square, meaning that \mathbf{B} and \mathbf{A} can have different numbers of rows.

Here we assume $\mathbf{B} \in \mathbb{F}^{M \times N}$ but $\mathbf{A} \in \mathbb{F}^{K \times N}$ so $\mathbf{Q} \in \mathbb{F}^{M \times K}$.

We still want \mathbf{Q} to have orthonormal columns, so we must have $1 \leq K \leq M$.

Define. The **Stiefel manifold** $\mathcal{V}_K(\mathbb{F}^M)$ is the set of $M \times K$ matrices having orthonormal columns

$$\mathcal{V}_K(\mathbb{F}^M) = \{\mathbf{Q} \in \mathbb{F}^{M \times K} : \mathbf{Q}'\mathbf{Q} = \mathbf{I}_K\}.$$

Special cases:

$\mathcal{V}_M(\mathbb{R}^M)$ is the set of $M \times M$ orthogonal matrices

$\mathcal{V}_M(\mathbb{C}^M)$ is the set of $M \times M$ unitary matrices

If $\mathbf{Q} \in \mathcal{V}_K(\mathbb{C}^M)$ then \mathbf{Q} is the first K columns of some $M \times M$ unitary matrix.

In many practical applications of the **Procrustes problem**, there can be both rotation and an unknown **translation** between the two sets of coordinates. Instead of the model $\mathbf{B}_{:,n} \approx \mathbf{Q}\mathbf{A}_{:,n}$ a more realistic model is $\mathbf{B}_{:,n} \approx \mathbf{Q}\mathbf{A}_{:,n} + \mathbf{d}$ where $\mathbf{d} \in \mathbb{F}^M$ is an unknown **displacement vector**. In matrix form:

$$\mathbf{B} \approx \mathbf{Q}\mathbf{A} + \mathbf{d}\mathbf{1}'_N.$$

Now we must determine both a matrix $\mathbf{Q} \in \mathcal{V}_K(\mathbb{F}^M)$ in the Stiefel manifold, and the vector $\mathbf{d} \in \mathbb{C}^M$ by a double minimization using a Frobenius norm:

$$(\hat{\mathbf{Q}}, \hat{\mathbf{d}}) \triangleq \arg \min_{\mathbf{Q} \in \mathcal{V}_K(\mathbb{F}^M)} \arg \min_{\mathbf{d} \in \mathbb{F}^M} g(\mathbf{d}, \mathbf{Q}), \quad g(\mathbf{d}, \mathbf{Q}) \triangleq \| \mathbf{B} - (\mathbf{Q}\mathbf{A} + \mathbf{d}\mathbf{1}'_N) \|_{\text{F}}^2.$$

We first focus on the inner minimization over the displacement \mathbf{d} for any given \mathbf{Q} :

$$\begin{aligned} g(\mathbf{d}, \mathbf{Q}) &= \| \mathbf{B} - (\mathbf{Q}\mathbf{A} + \mathbf{d}\mathbf{1}'_N) \|_{\text{F}}^2 = \text{trace}\{(\mathbf{Z} - \mathbf{d}\mathbf{1}'_N)'(\mathbf{Z}\mathbf{A} - \mathbf{d}\mathbf{1}'_N)\}, \quad \mathbf{Z} \triangleq \mathbf{B} - \mathbf{Q}\mathbf{A} \\ &= \text{trace}\{\mathbf{Z}'\mathbf{Z}\} - \text{trace}\{\mathbf{Z}'\mathbf{d}\mathbf{1}'_N\} - \text{trace}\{\mathbf{1}_N\mathbf{d}'\mathbf{Z}\} + \text{trace}\{\mathbf{1}_N\mathbf{d}'\mathbf{d}\mathbf{1}'_N\} \\ &= \text{trace}\{\mathbf{Z}'\mathbf{Z}\} - \text{trace}\{\mathbf{1}'_N\mathbf{Z}'\mathbf{d}\} - \text{trace}\{\mathbf{d}'\mathbf{Z}\mathbf{1}_N\} + \text{trace}\{\mathbf{d}'\mathbf{d}\mathbf{1}'_N\mathbf{1}_N\} \\ &= \text{trace}\{\mathbf{Z}'\mathbf{Z}\} - \mathbf{1}'_N\mathbf{Z}'\mathbf{d} - \mathbf{d}'\mathbf{Z}\mathbf{1}_N + N\mathbf{d}'\mathbf{d} \\ &= \text{trace}\{\mathbf{Z}'\mathbf{Z}\} - \frac{1}{N} \|\mathbf{Z}\mathbf{1}_N\|_2^2 + N \left\| \mathbf{d} - \frac{1}{N} \mathbf{Z}\mathbf{1}_N \right\|_2^2. \end{aligned}$$

It is clear from this expression that the optimal estimate of the displacement \mathbf{d} for any \mathbf{Q} is:

$$\hat{\mathbf{d}}(\mathbf{Q}) = \frac{1}{N} \mathbf{Z}\mathbf{1}_N = \frac{1}{N} (\mathbf{B} - \mathbf{Q}\mathbf{A})\mathbf{1}_N.$$

Now to find the optimal matrix \mathbf{Q} we must solve the outer minimization:

$$\begin{aligned}
\hat{\mathbf{Q}} &\triangleq \arg \min_{\mathbf{Q} \in \mathcal{V}_K(\mathbb{F}^M)} f(\mathbf{Q}), \quad f(\mathbf{Q}) \triangleq g\left(\hat{\mathbf{d}}(\mathbf{Q}), \mathbf{Q}\right) \\
f(\mathbf{Q}) &= \text{trace}\{(\mathbf{B} - \mathbf{Q}\mathbf{A})'(\mathbf{B} - \mathbf{Q}\mathbf{A})\} - \frac{1}{N} \|(\mathbf{B} - \mathbf{Q}\mathbf{A})\mathbf{1}_N\|_2^2 \\
&\stackrel{c}{=} -2 \text{real}\{\text{trace}\{\mathbf{Q}'\mathbf{B}\mathbf{A}'\}\} + \frac{2}{N} \text{real}\{\mathbf{1}'_N \mathbf{A}' \mathbf{Q}' \mathbf{B} \mathbf{1}_N\} \\
&= -2 \text{real}\{\text{trace}\{\mathbf{Q}'\mathbf{B}\mathbf{A}'\}\} + 2 \text{real}\left\{\text{trace}\left\{\mathbf{Q}'\mathbf{B} \frac{1}{N} \mathbf{1}_N \mathbf{1}'_N \mathbf{A}'\right\}\right\} \\
&= -2 \text{real}\left\{\text{trace}\left\{\mathbf{Q}'\tilde{\mathbf{C}}\right\}\right\}, \quad \tilde{\mathbf{C}} \triangleq \underbrace{\mathbf{B}\mathbf{M}\mathbf{A}'}_{M \times K}, \quad \mathbf{M} \triangleq \mathbf{I} - \frac{1}{N} \mathbf{1}_N \mathbf{1}'_N,
\end{aligned}$$

where $\stackrel{c}{=}$ means “equal to within constant terms that are irrelevant for minimization.”

After finding a (full) **SVD** $\tilde{\mathbf{C}} = \underbrace{\mathbf{U}}_{M \times M} \underbrace{\Sigma}_{M \times K} \underbrace{\mathbf{V}'}_{K \times K}$, we want:

$$\hat{\mathbf{Q}} = \arg \max_{\mathbf{Q} \in \mathcal{V}_K(\mathbb{F}^M)} \text{real}\left\{\text{trace}\left\{\mathbf{Q}'\tilde{\mathbf{C}}\right\}\right\}$$

where using the Frobenius inner product inequality (5.12):

$$\begin{aligned}
\text{real}\left\{\text{trace}\left\{\mathbf{Q}'\tilde{\mathbf{C}}\right\}\right\} &= \text{real}\{\text{trace}\{\mathbf{Q}'\mathbf{U}\Sigma\mathbf{V}'\}\} = \text{real}\{\text{trace}\{\mathbf{W}'\Sigma\}\}, \quad \text{where } \mathbf{W} \triangleq \mathbf{U}'\mathbf{Q}\mathbf{V} \in \mathcal{V}_K(\mathbb{F}^M) \\
&= \text{real}\{\langle \Sigma, \mathbf{W} \rangle\} \leq |\langle \Sigma, \mathbf{W} \rangle| \leq \|\text{vec}(\Sigma)\|_1 \|\text{vec}(\mathbf{W})\|_\infty = \|\Sigma\|_* \|\text{vec}(\mathbf{W})\|_\infty.
\end{aligned}$$

Because $\Sigma = \begin{bmatrix} \Sigma_K \\ \mathbf{0}_{(M-K) \times K} \end{bmatrix}$ is rectangular diagonal, the matrix $\mathbf{W} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{0}_{(M-K) \times K} \end{bmatrix} \in \mathcal{V}_K(\mathbb{F}^M)$ achieves the upper bound. Solving for \mathbf{Q} yields

$$\hat{\mathbf{Q}} = \mathbf{U} \mathbf{W} \mathbf{V}' = \mathbf{U} \begin{bmatrix} \mathbf{I}_K \\ \mathbf{0}_{(M-K) \times K} \end{bmatrix} \mathbf{V}' = \mathbf{U}_K \mathbf{V}',$$

where \mathbf{U}_K denotes the first K columns of the $M \times M$ matrix \mathbf{U} .

In summary, the optimal \mathbf{Q} is

$$\mathbf{Q} = \mathbf{U}_K \mathbf{V}', \text{ where } \tilde{\mathbf{C}} \triangleq \mathbf{B} \mathbf{M} \mathbf{A}' = \mathbf{U} \Sigma \mathbf{V}'.$$

The matrix \mathbf{M} is called a “de-meaning” or “centering” operator because $\mathbf{y} = \mathbf{M}\mathbf{x}$ subtracts the mean of \mathbf{x} from each element of \mathbf{x} . In code: `y = x .- mean(x)`

The de-meaning matrix \mathbf{M} is a symmetric **idempotent matrix** so $\mathbf{M} = \mathbf{M}\mathbf{M}'$ and we can rewrite $\tilde{\mathbf{C}}$ above as $\tilde{\mathbf{C}} = (\mathbf{B}\mathbf{M})(\mathbf{A}\mathbf{M})' = \tilde{\mathbf{B}}\tilde{\mathbf{A}}'$ where $\tilde{\mathbf{A}} \triangleq \mathbf{A}\mathbf{M}$, $\tilde{\mathbf{B}} \triangleq \mathbf{B}\mathbf{M}$ are versions of \mathbf{A} and \mathbf{B} where each column has its mean subtracted out.

In words, to find the optimal rotation matrix when there is possible translation, we first de-mean each column of \mathbf{A} and \mathbf{B} , and then compute the usual SVD of $\tilde{\mathbf{B}}\tilde{\mathbf{A}}'$ and use the left and right bases via $\mathbf{Q} = \mathbf{U}_K \mathbf{V}'$.

Exercise. Do a sanity check in the case where $\mathbf{B} = \alpha \mathbf{Q} \mathbf{A} + d\mathbf{1}_N'$.

Subspace / span comparisons

(Read)

Another application of the **orthogonal Procrustes problem** is quantifying the “alignment” between two subspace bases.

Suppose \mathbf{B}_1 and \mathbf{B}_2 are $M \times N$ matrices that we think span the same (or similar) subspace in \mathbb{F}^M . In general it does not make sense to use $d(\mathbf{B}_1, \mathbf{B}_2) = \|\mathbf{B}_1 - \mathbf{B}_2\|_F$ as a measure of dissimilarity because we could have $\mathcal{R}(\mathbf{B}_1) = \mathcal{R}(\mathbf{B}_2)$ even if \mathbf{B}_1 and \mathbf{B}_2 are themselves different, *e.g.*, if $\mathbf{B}_1 = -\mathbf{B}_2$.

A more useful measure of dissimilarity involves first rotating the basis for one subspace to be as similar to the other as possible, and then examining the difference, *i.e.*:

$$d(\mathbf{B}_1, \mathbf{B}_2) \triangleq \min_{\mathbf{Q} \in \mathcal{V}_N(\mathbb{F}^N)} \|\mathbf{B}_1 - \mathbf{B}_2 \mathbf{Q}'\|_F = \min_{\mathbf{Q} \in \mathcal{V}_N(\mathbb{F}^N)} \|\mathbf{B}'_1 - \mathbf{Q} \mathbf{B}'_2\|_F.$$

The best \mathbf{Q} is $\hat{\mathbf{Q}} = \mathbf{U}\mathbf{V}'$ where $\mathbf{C} = \mathbf{B}'_1 \mathbf{B}_2 = \mathbf{U}\Sigma\mathbf{V}'$, so the simplified dissimilarity measure is

$$d(\mathbf{B}_1, \mathbf{B}_2) = \|\mathbf{B}_1 - \mathbf{B}_2 \mathbf{V}\mathbf{U}'\|_F.$$

If the \mathbf{B} matrices are not in the **Stiefel manifold**, then one should include a scale factor like (5.25).

Practical implementation

The solution to the Procrustes problem requires just a couple JULIA statements. The key ingredient is simply the `svd` command. See the example notebook:

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/05_procrustes1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/05_procrustes1.ipynb

5.5 Summary

- We use **vector norms** and **matrix norms** are used to measure sizes and distances.
- Some **matrix norms** are essentially just vector norms in terms of $\text{vec}(\mathbf{A})$, some matrix norms satisfy the important **sub-multiplicative** property, and **operator norms** are induced by vector norms.
- Many of the matrix norms can be expressed in terms of singular values, and those are **unitarily invariant**.
- Classical methods (like linear LS) use 2-norms, but many modern methods use other norms. One vector norm of recent interest is the **ordered weighted ℓ_1 (OWL)** norm [16].
- We assess **convergence** of a sequence of vectors or matrices using norms.
- The **spectral radius** is a related quantity for square matrices, where $\sigma_1(\mathbf{A}) = \sqrt{\sigma_1(\mathbf{A}'\mathbf{A})} = \sqrt{\rho(\mathbf{A}'\mathbf{A})}$.
- The **Moore-Penrose pseudo-inverse** is the **generalized inverse** having minimum **Frobenius norm**.

The **orthogonal Procrustes problem** has an **SVD**-based solution:

$$\hat{\mathbf{Q}} = \arg \min_{\mathbf{Q} : \mathbf{Q}'\mathbf{Q} = \mathbf{I}_M} \|\mathbf{B} - \mathbf{Q}\mathbf{A}\|_{\text{F}}^2 = \mathbf{U}\mathbf{V}', \quad \mathbf{C} = \mathbf{B}\mathbf{A}' = \mathbf{U}\Sigma\mathbf{V}'.$$

- The solution is invariant to scaling factors $\alpha \mathbf{Q}\mathbf{A}$
- Unknown displacement (translation) simply requires de-meaning \mathbf{A} and \mathbf{B} before doing SVD
- Displacement estimate (if needed) is $\frac{1}{N} (\mathbf{B} - \hat{\mathbf{Q}}\mathbf{A}) \mathbf{1}_N$.

Deriving the solution to this problem used *many* of the tools discussed so far: Frobenius norm, matrix trace and its properties, SVD, matrix/vector algebra.

Exercise. Suppose \mathbf{A} and \mathbf{B} are both real $1 \times N$ vectors (each with mean 0 for simplicity). How can we interpret the orthogonal Procrustes solution in this case geometrically?

Hint: What is SVD of \mathbf{BA}' here?

If $\mathbf{A} = \mathbf{x}'$ and $\mathbf{B} = \mathbf{y}'$ where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$, then $\mathbf{BA}' = \mathbf{y}'\mathbf{x} = \underbrace{\mathbf{sgn}(\mathbf{y}'\mathbf{x})}_{U} \underbrace{|\mathbf{y}'\mathbf{x}|}_{\sigma_1} \underbrace{1}_{V}$

so $\mathbf{Q} = \mathbf{UV}' = \mathbf{sgn}(\mathbf{y}'\mathbf{x}) = \pm 1$. Here the “rotation” is just possibly negating the sign to match in 1D.

Exercise. What if $\mathbf{B} = e^{i\phi} \mathbf{A}$?

(in class if possible)

??

Challenge (for much later in the course). The Frobenius norm is not robust to **outlier** data. Using something like an ℓ_1 norm instead would provide better robustness [17].

Bibliography

- [1] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005.
- [2] D. G. Luenberger. *Optimization by vector space methods*. New York: Wiley, 1969.
- [3] R. H. Chan and M. K. Ng. “Conjugate gradient methods for Toeplitz systems”. In: *SIAM Review* 38.3 (Sept. 1996), 427–82.
- [4] V-S. Chellaboina and W. M. Haddad. “Is the Frobenius matrix norm induced?” In: *IEEE Trans. Auto. Control* 40.12 (Dec. 1995), 2137–9.
- [5] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge: Cambridge Univ. Press, 1985.
- [6] S. F. Cotter, B. D. Rao, K. Engan, and K. Kreutz-Delgado. “Sparse solutions to linear inverse problems with multiple measurement vectors”. In: *IEEE Trans. Sig. Proc.* 53.7 (July 2005), 2477–88.
- [7] J. A. Tropp, A. C. Gilbert, and M. J. Strauss. “Algorithms for simultaneous sparse approximation. Part I: Greedy pursuit”. In: *Signal Processing* 86.3 (Mar. 2006), 572–88.
- [8] S. Feizi and D. Tse. *Maximally correlated principal component analysis*. 2017.
- [9] R. Bhatia. *Matrix analysis*. New York: Springer, 1997.
- [10] R. Pascanu, T. Mikolov, and Y. Bengio. “On the difficulty of training recurrent neural networks”. In: *pmlr* 28.3 (2013), 1310–8.
- [11] I. Dokmanic and Remi Gribonval. *Beyond Moore-Penrose part I: Generalized inverses that minimize matrix norms*. 2017.

- [12] P. H. Schonemann. “A generalized solution of the orthogonal Procrustes problem”. In: *Psychometrika* 31.1 (Mar. 1966), 1–10.
- [13] S. Ahmed and I. M. Jaimoukha. “A relaxation-based approach for the orthogonal Procrustes problem with data uncertainties”. In: *Proc. UKACC Intl. Conf. Control.* 2012, 906–11.
- [14] F. C. Ghesu, B. Georgescu, S. Grbic, A. K. Maier, J. Hornegger, and D. Comaniciu. “Robust multi-scale anatomical landmark detection in incomplete 3D-CT data”. In: *Medical Image Computing and Computer-Assisted Intervention.* 2017, 194–202.
- [15] S. Ravishankar and Y. Bresler. “ l_0 sparsifying transform learning with efficient optimal updates and convergence guarantees”. In: *IEEE Trans. Sig. Proc.* 63.9 (May 2015), 2389–404.
- [16] M. A. T. Figueiredo and R. D. Nowak. “Ordered weighted l_1 regularized regression with strongly correlated covariates: Theoretical aspects”. In: *aistats.* 2016, 930–8.
- [17] P. J. F. Groenen, P. Giaquinto, and H. A. L. Kiers. “An improved majorization algorithm for robust Procrustes analysis”. In: *New Developments in Classification and Data Analysis: Proceedings of the Meeting of the Classification and Data Analysis Group (CLADAG) of the Italian Statistical Society, University of Bologna.* Berlin: Springer, 2005, pp. 151–8.

Chapter 6

Low-rank approximation

Contents (final version)

6.0 Introduction	6.2
6.1 Low-rank approximation via Frobenius norm	6.3
Implementation	6.8
1D example	6.15
Generalization to other norms	6.17
Bases for $\mathbb{F}^{M \times N}$	6.19
Low-rank approximation summary	6.22
Rank and stability	6.23
6.2 Sensor localization application (Multidimensional scaling)	6.24
Practical implementation	6.31
6.3 Alternative low-rank approximation formulations	6.34
6.4 Choosing the rank or regularization parameter	6.44
OptShrink	6.48
6.5 Related methods: autoencoders and PCA	6.53
Relation to autoencoder with linear hidden layer	6.53

Relation to principal component analysis (PCA)	6.55
6.6 Subspace learning	6.57

6.0 Introduction

L§8.1

In many applications, **dimensionality reduction** is important. This chapter focuses on **low-rank approximation** of a matrix. One of many applications is **image compression**, explored in a HW.

Another application is source localization via multidimensional scaling, explored on p. [6.24](#).

Source material for this chapter includes [[1](#), §8.1].

6.1 Low-rank approximation via Frobenius norm

We are given a matrix $\mathbf{A} \in \mathbb{F}^{M \times N}$ (often large), having rank $r \leq \min(M, N)$.

To perform **dimensionality reduction** we want to approximate \mathbf{A} by another matrix $\hat{\mathbf{B}}$ having rank $K \leq r$. Often we pick $K \ll r$.

To find the “best” \mathbf{B} we must define how closely \mathbf{B} approximates \mathbf{A} . The simplest metric is the **Frobenius norm** of the difference. This criterion leads to the follow **low-rank approximation problem**:

$$\hat{\mathbf{B}} \triangleq \arg \min_{\mathbf{B} \in \mathcal{L}_K^{M \times N}} \underbrace{\|\mathbf{B} - \mathbf{A}\|_F}_{\hookrightarrow \text{Frobenius norm}}, \quad \mathcal{L}_K^{M \times N} \triangleq \{\mathbf{B} \in \mathbb{F}^{M \times N} : \text{rank}(\mathbf{B}) \leq K\}. \quad (6.1)$$

This is a **non-convex** problem because the set $\mathcal{L}_K^{M \times N}$ of rank- K matrices is not convex. Remarkably, there is a simple solution based on any **SVD** of \mathbf{A} [2].

Theorem. The best rank (at most) K approximation uses the first (largest) K singular components of \mathbf{A} :

$$\hat{\mathbf{B}} = \sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}'_k, \quad \text{where } \mathbf{A} = \mathbf{U} \Sigma \mathbf{V}' = \sum_{k=1}^{\min(M,N)} \sigma_k \mathbf{u}_k \mathbf{v}'_k. \quad (6.2)$$

The approximation error depends on the singular values of the remaining (discarded) terms:

$$\|\hat{\mathbf{B}} - \mathbf{A}\|_F = \sqrt{\sum_{k=K+1}^{\text{rank}(\mathbf{A})} \sigma_k^2} \leq \|\mathbf{B} - \mathbf{A}\|_F, \quad \forall \mathbf{B} \in \mathcal{L}_K^{M \times N}.$$

- Clearly the approximation error will be small if the remaining singular values are small compared to the first K singular values.
- Once again we see that the SVD is key tool. Here, the SVD is used both to construct the approximation and to quantify the approximation error.
- The original matrix \mathbf{A} has MN values. The approximation $\hat{\mathbf{B}}$ is formed from $K(M + N + 1)$ values. This saves memory if $K \ll \min(M, N)$.

What is the rank of $\sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}'_k$ when $1 \leq K \leq r = \text{rank}(\mathbf{A})$? (Choose best answer.)

- A: Always 1 B: Always r C: Usually r D: Always K E: Usually K

??

Diagonal case: proof sketch

First consider a $M \times N$ (rectangular) diagonal matrix Σ having rank r , with descending diagonal values, where we want to approximate it by a matrix C of rank at most $K \leq r$, i.e., we want to solve:

$$\begin{aligned} \hat{C} &\triangleq \arg \min_{C \in \mathcal{L}_K^{M \times N}} \|C - \Sigma\|_F^2 = \arg \min_{C \in \mathcal{L}_K^{M \times N}} \left\| \begin{bmatrix} c_{11} & \dots & c_{1N} \\ & \vdots & \\ c_{M1} & \dots & c_{MN} \end{bmatrix} - \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_K & & \\ & & & \sigma_{K+1} & \\ & & & & \ddots \\ & & & & & \sigma_r \\ & & & & & 0 \end{bmatrix} \right\|_F^2 \\ &= \arg \min_{C \in \mathcal{L}_K^{M \times N}} \sum_{k=1}^K (c_{kk} - \sigma_k)^2 + \sum_{k=K+1}^r (c_{kk} - \sigma_k)^2 + \sum_{k=r+1}^{\min(M,N)} (c_{kk} - 0)^2 + \sum_{m \neq n} (c_{mn} - 0)^2. \end{aligned}$$

To minimize this quadruple sum, subject to the constraint $\text{rank}(C) \leq K$, intuitively we prefer the last two terms to be zero. And to minimize the first two terms, because $\sigma_1 \geq \sigma_2 \geq \dots$, it seems natural to keep the first K (using $c_{kk} = \sigma_k$) and set the rest to zero, yielding the following solution:

$$\hat{c}_{mn} = \begin{cases} \sigma_m, & m = n \leq K \\ 0, & \text{otherwise} \end{cases} \implies \hat{C} = \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_K & & \\ & & & \mathbf{0}_{r-K \times r-K} & \\ & & & & \mathbf{0}_{M-r \times N-r} \end{bmatrix} = \sum_{k=1}^K \sigma_k \mathbf{e}_k \tilde{\mathbf{e}}'_k, \quad (6.3)$$

where e_k is the k th unit vector in \mathbb{F}^M and \tilde{e}_k is the k th unit vector in \mathbb{F}^N .

The conclusion (6.3) is correct, but the above reasoning is not rigorous. For a rigorous proof, see [wiki] [3].

Proof for general case

Now we assume that (6.3) is correct (it is, though not proven here), and use it to prove the general case.

Denote an SVD of A by $A = U\Sigma V'$. Rewrite any $B \in \mathbb{F}^{M \times N}$ in terms of the U and V bases as follows:

$$B = \underbrace{(UU')}_{I} B \underbrace{(VV')}_{I} = U \underbrace{(U'BV)}_{\hookrightarrow \triangleq C \text{ (not diagonal in general)}} V' = UCV'. \quad (6.4)$$

Because U and V are unitary, $\text{rank}(B) = \text{rank}(C)$, so (6.1) is equivalent to

$$\begin{aligned} \hat{B} &= U\hat{C}V', \quad \hat{C} \triangleq \arg \min_{C \in \mathcal{L}_K^{M \times N}} \left\| \underbrace{UCV'}_B - \underbrace{U\Sigma V'}_A \right\|_F = \arg \min_{C \in \mathcal{L}_K^{M \times N}} \|U(C - \Sigma)V'\|_F \\ &= \arg \min_{C \in \mathcal{L}_K^{M \times N}} \|C - \Sigma\|_F = \sum_{k=1}^K \sigma_k e_k \tilde{e}'_k, \end{aligned}$$

using the result (6.3) for the diagonal case and the fact that the **Frobenius norm** is **unitarily invariant**.

Thus the final best approximation to \mathbf{A} having rank (at most) K is:

$$\hat{\mathbf{B}} = \mathbf{U}\hat{\mathbf{C}}\mathbf{V}' = \sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}'_k, \text{ where } \mathbf{A} = \mathbf{U}\Sigma\mathbf{V}' = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k,$$

because $\mathbf{U}\mathbf{e}_k = \mathbf{u}_k$. The approximation error depends on the remaining singular values:

$$\|\hat{\mathbf{B}} - \mathbf{A}\|_F = \|\mathbf{U}\hat{\mathbf{C}}\mathbf{V}' - \mathbf{U}\Sigma\mathbf{V}'\|_F = \|\hat{\mathbf{C}} - \Sigma\|_F = \sqrt{\sum_{k=K+1}^r \sigma_k^2}.$$

Alternative way to derive the approximation error expression:

(Read)

$$\begin{aligned} \mathbf{A} - \hat{\mathbf{B}} &= \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k - \sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}'_k = \sum_{k=K+1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k \implies \\ \|\hat{\mathbf{B}} - \mathbf{A}\|_F &= \left\| \sum_{k=K+1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k \right\|_F = \left\| \mathbf{U}' \sum_{k=K+1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k \mathbf{V} \right\|_F = \left\| \sum_{k=K+1}^r \sigma_k \mathbf{e}_k \mathbf{e}'_k \right\|_F = \sqrt{\sum_{k=K+1}^r \sigma_k^2}, \end{aligned}$$

which again is related to **Parseval's theorem**.

□

Summary

In words: the best rank (at most) K approximation to a matrix is given by its **truncated SVD** (to K terms).

Implementation

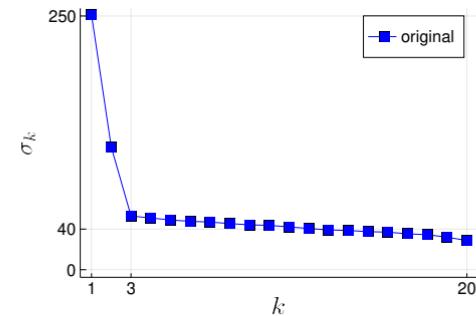
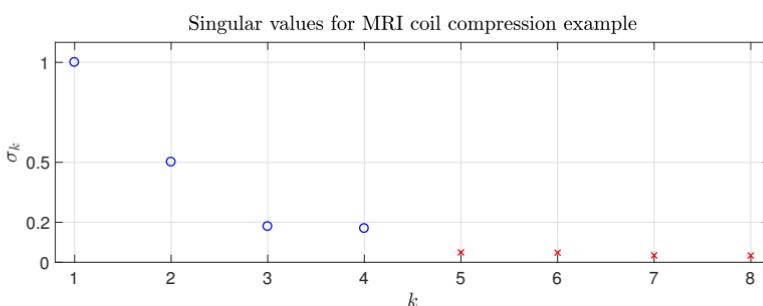
For implementation, we need to choose $K \leq r = \text{rank}(\mathbf{A}) \leq \min(M, N)$.

The approximation error is given by the 2-norm of the excluded singular values ($\sigma_{K+1}, \dots, \sigma_r$), so it is useful to plot all the singular values versus k and look for a “knee in the curve.”

Low-rank approximation is related to **factor analysis**, and a **scree plot** shows the singular values [4].

Example.

(Left figure from MRI example on p. 6.12; right figure from a textbook example with $r = 2$ and noise.)



You hope that your scree plot has a roughly linear section corresponding to the noise.

For moderate sized matrices, low-rank approximation needs just a few lines of JULIA code. The simplest approach uses the sum of outer products solution (6.2) literally:

```
using LinearAlgebra: svd
(U, s, V) = svd(A)
B = zeros(size(A))
for k=1:K
    global B += U[:,k] * s[k] * V[:,k]' # sum of rank-1 outer products
end
```

To avoid a loop one can use matrix multiplication:

```
using LinearAlgebra: svd, Diagonal
(U, s, V) = svd(A) # default full=false saves memory
B = U[:,1:K] * Diagonal(s[1:K]) * V[:,1:K]'
```

Why small `s` above? Reminds me that output is a vector, not a matrix.

Quality code would also include a bounds check that $1 \leq K \leq \min(M, N)$.

Yet another option is to compute only the first K SVD components using `svds`:

```
using LinearAlgebra: Diagonal
using Arpack: svds
tmp = svds(A, nsv=K, ritzvec=true)[1] # special SVD data structure
(U, s, Vt) = (tmp.U, tmp.S, tmp.Vt) # extract needed components
B = U * Diagonal(s) * Vt
```

Why `Vt` above? Avoids an unnecessary transpose operation.

Choosing rank via permutation

Looking for a jump in the **scree plot** seems subjective. A more quantitative approach uses a **permutation method** [5].

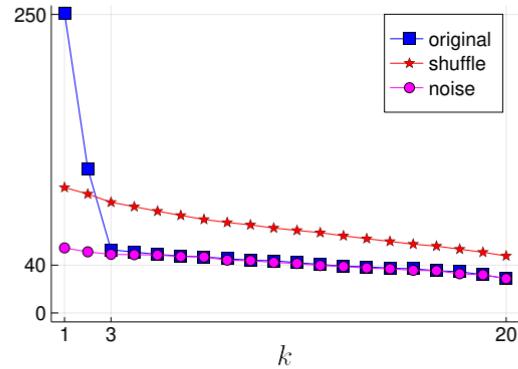
Consider a matrix that is the sum of a low-rank component and IID random noise. If we permute (randomly) each of the columns (or rows?) of that matrix, the noise part will remain the same (statistically) but the low-rank structure will be destroyed. Now we plot the singular values of this shuffled matrix, which will basically correspond to noise. The singular values of the original data that are larger than those of the shuffled data might be considered to be the “significant” ones and can help guide rank choice. See [5] for theoretical analysis.

One might need to permute multiple times and average to get a reliable baseline.

Example. This is from synthetic data that is rank 2 plus noise. The two largest singular values stand out above the shuffled case.

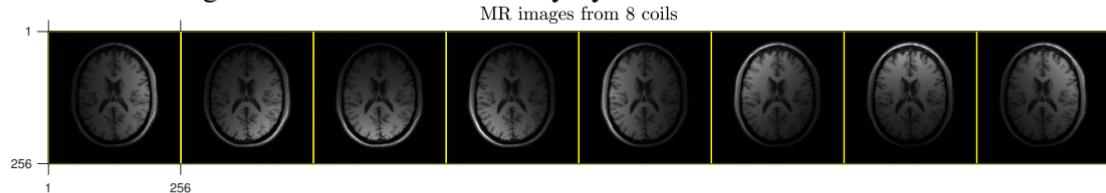
Here is the key code for permuting:

```
using Random: shuffle  
Y = mapslices(shuffle, X, dims=1)
```

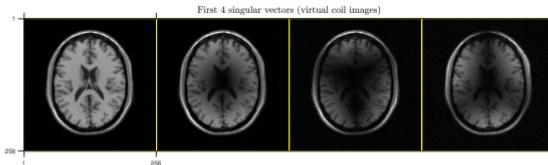


MRI Example

Modern MRI scanners use multiple receive coils to collect more image information at the same time. For example, the fMRI Center on UM's North Campus uses a 32-channel head coil. Processing all 32 sets of 3D images can require undesirably large computation times, and typically the data recorded by 32 coils is close to being linearly dependent. A technique called **coil compression** reduces data size [6, 7], essentially by making a **low-rank** approximation to the data and then just storing and processing the K singular vectors. Here are 8 MRI brain images recorded simultaneously by 8 coils around the head:



The scree plot on p. 6.8 suggests that most of the information is in the first 4 components. We reshape the data into a ($M = 256^2 \times N = 8$) matrix, use an SVD to find the rank-4 approximation, and reshape the first $K = 4$ vectors $\mathbf{u}_1, \dots, \mathbf{u}_4$ into 256×256 “virtual coil” images for display:



(In practice this operation is done on the raw data before making images.)



Non-uniqueness of SVD and low-rank approximation

As noted previously, “the” SVD of a matrix \mathbf{A} is not unique. In particular, if two singular values are the same, then one can swap the corresponding columns of \mathbf{U} and \mathbf{V} (or perform a more general rotation of their subspaces) and have two equally valid SVD expressions. Mathematically, when we write $\mathbf{A} = \sum_{k=1}^{\min(M,N)} \sigma_k \mathbf{u}_k \mathbf{v}'_k$, the order of the terms in the sum is unique if $\sigma_1 > \sigma_2 > \dots > \sigma_{\min(M,N)}$, but if any of the singular values are identical (including two or more zero singular values) then the ordering is not unique, and one can swap corresponding columns of \mathbf{U} and \mathbf{V} .

Example. Here are two different SVDs for a diagonal matrix with $7 = \sigma_1 > \sigma_2 = \sigma_3 = 5$:

$$\underbrace{\begin{bmatrix} 7 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 7 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{V}'} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\tilde{\mathbf{U}}} \underbrace{\begin{bmatrix} 7 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\tilde{\mathbf{V}}'}$$

Even if all the singular values are distinct, “the” SVD is still not unique because we can multiply both \mathbf{u}_k and \mathbf{v}_k by $e^{i\phi}$ and get a “different” \mathbf{U} and \mathbf{V} . However, the low-rank approximation is still the same in this case because $\sigma_k(e^{i\phi} \mathbf{u}_k)(e^{i\phi} \mathbf{v}_k)' = \sigma_k \mathbf{u}_k \mathbf{v}'_k$.

We have not focused on this issue too much yet because often uniqueness is relatively unimportant.

Now consider the question of whether the rank- K approximation of \mathbf{A} is **unique**. This is simpler to answer.

- If $\sigma_K > \sigma_{K+1}$ then one can extend the proof on p. 6.6 to show that the rank- K approximation is **unique**.
- If $\sigma_K = \sigma_{K+1}$, then the rank- K approximation is *not unique* because we could swap the K th and $(K+1)$ terms.

Example. Here are two distinct rank-1 approximations to a simple diagonal matrix:

$$\mathbf{A} \triangleq \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\mathbf{A} \approx \mathbf{B}_1 \triangleq \sigma_1 \mathbf{u}_1 \mathbf{v}'_1 = 5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{A} \approx \mathbf{B}_2 \triangleq \sigma_1 \tilde{\mathbf{u}}_1 \tilde{\mathbf{v}}'_1 = 5 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix}.$$

What is the approximation error $\|\mathbf{A} - \mathbf{B}_1\|_{\text{F}}$ in the preceding example?

A: 0

B: $\sqrt{5}$

C: 5

D: 10

E: 5^2

??

In practice, any rank- K approximation (for suitable K) often is equally useful, so one infrequently worries about non-uniqueness.

Furthermore, in practical finite precision computing, it is dubious to test whether two floating-point values σ_K and σ_{K+1} are “exactly” equal. Avoid code like `if (a == b)` with floating-point numbers!

1D example

Here we consider a 2×9 matrix where each column is an (x_n, y_n) coordinate for $n = 1, \dots, N = 9$, i.e.,

$$\mathbf{A} = \begin{bmatrix} x_1 & \dots & x_9 \\ y_1 & \dots & y_9 \end{bmatrix}.$$

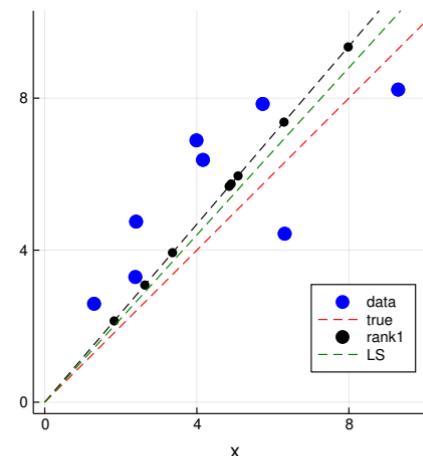
Based on the picture below, what is $\text{rank}(\mathbf{A})$? ??

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_low_rank1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_low_rank1.ipynb

Suppose we want to perform **dimensionality reduction** of the 2D data by reducing it to a 1D line.

- One option for processing this data is to perform a **rank-1 approximation**, leading to dashed black line in the figure. The black points are the nearest points in the subspace for each of the data points. ➤
- An alternative is to perform a **linear least-squares** fit assuming $y_n \approx \alpha x_n$. This leads to the green line with slope $\hat{\alpha}$ in the figure.

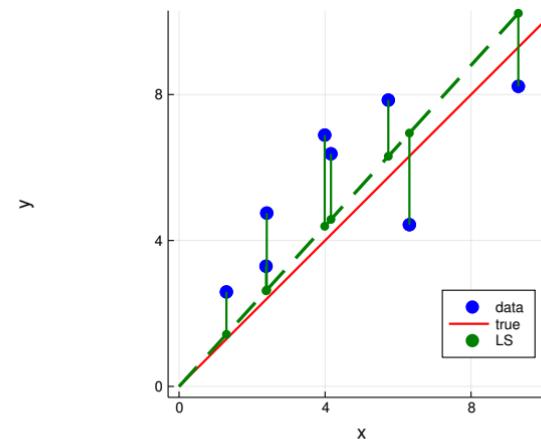
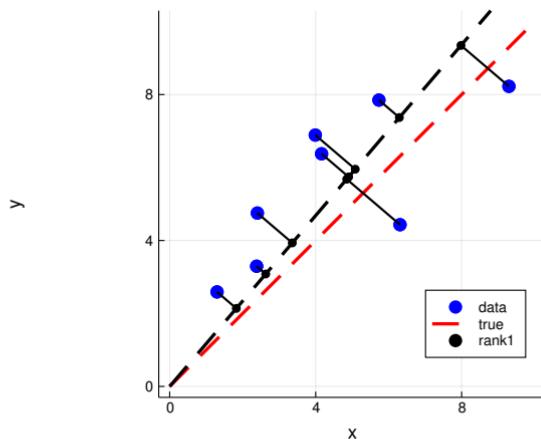


You might find it surprising that these two methods lead to different fits.

The **rank-1 approximation** is $\arg \min_{B \in \mathcal{L}_1^{M \times N}} \left\| \begin{bmatrix} x_1 & \dots & x_N \\ y_1 & \dots & y_N \end{bmatrix} - B \right\|_F^2 = \arg \min_{B \in \mathcal{L}_1^{M \times N}} \sum_{n=1}^N \left\| \begin{bmatrix} x_n \\ y_n \end{bmatrix} - B_{:,n} \right\|_2^2$.

The **linear least-squares** estimator is $\arg \min_{\alpha} \left\| \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \alpha \right\|_2^2 = \arg \min_{\alpha} \sum_{n=1}^N |y_n - \alpha x_n|^2$.

These two methods measure approximation error differently, as illustrated in the following two plots.



Generalization to other norms

Theorem (**Eckart-Young-Mirsky**) (See [3] and [8] for a proof.)

For any **unitarily invariant** matrix norm $\|\cdot\|_{\text{UI}}$, the **low-rank approximation problem** has the same solution using the first (largest) K singular components of $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}' = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k$:

$$\hat{\mathbf{B}}_K \triangleq \arg \min_{\mathbf{B} \in \mathcal{L}_K^{M \times N}} \underbrace{\|\mathbf{B} - \mathbf{A}\|_{\text{UI}}}_{\hookrightarrow \text{unitarily invariant norm}} = \sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}'_k.$$

So even though the proof sketch on earlier pages was for the Frobenius norm, remarkably the same result holds (with a different proof) for other unitarily invariant norms such as the **spectral norm**.

To be robust to data outliers, sometimes one prefers other norms that are *not* unitarily invariant [9, 10], especially $\|\cdot\|_1$. Those other norms require different solution methods (typically iterative algorithms). SVD operations are still used as one part of many such algorithms [10].

What is $\|\hat{\mathbf{B}}_K - \mathbf{A}\|_2$ when \mathbf{A} has rank r ?

- A: $\sqrt{\sum_{k=K+1}^r \sigma_k^2}$ B: $\sum_{k=K+1}^r \sigma_k$ C: σ_{K+1} D: 0 E: None of these

??

What is $\|\hat{\mathbf{B}}_K - \mathbf{A}\|_*$? (Choose from same answer list.) ??

Proof for spectral norm

(Read)

Claim: $\text{rank}(\mathbf{B}) \leq K \implies \left\| \mathbf{A} - \tilde{\mathbf{A}}_K \right\|_2^2 \leq \|\mathbf{A} - \mathbf{B}\|_2^2$ where $\tilde{\mathbf{A}}_K \triangleq \sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}'_k$.

Proof. Let $\mathbf{W} = [\mathbf{v}_1 \ \dots \ \mathbf{v}_{K+1}]$ so $\dim(\mathcal{R}(\mathbf{W})) = K + 1$ because columns of \mathbf{W} are orthonormal.

$\text{rank}(\mathbf{B}) \leq K \implies \dim(\mathcal{N}(\mathbf{B})) \geq N - K \implies \dim(\mathcal{N}(\mathbf{B})) + \dim(\mathcal{R}(\mathbf{W})) \geq (N - K) + (K + 1) = N + 1$. But both $\mathcal{N}(\mathbf{B})$ and $\mathcal{R}(\mathbf{W})$ are subspaces in \mathbb{F}^N , so they must have a nontrivial intersection, by (3.4).

Thus there exists some $\mathbf{x} \neq \mathbf{0}$ such that $\mathbf{x} \in \mathcal{N}(\mathbf{B})$ and $\mathbf{x} \in \mathcal{R}(\mathbf{W})$. WLOG take $\|\mathbf{x}\|_2 = 1$.

$\mathbf{x} \in \mathcal{R}(\mathbf{W}) \implies \mathbf{x} = \mathbf{W}\mathbf{z}$ for $\mathbf{z} \in \mathbb{F}^{K+1} \implies 1 = \|\mathbf{x}\|_2 = \|\mathbf{W}\mathbf{z}\|_2 = \|\mathbf{z}\|_2 = \|\mathbf{W}'\mathbf{W}\mathbf{z}\|_2 = \|\mathbf{W}'\mathbf{x}\|_2$.

To complete the proof:

$$\begin{aligned} \left\| \mathbf{A} - \tilde{\mathbf{A}}_K \right\|_2^2 &= \left\| \sum_{k=K+1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k \right\|_2^2 = \sigma_{K+1}^2 = \sigma_{K+1}^2 \|\mathbf{x}\|_2^2 = \sigma_{K+1}^2 \|\mathbf{W}'\mathbf{x}\|_2^2 \text{ because } \|\mathbf{x}\|_2 = 1 = \|\mathbf{W}'\mathbf{x}\|_2 \\ &= \sigma_{K+1}^2 \sum_{k=1}^{K+1} \|\mathbf{v}'_k \mathbf{x}\|_2^2 \leq \sum_{k=1}^{K+1} \sigma_k^2 \|\mathbf{v}'_k \mathbf{x}\|_2^2 = \|\mathbf{A}\mathbf{x}\|_2^2 \text{ because } \mathbf{x} \in \mathcal{R}(\mathbf{W}) \\ &= \|(\mathbf{A} - \mathbf{B})\mathbf{x}\|_2^2 \text{ because } \mathbf{x} \in \mathcal{N}(\mathbf{B}) \\ &\leq \|\mathbf{A} - \mathbf{B}\|_2^2 \|\mathbf{x}\|_2^2 = \|\mathbf{A} - \mathbf{B}\|_2^2. \end{aligned}$$

□

Bases for $\mathbb{F}^{M \times N}$

Recall that a set of **linearly independent** vectors $\mathbf{b}_1, \mathbf{b}_2, \dots$ is a **basis** for a vector space \mathcal{V} iff

$$\text{span}(\{\mathbf{b}_1, \mathbf{b}_2, \dots\}) = \mathcal{V},$$

i.e., we can write every vector in \mathcal{V} as a linear combination of the basis vectors: $\mathbf{v} \in \mathcal{V} \implies \mathbf{v} = \sum_k \mathbf{b}_k \alpha_k$ for some $\alpha_k \in \mathbb{F}$. Now we consider two types of bases for the vector space of **matrices** $\mathbb{F}^{M \times N}$.

Canonical basis for $\mathbb{F}^{M \times N}$

First, the obvious (canonical) basis for $\mathbb{F}^{M \times N}$ is: $\mathcal{B} = \{\mathbf{e}_m \tilde{\mathbf{e}}'_n : m = 1, \dots, M, n = 1, \dots, N\}$

where \mathbf{e}_m denotes the m th unit vector of length M and $\tilde{\mathbf{e}}_n$ denotes the n th unit vector of length N . These are

clearly linearly independent and $\text{span}(\mathcal{B}) = \mathbb{F}^{M \times N}$ because $\mathbf{A} \in \mathbb{F}^{M \times N} \implies \mathbf{A} = \sum_{m=1}^M \sum_{n=1}^N a_{mn} \mathbf{e}_m \tilde{\mathbf{e}}'_n$.

To attempt “**dimensionality reduction**” using this basis, we could seek an approximation of the form

$\mathbf{B} = \sum_{m=1}^M \sum_{n=1}^N b_{mn} \mathbf{e}_m \tilde{\mathbf{e}}'_n$, where we limit the number of nonzero coefficients b_{mn} .

Let $\|\mathbf{B}\|_0$ denote the number of nonzero elements of matrix \mathbf{B} . Then a natural optimization problem is:

$$\hat{\mathbf{B}} = \underset{\mathbf{B} \in \mathbb{F}^{M \times N}, \|\mathbf{B}\|_0 \leq K}{\arg \min} \|\mathbf{A} - \mathbf{B}\|_{\text{F}}$$

This problem has a trivial solution: choose the K elements of \mathbf{A} having the largest magnitudes $|a_{mn}|$ and set

all other elements of $\hat{\mathbf{B}}$ to zero. The approximation error is the square root of the sum of squared magnitudes of all those other elements.

Example.

(Read)

$$\hat{\mathbf{B}} = \arg \min_{\mathbf{B} \in \mathbb{F}^{M \times N}, \|\mathbf{B}\|_0 \leq 3} \left\| \begin{bmatrix} 3 & 5 & 7 \\ 6 & 0 & 2 \end{bmatrix} - \mathbf{B} \right\|_{\text{F}} = \begin{bmatrix} 0 & 5 & 7 \\ 6 & 0 & 0 \end{bmatrix}.$$

The approximation error is $\left\| \mathbf{A} - \hat{\mathbf{B}} \right\|_{\text{F}} = \left\| \begin{bmatrix} 3 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix} \right\|_{\text{F}} = \sqrt{2^2 + 3^2}$.

How large must K be here for this approximation to have zero error?

K = 5 because \mathbf{A} has 5 nonzero elements.

How large must K be here for the rank- K approximation to have zero error?

K = 2 because \mathbf{A} has rank 2.

The **low-rank approximation** seems more parsimonious than approximation using \mathcal{B} .

To state this rigorously, we must first show that our low-rank approximation also uses a **basis** for $\mathbb{F}^{M \times N}$.

Orthonormal bases for $\mathbb{F}^{M \times N}$

Let \mathbf{U} denote any **unitary** $M \times M$ matrix and \mathbf{V} denote any **unitary** $N \times N$ matrix.

Now endow $\mathbb{F}^{M \times N}$ with the (Frobenius) **inner product**: $\langle \mathbf{A}, \mathbf{B} \rangle = \text{trace}\{\mathbf{B}'\mathbf{A}\}$

and corresponding (Frobenius) norm:

$$\|\mathbf{A}\|_{\text{F}} = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle} = \sqrt{\text{trace}\{\mathbf{A}'\mathbf{A}\}}.$$

Claim: the following collection of MN rank-1 $M \times N$ matrices is an **orthonormal basis** for $\mathbb{F}^{M \times N}$:

$$\{\mathbf{u}_m \mathbf{v}'_n : m = 1, \dots, M, n = 1, \dots, N\}.$$

Proof of orthonormality (for the Frobenius inner product):

$$\begin{aligned}\langle \mathbf{u}_m \mathbf{v}'_n, \mathbf{u}_k \mathbf{v}'_l \rangle &= \text{trace}\{(\mathbf{u}_k \mathbf{v}'_l)' (\mathbf{u}_m \mathbf{v}'_n)\} = \text{trace}\{\mathbf{v}_l \mathbf{u}'_k (\mathbf{u}_m \mathbf{v}'_n)\} = \text{trace}\{\mathbf{u}'_k \mathbf{u}_m \mathbf{v}'_n \mathbf{v}_l\} \\ &= (\mathbf{u}'_k \mathbf{u}_m)(\mathbf{v}'_n \mathbf{v}_l) = \begin{cases} 1, & k = m, l = n \\ 0, & \text{otherwise.} \end{cases}\end{aligned}$$

Being orthonormal, the set $\{\mathbf{u}_m \mathbf{v}'_n\}$ spans a MN -dimensional space and is a basis for $\mathbb{F}^{M \times N}$. Thus one can write any $\mathbf{X} \in \mathbb{F}^{M \times N}$ in terms of that basis, i.e., $\mathbf{X} = \mathbf{U} \mathbf{C} \mathbf{V}' = \sum_{m=1}^M \sum_{n=1}^N c_{mn} \mathbf{u}_m \mathbf{v}'_n$, where $\mathbf{C} \triangleq \mathbf{U}' \mathbf{X} \mathbf{V}$, consistent with (6.4).

When working with a matrix \mathbf{A} , using the basis for $\mathbb{F}^{M \times N}$ that is formed from its own left and right **singular vectors**, i.e., \mathbf{U} and \mathbf{V} , respectively, turns out to be provide the most parsimonious representation. Note that the original low-rank problem formulation (6.1) did not involve any SVD, but an **SVD** arose in the solution.

Does the canonical basis $\{\mathbf{e}_m \tilde{\mathbf{e}}'_n : m = 1, \dots, M, n = 1, \dots, N\}$ form an orthonormal basis for $\mathbb{F}^{M \times N}$ when using the Frobenius inner product?

A: Yes

B: No

C: Insufficient information

??

Low-rank approximation summary

The low-rank approximation problem (6.1) and its SVD-based solution (6.2) provides the best low-rank *representation* (or approximation) of a given matrix (in the Frobenius norm sense). Specifically, if \mathbf{B} is any rank- K matrix having the same size as a matrix $\mathbf{A} = \sum_{k=1}^r \mathbf{u}_k \mathbf{v}'_k$, then:

$$\sqrt{\sum_{k=K+1}^r \sigma_k^2} = \left\| \mathbf{A} - \sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}'_k \right\|_{\text{F}} \leq \| \mathbf{X} - \mathbf{B} \|_{\text{F}}, \quad \forall \mathbf{B} \in \mathcal{L}_K^{M \times N}.$$

Generalizations

(Read)

What if the data is corrupted by noise? What if

$$\mathbf{Y} = \mathbf{X} + \boldsymbol{\varepsilon}$$

where \mathbf{X} is a low-rank matrix (or close to low rank). Due to the noise matrix $\boldsymbol{\varepsilon}$, typically \mathbf{Y} is not low-rank. We really want to recover \mathbf{X} here, not just represent or approximate \mathbf{Y} . How do we do that? An answer called OptShrink is given in [11] and discussed on p. 6.48.

Low-rank approximation is related closely to **principal component analysis (PCA)**, discussed on p. 6.55. There are many interesting **PCA generalizations** including robust methods, nonlinear models, multi-linear (tensor) models, **nonnegative matrix factorizations (NMF)** [12], methods that use sparsity, ...

Rank and stability

(Read)

Although **rank** is a simple mathematical concept, it is not stable numerically. Precisely, it is not a continuous function of the elements of a matrix.

Example. Consider the following two matrices:

$$\mathbf{A} = \begin{bmatrix} 1.7 & 1.7 \\ 1.7 & 1.7 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1.7 & 1.7 \\ 1.7 + \varepsilon & 1.7 \end{bmatrix}, \quad 0 < |\varepsilon| \ll 1.$$

These two matrices are nearly indistinguishable numerically, yet $\text{rank}(\mathbf{A}) = 1$ and $\text{rank}(\mathbf{B}) = 2$.

An alternative matrix property used in some situations is the **stable rank** [13], defined (for $\mathbf{A} \neq 0$) as:

$$\frac{\|\mathbf{A}\|_{\text{F}}^2}{\|\mathbf{A}\|_2^2}, \text{ where } 1 \leq \frac{\|\mathbf{A}\|_{\text{F}}^2}{\|\mathbf{A}\|_2^2} = \frac{1}{\sigma_1^2} \sum_{k=1}^{\min(M,N)} \sigma_k^2 = 1 + \sum_{k=2}^{\min(M,N)} \frac{\sigma_k^2}{\sigma_1^2} \leq r,$$

because (see HW) $\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_{\text{F}} \leq \sqrt{r} \|\mathbf{A}\|_2$. For the above example, $\|\mathbf{B}\|_{\text{F}}^2 / \|\mathbf{B}\|_2^2 \approx 1 + \varepsilon/2$.

Example. For $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$, the stable rank is $\frac{\|\mathbf{A}\|_{\text{F}}^2}{\|\mathbf{A}\|_2^2} = \frac{2^2 + 1^2}{2^2} = 1.25 \in [1, 2]$.

What is the **stable rank** of $\mathbf{A} = \mathbf{x}\mathbf{y}'$?

A: 0

B: 1

C: $\|\mathbf{x}\|_2 \|\mathbf{y}\|_2$ D: $\|\mathbf{x}\|_2^2 \|\mathbf{y}\|_2^2$

??

6.2 Sensor localization application (Multidimensional scaling)

We now turn to another application of matrix factorization: **sensor localization** via **multidimensional scaling**. Suppose $J \geq d$ sensors are located at unknown locations $\mathbf{c}_1, \dots, \mathbf{c}_J \in \mathbb{R}^d$, where typically $d = 2$ or $d = 3$. All we are given is the $J \times J$ **distance matrix** \mathbf{D} having elements

$$d_{ij} = d_{ji} = \underbrace{\|\mathbf{c}_i - \mathbf{c}_j\|_2}_{\hookrightarrow \text{usual Euclidean distance}}, \quad i, j = 1, \dots, J.$$

Clearly by definition the diagonal elements of \mathbf{D} are zero: $d_{jj} = 0$.

Goal: given \mathbf{D} , determine the locations $\{\mathbf{c}_j\}$.

Limitation: there is a fundamental ambiguity because translating or rotating coordinates does not change \mathbf{D} . Specifically, if $\tilde{\mathbf{c}}_j = \mathbf{Q}\mathbf{c}_j + \mathbf{d}$ where \mathbf{Q} is a $d \times d$ orthogonal matrix (such as a rotation matrix) and $\mathbf{d} \in \mathbb{R}^d$ is a displacement vector, then

$$\|\tilde{\mathbf{c}}_i - \tilde{\mathbf{c}}_j\|_2 = \|(\mathbf{Q}\mathbf{c}_i + \mathbf{d}) - (\mathbf{Q}\mathbf{c}_j + \mathbf{d})\|_2 = \|\mathbf{Q}(\mathbf{c}_i - \mathbf{c}_j)\|_2 = \|\mathbf{c}_i - \mathbf{c}_j\|_2$$

because the Euclidean norm is **unitarily invariant**. So \mathbf{D} would be the same for $\{\tilde{\mathbf{c}}_j\}$ and $\{\mathbf{c}_j\}$.

Thus we must be content with determining locations $\{\mathbf{c}_j\}$ to within some translation and rotation factor.

Note the choice to use a matrix representation of the data! And of course the locations are vectors. To derive a solution, define another matrix \mathbf{S} by the square of the elements of \mathbf{D} :

$$s_{ij} \triangleq d_{ij}^2 = \|\mathbf{c}_i - \mathbf{c}_j\|_2^2 = \|\mathbf{c}_i\|_2^2 + \|\mathbf{c}_j\|_2^2 - 2 \langle \mathbf{c}_i, \mathbf{c}_j \rangle.$$

Naturally we write \mathbf{S} in matrix form:

$$\begin{aligned} \mathbf{S} &= \begin{bmatrix} s_{11} & \dots & s_{1J} \\ \vdots & \dots & \vdots \\ s_{J1} & \dots & s_{JJ} \end{bmatrix} = \begin{bmatrix} \|\mathbf{c}_1\|_2^2 & \dots & \|\mathbf{c}_1\|_2^2 \\ \vdots & \dots & \vdots \\ \|\mathbf{c}_J\|_2^2 & \dots & \|\mathbf{c}_J\|_2^2 \end{bmatrix} + \begin{bmatrix} \|\mathbf{c}_1\|_2^2 & \dots & \|\mathbf{c}_J\|_2^2 \\ \vdots & \dots & \vdots \\ \|\mathbf{c}_1\|_2^2 & \dots & \|\mathbf{c}_J\|_2^2 \end{bmatrix} - 2 \begin{bmatrix} \langle \mathbf{c}_1, \mathbf{c}_1 \rangle & \dots & \langle \mathbf{c}_1, \mathbf{c}_J \rangle \\ \vdots & \dots & \vdots \\ \langle \mathbf{c}_J, \mathbf{c}_1 \rangle & \dots & \langle \mathbf{c}_J, \mathbf{c}_J \rangle \end{bmatrix} \\ &= \begin{bmatrix} \|\mathbf{c}_1\|_2^2 \\ \vdots \\ \|\mathbf{c}_J\|_2^2 \end{bmatrix} \mathbf{1}'_J + \mathbf{1}_J \begin{bmatrix} \|\mathbf{c}_1\|_2^2 & \dots & \|\mathbf{c}_J\|_2^2 \end{bmatrix} - 2 \begin{bmatrix} \mathbf{c}'_1 \\ \vdots \\ \mathbf{c}'_J \end{bmatrix} [\mathbf{c}_1 \ \dots \ \mathbf{c}_J] \\ &= \mathbf{r} \mathbf{1}'_J + \mathbf{1}_J \mathbf{r}' - 2 \mathbf{C}' \mathbf{C}, \quad \mathbf{r} \triangleq \begin{bmatrix} \|\mathbf{c}_1\|_2^2 \\ \vdots \\ \|\mathbf{c}_J\|_2^2 \end{bmatrix}, \quad \mathbf{C} \triangleq \underbrace{[\mathbf{c}_1 \ \dots \ \mathbf{c}_J]}_{d \times J \text{ unknown locations}}. \end{aligned} \tag{6.5}$$

Unfortunately we do not know the values in the vector \mathbf{r} .

So we know the $J \times J$ matrix \mathbf{S} and we want to solve for the $d \times J$ unknown location matrix \mathbf{C} using

$$\mathbf{S} = \mathbf{r}\mathbf{1}'_J + \mathbf{1}_J\mathbf{r}' - 2\mathbf{C}'\mathbf{C}. \quad (6.6)$$

Because \mathbf{S} on the left is symmetric and zero along its diagonal, it contains $(J^2 - J)/2$ distinct known values. The right-hand side depends on the dJ unknown values in \mathbf{C} .

Recall that source localization has a translation ambiguity, so we may as well use a coordinate system where the centroid is $\mathbf{0}$, *i.e.*, we can assume $\mathbf{C}\mathbf{1}_J = \sum_{j=1}^J \mathbf{c}_j = \mathbf{0}$ without losing any further generality.

Define the following “de-meaning” operator that projects onto the **orthogonal complement** of $\mathcal{R}(\mathbf{1}_J)$:

$$\mathbf{P}^\perp \triangleq \mathbf{I} - \frac{1}{J}\mathbf{1}_J\mathbf{1}'_J.$$

Multiplying \mathbf{P}^\perp by any vector produces a new vector whose mean value is zero, so we say it “de-means” the input vector. (The English word “demean” has a very different definition.) In winter, sometimes an airplane must be “de-iced” by removing ice from the wings. Here **de-mean** has similar use: removing the mean.

Clearly $\mathbf{P}^\perp\mathbf{1}_J = \mathbf{0}$, so $\mathbf{1}'_J\mathbf{P}^\perp = \mathbf{0}'$ and $\mathbf{C}\mathbf{P}^\perp = \mathbf{C}(\mathbf{I} - \frac{1}{J}\mathbf{1}_J\mathbf{1}'_J) = \mathbf{C}$ so from (6.6):

$$\mathbf{P}^\perp\mathbf{S}\mathbf{P}^\perp = \mathbf{P}^\perp(\mathbf{r}\mathbf{1}'_J + \mathbf{1}_J\mathbf{r}' - 2\mathbf{C}'\mathbf{C})\mathbf{P}^\perp = -2\mathbf{C}'\mathbf{C}.$$

Thus we have the following simple expression for finding the **Gram matrix**:

$$\mathbf{G} \triangleq \mathbf{C}'\mathbf{C} = -\frac{1}{2}\mathbf{P}^\perp\mathbf{S}\mathbf{P}^\perp.$$

In words, we remove the row and column means of \mathbf{S} and divide by -2 .

Now we have the **Gram matrix** $\mathbf{G} = \mathbf{C}'\mathbf{C}$, but we really want the coordinates matrix \mathbf{C} itself.

Although $\mathbf{C}'\mathbf{C}$ is a $J \times J$ matrix, $d \leq J$ so typically $\text{rank}(\mathbf{C}'\mathbf{C}) = d$. If the locations happen to be linearly dependent, *i.e.*, if the sensors are along one line through the origin in 2D or in the same plane through the origin in 3D, then $\text{rank}(\mathbf{C}'\mathbf{C}) < d$.

To elaborate, consider $d = 2$ where

$$\mathbf{C} \triangleq [\mathbf{c}_1 \quad \dots \quad \mathbf{c}_J] = \begin{bmatrix} x_1 & \dots & x_J \\ y_1 & \dots & y_J \end{bmatrix},$$

where $\mathbf{c}_j = (x_j, y_j)$. Because \mathbf{C} has two rows, $\text{rank}(\mathbf{C}'\mathbf{C}) = \text{rank}(\mathbf{C}) \leq 2$.

When does $\text{rank}(\mathbf{C}) = 1$? Only if the two rows of \mathbf{C} are linearly dependent, *i.e.*, if $\mathbf{y} = \alpha\mathbf{x}$ for some $\alpha \in \mathbb{R}$, *i.e.*, if $y_j = \alpha x_j$ for all j , which is the equation for points along a line through the origin with slope α .

The matrix $\mathbf{C}'\mathbf{C}$ consists of inner products, and it is invariant to rotations of the source locations:

$$\tilde{\mathbf{C}} = \mathbf{Q}\mathbf{C} \implies \tilde{\mathbf{C}}'\tilde{\mathbf{C}} = (\mathbf{Q}\mathbf{C})'(\mathbf{Q}\mathbf{C}) = \mathbf{C}'\mathbf{Q}'\mathbf{Q}\mathbf{C} = \mathbf{C}'\mathbf{C}.$$

Given the $J \times J$ Gram matrix $\mathbf{G} \triangleq \mathbf{C}'\mathbf{C}$, we determine its **compact SVD**:

$$\mathbf{G} = \mathbf{V}\Sigma\mathbf{V}' = \sum_{k=1}^J \sigma_j \mathbf{v}_j \mathbf{v}_j' = \sum_{j=1}^d \sigma_j \mathbf{v}_j \mathbf{v}_j' = \mathbf{V}_d \Sigma_d \mathbf{V}_d', \quad \Sigma_d \triangleq \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_d \end{bmatrix}}_{d \times d}, \quad \mathbf{V}_d \triangleq \underbrace{\begin{bmatrix} | & & | \\ \mathbf{v}_1 & \dots & \mathbf{v}_d \\ | & & | \end{bmatrix}}_{J \times d}.$$

- Because \mathbf{G} is Hermitian (in the absence of noise), we know $\mathbf{U} = \mathbf{V}$.
- Because \mathbf{G} has rank at most d (in the absence of noise), the above **low-rank** form is *exact*, not an approximation like we used earlier in this chapter.

We define our estimate of the source locations to be the J columns of this $d \times J$ matrix:

$$\hat{\mathbf{C}} = \Sigma_d^{1/2} \mathbf{V}_d' = \underbrace{\begin{bmatrix} \sqrt{\sigma_1} & & \\ & \ddots & \\ & & \sqrt{\sigma_d} \end{bmatrix}}_{d \times d} \underbrace{\begin{bmatrix} - & \mathbf{v}_1' & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{v}_d' & - \end{bmatrix}}_{d \times J},$$

because this estimate $\hat{\mathbf{C}}$ is consistent with \mathbf{C} to within an unknown rotation and translation:

$$\hat{\mathbf{C}}'\hat{\mathbf{C}} = (\Sigma_d^{1/2} \mathbf{V}_d')' (\Sigma_d^{1/2} \mathbf{V}_d) = \mathbf{V}_d \Sigma_d \mathbf{V}_d' = \underbrace{\mathbf{V}\Sigma\mathbf{V}'}_! = \mathbf{G} = \mathbf{C}'\mathbf{C}.$$

Now it seems like we are done because we have an expression for \mathbf{G} that is computable from \mathbf{S} , so we can use its eigendecomposition to find the source location estimates $\hat{\mathbf{C}}$. However, we assumed that $\mathbf{C}\mathbf{1}_J = \mathbf{0}$ so we must verify that $\hat{\mathbf{C}}\mathbf{1}_J = \mathbf{0}$ to ensure that our approach is self consistent.

Because $\mathbf{P}^\perp\mathbf{1}_J = \mathbf{0}$, it follows that $\mathbf{G}\mathbf{1}_J = \mathbf{0}$, so $\mathbf{V}\Sigma\mathbf{V}'\mathbf{1}_J = \mathbf{V}_d\Sigma_d\mathbf{V}'_d\mathbf{1}_J = \mathbf{0}$, which in turn implies that $\hat{\mathbf{C}}\mathbf{1}_J = \Sigma_d^{1/2}\mathbf{V}'_d\mathbf{1}_J = \mathbf{0}$.

What is the size of the final $\mathbf{0}$ in the immediately preceding line?

A: $J \times J$ B: $J \times 1$ C: $J \times d$ D: $d \times J$ E: $d \times 1$

??

Multidimensional scaling

Summary of method for finding locations from distances:

- Given distances arranged in a matrix \mathbf{D} .
- Compute \mathbf{S} from \mathbf{D} by squaring elements.
- Compute $\mathbf{G} = \mathbf{C}'\mathbf{C} = -\frac{1}{2}\mathbf{P}^\perp\mathbf{S}\mathbf{P}^\perp$ by de-meaning.
- Compute the **compact SVD** $\mathbf{G} = \mathbf{V}_d\Sigma_d\mathbf{V}'_d$.
- Use rank- d terms for source location estimate: $\hat{\mathbf{C}} = \Sigma_d^{1/2}\mathbf{V}'_d$.

This remarkably simple algorithm for finding $\hat{\mathbf{C}}$ is called (classical) **multidimensional scaling** [14, Ch. 12], and using the low-rank compact SVD is a key step. Because $\mathbf{G} = \mathbf{C}'\mathbf{C}$, \mathbf{G} is symmetric positive semidefinite, (in the absence of noise), its eigenvalues are all real and nonnegative so alternatively one could use an eigendecomposition of \mathbf{G} with suitably ordered eigenvalues.

Questions

Consider a given distance matrix D with $J > d$. The position estimates \hat{C} returned by the SVD-based MDS algorithm are unique (to within numerical precision), *i.e.*, are the same for any SVD of the Gram matrix.

A: True

B: False

??

The product $\hat{C}'\hat{C}$ is unique (to within numerical precision) for any SVD of the Gram matrix.

A: never

B: usually

C: always

??

Practical implementation

The `MultivariateStats` package of JULIA has a `classical_mds` command.

But the method is so simple that it is just as easy to code our own.

Example. Consider $J = 3$ sources that are all equally distant: $S = D .^2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$.

Just a few lines of JULIA suffices (see notebook linked on next page):

```
using Plots; using LinearAlgebra: svd, Diagonal; using Statistics: mean
D = [0 1 1; 1 0 1; 1 1 0] + 0e-9 * randn(3,3) # given distances
S = D.^2
tmp = S .- mean(S, dims=1)
G = -0.5 * (tmp .- mean(tmp, dims=2))
(_, s, V) = svd(G)
C = Diagonal(sqrt.(s[1:2])) * V[:,1:2]'
```

```
scatter(C[1,:], C[2,:], aspect_ratio=1, label="",
        xlabel="x", ylabel="y", title="s=$(s)",
        xtick=-.5:.5:.5, ytick=-.5:.5:.5, xlim=(-0.6,0.6), ylim=(-0.6,0.6))
```

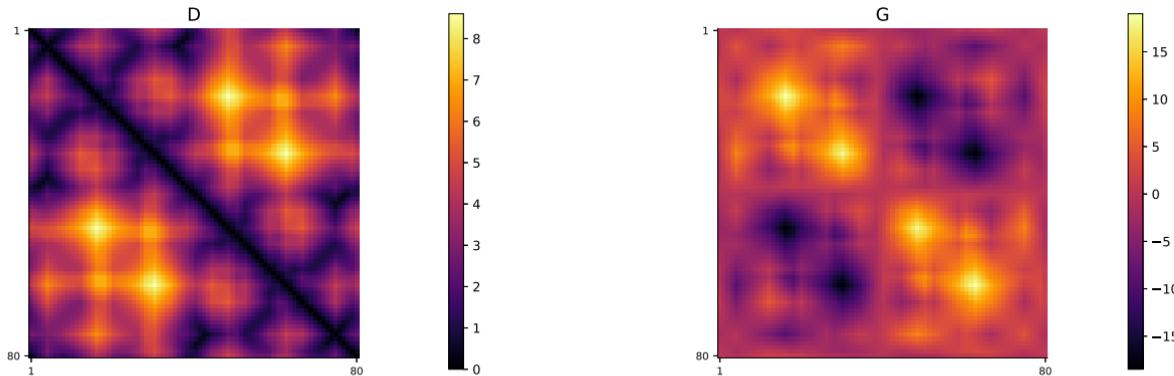
What “shape” should appear? ??

Example. An unknown collection of $J = 80$ source points:

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_source_local1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_source_local1.ipynb

Images of D and G :



Run demo files to show \hat{C} and consider noisy case to examine robustness.

Extensions

- When the distance measurements are noisy then D may not be perfectly symmetric and $P^\perp S P^\perp$ may have (hopefully small) negative eigenvalues.
Use **truncated SVD**: truncate rank to dimension d of embedding.
- What if D is **geodesic distances** or distances along roads between locations?
Distortion in location estimates.
- What if some sensors cannot reach some other sensors, *i.e.*, if D has some missing elements?
- Is there a way to formulate the problem that considers the possibility of noise (errors in the distance values) at the outset, instead of simply empirically investigating the effects of noise on our solution.
- What if there are outliers in the data, *e.g.*, some sensors that give incorrect values (either due to errors or because the sensor was hacked or otherwise compromised). Is MDS robust to such errors, or is a different formulation needed, perhaps based on a robust distance measure like $\|\cdot\|_1$?
- If we want to compare the location estimates \hat{C} to some ground truth, we can use the Procrustes method to align them, compensating for the inherent rotation/translation ambiguity.

6.3 Alternative low-rank approximation formulations

If $\mathbf{Y} \in \mathbb{F}^{M \times N}$ is a given data matrix with SVD $\mathbf{Y} = \mathbf{U}\Sigma\mathbf{V}'$ and $\mathbf{Y} = \mathbf{X} + \boldsymbol{\varepsilon}$ where we believe $\text{rank}(\mathbf{X}) \leq K$ and $\boldsymbol{\varepsilon}$ denotes a $M \times N$ noise matrix, then the constrained formulation discussed so far is natural [2]:

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathcal{L}_K^{M \times N}} \|\mathbf{Y} - \mathbf{X}\|_F^2 = \sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}_k' = \sum_{k=1}^r \sigma_k \mathbb{I}_{\{k \leq K\}} \mathbf{u}_k \mathbf{v}_k'.$$

This problem is nonconvex, because $\text{rank}(\cdot)$ is a nonconvex function, but still has a nice solution (6.2), reiterated above in two different forms. The second form reminds us that we have “set to 0” all the singular values for $k > K$.

Why rank is a non-convex function

To see why $\text{rank}(\cdot)$ is a non-convex function, consider: $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, $\alpha \in (0, 1) \implies$

$$\text{rank}(\alpha \mathbf{A} + (1 - \alpha) \mathbf{B}) = \text{rank} \left(\begin{bmatrix} \alpha & 0 \\ 0 & 1 - \alpha \end{bmatrix} \right) = 2 \not\leq \alpha \text{rank}(\mathbf{A}) + (1 - \alpha) \text{rank}(\mathbf{B}) = \alpha \cdot 1 + (1 - \alpha) \cdot 1 = 1.$$

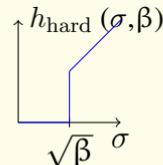
Unconstrained / regularized formulation

If the rank is unknown, an alternative approach is to consider an *unconstrained* cost function that discourages (penalizes) high rank (high complexity) rather than constraining it [15]:

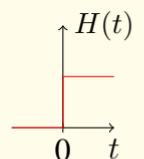
$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathbb{F}^{M \times N}} \|\mathbf{Y} - \mathbf{X}\|_{\text{F}}^2 + \beta \operatorname{rank}(\mathbf{X}) = \sum_{k=1}^r h_{\text{hard}}(\sigma_k, \beta) \mathbf{u}_k \mathbf{v}'_k, \quad (6.7)$$

where (see proof sketch below) $h_{\text{hard}}(\cdot, \beta)$ is the hard-thresholding function (*cf.* **Heaviside step function**):

$$h_{\text{hard}}(\sigma, \beta) = \frac{\sigma H(\sigma - \sqrt{\beta})}{\sigma \mathbb{I}_{\{\sigma > \sqrt{\beta}\}}} = \begin{cases} \sigma, & \sigma > \sqrt{\beta}, \\ 0, & \text{otherwise,} \end{cases}$$



$$H(t) = \begin{cases} 1, & t > 0, \\ 0, & \text{otherwise.} \end{cases}$$



General unitarily invariant formulations

There are several ways to form low-rank approximations that balance between data-fit and model complexity, all having the general form:

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathbb{F}^{M \times N}} \| \mathbf{Y} - \mathbf{X} \|_{\text{UI}}^2 + \beta R(\mathbf{X}), \quad (6.8)$$

for some **unitarily invariant** matrix norm $\| \cdot \|$ and for some **regularizer** $R(\mathbf{X})$, where $R : \mathbb{F}^{M \times N} \mapsto \mathbb{R}$, such as $R(\mathbf{X}) = \text{rank}(\mathbf{X})$. The regularization parameter (aka **hyperparameter**) $\beta > 0$ controls the trade-off between fit to the data \mathbf{Y} , as measured by $\| \mathbf{Y} - \mathbf{X} \|$, and model complexity, as quantified by $R(\mathbf{X})$.

We assume hereafter that the regularizer is also a unitarily invariant function, *i.e.*, $R(\mathbf{U}\mathbf{X}\mathbf{V}) = R(\mathbf{X})$ for *any* suitably sized unitary matrices \mathbf{U} and \mathbf{V} . Because both the data-fit norm and the regularizer are unitarily invariant, using the symmetric gauge principles of [3, 8], one can show that any minimizer has the form

$$\hat{\mathbf{X}} = \sum_{k=1}^r \hat{w}_k \mathbf{u}_k \mathbf{v}'_k = \mathbf{U}_r \hat{\Sigma}_r \mathbf{V}'_r \text{ where } \hat{\Sigma}_r = \text{diag}\{\hat{w}_k\}, \quad \hat{w}_k = h_k(\sigma_1, \dots, \sigma_r; \beta), \quad \mathbf{Y} = \mathbf{U}_r \Sigma_r \mathbf{V}'_r, \quad (6.9)$$

for some **shrinkage** or **thresholding** function $h_k(\cdot; \beta)$ that depends on the data-fit norm and the regularizer.

What is the rank of the $\hat{\mathbf{X}}$ in (6.9) when $\text{rank}(\mathbf{Y}) = r$? (Choose best answer.)

- A: Usually 1 B: Always 1 C: Usually r D: Always r E: None of these

??

Accepting that (6.9) is the form of the solution, we can rewrite the general optimization problem (6.8) as

$$\hat{\mathbf{X}} = \mathbf{U}_r \hat{\Sigma}_r \mathbf{V}'_r, \quad \hat{\Sigma}_r = \arg \min_{s_1, \dots, s_r} \|\Sigma_r - \mathbf{S}\|_{\text{UI}}^2 + \beta R(\mathbf{S}), \quad \mathbf{S} = \text{diag}\{s_1, \dots, s_r\}. \quad (6.10)$$

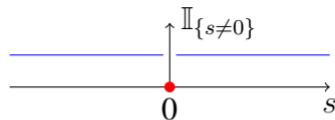
Singular value hard thresholding

Now return to the special case (6.7) with the Frobenius norm and the rank penalty. The equivalent minimization problem in terms of the singular values requires minimizing over $\{s_1, \dots, s_r\}$:

$$\|\Sigma_r - \mathbf{S}\|_{\text{F}}^2 + \beta \text{rank}(\mathbf{S}) = \sum_{k=1}^r ((\sigma_k - s_k)^2 + \beta \mathbb{I}_{\{s_k \neq 0\}}) \implies \hat{w}_k = \arg \min_{s \geq 0} (\sigma_k - s)^2 + \beta \mathbb{I}_{\{s \neq 0\}}. \quad (6.11)$$

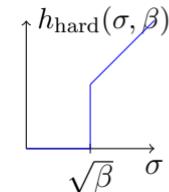
We started with the cost function (6.7), simplified it to the equivalent problem with diagonal matrices (6.10), and now we have r separate 1D problems that we can solve fairly easily.

We cannot just differentiate w.r.t. s and set the derivative to zero, because $\mathbb{I}_{\{s \neq 0\}}$ is a non-differentiable function. So we must solve it by “cases and braces.”



In (6.11), each minimizer \hat{w}_k can be either 0 or nonzero. If $\hat{w}_k = 0$ then the first term is σ_k^2 and the second term is 0. If $\hat{w}_k \neq 0$ then the second term is β and the best non-zero value is $\hat{w}_k = \sigma_k$ to make the first term zero. So the minimizer depends on whether β or σ_k^2 is larger. If $\sigma_k^2 < \beta$ then $\hat{w}_k = 0$ gives the lower cost. Otherwise $\hat{w}_k = \sigma_k$ gives the lower cost. In other words, in terms of the general expression (6.9) we have

$$\hat{w}_k = h_{\text{hard}}(\sigma_k, \beta), \quad h_{\text{hard}}(\sigma, \beta) \triangleq \begin{cases} \sigma, & \sigma > \sqrt{\beta} \\ 0, & \text{otherwise.} \end{cases}$$



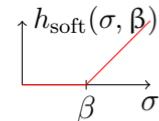
This case-wise analysis leads to the method (6.7) that is called **singular value hard thresholding**.

Singular value soft thresholding

Any formulation involving $\text{rank}(\cdot)$ is non-convex, and it is somewhat remarkable that a nice solution to (6.7) exists despite that non-convexity. Often it can be preferable to have a **convex** formulation. The **convex relaxation** [16] of rank is the **nuclear norm** that leads to soft thresholding of singular values:

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathbb{F}^{M \times N}} \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\|_{\text{F}}^2 + \beta \|\mathbf{X}\|_* = \sum_{k=1}^r [\sigma_k - \beta]_+ \mathbf{u}_k \mathbf{v}'_k, \quad \text{where } [t]_+ = \begin{cases} t, & t > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (6.12)$$

So $\hat{w}_k = h_{\text{soft}}(\sigma_k, \beta)$, $h_{\text{soft}}(\sigma, \beta) \triangleq [\sigma - \beta]_+ = \begin{cases} \sigma - \beta, & \sigma > \beta, \\ 0, & \text{otherwise.} \end{cases}$



This approach is called **singular value soft thresholding (SVST)**.

Proof sketch:

$$\frac{1}{2} \|\Sigma_r - \mathbf{S}\|_{\text{F}}^2 + \beta \|\mathbf{S}\|_* = \sum_{k=1}^r \frac{1}{2} (\sigma_k - s_k)^2 + \beta s_k \implies \hat{w}_k = \arg \min_s \frac{1}{2} (\sigma_k - s)^2 + \beta |s|.$$

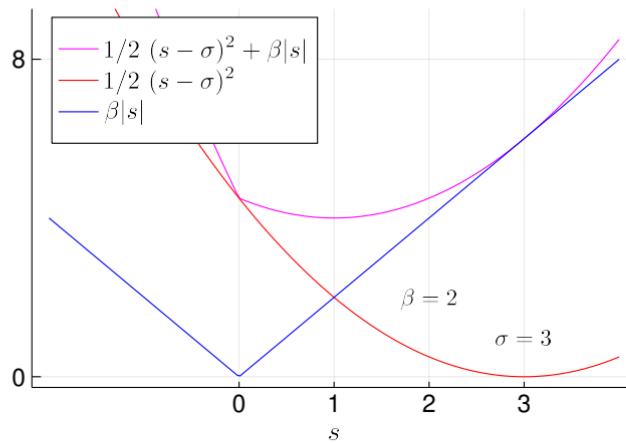
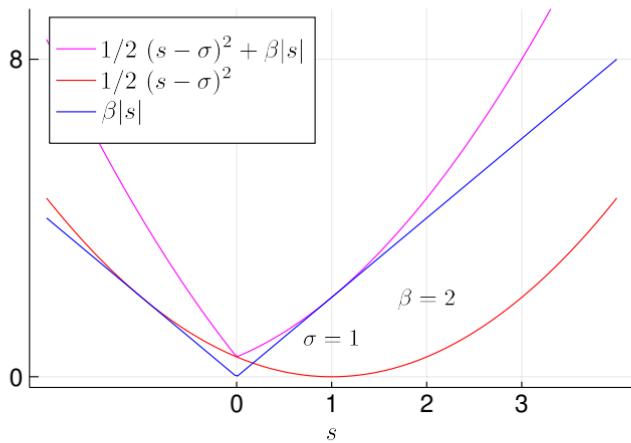
Sketching we see that the minimizer will be some $s \geq 0$ because $\sigma_k \geq 0$.

Differentiating: $0 = (s - \sigma_k) + \beta \Big|_{s=\hat{w}_k} \implies \hat{w}_k = \sigma_k - \beta$ but we need $\hat{w}_k \geq 0$ so $\hat{w}_k = [\sigma_k - \beta]_+$.

The following figures illustrate the cases where $\sigma_k < \beta$ and $\sigma_k > \beta$, by plotting the following functions:

$$\frac{1}{2}(s - \sigma)^2 + \beta |s|.$$

- When $0 < \beta \leq \sigma$ the minimizer is at $s = \sigma - \beta$.
- When $0 \leq \sigma \leq \beta$, the minimizer is at $s = 0$.



One can generalize this to complex numbers and show that

$$\arg \min_{s \in \mathbb{C}} \frac{1}{2} |s - z|^2 + \beta |z| = [|z| - \beta]_+ e^{i\angle z}.$$

If \mathbf{Y} has nonzero singular values 3, 5, 6, 7, 9 and $\beta = 6$, what is the rank of $\hat{\mathbf{X}}$ here?

- A: 1 B: 2 C: 3 D: 4 E: 5

??

This $\hat{\mathbf{X}}$ equals the rank- K approximation to \mathbf{Y} , where K is answer to previous problem. (?)

- A: True B: False

??

In general, what is $\|\hat{\mathbf{X}}\|_2$ in terms of the notation used throughout here?

- A: β B: σ_1 C: $\sigma_1 - \beta$ D: $\sqrt{\sigma_r - \beta}$ E: $[\sigma_1 - \beta]_+$

??

In general, if $\|\mathbf{Y}\|_2 \geq \beta$, then what is $\|\mathbf{Y} - \hat{\mathbf{X}}\|_2$ in terms of the notation used throughout here?

- A: β B: σ_1 C: $\sigma_1 - \beta$ D: $\sqrt{\sigma_r - \beta}$ E: $[\sigma_1 - \beta]_+$

??

One motivation for these unconstrained formulations is for solving **matrix completion** or **matrix sensing** problems where we must replace $\|\mathbf{Y} - \mathbf{X}\|_{\text{F}}^2$ with something like $\|\mathbf{y} - \mathbf{A} \text{vec}(\mathbf{X})\|_2^2$ where \mathbf{A} is a known matrix. These problems do not have closed-form solutions and require iterative methods, and convergence of iterative methods is better understood for convex problems.

Other extensions of low-rank approximation

- Sometimes we might want to retain some columns of the data exactly, leading to a different type of constrained low-rank approximation [17].
- Another variation replaces the Frobenius norm by the spectral norm: (HW)

$$\arg \min_{\mathbf{X}} \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\|_F^2 + \beta \|\mathbf{X}\|_*.$$

- Another variation replaces the Frobenius norm by the nuclear norm:

$$\arg \min_{\mathbf{X}} \|\mathbf{Y} - \mathbf{X}\|_* + \beta \|\mathbf{X}\|_*.$$

The solution to this variation is left as an exercise. It seems not to be useful.

- For further generalizations using weighted norms, see [18] [19, 20].

If we use two Frobenius norms, the solution has the following general form:

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X}} \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\|_{\text{F}}^2 + \beta \frac{1}{2} \|\mathbf{X}\|_{\text{F}}^2 = \sum_{k=1}^r \hat{w}_k \mathbf{u}_k \mathbf{v}'_k, \quad \hat{w}_k = h_{\text{FF}}(\sigma_k, \beta).$$

What is $h_{\text{FF}}(\sigma, \beta)$?

- A: $[\sigma - \beta]_+$ B: $\sigma H(\sigma - \sqrt{\beta})$ C: $\max(\sigma, \beta)$ D: $\frac{\sigma}{1 + \beta}$ E: $(\sigma - \beta)^2$

??

If \mathbf{Y} is $M \times N$ with rank r , then what is the rank of the solution $\hat{\mathbf{X}}$ here?

- A: 0 B: 1 C: r D: $\min(N, M)$ E: None of these.

??

6.4 Choosing the rank or regularization parameter

In all the above low-rank approximation formulations, we must either

- choose the rank K directly, or
- choose the regularization parameter β that influences the rank of $\hat{\mathbf{X}}$.

This selection process is non-trivial because when $\mathbf{Y} = \mathbf{X} + \varepsilon$ and the **latent** matrix \mathbf{X} has low-rank:

- $\left\| \hat{\mathbf{X}} - \mathbf{Y} \right\|_{\text{F}}$: the difference between the noisy data \mathbf{Y} and the low-rank approximation $\hat{\mathbf{X}}$ always
 - decreases monotonically as K increases (discretely)
 - increases monotonically as β increases (continuously).
- $\left\| \hat{\mathbf{X}} - \mathbf{X} \right\|_{\text{F}}$: the difference between the low-rank matrix \mathbf{X} and the low-rank approximation $\hat{\mathbf{X}}$:
 - usually decreases initially as K increases, then increases
 - usually decreases initially as β increases, then increases.

The figures on the next page illustrate these properties.

Often we normalize the difference between $\hat{\mathbf{X}}$ and \mathbf{Y} or that between $\hat{\mathbf{X}}$ and \mathbf{X} and define the (unitless) **normalized root mean-squared difference (NRMSD)** and **normalized root mean-squared error (NRMSE)**:

$$\text{NRMSD} = \frac{\left\| \hat{\mathbf{X}} - \mathbf{Y} \right\|_{\text{F}}}{\left\| \mathbf{Y} \right\|_{\text{F}}} \cdot 100\%, \quad \text{NRMSE} = \frac{\left\| \hat{\mathbf{X}} - \mathbf{X} \right\|_{\text{F}}}{\left\| \mathbf{X} \right\|_{\text{F}}} \cdot 100\%.$$

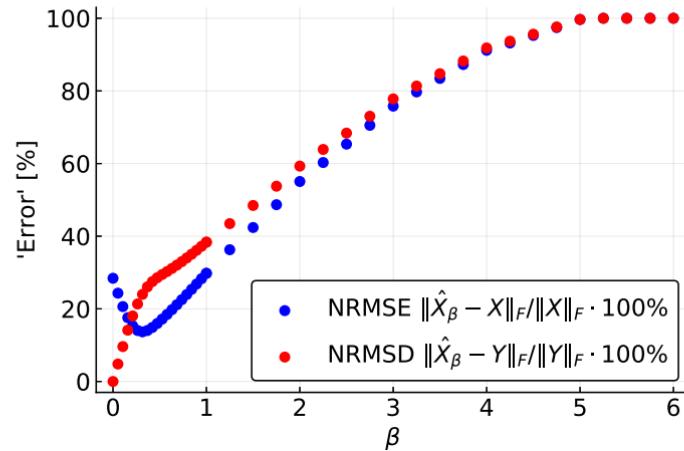
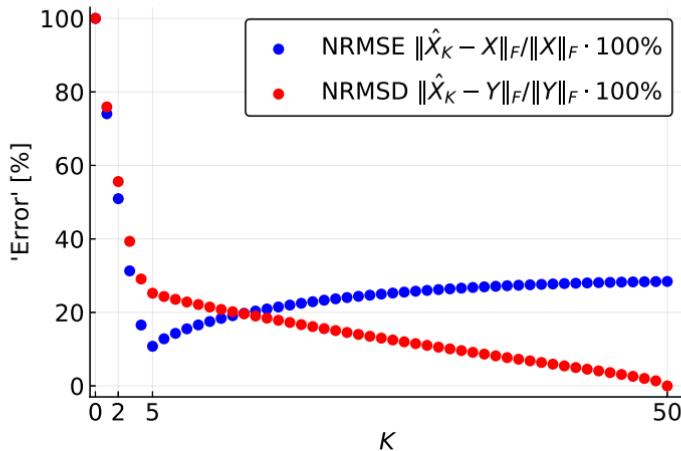
(There are many variations of these definitions in the literature.)

Example. In this case the 100×50 matrix \mathbf{X} has rank 5 and $\mathbf{Y} = \mathbf{X} + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon}$ is $M \times N$ matrix with IID Gaussian noise.

See JULIA demo:

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_lr_sure1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_lr_sure1.ipynb



Ideally we would like to find K_* or β_* that minimizes $\left\| \hat{\mathbf{X}} - \mathbf{X} \right\|_F$, the minimizers of the blue curves above. Finding either of the choices exactly is impossible because \mathbf{X} is unknown in practice!

All we have is $\mathbf{Y} = \mathbf{X} + \boldsymbol{\varepsilon}$, and usually we are *hoping* that \mathbf{X} is low rank, without knowing for sure.

Researchers have developed surrogates for $\left\| \hat{\mathbf{X}} - \mathbf{X} \right\|_F$ that help choose β .

One notable method is **Stein's unbiased risk estimate (SURE)**. If $\hat{\mathbf{x}}(\mathbf{y}; \beta)$ is a **weakly differentiable** estimate of a parameter $\mathbf{x} \in \mathbb{R}^d$ from data with Gaussian noise: $\mathbf{y} \sim N(\mathbf{x}, \sigma_0^2 \mathbf{I})$, then an unbiased estimate of the **risk**, *i.e.*, the mean-squared error (**MSE**), is:

$$\text{MSE}\{\beta\} = E[\|\hat{\mathbf{x}}(\mathbf{y}; \beta) - \mathbf{x}\|^2] = \text{SURE}\{\beta\} \triangleq \underbrace{\|\hat{\mathbf{x}}(\mathbf{y}; \beta) - \mathbf{y}\|_2^2 - d\sigma_0^2 + 2\sigma_0^2 \sum_i \frac{\partial}{\partial y_i} \hat{\mathbf{x}}_i(\mathbf{y}; \beta)}_{\text{Independent of } \mathbf{x}!}.$$

Divergence of $\hat{\mathbf{x}}(\cdot; \beta)$

For a proof, see [\[wiki\]](#). Note that σ_0^2 is a noise variance, not a singular value!

The SURE approach uses the approximation to select β as follows:

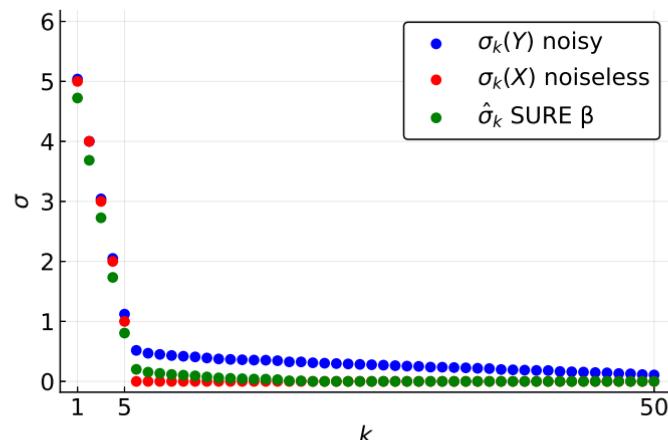
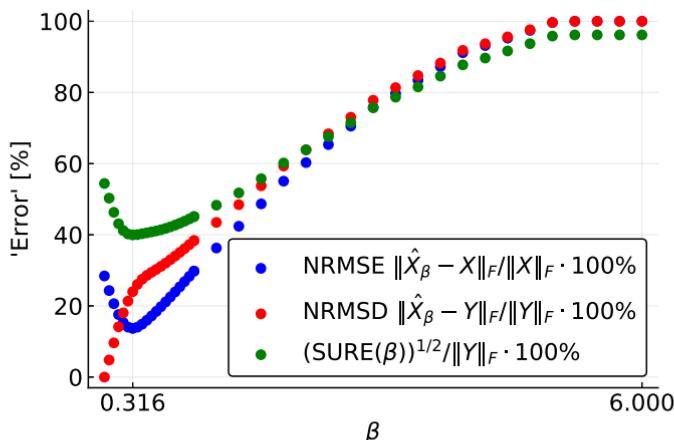
$$\beta_* \triangleq \arg \min_{\beta} \|\hat{\mathbf{x}}_{\beta} - \mathbf{x}\|^2 \approx \arg \min_{\beta} E[\|\hat{\mathbf{x}}_{\beta} - \mathbf{y}\|^2] = \arg \min_{\beta} \text{SURE}\{\beta\}.$$

The hard thresholding function is *not* weakly differentiable, so we cannot apply the SURE method to rank constrained or rank regularized cases. The soft thresholding function *is* weakly differentiable. Remarkably the divergence expression derived in [\[15\]](#) for any method of the form $\hat{\mathbf{X}} = \sum_k \mathbf{u}_k h_k(\sigma_k; \beta) \mathbf{v}'_k$ turns out to

depend only on the singular values so it is practical to evaluate. When the singular values of \mathbf{Y} are distinct:

$$\text{SURE}\{\beta\} = \left\| \hat{\mathbf{X}} - \mathbf{Y} \right\|_F^2 - MN\sigma_0^2 + 2\sigma_0^2 \left(|M-N| \sum_{i=1}^{\min(M,N)} \frac{h(\sigma_i; \beta)}{\sigma_i} + \sum_{i=1}^{\min(M,N)} h_i(\sigma_i; \beta) + 2 \sum_{i \neq j}^{\min(M,N)} \frac{\sigma_i h_i(\sigma_i; \beta)}{\sigma_i^2 - \sigma_j^2} \right).$$

See the demo notebook. A practical challenge is that one must know σ_0^2 to apply this method.



The minimizer β_* of the SURE function is remarkably close to the minimum MSE choice of β .

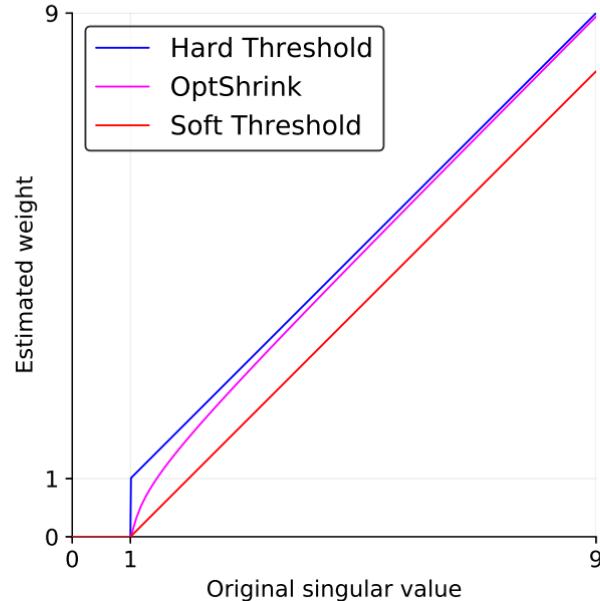
OptShrink

3002

IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 60, NO. 5, MAY 2014

OptShrink: An Algorithm for Improved Low-Rank Signal Matrix Denoising by Optimal, Data-Driven Singular Value Shrinkage

Raj Rao Nadakuditi, Member, IEEE



Model: $\mathbf{Y} = \mathbf{X} + \boldsymbol{\varepsilon}$. Random matrix theory says, for quite general noise models (bi-unitarily invariant), the best estimate of \mathbf{X} is $\hat{\mathbf{X}} = \sum_{k=1}^r \hat{w}_k \mathbf{u}_k \mathbf{v}'_k$, where \hat{w}_k is the best “weight” for the k th SVD component [11].

Algorithm 1 OptShrink: A New Algorithm for Low-Rank Matrix Denoising by Optimal, Data-Driven Singular Value Shrinkage

- 1: Input: $\tilde{X} = n \times m$ signal-plus-noise matrix
 - 2: Input: \hat{r} = Estimate of the effective rank of the latent low-rank signal matrix
 - 3: Compute $\tilde{X} = \sum_{i=1}^q \hat{\sigma}_i \hat{u}_i \hat{v}_i^H$
 - 4: Compute $\hat{\Sigma}_{\hat{r}} = \text{diag}(\hat{\sigma}_{\hat{r}+1}, \dots, \hat{\sigma}_q) \in \mathbb{R}^{(n-\hat{r}) \times (m-\hat{r})}$
 - 5: **for** $i = 1, \dots, \hat{r}$ **do**
 - 6: Compute $\hat{D}(\hat{\sigma}_i; \hat{\Sigma}_{\hat{r}})$ using (16a) and $\hat{D}'(\hat{\sigma}_i; \hat{\Sigma}_{\hat{r}})$ using (16b)
 - 7: Compute $\hat{w}_{i,\hat{r}}^{\text{opt}} = -2 \frac{\hat{D}(\hat{\sigma}_i; \hat{\Sigma}_{\hat{r}})}{\hat{D}'(\hat{\sigma}_i; \hat{\Sigma}_{\hat{r}})}$
 - 8: **end for**
 - 9: **return** $\hat{S}_{\text{opt}} = \sum_{i=1}^{\hat{r}} \hat{w}_{i,\hat{r}}^{\text{opt}} \hat{u}_i \hat{v}_i^H$ = denoised estimate of the rank \hat{r} signal matrix
 - 10: **return** (optional) Compute estimate of MSE using (17a)
 - 11: **return** (optional) Compute estimate of relative MSE using (17b)
-

For a matrix $X \in \mathbb{K}^{n \times m}$. Define

$$\begin{aligned}\hat{D}(z; X) &:= \frac{1}{n} \text{Tr} \left(z (z^2 I - XX^H)^{-1} \right) \\ &\quad + \frac{1}{m} \text{Tr} \left(z (z^2 I - X^H X)^{-1} \right)\end{aligned}\quad (16a)$$

and

$$\begin{aligned}\hat{D}'(z; X) &:= \frac{1}{n} \text{Tr} \left[z (z^2 I - XX^H)^{-1} \right] \\ &\quad + \frac{1}{m} \text{Tr} \left[-2z^2 (z^2 I - X^H X)^{-2} + (z^2 I - X^H X)^{-1} \right] \\ &\quad + \frac{1}{m} \text{Tr} \left[z (z^2 I - X^H X)^{-1} \right] \\ &\quad + \frac{1}{n} \text{Tr} \left[-2z^2 (z^2 I - XX^H)^{-2} + (z^2 I - XX^H)^{-1} \right].\end{aligned}\quad (16b)$$

Note: “ X ” in (16a) and (16b) is the $(n - \hat{r}) \times (m - \hat{r})$ (rectangular) diagonal matrix $\hat{\Sigma}_{\hat{r}}$.

The online code <http://web.eecs.umich.edu/~rajnrao/optshrink/> and the equations above are practical only if both n and m are sufficiently small. Here we adapt the approach to allow one of n or m (but not both) to be large.

Let \mathbf{S} denote a $K \times L$ rectangular diagonal matrix with entries $s_1, \dots, s_{\min(K,L)}$. We want to evaluate:

$$D(z, \mathbf{S}) = \frac{1}{K} \text{trace}\{z(z^2 \mathbf{I} - \mathbf{S}\mathbf{S}')^{-1}\} \frac{1}{L} \text{trace}\{z(z^2 \mathbf{I} - \mathbf{S}'\mathbf{S})^{-1}\}.$$

Note that $D(z, \mathbf{S}) = D(z, \mathbf{S}')$ so WLOG assume $K \geq L$ (tall). Then $\mathbf{S} = \begin{bmatrix} \mathbf{S}_L \\ \mathbf{0} \end{bmatrix}$ and

$$\begin{aligned} z^2 \mathbf{I} - \mathbf{S}\mathbf{S}' &= z^2 \mathbf{I} - \begin{bmatrix} \mathbf{S}_L^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} z^2 \mathbf{I} - \mathbf{S}_L^2 & \mathbf{0} \\ \mathbf{0} & z^2 \mathbf{I} \end{bmatrix} \\ \implies \text{trace}\{z(z^2 \mathbf{I} - \mathbf{S}\mathbf{S}')^{-1}\} &= \text{trace}\left\{z \begin{bmatrix} z^2 \mathbf{I} - \mathbf{S}_L^2 & \mathbf{0} \\ \mathbf{0} & z^2 \mathbf{I} \end{bmatrix}^{-1}\right\} = \sum_{k=1}^L \frac{z}{z^2 - s_k^2} + \frac{K-L}{z}. \end{aligned}$$

Similarly, as long as $z \neq s_k$:

$$\text{trace}\{z(z^2 \mathbf{I} - \mathbf{S}'\mathbf{S})^{-1}\} = \sum_{k=1}^L \frac{z}{z^2 - s_k^2}, \quad \implies D(z, \mathbf{S}) = \frac{1}{KL} \left(\sum_{k=1}^L \frac{z}{z^2 - s_k^2} + \frac{K-L}{z} \right) \left(\sum_{k=1}^L \frac{z}{z^2 - s_k^2} \right).$$

A similar simplification is feasible for $D'(z, \mathbf{S})$. (HW)

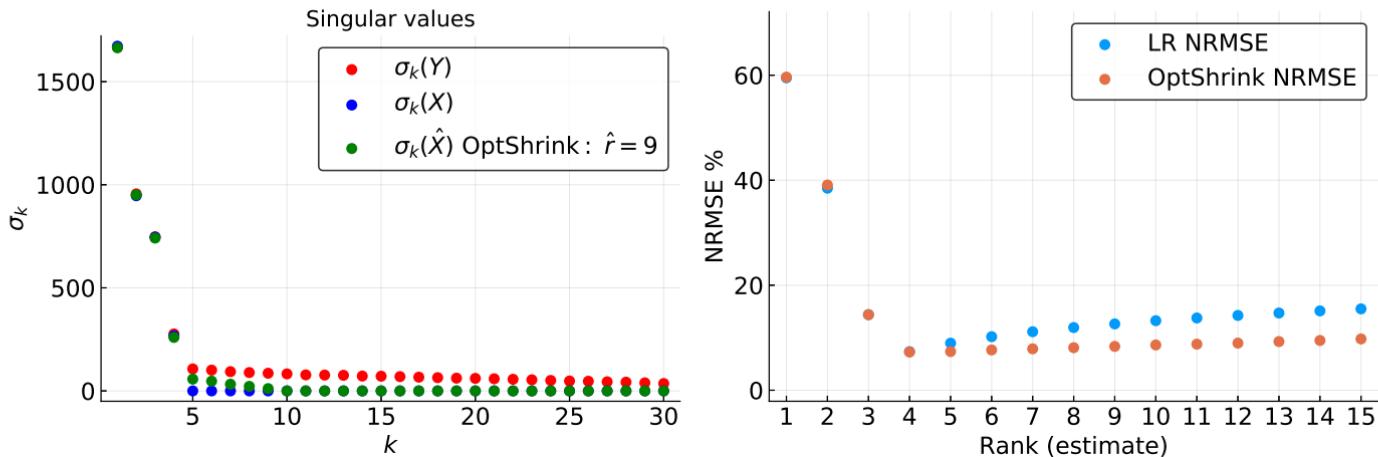
Combining, one can implement OptShrink to calculate the optimal SVD component weights when at most one of m or n is large.

Example showing OptShrink's robustness to rank estimate

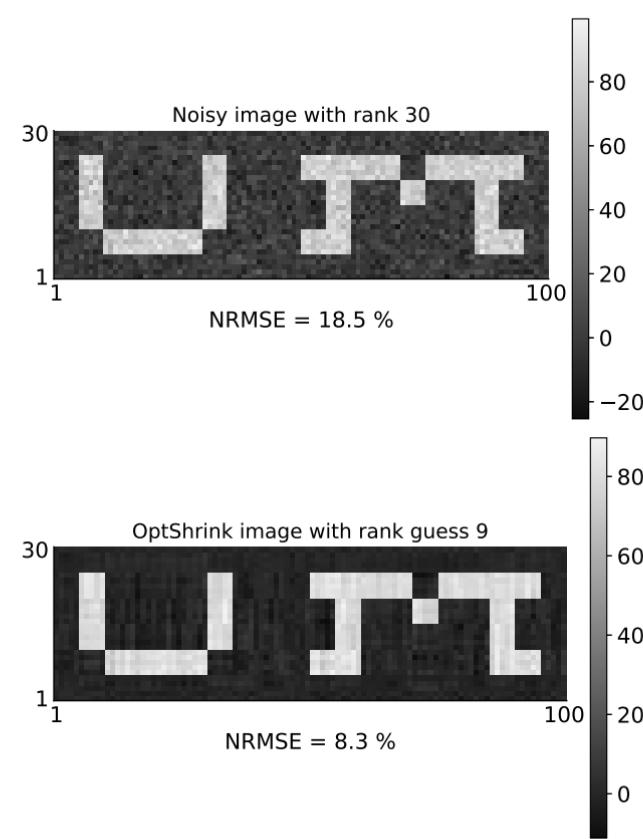
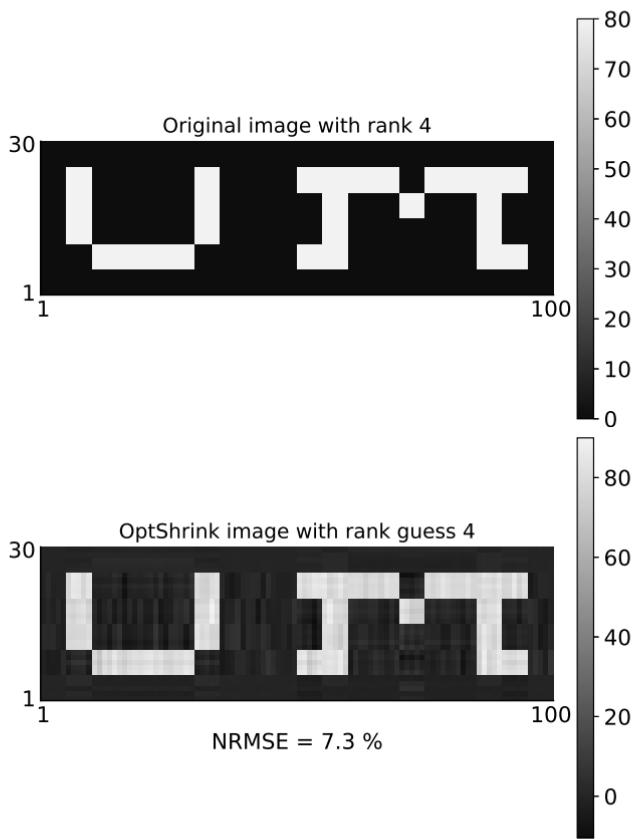
\mathbf{X} is a 100×30 logo image (aka 2D array aka matrix) with true rank 4. The noisy data is $\mathbf{Y} = \mathbf{X} + \boldsymbol{\varepsilon}$.

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_optshrink1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/06_optshrink1.ipynb



The error of $\hat{\mathbf{X}}$ for conventional low-rank approximation increases faster (due to over-fitting noise) than that of OptShrink. OptShrink finds the “threshold” for singular value shrinkage *adaptively* from the data, namely from the tail singular values, using random matrix theory.



6.5 Related methods: autoencoders and PCA

Relation to autoencoder with linear hidden layer

An **autoencoder** is a type of **artificial neural network (ANN)** that is designed to perform **dimensionality reduction**. An autoencoder consists of an **encoder** and a **decoder**:

$$\mathbf{x} \in \mathbb{F}^M \rightarrow \boxed{\text{encoder } \phi} \rightarrow \mathbf{z} \in \mathbb{F}^K \rightarrow \boxed{\text{decoder } \psi} \rightarrow \hat{\mathbf{x}} = \psi(\phi(\mathbf{x})) \in \mathbb{F}^M,$$

where typically $K \ll M$, *i.e.*, the dimension of the **code** or **latent variable** \mathbf{z} is much less than that of a data point \mathbf{x} .

The simplest possible case is where the encoder and decoder are each linear, and hence each can be represented via a **matrix**:

$$\phi(\mathbf{x}) = \mathbf{A}\mathbf{x}, \quad \mathbf{A} \in \mathbb{F}^{K \times M} \text{ (wide)}$$

$$\psi(\mathbf{z}) = \mathbf{B}\mathbf{z}, \quad \mathbf{B} \in \mathbb{F}^{M \times K} \text{ (tall).}$$

In this case, the output is $\hat{\mathbf{x}} = \mathbf{B}\mathbf{A}\mathbf{x}$ and the natural approach to **training** this ANN given training data $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{F}^M$ is

$$(\hat{\mathbf{A}}, \hat{\mathbf{B}}) = \arg \min_{\mathbf{A} \in \mathbb{F}^{K \times M}, \mathbf{B} \in \mathbb{F}^{M \times K}} \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{B}\mathbf{A}\mathbf{x}_n\|_2^2 = \arg \min_{\mathbf{A} \in \mathbb{F}^{K \times M}, \mathbf{B} \in \mathbb{F}^{M \times K}} \|\mathbf{X} - \mathbf{B}\mathbf{A}\mathbf{X}\|_{\text{F}}^2, \quad \mathbf{X} \triangleq [\mathbf{x}_1 \ \dots \ \mathbf{x}_N].$$

Note that the product \mathbf{BAX} has at most rank K due to the dimensions of \mathbf{A} and \mathbf{B} . So we can use the low-rank approximation work to observe that one solution to this training problem is

$$\mathbf{A} = \mathbf{U}'_K, \quad \mathbf{B} = \mathbf{U}_K,$$

where \mathbf{U}_K is the first K left singular vectors of \mathbf{U} , and an **SVD** is $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}'$. For that choice of \mathbf{A} and \mathbf{B} , and assuming $K \leq \min(M, N)$, the product becomes

$$\begin{aligned} \mathbf{BAX} &= \mathbf{U}_K \mathbf{U}'_K \underbrace{\mathbf{U}\Sigma\mathbf{V}'}_{\mathbf{X}} = \mathbf{U}_K \mathbf{U}'_K [\mathbf{U}_K \quad \mathbf{U}_{:, (K+1):M}] \Sigma \mathbf{V}' \\ &= \mathbf{U}_K [\mathbf{I}_K \quad \mathbf{0}_{K \times M}] \begin{bmatrix} \Sigma_K & \mathbf{0}_{K \times (N-K)} \\ \mathbf{0}_{(M-K) \times K} & \Sigma_{(M-K) \times (N-K)} \end{bmatrix} [\mathbf{V}_K \quad \mathbf{V}_{:, (K+1):N}]' = \mathbf{U}_K \Sigma_K \mathbf{V}'_K = \mathbf{X}_K, \end{aligned}$$

namely the optimal rank at most K approximation to \mathbf{X} .

So performing **low-rank approximation** of training data is comparable to learning an autoencoder with a single linear hidden layer of reduced dimension.

Nonlinear autoencoders are simply generalizations with nonlinear operations such as $\phi(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ where σ is some nonlinear function, \mathbf{W} is a $M \times K$ matrix and $\mathbf{b} \in \mathbb{F}^M$ is **bias vector**.

Exercise. Revisit the training derivation above in the case where the encoder and decoder are **affine** instead of linear, *i.e.*, $\phi(\mathbf{x}) = \mathbf{Ax} + \mathbf{b}_1$, $\psi(\mathbf{z}) = \mathbf{Bz} + \mathbf{b}_2$.

Relation to principal component analysis (PCA)

So far we have focused on low-rank approximation of a matrix. Often the low-dimensional subspaces $\text{span}(\mathbf{U}_K)$ and $\text{span}(\mathbf{V}_K)$ are more important to us than the overall approximation $\hat{\mathbf{B}} = \mathbf{U}_K \Sigma_K \mathbf{V}_K'$.

The **principal component analysis (PCA)** approach focuses on learning a linear transform of the data (having **orthonormal** columns) that maximizes variance. PCA starts with a set of N training vectors $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{F}^M$ and first subtracts from each the mean of all training vectors to form a first de-meaned data matrix:

$$\mathbf{X}_1 \triangleq [\mathbf{x}_1 - \boldsymbol{\mu} \quad \dots \quad \mathbf{x}_N - \boldsymbol{\mu}] = \mathbf{X} - \mathbf{X} \frac{1}{N} \mathbf{1}_N \mathbf{1}'_N, \quad \text{where } \mathbf{X} \triangleq [\mathbf{x}_1 \quad \dots \quad \mathbf{x}_N], \quad \boldsymbol{\mu} \triangleq \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n.$$

Note that $\mathbf{X}_1 \mathbf{1}_N = \mathbf{0}_M$. Now PCA seeks the (unit norm) linear combination of rows of \mathbf{X} (elements of \mathbf{x}) that maximizes (empirical) variance as follows:

$$\arg \max_{\mathbf{u}: \|\mathbf{u}\|=1} \frac{1}{N} \sum_{n=1}^N |\mathbf{u}' \mathbf{X}_1[:, n]|^2 = \arg \max_{\mathbf{u}: \|\mathbf{u}\|=1} \|(\mathbf{u}' \mathbf{X}_1)'\|_2 = \arg \max_{\mathbf{u}: \|\mathbf{u}\|=1} \|\mathbf{X}_1' \mathbf{u}\|_2 = \mathbf{u}_1,$$

where an **SVD** of \mathbf{X}_1 is $\mathbf{X}_1 = \mathbf{U} \Sigma \mathbf{V}'$. Another way of writing this uses a **Rayleigh quotient**:

$$\arg \max_{\mathbf{u} \in \mathbb{F}^M - \{\mathbf{0}\}} \frac{\mathbf{u}' \mathbf{X}_1 \mathbf{X}_1' \mathbf{u}}{\|\mathbf{u}\|_2^2},$$

where we recognize $\mathbf{X}_1 \mathbf{X}'_1 / N \in \mathbb{F}^{M \times M}$ as an **empirical covariance matrix**.

Having found that \mathbf{u}_1 is the (unit norm) linear combination that maximizes the (empirical) variance, we can now remove the component along that direction and then seek another linear combination that maximizes the variance of what is left. To remove the component along \mathbf{u}_1 we construct a new matrix as follows:

$$\mathbf{X}_2 \triangleq \mathbf{P}_{\mathcal{R}(\mathbf{u}_1)}^\perp \mathbf{X}_1 = (\mathbf{I} - \mathbf{P}_{\mathcal{R}(\mathbf{u}_1)}) \mathbf{X}_1 = (\mathbf{I} - \mathbf{u}_1 \mathbf{u}'_1) \mathbf{X}_1.$$

Note that $\mathbf{X}_2 \mathbf{1} = \mathbf{0}$ so we still have a zero-mean array. Now we want to find

$$\arg \max_{\mathbf{u} : \|\mathbf{u}\|=1} \|(\mathbf{u}' \mathbf{X}_2)'\|_2 = \arg \max_{\mathbf{u} : \|\mathbf{u}\|=1} \|\mathbf{X}'_1 (\mathbf{I} - \mathbf{u}_1 \mathbf{u}'_1) \mathbf{u}\|_2 = \mathbf{u}_2.$$

To see why \mathbf{u}_2 is the solution here, use an SVD of \mathbf{X}_1 to write

$$\mathbf{X}_2 = (\mathbf{I} - \mathbf{u}_1 \mathbf{u}'_1) \mathbf{X}_1 = (\mathbf{I} - \mathbf{u}_1 \mathbf{u}'_1) \mathbf{U} \Sigma \mathbf{V}' = \mathbf{U} \operatorname{diag}\{0, \sigma_2, \sigma_3, \dots\} \mathbf{V}',$$

which is almost an SVD of \mathbf{X}_2 except for a permutation. Clearly the nonzero singular values of \mathbf{X}_2 are $(\sigma_2, \sigma_3, \dots, \sigma_r)$ and the spectral norm of \mathbf{X}_2 is σ_2 and the corresponding left singular vector, which is its principal left singular vector, is \mathbf{u}_2 .

One can continue this reasoning to show that \mathbf{U}_K , the first K columns of \mathbf{U} , provides a linear transform with orthonormal columns that maximizes the resulting variances of the **scores** $\mathbf{U}'_K \mathbf{X}_1$.

Using the same SVD of \mathbf{X}_1 , we have $\mathbf{U}'_K \mathbf{X}_1 = \mathbf{U}'_K \mathbf{U} \Sigma \mathbf{V}' = \Sigma_K \mathbf{V}'_K$, where \mathbf{V}_K denotes the first K columns of \mathbf{V} . Thus instead of storing all MN elements of \mathbf{X}_1 it suffices to store the MK and NK elements of \mathbf{U}_K and \mathbf{V}_K , reflecting dimensionality reduction.

6.6 Subspace learning

This section discusses an application of low-rank matrix approximation to subspace learning. In this setting, the low-dimensional subspace $\mathcal{R}(\mathbf{U}_K)$ (or $\mathcal{R}(\mathbf{V}_K)$) is more important to us than the overall approximation $\hat{\mathbf{B}} = \mathbf{U}_K \Sigma_K \mathbf{V}'_K$.

In supervised classification, we are given N labeled training data samples $\mathbf{x}_{j,1}, \dots, \mathbf{x}_{j,N}$ for each of J classes of objects, where each feature vector $\mathbf{x}_{j,n} \in \mathbb{F}^M$. We would like to learn something about the nature of each class so that later when we get a new sample vector \mathbf{x}_0 we can assess which class it is most like.

One basic approach to classification is the **K-nearest-neighbors method**. When $K = 1$, this method simply chooses the class of the nearest neighbor among the training samples:

$$\hat{j} = \arg \min_{j \in \{1, \dots, J\}} \min_{n=1, \dots, N} \|\mathbf{x}_0 - \mathbf{x}_{j,n}\|.$$

This basic approach requires JN comparisons, so complexity grows with training data size. Also, its performance may not improve as N increases when the marginal distributions overlap.

Instead, often we try to learn a model from the training data, and then use fit to the model as the basis for classification. Here we focus on learning a subspace model for each class. Then to classify a test point \mathbf{x}_0 we simply compute the distance of \mathbf{x}_0 to each of the J subspaces and see which is closest.

To simplify notation, we drop the class subscript j and focus on learning a subspace for a set of samples for one class type, $\mathbf{x}_1, \dots, \mathbf{x}_N$ where each $\mathbf{x}_n \in \mathbb{F}^M$.

Saying that all the \mathbf{x}_n vectors lie in a subspace of dimension K means that there is some $M \times K$ matrix \mathbf{Q} having orthonormal columns such that $\mathbf{x}_n \in \mathcal{R}(\mathbf{Q})$. The columns of \mathbf{Q} form an **orthonormal basis** for the subspace. In practice the data will only *approximately* lie in a subspace, so $\mathbf{x}_n \approx \mathbf{Q}\mathbf{z}_n$ for some coefficient vector $\mathbf{z}_n \in \mathbb{F}^K$. We want to *learn* the subspace basis matrix \mathbf{Q} from the training data $\mathbf{X} = [\mathbf{x}_1 \ \dots \ \mathbf{x}_N]$, a $M \times N$ matrix. Writing $\mathbf{x}_n \approx \mathbf{Q}\mathbf{z}_n$ in matrix form:

$$\underbrace{\mathbf{X}}_{M \times N} \approx \underbrace{\mathbf{Q}}_{M \times K} \underbrace{\mathbf{Z}}_{K \times N}, \quad \text{where } \mathbf{Z} \triangleq [\mathbf{z}_1 \ \dots \ \mathbf{z}_N].$$

To find \mathbf{Q} and \mathbf{Z} we could pursue the following optimization problem:

$$\hat{\mathbf{Q}} = \arg \min_{\mathbf{Q} \in \mathbb{F}^{M \times K}} \min_{\mathbf{Z} \in \mathbb{F}^{K \times N}} \|\mathbf{X} - \mathbf{Q}\mathbf{Z}\|_F, \quad \text{s.t. } \mathbf{Q}'\mathbf{Q} = \mathbf{I}_K. \quad (6.13)$$

In this setting $K < \min(M, N)$ so the product $\mathbf{Q}\mathbf{Z}$ is matrix with (at most) rank K .

Thus this problem is essentially a low-rank approximation problem except that here we really care only about the subspace basis \mathbf{Q} and not the coefficients \mathbf{Z} (nor their product). The low-rank solution is

$$\hat{\mathbf{X}}_K = \sum_{k=1}^K \sigma_k \mathbf{u}_k \mathbf{v}'_k = \mathbf{U}_K \Sigma_K \mathbf{V}'_K = \underbrace{\mathbf{U}_K}_{\mathbf{Q}} \underbrace{\Sigma_K}_{\mathbf{Z}} \underbrace{\mathbf{V}'_K}_{\mathbf{Z}'_K}.$$

So to learn the subspace \mathbf{Q} we simply use the first K singular vectors of the training data \mathbf{X} .

After learning a subspace \mathbf{Q}_j for each of the J classes, we perform classification of a test point \mathbf{x}_0 by finding the nearest subspace:

$$\hat{j} = \arg \min_{j \in \{1, \dots, J\}} \min_{\mathbf{z} \in \mathbb{R}^k} \|\mathbf{x}_0 - \mathbf{Q}_j \mathbf{z}\|_2 = \arg \min_{j \in \{1, \dots, J\}} f_j(\mathbf{x}_0)$$

$$f_j(\mathbf{x}_0) \triangleq \underbrace{\|\mathbf{x}_0 - \mathbf{Q}_j (\mathbf{Q}_j' \mathbf{x}_0)\|_2}_{(a)} = \underbrace{\|\mathbf{x}_0 - \mathbf{P}_{\mathbf{Q}_j} \mathbf{x}_0\|_2}_{(b)} = \underbrace{\|\mathbf{P}_{\mathbf{Q}_j^\perp} \mathbf{x}_0\|_2}_{(c)},$$

where $\mathbf{P}_{\mathbf{Q}} \mathbf{x}$ denotes the orthogonal projection of \mathbf{x} onto the subspace $\mathcal{R}(\mathbf{Q})$.

A Discussion section task sheet will explore this method with hand-written digits from the **MNIST** data.

Learning subspaces via (6.13) is optimal in the Frobenius norm sense of low-rank approximation, but is *not* necessarily optimal for the purposes of classification. An extension called **supervised PCA** aims to find subspaces that are good for both approximation and classification [21, 22, 0].

See [23] for a special issue on subspace learning and a generalization called submanifold learning.

Which form above is the most compute efficient?

A: (a)

B: (b)

C: (c)

??

For that efficient form, how many multiplies are needed per test sample?

A: $J2M(K + 1)$

B: $JM(M + 1)$

C: JM^2

D: $J2MK$

??

Subspace clustering

(Read)

The **subspace learning** approach described above, especially (6.13), assumes that the training data is **labeled** by class type, so it is **supervised learning**. There are also **unsupervised** subspace learning methods that perform **subspace clustering** as part of the subspace learning process [24, 25] [8].

Given unlabeled training data $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{F}^M$, in a J -subspace clustering approach we want to learn a set of J orthogonal bases $\mathbf{Q}_1, \dots, \mathbf{Q}_J \in \mathcal{V}_K(\mathbb{F}^M) \subset \mathbb{F}^{M \times K}$, such that each training point \mathbf{x}_n is close to one of the J subspaces $\{\mathcal{R}(\mathbf{Q}_j)\}$. One possible formulation is

$$\arg \min_{\mathbf{Q}_1, \dots, \mathbf{Q}_J \in \mathcal{V}_K(\mathbb{F}^M)} \min_{j_1, \dots, j_N \in \{1, \dots, J\}} \sum_{n=1}^N \|\mathbf{x}_n - \mathbf{P}_{\mathbf{Q}_{j_n}} \mathbf{x}_n\|_2^2.$$

An alternative way of writing this is to use weights where w_{nj} indicates whether the n th training point is assigned to the j class:

$$\arg \min_{\mathbf{Q}_1, \dots, \mathbf{Q}_J \in \mathcal{V}_K(\mathbb{F}^M)} \min_{\mathbf{W} \in \mathcal{W}} \sum_{n=1}^N \sum_{j=1}^J w_{nj} \|\mathbf{x}_n - \mathbf{P}_{\mathbf{Q}_j} \mathbf{x}_n\|_2^2, \quad \mathcal{W} \triangleq \left\{ \mathbf{W} \in \mathbb{R}^{N \times J} : w_{nj} \in \{0, 1\}, \sum_{j=1}^J w_{nj} = 1 \right\}.$$

These are challenging discrete optimization problems called **integer programming** problems. One can alternate between updating the basis $\{\mathbf{Q}_j\}$ and the cluster assignments ($\{j_n\}$ or \mathbf{W}).

An alternative formulation called **sparse subspace clustering (SSC)** relaxes the discrete problem [24].

A related concept for unlabeled data is called **generalized principal component analysis (GPCA)** [26].

Bibliography

- [1] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005.
- [2] C. Eckart and G. Young. “The approximation of one matrix by another of lower rank”. In: *Psychometrika* 1.3 (1936), 211–8.
- [3] L. Mirsky. “Symmetric gauge functions and unitarily invariant norms”. In: *The quarterly journal of mathematics* 11.1 (Jan. 1960), 50–9.
- [4] R. B. Cattell. “The scree test for the number of factors”. In: *Multivariate Behavioral Research* 1.2 (1966), 245–76.
- [5] E. Dobriban. *Factor selection by permutation*. 2017.
- [6] M. Buehrer, K. P. Pruessmann, P. Boesiger, and S. Kozerke. “Array compression for MRI with large coil arrays”. In: *Mag. Res. Med.* 57.6 (June 2007), 1131–9.
- [7] T. Zhang, J. M. Pauly, S. S. Vasanawala, and M. Lustig. “Coil compression for accelerated imaging with Cartesian sampling”. In: *Mag. Res. Med.* 69.2 (Feb. 2013), 571–82.
- [8] Y-L. Yu and D. Schuurmans. “Rank/norm regularization with closed-form solutions: application to subspace clustering”. In: *Proc. 27th Conf. Uncertainty in AI*. 2011, 778–85.
- [9] P. Verboon and W. J. Heiser. “Resistant lower rank approximation of matrices by iterative majorization”. In: *Comp. Stat. Data Anal.* 18.4 (Nov. 1994), 457–67.
- [10] E. J. Candes, X. Li, Y. Ma, and J. Wright. “Robust principal component analysis?” In: *J. Assoc. Comput. Mach.* 58.3 (May 2011), 1–37.

- [11] R. R. Nadakuditi. “OptShrink: an algorithm for improved low-rank signal matrix denoising by optimal, data-driven singular value shrinkage”. In: *IEEE Trans. Info. Theory* 60.5 (May 2014), 3002–18.
- [12] Y. Shitov. “The nonnegative rank of a matrix: hard problems, easy solutions”. In: *SIAM Review* 59.4 (2017), 794–800.
- [13] M. B. Cohen, J. Nelson, and D. P. Woodruff. *Optimal approximate matrix product in terms of stable rank*. 2016.
- [14] I. Borg and P. J. F. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005.
- [15] E. J. Candes, C. A. Sing-Long, and J. D. Trzasko. “Unbiased risk estimates for singular value thresholding and spectral estimators”. In: *IEEE Trans. Sig. Proc.* 61.19 (Oct. 2013), 4643–57.
- [16] E. J. Candes and T. Tao. “The power of convex relaxation: near-optimal matrix completion”. In: *IEEE Trans. Info. Theory* 56.5 (May 2010), 2053–80.
- [17] G. Golub, A. Hoffman, and G. Stewart. “A generalization of the Eckart-Young-Mirsky matrix approximation theorem”. In: *Linear Algebra and its Applications* 88 (Apr. 1987), 317–27.
- [18] S. Gu, L. Zhang, W. Zuo, and X. Feng. “Weighted nuclear norm minimization with application to image denoising”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2014, 2862–9.
- [19] Y. Xie, S. Gu, Y. Liu, W. Zuo, W. Zhang, and L. Zhang. “Weighted Schatten p-norm minimization for image denoising and background subtraction”. In: *IEEE Trans. Im. Proc.* 25.10 (Oct. 2016), 4842–57.
- [20] X. Liu, X-Y. Jing, G. Tang, F. Wu, and Q. Ge. “Image denoising using weighted nuclear norm minimization with multiple strategies”. In: *Signal Processing* 135 (June 2017), 239–52.
- [21] E. Bair, T. Hastie, D. Paul, and R. Tibshirani. “Prediction by supervised principal components”. In: *J. Am. Stat. Assoc.* 101.473 (2006), 119–37.

- [22] E. Barshan, A. Ghodsi, Z. Azimifar, and M. Z. Jahromi. “Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds”. In: *Pattern Recognition* 44.7 (July 2011), 1357–71.
- [0] A. Ritchie and C. Scott. *Supervised principal component analysis: A regularization approach*. 2018.
- [23] Y. Ma, P. Niyogi, G. Sapiro, and R. Vidal. “Dimensionality reduction via subspace and submanifold learning [From the guest editors]”. In: *IEEE Sig. Proc. Mag.* 28.2 (Mar. 2011), 14–126.
- [24] R. Vidal. “Subspace clustering”. In: *IEEE Sig. Proc. Mag.* 28.2 (Mar. 2011), 52–68.
- [25] J. Lipor, D. Hong, D. Zhang, and L. Balzano. *Subspace clustering using ensembles of K-subspaces*. 2017.
- [26] R. Vidal, Y. Ma, and S. Sastry. “Generalized principal component analysis (GPCA)”. In: *IEEE Trans. Patt. Anal. Mach. Int.* 27.12 (Dec. 2005), 1945–59.

Chapter 7

Special matrices

Contents (final version)

7.0 Introduction	7.1
7.1 Companion matrices	7.2
Using companion matrices to check for common roots of two polynomials	7.8
7.2 Circulant matrices	7.10
7.3 Power iteration	7.16
7.4 Nonnegative matrices and Perron-Frobenius theorem	7.20
Markov chains	7.25
Irreducible matrix	7.34
Google's PageRank method	7.40
7.5 Summary	7.44

7.0 Introduction

This chapter contains topics related to matrices with special structures.

7.1 Companion matrices

L§10.4

Previously we defined the **eigenvalues** of a square matrix A to be the roots of the **characteristic polynomial**:

$$\det\{A - zI\} = 0.$$

Here we work somewhat in the reverse direction by starting with a polynomial and then defining a matrix from it. Consider the **monic polynomial**

$$p(z) = z^n + c_{n-1}z^{n-1} + \cdots + c_1z + c_0, \quad z \in \mathbb{C},$$

and now define the following $n \times n$ matrix called a **companion matrix** of that polynomial:

$$A \triangleq \begin{bmatrix} -c_{n-1} & -c_{n-2} & \dots & -c_1 & -c_0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}.$$

Which matrix class is this?

- A: Lower triangular B: Upper triangular C: Lower Hessenberg D: Upper Hessenberg E: None of these

??

Example. For $n = 3$ we have

$$\mathbf{A} = \begin{bmatrix} -c_2 & -c_1 & -c_0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

To examine the eigenvalues of \mathbf{A} , evaluate the **determinant** using the usual minors:

$$\begin{aligned} \det\{z\mathbf{I} - \mathbf{A}\} &= \det \left\{ \begin{bmatrix} z + c_2 & c_1 & c_0 \\ -1 & z & 0 \\ 0 & -1 & z \end{bmatrix} \right\} \\ &= (z + c_2) \det \left\{ \begin{bmatrix} z & 0 \\ -1 & z \end{bmatrix} \right\} - c_1 \det \left\{ \begin{bmatrix} -1 & 0 \\ 0 & z \end{bmatrix} \right\} + c_0 \det \left\{ \begin{bmatrix} -1 & z \\ 0 & -1 \end{bmatrix} \right\} \\ &= (z + c_2)(z^2 + 1 \cdot 0) - c_1(-z - 0^2) + c_0((-1)^2 - 0z) \\ &= z^3 + c_2z^2 + c_1z + c_0 = p(z). \end{aligned}$$

So the eigenvalues of the companion matrix \mathbf{A} are exactly the roots of the monic polynomial whose (negative) coefficients correspond to the first row of \mathbf{A} . This is not a coincidence; it is by design.

For the general $n \times n$ case the same process yields:

$$\begin{aligned} \det\{zI - A\} &= \det \left\{ \begin{bmatrix} z + c_{n-1} & c_{n-2} & \dots & c_1 & c_0 \\ -1 & z & 0 & \dots & 0 \\ 0 & -1 & z & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & -1 & z \end{bmatrix} \right\} = (z + c_{n-1}) \underbrace{\det \left\{ \begin{bmatrix} z & 0 & \dots & 0 \\ -1 & z & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & -1 & z \end{bmatrix} \right\}}_{(n-1) \times (n-1)} \\ &\quad - c_{n-2} \underbrace{\det \left\{ \begin{array}{c|ccccc} -1 & 0 & 0 & \dots & 0 \\ \hline 0 & z & 0 & \dots & 0 \\ 0 & -1 & z & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & -1 & z \end{array} \right\}}_{(n-1) \times (n-1)} + c_{n-3} \underbrace{\det \left\{ \begin{array}{c|ccccc} -1 & z & 0 & 0 & \dots & 0 \\ 0 & -1 & 0 & 0 & \dots & 0 \\ \hline 0 & 0 & z & 0 & \dots & 0 \\ 0 & 0 & -1 & z & \dots & 0 \\ \vdots & \vdots & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -1 & z \end{array} \right\}}_{(n-1) \times (n-1)} + c_{n-4} \dots \\ &= (z + c_{n-1})z^{n-1} + c_{n-2}z^{n-2} + c_{n-3}z^{n-3} + \dots \end{aligned}$$

The general pattern is that we have a **block diagonal** matrix for the c_{n-k} term, where the first block is $(k-1) \times (k-1)$ **upper triangular** and has determinant $(-1)^{k-1}$, and the second block is **lower triangular** and has determinant z^{n-k} , leading to a final overall contribution to the determinant of $c_{n-k}z^{n-k}$, for $k = 1, \dots, n$,

because $\det\left\{\begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} \end{bmatrix}\right\} = \det\{\mathbf{B}\} \det\{\mathbf{C}\}$. Thus

$$\det\{z\mathbf{I} - \mathbf{A}\} = c_0 + c_1 z + \cdots + c_{n-1} z^{n-1} + z^n = p(z).$$

In words: the polynomial $p(z)$ is the **characteristic polynomial** of \mathbf{A} . It is also its **minimum polynomial**. Due to this property, the matrix \mathbf{A} so defined is called a **companion matrix** of the monic polynomial $p(z)$.

The **eigenvalues** of the **companion matrix** \mathbf{A} are exactly the **roots** of the **monic polynomial** whose (negative) coefficients correspond to the first row of \mathbf{A} .

There are other rearrangements of \mathbf{A} (transposes and other permutations) that have the same property.

Practical implementation

To match the arrangement given above, use this one-line JULIA function:

```
compan = c -> [-reverse(c,dims=1)'; [I zeros(length(c)-1)]]
```

```
A = compan(c)
```

where `c` is the (column) vector with elements $(c_0, c_1, \dots, c_{n-1})$.

(This code needs $n \geq 1$ to work.)

However, in practice this matrix is perhaps used more for analysis than for implementation.

Eigenvectors of companion matrices

One can find **eigenvectors** of **companion matrices** by inspection (*i.e.*, by guess and check).

$$\begin{aligned} \text{If } \mathbf{v} = \begin{bmatrix} t^{n-1} \\ \vdots \\ t \\ 1 \end{bmatrix}, t \in \mathbb{C}, \text{ then } A\mathbf{v} &= \begin{bmatrix} -c_{n-1} & -c_{n-2} & \dots & -c_1 & -c_0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} t^{n-1} \\ \vdots \\ t \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} -c_{n-1}t^{n-1} - \dots - c_1t - c_0 \\ t^{n-1} \\ \vdots \\ t \end{bmatrix} \stackrel{?}{=} \begin{bmatrix} t^n \\ t^{n-1} \\ \vdots \\ t \\ 1 \end{bmatrix} = t \begin{bmatrix} t^{n-1} \\ \vdots \\ t \\ 1 \end{bmatrix} = t\mathbf{v}, \end{aligned}$$

where the inner equality holds iff

$$-c_{n-1}t^{n-1} - \dots - c_1t - c_0 = t^n, \text{ i.e., } 0 = t^n + c_{n-1}t^{n-1} + \dots + c_1t + c_0 = p(t).$$

In other words, if t is a root of the polynomial $p(z)$, then the corresponding vector \mathbf{v} is an eigenvector of \mathbf{v} , with eigenvalue t . Note that \mathbf{v} is a nonzero vector by definition.

Diagonalizing a companion matrix

Each root of $p(z)$ has a corresponding eigenvector (that we can normalize to have unit norm).

If $p(z)$ has any repeated roots, then the matrix \mathbf{A} is not diagonalizable.

(Fact: \mathbf{A} is **diagonalizable** iff its minimal polynomial is a product of distinct factors. [\[wiki\]](#))

If the roots of $p(z)$ are *distinct*, then the companion matrix \mathbf{A} is diagonalizable:

$$\mathbf{V}^{-1} \mathbf{A} \mathbf{V} = \text{diag}\{z_1, \dots, z_n\},$$

where $\{z_1, \dots, z_n\}$ denote the roots of the polynomial (*i.e.*, the eigenvalues of \mathbf{A}), and where the eigenvector matrix \mathbf{V} is an $n \times n$

$$\mathbf{V} = \begin{bmatrix} z_1^{n-1} & z_2^{n-1} & \dots & z_n^{n-1} \\ z_1^{n-2} & z_2^{n-2} & \dots & z_n^{n-2} \\ \vdots & \vdots & \vdots & \vdots \\ z_1 & z_2 & \dots & z_n \\ 1 & 1 & \dots & 1 \end{bmatrix}.$$

Vandermonde matrix of the form on the right:

Fact. A **Vandermonde matrix** is **invertible** iff the z_k values are distinct.

Vandermonde matrices are often transposed or `reverse` versions of the matrix shown above.

Example. In signal processing, the most important Vandermonde matrix is the N -point **DFT matrix**, which is a transposed and flipped version of the above matrix with $z_k = e^{-j2\pi k/N}$.

Rectangular Vandermonde matrices have been used as **frames** [1].



The companion matrix for the polynomial $p(z) = z^2 - 9z + 6$ is diagonalizable.

A: True

B: False

??

Using companion matrices to check for common roots of two polynomials

(Read)

First we need another tool.

Kronecker sum

Define. The **Kronecker sum** of $M \times M$ matrix \mathbf{A} with a $N \times N$ matrix \mathbf{B} is the following $MN \times MN$ matrix [2, p.143]:

$$\mathbf{A} \oplus \mathbf{B} = (\mathbf{I}_N \otimes \mathbf{A}) + (\mathbf{B} \otimes \mathbf{I}_M).$$

L§13.2

Wikipedia uses this definition:

$$\mathbf{A} \oplus \mathbf{B} = (\mathbf{A} \otimes \mathbf{I}_N) + (\mathbf{I}_M \otimes \mathbf{B}).$$

The two definitions are the same to within a permutation, *i.e.*, there is a $MN \times MN$ permutation matrix \mathbf{P} such that

$$\mathbf{P} ((\mathbf{I}_N \otimes \mathbf{A}) + (\mathbf{B} \otimes \mathbf{I}_M)) \mathbf{P}' = (\mathbf{A} \otimes \mathbf{I}_N) + (\mathbf{I}_M \otimes \mathbf{B}).$$

Because eigenvalues are invariant to similarity transforms, both definitions have the same eigenvalues so the following Fact holds for both definitions.

Fact. [2, Thm. 13.16]

If \mathbf{A} has eigenvalues $\{\lambda_m, m = 1, \dots, M\}$ and \mathbf{B} has eigenvalues $\{\mu_n, n = 1, \dots, N\}$, then the MN eigenvalues of $\mathbf{A} \oplus \mathbf{B}$ are

$$\lambda_m + \mu_n, \quad m = 1, \dots, M, \quad n = 1, \dots, N.$$

Proof sketch. If $\mathbf{Ax} = \lambda\mathbf{x}$ and $\mathbf{By} = \mu\mathbf{y}$, then

$$\begin{aligned} (\mathbf{A} \oplus \mathbf{B})(\mathbf{y} \otimes \mathbf{x}) &= (\mathbf{I}_N \otimes \mathbf{A})(\mathbf{y} \otimes \mathbf{x}) + (\mathbf{B} \otimes \mathbf{I}_M)(\mathbf{y} \otimes \mathbf{x}) = (\mathbf{y} \otimes (\mathbf{Ax})) + ((\mathbf{By}) \otimes \mathbf{x}) \\ &= (\mathbf{y} \otimes (\lambda\mathbf{x})) + ((\mu\mathbf{y}) \otimes \mathbf{x}) = (\lambda + \mu)(\mathbf{y} \otimes \mathbf{x}). \end{aligned}$$

Application: checking for common roots of two polynomials

If $p_1(z)$ and $p_2(z)$ are two monic polynomials (possibly of different degrees) having corresponding companion matrices \mathbf{A} and \mathbf{B} , then $p_1(z)$ and $p_2(z)$ share a common root iff \mathbf{A} and \mathbf{B} have a common eigenvalue, *i.e.*, iff there exists some λ_m and μ_n such that $\lambda_m = \mu_n$. In other words, $p_1(z)$ and $p_2(z)$ share a common root iff the matrix $\mathbf{A} \oplus (-\mathbf{B})$ has a zero eigenvalue, *i.e.*, is singular. Thus we can determine if two polynomials have a common root *without* performing any eigendecomposition. This property is explored in HW.

7.2 Circulant matrices

A special case of the **companion matrix** considered above corresponds to the simple monic polynomial

$$p(z) = z^N - 1.$$

Here $c_0 = -1$ and all other coefficients are 0, so the corresponding companion matrix is simply:

$$\mathbf{G}_N \triangleq \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}.$$

This special case is a **circulant matrix**. (Indeed it is the parent of all circulant matrices, as shown below.)

What are the eigenvectors of this matrix \mathbf{G}_N ? The roots of $p(z)$ here are solutions to $z^N - 1 = 0$, i.e., the N th roots of unity: $z_k = e^{-i2\pi k/N}$, $k = 0, \dots, N-1$. An eigenvector corresponding to the k th root z_k is

$$\mathbf{v}_k \triangleq \frac{1}{\sqrt{N}} z_k^{1-N} \begin{bmatrix} z_k^{N-1} \\ z_k^{N-2} \\ \vdots \\ z_k \\ 1 \end{bmatrix} = \frac{1}{\sqrt{N}} \begin{bmatrix} z_k^0 \\ z_k^{-1} \\ \vdots \\ z_k^{2-N} \\ z_k^{1-N} \end{bmatrix} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ w_k \\ \vdots \\ w_k^{N-2} \\ w_k^{N-1} \end{bmatrix}, \quad w_k \triangleq e^{i2\pi k/N},$$

with corresponding eigenvalue z_k .

Note that w_k for $k = 0, \dots, N - 1$ are also (distinct!) roots of $p(z)$, spaced equally around the unit circle in the complex plane.

The $N \times N$ matrix formed from all N of the eigenvectors is another Vandermonde matrix:

$$\mathbf{Q} \triangleq [\mathbf{v}_0 \ \dots \ \mathbf{v}_{N-1}] = \frac{1}{\sqrt{N}} \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} & \dots & \begin{bmatrix} 1 \\ w_k \\ \vdots \\ w_k^{N-2} \\ w_k^{N-1} \end{bmatrix} & \dots & \begin{bmatrix} 1 \\ w_{N-1} \\ \vdots \\ w_{N-2}^{N-2} \\ w_{N-1}^{N-1} \\ w_{N-1}^{N-1} \end{bmatrix} \end{bmatrix}. \quad (7.1)$$

Because the roots $\{w_k\}$ are distinct, this Vandermonde matrix is invertible. But here we can say much more: the columns of \mathbf{Q} here are **orthonormal**, so \mathbf{Q} is a **unitary matrix**. Thus

$$\mathbf{G}_N = \mathbf{Q} \operatorname{diag}\{z_0, \dots, z_{N-1}\} \mathbf{Q}'.$$

Multiplying matrix \mathbf{Q}' by a vector \mathbf{x} of length N corresponds to taking the N -point (orthonormal) **DFT** of \mathbf{x} .

Exercise. Verify that the columns of \mathbf{Q} are orthonormal vectors, *i.e.*, $\langle \mathbf{v}_k, \mathbf{v}_l \rangle = 0$ for $k \neq l$.

Now consider a general **circulant matrix**:

$$\mathbf{C} = \begin{bmatrix} c_0 & c_{n-1} & \dots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \dots & c_1 & c_0 \end{bmatrix}.$$

Because \mathbf{G}_N is a circulant permutation matrix, powers of \mathbf{G}_N provide other shifts:

$$\mathbf{G}_N^2 = \mathbf{G}_N \mathbf{G}_N = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \dots & 1 & 0 & 0 \end{bmatrix}.$$

So by defining $\mathbf{G}_N^0 = \mathbf{I}$, we write the general circulant matrix \mathbf{C} in terms of powers of \mathbf{G}_N as follows:

$$\mathbf{C} = c_0 \mathbf{I} + c_1 \mathbf{G}_N + c_2 \mathbf{G}_N^2 + \dots + c_{N-1} \mathbf{G}_N^{N-1} = c_0 \mathbf{I} + \sum_{n=1}^{N-1} c_n \mathbf{G}_N^n = \sum_{n=0}^{N-1} c_n \mathbf{G}_N^n.$$

The matrix \mathbf{G}_N^n has the same (orthonormal) eigenvectors as \mathbf{G}_N because $\mathbf{G}_N^2 \mathbf{v}_k = \mathbf{G}_N(\mathbf{G}_N \mathbf{v}_k) = z_k^2 \mathbf{v}_k$.

Thus

$$\begin{aligned} \mathbf{Q}' \mathbf{C} \mathbf{Q} &= \mathbf{Q}' \left(\sum_{n=0}^{N-1} c_n \mathbf{G}_N^n \right) \mathbf{Q} = \sum_{n=0}^{N-1} c_n (\mathbf{Q}' \mathbf{G}_N^n \mathbf{Q}) = \sum_{n=0}^{N-1} c_n \operatorname{diag}\{z_0^n, \dots, z_{N-1}^n\} \\ &= \operatorname{diag} \left\{ \sum_{n=0}^{N-1} c_n z_0^n, \dots, \sum_{n=0}^{N-1} c_n z_{N-1}^n \right\}, \end{aligned}$$

so the eigendecomposition of \mathbf{C} is

$$\mathbf{C} = \mathbf{Q} \operatorname{diag} \left\{ \sum_{n=0}^{N-1} c_n z_0^n, \dots, \sum_{n=0}^{N-1} c_n z_{N-1}^n \right\} \mathbf{Q}'. \quad (7.2)$$

This decomposition holds for *any* circulant matrix \mathbf{C} . In other words, all circulant matrices have the same unitary eigenvectors (the orthonormal DFT basis).

The k th eigenvalue of \mathbf{C} is

$$\sum_{n=0}^{N-1} c_n z_k^n = \sum_{n=0}^{N-1} c_n e^{-i2\pi nk/N}, \quad k = 0, \dots, N-1,$$

which is the (ordinary, *not* orthonormal) DFT of the *first column* of \mathbf{C} .

One can use the **fast Fourier transform (FFT)** to find all N eigenvalues with $O(N \log N)$ operations,

Relationship to DFT properties from DSP

(Read)

- The eigendecomposition (7.2) is the linear algebra expression of the DFT property that circular convolution corresponds to multiplying spectra:

$$h[n] * x[n] \xleftrightarrow{\text{DFT}} H[k]X[k].$$

- The fact that all circulant matrices have the same unitary basis, means that all circulant matrices **commute**, which is the linear algebra expression of the DFT property that circular convolution operations commute:

$$h[n] * x[n] = x[n] * h[n] \xleftrightarrow{\text{DFT}} H[k]X[k] = X[k]H[k].$$

- The associative property of circular convolution is also related to the fact all circulant matrices commute:

$$h[n] * (g[n] * x[n]) = (h[n] * g[n]) * x[n].$$

- The identity matrix \mathbf{I} is a circulant matrix, so $\mathbf{I} = \mathbf{Q}\mathbf{I}\mathbf{Q}'$ where \mathbf{Q} is the orthonormal DFT matrix defined in (7.1). The fact that the DFT of the Kronecker impulse signal $x[n] = [1 \ 0 \ \dots \ 0]$ is a flat spectrum $X[k] = 1$ is equivalent to the fact that the eigenvalues of \mathbf{I} are all unity.

The number of distinct eigenvalues of the matrix G_6^3 is:

A: 1

B: 2

C: 3

D: 4

E: 6

??

Practical implementation

A **circulant matrix** is a special case of a **Toeplitz matrix**.

To construct a circulant matrix in JULIA, one option is to use:

```
using ToeplitzMatrices  
A = Circulant(1:3)
```

However, it is very rare that we actually use a circulant matrix like this!

Directly multiplying a $N \times N$ circulant matrix C by a length- N vector, *i.e.*, $y = Cx$ via $y = C * x$, would require $O(N^2)$ operations.

We can compute the same product in $O(N \log_2 N)$ operations [3] using the **fast Fourier transform (FFT)** as follows:

```
y = ifft(fft(c) .* fft(x))
```

where c is the first column of C .

The resulting y will be the same as if we did $y = C * x$ except for some small numerical differences due to finite precision. These differences are rarely (if ever) important in practice.

One cautionary note is that if C and x are both real then of course $y = Cx$ is also real. However, the result of the above set of `fft` operations may end up with some very small imaginary part, depending on the implementation, so you might need to take the real part at the end.

7.3 Power iteration

(Read)

We showed previously that if square matrix \mathbf{A} is **diagonalizable**, then

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1} \implies \mathbf{A}^k = \mathbf{V}\Lambda^k\mathbf{V}^{-1}, \quad \forall k \in \mathbb{N}.$$

Thus for a vector \mathbf{x}_0 of appropriate size:

$$\mathbf{A}^k \mathbf{x}_0 = \mathbf{V}\Lambda^k \overbrace{\mathbf{V}^{-1} \mathbf{x}_0}^{\triangleq \mathbf{z}} = \sum_{n=1}^N (\lambda_n^k z_n) \mathbf{v}_n. \quad (7.3)$$

Now make some assumptions:

- WLOG we order the eigenvalues with decreasing magnitudes: $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_N|$.
- Assume $\lambda_1 \neq 0$ to avoid the trivial case of $\mathbf{A} = \mathbf{0}$.
- Assume $z_1 = [\mathbf{V}^{-1} \mathbf{x}_0]_1 \neq 0$, i.e., the initial vector \mathbf{x}_0 has a nonzero coordinate in the direction associated with the dominant eigenvalue λ_1 . Choosing \mathbf{x}_0 at random ensures $z_1 \neq 0$ with probability 1.

Then it follows from the linear independence of the columns of \mathbf{V} and the summation (7.3) that $\mathbf{A}^k \mathbf{x}_0 \neq \mathbf{0}$ for all $k \in \mathbb{N}$, so we can normalize it to have unit norm and define:

$$\mathbf{x}_k \triangleq \frac{\mathbf{A}^k \mathbf{x}_0}{\|\mathbf{A}^k \mathbf{x}_0\|_2}, \quad k = 1, 2, \dots \quad (7.4)$$

This definition leads to the following simple recursion:

$$\mathbf{x}_{k+1} = \frac{\mathbf{A}^{k+1}\mathbf{x}_0}{\|\mathbf{A}^{k+1}\mathbf{x}_0\|_2} = \frac{\mathbf{A}(\mathbf{A}^k\mathbf{x}_0)}{\|\mathbf{A}(\mathbf{A}^k\mathbf{x}_0)\|_2} = \frac{\mathbf{A}(\mathbf{A}^k\mathbf{x}_0)/\|\mathbf{A}^k\mathbf{x}_0\|_2}{\|\mathbf{A}(\mathbf{A}^k\mathbf{x}_0)/\|\mathbf{A}^k\mathbf{x}_0\|_2\|_2} = \frac{\mathbf{A}\mathbf{x}_k}{\|\mathbf{A}\mathbf{x}_k\|_2}, \quad k = 0, 1, \dots$$

For implementation we use this recursive form that is called the **power iteration**:

$$\mathbf{x}_{k+1} = \frac{\mathbf{A}\mathbf{x}_k}{\|\mathbf{A}\mathbf{x}_k\|_2}, \quad k = 0, 1, \dots \tag{7.5}$$

Returning to the summation form (7.3), one can show with some algebra [wiki] that

$$\mathbf{x}_k \propto \mathbf{A}^k \mathbf{x}_0 = z_1 \lambda_1^k \left(\mathbf{v}_1 + \sum_{n=2}^N \left(\left(\frac{\lambda_n}{\lambda_1} \right)^k \frac{z_n}{z_1} \right) \mathbf{v}_n \right),$$

where \mathbf{v}_n denotes a (unit-norm) eigenvector of \mathbf{A} corresponding to λ_n , i.e., the n th column of \mathbf{V} .

Example. Consider the 1×1 matrix $\mathbf{A} = i$. Here $\mathbf{A}^k \mathbf{x}_0 = z_1 i^k$ so $\mathbf{x}_k = \mathbf{A}^k \mathbf{x}_0 / \|\mathbf{A}^k \mathbf{x}_0\| = z_1 i^k / |z_1 i^k| = e^{i\angle z_1} i^k$, which does not converge due to the oscillations of the i^k term! Casually speaking, people say “the power iteration converges to \mathbf{v}_1 ” but this 1×1 example shows such words are imprecise. In general one cannot claim that $\|\mathbf{x}_k - \alpha \mathbf{v}_1\| \rightarrow 0$, no matter how we might pick α .

Convergence of power iteration

However, if the first eigenvalue magnitude dominates all others, *i.e.*, if

$$|\lambda_n| < |\lambda_1|, \quad n = 2, \dots, N,$$

then one can show that $\|e^{-i\angle z_1} (e^{-i\angle \lambda_1})^k \mathbf{x}_k - \mathbf{v}_1\| \rightarrow 0$, because $e^{-i\angle z_1} z_1 = |z_1|$ so

$$e^{-i\angle z_1} (e^{-i\angle \lambda_1})^k \mathbf{x}_k = \frac{|z_1| |\lambda_1|^k \left(\mathbf{v}_1 + \sum_{n=2}^N \left(\left(\frac{\lambda_n}{\lambda_1} \right)^k \frac{z_n}{z_1} \right) \mathbf{v}_n \right)}{\left\| z_1 \lambda_1^k \left(\mathbf{v}_1 + \sum_{n=2}^N \left(\left(\frac{\lambda_n}{\lambda_1} \right)^k \frac{z_n}{z_1} \right) \mathbf{v}_n \right) \right\|_2} \rightarrow \mathbf{v}_1.$$

In other words, as k increases, \mathbf{x}_k is approximately $e^{i\angle z_1} e^{ik\angle \lambda_1} \mathbf{v}_1$, meaning that it is approximately \mathbf{v}_1 times a unit-magnitude complex number.

Another way of saying this is $\|\mathbf{P}_{\mathbf{v}_1^\perp} \mathbf{x}_k\| \rightarrow 0$ as $k \rightarrow \infty$.

Under all the conditions assumed above, one can also show that the **eigenvalue** converges:

$$\mathbf{x}_k' \mathbf{A} \mathbf{x}_k \rightarrow \lambda_1.$$

Here we do not need to worry about the extra phase term $e^{i\angle z_1} e^{ik\angle \lambda_1}$ because it cancels out due to \mathbf{x}_k' and \mathbf{x}_k :

$$(e^{i\phi} \mathbf{v}_1)' \mathbf{A} (e^{i\phi} \mathbf{v}_1) = e^{-i\phi} e^{i\phi} \mathbf{v}_1' (\mathbf{A} \mathbf{v}_1) = \mathbf{v}_1' (\lambda_1 \mathbf{v}_1) = \lambda_1.$$

- In words, if the magnitude of one eigenvalue magnitude dominates all others, and if $z_1 \neq 0$, then for large k the output of the power iteration \mathbf{x}_k is approximately a unit-norm eigenvector corresponding to that largest magnitude eigenvalue.
- Casually speaking we say “the power iteration converges to the eigenvector corresponding to the largest magnitude eigenvalue” but it is imprecise to say “the” eigenvector since we can always scale by -1 or $e^{i\phi}$ more generally and to make rigorous claims about convergence we must also think about the phase.
- The convergence rate of the power iteration is governed by $|\lambda_2/\lambda_1|$, so the bigger the “gap” between the first and second largest eigenvalue magnitudes, the faster the convergence.
- We have assumed A is diagonalizable here, but the **power iteration convergence analysis** generalizes to non-diagonalizable matrices using the **Jordan form**. So sufficient conditions are:
 - A is square
 - $z_1 = [\mathbf{A}\mathbf{x}_0]_1 \neq 0$
 - $|\lambda_1|$ dominates the other eigenvalue magnitudes
- For analyzing Markov chains (next), we will see that the dominant eigenvalue of the transition matrix P is real and positive. For any matrix where λ_1 is real and positive, we have that $e^{i\angle\lambda_1} = 1$ so there is no problematic oscillation term. In this case, under the sufficient conditions above, we can say that the power iteration converges:

$$\mathbf{x}_k \rightarrow e^{i\angle z_1} \mathbf{v}_1.$$

For Markov chains, we will also see that \mathbf{v}_1 can be a nonnegative vector, so we can normalize it to *sum* to one, and if we initialize with a nonnegative vector \mathbf{x}_0 that sums to one then $e^{i\angle z_1} = 1$ and we can simply let $\mathbf{x}_{k+1} = P\mathbf{x}_k$ and conclude that $\mathbf{x}_k \rightarrow \mathbf{v}_1$.

7.4 Nonnegative matrices and Perron-Frobenius theorem

Define. A matrix A is called a **positive matrix** iff all of its elements are real and positive.

Define. A matrix A is called a **nonnegative matrix** iff all of its elements are real and nonnegative.

Define. A matrix A is called a **primitive matrix** iff it is a nonnegative square matrix and A^m is a **positive matrix** for some $m \in \mathbb{N}$.

The matrix $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ is (choose most specific answer):

- A: Square B: Nonnegative C: Primitive D: Positive E: None of these

??

These definitions are unrelated to **positive definite** and **positive semi-definite**, except in special cases like **diagonal** matrices.



There are many applications of such matrices.

- Such matrices arise when describing transition probabilities in certain **Markov chains**.
- Google's **PageRank** algorithm for ranking web sites is built on a certain nonnegative matrix.
- Other applications include power control, commodity pricing, and population growth [4].

Any positive semi-definite diagonal matrix is a nonnegative matrix. (?)

A: True

B: False

??

Any positive definite diagonal matrix is a positive matrix. (?)

A: True

B: False

??

The circulant generator matrix G_N from the previous class is (choose most specific answer):

A: Square

B: Nonnegative

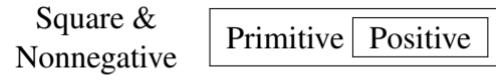
C: Primitive

D: Positive

E: None of these

??

Venn diagram (for square matrices only):



(There are rectangular matrices that are nonnegative and/or positive, but those are not of interest here.)

Square positive, primitive, and nonnegative matrices have various special properties in terms of their eigenvalues and eigenvectors that are important in applications. We focus on those properties next.

Properties of nonnegative matrices

We start with the broadest category: **nonnegative matrices**.

Recall that the **spectral radius** of a square matrix is its largest eigenvalue *magnitude*: $\rho(\mathbf{A}) = \max_i |\lambda_i(\mathbf{A})|$.

Fact. If \mathbf{A} is **square** and **nonnegative**, then we have the following bounds on its **spectral radius** [4]:

- Row sums provide bounds on the spectral radius: $\min_i \sum_j a_{ij} \leq \rho(\mathbf{A}) \leq \max_i \sum_j a_{ij}$
- Column sums provide bounds on the spectral radius: $\min_j \sum_i a_{ij} \leq \rho(\mathbf{A}) \leq \max_j \sum_i a_{ij}$

The upper bounds follow from the fact that $\rho(\mathbf{A}) \leq \|\mathbf{A}\|$ for any induced norm, so use $\|\mathbf{A}\|_1$ and $\|\mathbf{A}\|_\infty$.

One can also prove the upper bound using the **Geršgorin disk theorem**. The i th disk is

$\left\{ \lambda \in \mathbb{C} : |\lambda - a_{ii}| \leq r_i = \sum_{j \neq i} a_{ij} \right\}$ and if λ is in that disk, then $|\lambda| \leq a_{ii} + r_i = a_{ii} + \sum_{j \neq i} a_{ij} = \sum_j a_{ij}$. Thus $\rho(\mathbf{A}) \leq \max_i |\lambda_i| \leq \max_i \sum_j a_{ij}$.

The **Perron-Frobenius theorem** for any square, **nonnegative** matrix \mathbf{A} states the following:

- $r \triangleq \rho(\mathbf{A}) \geq 0$ is an eigenvalue of \mathbf{A} .

In other words, the largest magnitude eigenvalue is a real, nonnegative eigenvalue. Think $\lambda_1 = r \geq 0$.

We call r the **Perron root** or **Perron-Frobenius eigenvalue**.

- \mathbf{A} has a (right) eigenvector \mathbf{v} with *nonnegative elements* corresponding to that eigenvalue, *i.e.*, $\mathbf{A}\mathbf{v} = r\mathbf{v}$.
- There likewise exists a left eigenvector \mathbf{w} having all nonnegative elements for which $\mathbf{w}'\mathbf{A} = r\mathbf{w}'$.

We refer to such \mathbf{v} and \mathbf{w} as right and left **Perron vectors**, respectively.

Remarks.

- Because \mathbf{A} and \mathbf{A}' have the same eigenvalues, the existence of the nonnegative left eigenvector follows directly from the existence of the nonnegative right eigenvector, because \mathbf{A}' is also a nonnegative matrix.
- The fact that the largest magnitude eigenvalue is real and nonnegative takes some work to show, and does not follow from the **Geršgorin disk theorem**. One version uses the **proof for positive matrices** based on **Gelfand's formula** (5.22), followed by noting that any nonnegative matrix is a **limit of positive matrices**.
- By definition of spectra radius: $|\lambda_i(\mathbf{A})| \leq \rho(\mathbf{A})$, so for a square nonnegative matrix $|\lambda_i(\mathbf{A})| \leq r = \lambda_1$.
- This inequality is not strict and in general there can be multiple eigenvalues with the same magnitude. This matters because our convergence theory for the **power method** assumed there was one eigenvalue with dominant magnitude. So we cannot say much about powers of nonnegative matrices.
- Likewise, there can be multiple non-collinear right nonnegative eigenvectors having an eigenvalue where $|\lambda_i| = r$.
- In other words, we cannot say anything about uniqueness (or **multiplicity**) of an eigenvector having nonnegative elements. This means we cannot say much about the limiting behavior of Markov chains having merely nonnegative transition matrices.

Example. For $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ the eigenvalues are 1, 1. Any nonzero vector of the form $\begin{bmatrix} x \\ y \end{bmatrix}$ with $x, y \geq 0$ is a (right and left) Perron vector.

Example. For $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ the eigenvalues are $+1, -1$. The vector $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is a (right and left) Perron vector; the vector $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ is also an eigenvector whose eigenvalue, -1 , also has magnitude 1. Powers \mathbf{A}^k do not converge.

Properties of primitive matrices

To make stronger statements about uniqueness, we make stronger assumptions.

The **Perron-Frobenius theorem** for any (square) **primitive matrix A** states that:

- All the conclusions for (square) nonnegative matrices hold.
- $r \triangleq \rho(\mathbf{A}) > 0$ is a (real) eigenvalue of \mathbf{A} .

In other words, the largest magnitude eigenvalue is a real, *positive* eigenvalue. Think $\lambda_1 = r > 0$.

We call r the **Perron root** or **Perron-Frobenius eigenvalue**.

- That eigenvalue has **multiplicity one**, both **algebraic multiplicity** and **geometric multiplicity**.
- Uniqueness: For any other eigenvalue λ , $|\lambda| < r = \rho(\mathbf{A})$, i.e., there is no other λ such that $|\lambda| = r$.
- There exists a unit-norm eigenvector v with eigenvalue r , i.e., $\mathbf{A}v = rv$, where v has all *positive* elements.
- There likewise exists a unit-norm left eigenvector w having all positive elements for which $w^T A = rw^T$.
We call this v and w the right and left **Perron vectors**, respectively
- Uniqueness: All other (left and right) unit-norm eigenvectors have negative and/or non-real elements.
So v and w are unique in that sense.

All of these properties hold for (square) positive matrices because any such matrix is also primitive.

Because a primitive matrix A has one eigenvalue whose magnitude dominates all others, the **power iteration converges** to the **Perron vector** (to within a constant $e^{i\phi}$).

Some, but not all, of these properties generalize to the broader class of **irreducible matrices** [4].

Next we apply these definitions and properties to study Markov chains.

Markov chains

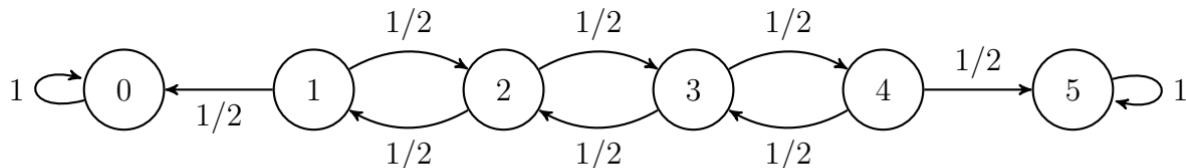
Define. A (first-order) **Markov chain** is a random process (or random sequence, *i.e.*, a sequence of random variables) X_1, X_2, \dots whose joint distribution is such that the conditional distribution of X_{k+1} given the values (states) of all preceding variables depends only on the preceding variable's value (state) and not how the process reached that state:

$$\mathsf{p}(X_{k+1} = x \mid X_k = x_k, X_{k-1} = x_{k-1}, \dots, X_1 = x_1) = \mathsf{p}(X_{k+1} = x \mid X_k = x_k).$$

Example. You start with \$2 and bet \$1 on the outcome of a fair coin toss. If you win you then have \$3 and if you lose you then have \$1. And you continue until (w.p.1) you eventually lose all your money or you stop betting because you reach your goal of, say, \$5. If at time k point you have, say \$3, then

$$\mathsf{p}(X_{k+1} = x \mid X_k = 3, X_{k-1} = x_{k-1}, \dots, X_1 = x_1) = \mathsf{p}(X_{k+1} = x \mid X_k = 3) = \begin{cases} 1/2, & x = 4 \\ 1/2, & x = 2 \\ 0, & \text{otherwise.} \end{cases}$$

In words, the next state will be \$2 or \$4, (with probability 1/2), regardless of how you reached the state of \$3.



We focus on cases (like that example) where the random variables take a finite set of discrete values. In this case, WLOG we can use the values $\{1, \dots, N\}$, where N is the number of **states** (*i.e.*, values).

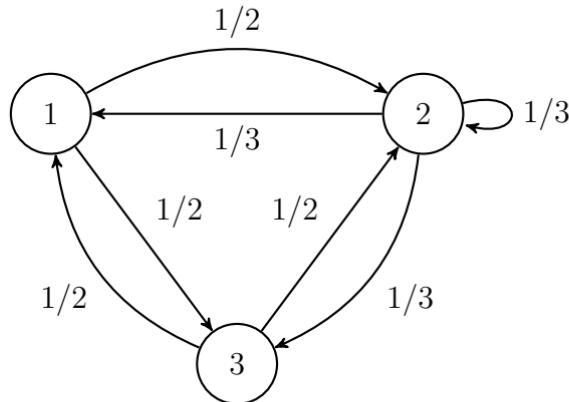
In this typical case, the statistical properties are governed by the **transition probabilities**:

$$\mathsf{p}(X_{k+1} = i \mid X_k = j), \quad i, j = 1, \dots, N.$$

In the typical case where the distributions are independent of “time” index k , we call it a **time-homogeneous Markov chain**, and we often describe the Markov chain by a **directed graph** that illustrates the transition probabilities between states. (See examples on previous and on next page.)

- Circles represent states.
- Arrows are labeled with transition probabilities.
- We draw arrows in such graphs only for nonzero transition probabilities.

Example.



Because this is a course on matrix methods, not on graphs or random processes, naturally we use a $N \times N$ **matrix** to describe these probabilities, called the **transition matrix**:

$$\mathbf{P}_{i,j} \triangleq p(X_{k+1} = i | X_k = j), \quad i, j = 1, \dots, N.$$

$$\begin{aligned} \mathbf{P} &= \begin{bmatrix} 0 & 1/3 & 1/2 \\ 1/2 & 1/3 & 1/2 \\ 1/2 & 1/3 & 0 \end{bmatrix} \\ &= \begin{bmatrix} p(X_{k+1} = i | X_k = j) \end{bmatrix} \end{aligned}$$

Sometimes the transpose of \mathbf{P} is also called the **transition matrix**. 

Note that the sum of the elements of each column is one, i.e., $\mathbf{1}'_3 \mathbf{P} = \mathbf{1}'_3$.

Here is a demo that also shows the limiting distribution discussed later:

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/07_markov_chain1.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/07_markov_chain1.ipynb

Properties of transition matrices

Any $N \times N$ transition matrix \mathbf{P} has useful properties.

- All transition matrices are **square** and **nonnegative**.
- Generalizing the previous example, the columns always sum to unity:

$$\mathbf{1}'_N \mathbf{P} = \mathbf{1}'_N \text{ because } \sum_{n=1}^N \mathsf{p}(X_{k+1} = n \mid X_k = j) = 1. \quad (7.6)$$

In words, the transition probabilities for leaving the j th state sum to one, for every j .

- Due to (7.6), a $N \times N$ transition matrix \mathbf{P} always has $\mathbf{1}_N$ as a left eigenvector with eigenvalue 1.
- Because all columns sum to unity: $\rho(\mathbf{P}) \leq 1$ so $|\lambda_i(\mathbf{P})| \leq 1$. (No eigenvalues are outside the unit disk.)
- Thus $\rho(\mathbf{P}) = 1$.

In general, there can be multiple nonnegative eigenvectors having eigenvalue equal to unity.

Example. The following transition matrix has two distinct eigenvectors with eigenvalue 1:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \implies \mathbf{v}_1 = \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \end{bmatrix} \text{ and } \mathbf{v}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \quad (7.7)$$

Is this \mathbf{P} **primitive**?

A: Yes

B: No

??

Equilibrium distributions of a Markov chain

Markov chains are a rich subject with numerous applications and theoretical considerations.

One particular aspect of them that we can examine using matrix methods is the existence of an **equilibrium distribution** or **stationary distribution** or **steady-state distribution**.

Define. Let π denote an N -dimensional vector $\pi = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_N \end{bmatrix}$. We call π a **stationary distribution** or **equilibrium distribution** or **steady-state distribution** of a Markov chain having **transition matrix** P iff

$$\pi_n \geq 0, \quad n = 1, \dots, N, \quad \sum_{n=1}^N \pi_n = \mathbf{1}'_N \pi = 1, \quad P\pi = \pi.$$

In words, π has nonnegative elements that sum to unity and is an eigenvector of P with eigenvalue 1. Such a π is sometimes called a **stochastic eigenvector**. See preceding example for two such π .

- Because a transition matrix P is nonnegative and has $\rho(P) = 1$, the **Perron-Frobenius theorem for nonnegative matrices** ensures that there always **exists** such a (nonzero) eigenvector having nonnegative elements with eigenvalue 1, so we can normalize that eigenvector to sum to unity, establishing existence of an equilibrium distribution.
- However, that eigenvalue of 1 need not be unique and there can be multiple stochastic eigenvectors. Again, see preceding example.

The importance of an equilibrium distribution is the following.

In general, by the **law of total probability**:

$$\mathsf{p}(X_{k+1} = i) = \sum_{j=1}^N \mathsf{p}(X_{k+1} = i \mid X_k = j) \mathsf{p}(X_k = j).$$

The chain is in **equilibrium** or **steady state** when $\mathsf{p}(X_{k+1} = n) = \mathsf{p}(X_k = n) = \pi_n$, *i.e.*, when:

$$\pi_i = \sum_{j=1}^N \mathsf{p}(X_{k+1} = i \mid X_k = j) \pi_j = \sum_{j=1}^N P_{ij} \pi_j.$$

In matrix vector form:

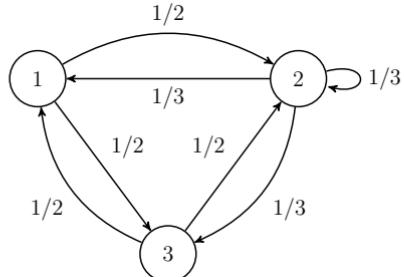
$$\boldsymbol{\pi} = \mathbf{P}\boldsymbol{\pi}.$$

In other words, an equilibrium distribution $\boldsymbol{\pi}$ is a stochastic eigenvector of the transition matrix \mathbf{P} associated with eigenvalue 1.

Caution: “**steady state**” does not mean “stuck in one state” in general, although one of the two equilibrium distributions for the preceding 3 state example happens to have that property.



Example. Returning to the earlier example, the eigenvalues of $P = \begin{bmatrix} 0 & 1/3 & 1/2 \\ 1/2 & 1/3 & 1/2 \\ 1/2 & 1/3 & 0 \end{bmatrix}$ are $\{1, -1/2, -1/6\}$.



A (unit norm) eigenvector associated with eigenvalue 1 is $v_1 = \begin{bmatrix} 0.485071 \\ 0.727607 \\ 0.485071 \end{bmatrix}$.

One can verify that $\|v_1\|_2 = 1$. But this is not the normalization we want for a probability distribution.

We want $\sum_n \pi_n = 1$ i.e., $1'_N \boldsymbol{\pi} = 1$.

So define $\boldsymbol{\pi} = v_1 / (1'_N v_1)$, possibly with sign flip if needed, for which $\boldsymbol{\pi} = \begin{bmatrix} 0.285714 \\ 0.428571 \\ 0.285714 \end{bmatrix}$.

This **stochastic eigenvector** is the **equilibrium distribution** and its elements show the (equilibrium) probability of each state. Note that state 2 is more likely, and this is intuitive in light of the graph.

Limiting distribution

We can also use matrix methods to examine the **limiting behavior** or **asymptotics** of Markov chains.

If we start the chain in some state (perhaps chosen at random, or perhaps not) and run the system for a long time, what is the probability of being in the n th state? Formally we want to examine:

$$\lim_{k \rightarrow \infty} p(X_k = n).$$

Fact. If the **transition matrix** is **primitive**, then it follows from a version of the Perron-Frobenius theorem [4] that:

- the equilibrium distribution π is **unique**
-

$$\lim_{k \rightarrow \infty} \mathbf{P}^k = \boldsymbol{\pi} \mathbf{1}'_N$$

Thus, regardless of the initial distribution,

$$\lim_{k \rightarrow \infty} p(X_k = n) = \pi_n,$$

i.e., the limiting distribution of a Markov chain having a primitive transition matrix is the unique equilibrium distribution.

Example.

For our running example with 3 states, the transition matrix *is* primitive.

```
julia> P = [0 1/3 1/2; 1/2 1/3 1/2; 1/2 1/3 0]
0.0  0.333333  0.5
0.5  0.333333  0.5
0.5  0.333333  0.0
```

```
julia> P^5
0.270062  0.285751  0.301312
0.428627  0.428498  0.428627
0.301312  0.285751  0.270062
```

```
julia> P^10
0.286203  0.285714  0.285226
0.428571  0.428571  0.428571
0.285226  0.285714  0.286203
```

If transition matrix P is a **primitive matrix**, then $\lim_{k \rightarrow \infty} P^k$ is a **projection matrix**. (?)

A: True

B: False

??

Irreducible matrix

Define. A square matrix A is called **irreducible** iff

$$\forall i, j, \exists m \in \mathbb{N} \text{ such that } [A^m]_{i,j} > 0.$$

(In general, m depends on i, j .) Otherwise, the matrix is called **reducible**.

There are other equivalent definitions of irreducible matrices.

Any **primitive** matrix is an **irreducible** matrix. (?)

A: True

B: False

??

The **circulant** generator matrix G_N is **irreducible**. (?)

A: True

B: False

??

Fact. If an nonnegative **irreducible** matrix A has at least one non-zero diagonal element, then A is **primitive** [5, p. 678].

Some of the Perron-Frobenius theorem statements hold for a nonnegative irreducible matrix, but not all.

In particular, we lose uniqueness of the maximum eigenvalue magnitude.

Example. For the circulant generator matrix G_N , the eigenvalue magnitudes are all $|e^{-i2\pi k/N}| = 1$.

That is why we focused on **primitive** matrices previously.

Matrix period

Define. For a square nonnegative matrix, the **period of the i th index** is the greatest common divisor of all $m \in \mathbb{N}$ such that $[A^m]_{i,i} > 0$, if any such m exists.

Define. If the period of the i th index exists and is 1 for all i , then A is called **aperiodic**.

Fact. For a **nonnegative irreducible** matrix, the **period of the i th index** always exists and is the same for all i , and is simply called the **period** of the matrix.

Any square **positive** matrix is **aperiodic**. (?)

A: True

B: False

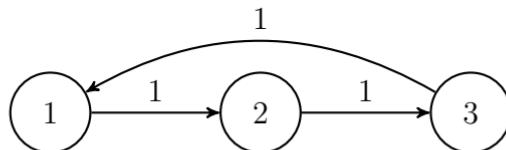
??

Example. The transition matrix corresponding to a loop of 3 states is $P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$.

This matrix is nonnegative and irreducible and it has period 3.

It has three eigenvalues whose magnitude is 1. Its (only) equilibrium distribution is $\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$.

However, P^m does not converge.



Fact. A square nonnegative matrix is **primitive** iff it is **irreducible** and **aperiodic**.

Properties of nonnegative irreducible matrices

The **Perron-Frobenius theorem** for any (square) nonnegative **irreducible matrix** \mathbf{A} states that:

- All the conclusions for (square) nonnegative matrices hold.
- $r \triangleq \rho(\mathbf{A}) > 0$ is a (real) eigenvalue of \mathbf{A} .

In other words, the largest magnitude eigenvalue is a real, *positive* eigenvalue. Think $\lambda_1 = r > 0$.

We call r the **Perron root** or **Perron-Frobenius eigenvalue**.

- That eigenvalue is **simple**, *i.e.*, has **multiplicity one**, meaning both its **algebraic multiplicity** is one (non-repeated root of the characteristic polynomial) and **geometric multiplicity** is one.
- If m denotes the period of \mathbf{A} , then there are exactly m eigenvalues for which $|\lambda| = r = \rho(\mathbf{A})$.
- There exists a unit-norm eigenvector \mathbf{v} with eigenvalue r , *i.e.*, $\mathbf{Av} = r\mathbf{v}$, where \mathbf{v} has all *positive* elements.
- There likewise exists a unit-norm left eigenvector \mathbf{w} having all positive elements for which $\mathbf{w}'\mathbf{A} = r\mathbf{w}'$.
We call this \mathbf{v} and \mathbf{w} the right and left **Perron vectors**, respectively
- Uniqueness: All other (left and right) unit-norm eigenvectors have negative and/or non-real elements.
So \mathbf{v} and \mathbf{w} are unique in that sense.

So in short, the property we “lose” in going from **primitive** matrices to the broader family of **irreducible matrices** is uniqueness of the magnitude of the Perron root. This prevents us from drawing conclusions using the power method.

Strongly connected graphs

Define. A directed graph is **strongly connected** if it is possible to reach any state from any other state (in one or more steps).

Fact. A graph is **strongly connected** iff the corresponding (transition) matrix A is **irreducible**.

All (square) primitive matrices correspond to strongly connected graphs. (?)

A: True

B: False

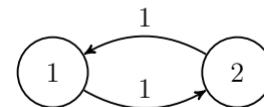
??

Example. The running example with 3 states on p. 7.31 *is* strongly connected.

Example. The three-state loop on p. 7.35 *is* strongly connected.

Example. The graph corresponding to the 3×3 matrix (7.7) *is not* strongly connected.

Example. This two-state Markov chain is **strongly connected**.



The corresponding transition matrix $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ is primitive. (?)

A: True

B: False

??

- The corresponding eigenvalues are $\{+1, -1\}$.
- This 2×2 matrix is irreducible, and its unit-norm Perron vector is $\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$.
- So the (unique) equilibrium distribution of this chain is $\begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$.

Let $\pi^{(n)} \triangleq P^{n-1}\pi^{(1)}$ where $\pi^{(1)} \triangleq \begin{bmatrix} P\{X_1 = 1\} \\ P\{X_1 = 2\} \end{bmatrix}$ denotes the starting probability distribution.

Does $\pi^{(n)}$ approach the equilibrium distribution as $n \rightarrow \infty$?

A: Yes, for any $\pi^{(1)} \geq 0$ s.t. $1'_2 \pi^{(1)} = 1$.

B: Yes, but only for certain $\pi^{(1)}$ choices.

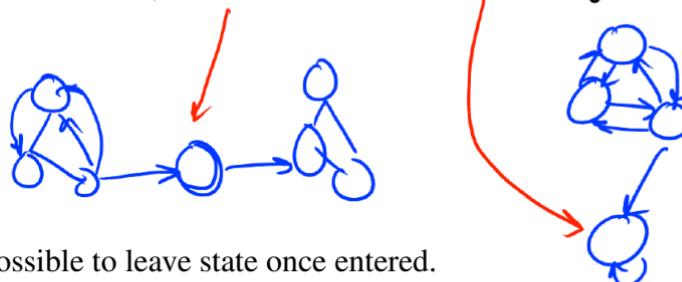
C: No, never, because P^n oscillates between $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

D: None of these.

??

A strongly connected graph cannot have **transient** or **absorbing** states:

A strongly connected graph cannot have transient or absorbing states



- Absorbing state: impossible to leave state once entered.
- Transient state: non-zero probability of never returning.

If a Markov chain has a **strongly connected graph**, then the following properties hold.

- Its transition matrix is **irreducible**.
- An **equilibrium distribution** exists and is *unique* and has all *positive* entries.
- The associated eigenvalue is 1 and is *unique*.
(However, there can be other eigenvalues whose *magnitude* is 1.)

We *cannot* conclude that the Markov chain approaches in the limit that equilibrium distribution.

Making the stronger assumption that the transition matrix is primitive ensures that limiting behavior.

Google's PageRank method

This section uses the preceding ideas to summarize **Google's PageRank** method [4].

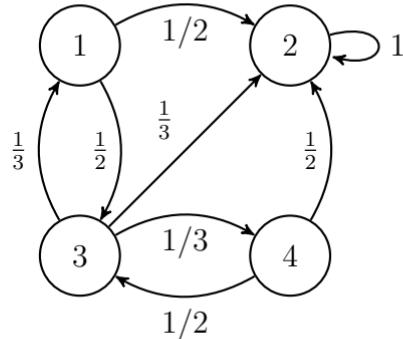
To rank web pages presented after a search, two natural criteria are

- The relevance of each page (number of related terms) etc.
- The “importance” (link popularity) of each web page.

One way to quantify importance is to imagine making a **random walk** between pages, choosing the next page *at random* from all the out links on the current page. This is a Markov chain having a transition matrix:

$$p_{ij} = \begin{cases} 1, & i = j \text{ and page } j \text{ has no out links} \\ 1/N_j, & i \neq j \text{ and page } j \text{ has } N_j \text{ out links, one of which is page } i \\ 0, & \text{otherwise.} \end{cases}$$

Example.



$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1/3 & 0 \\ 1/2 & 1 & 1/3 & 1/2 \\ 1/2 & 0 & 0 & 1/2 \\ 0 & 0 & 1/3 & 0 \end{bmatrix}$$

The idea is that someone who hypothetically walked pages at random this way would tend to visit more important pages more often because important pages have many incoming links.

So to rank pages by importance we would like to find “the” equilibrium distribution of this Markov chain and then rank importance based on sorting the probabilities from largest to smallest.

In practice, the transition matrix \mathbf{P} has columns that sum to one by construction, but the graph of web page outlinks is *not strongly connected*, so \mathbf{P} is not **irreducible**.

- Web has **absorbing states** (“dangling nodes”) *e.g.*, PDF documents with no links and many business sites
- Web is not strongly connected (*e.g.*, probably cannot reach `ctools.umich.edu` from `whitehouse.gov`)

Thus there is not a unique equilibrium distribution.

One solution is to instead use a modified transition matrix:

$$\tilde{\mathbf{P}}_\alpha \triangleq \alpha \mathbf{P} + (1 - \alpha) \frac{1}{N} \mathbf{1}_N \mathbf{1}'_N = \underbrace{\alpha \mathbf{P} + (1 - \alpha) \begin{bmatrix} 1/N & \dots & 1/N \\ \vdots & & \vdots \\ 1/N & \dots & 1/N \end{bmatrix}}_{\text{jump at random to any of the } N \text{ known web pages}}.$$

This matrix $\tilde{\mathbf{P}}_\alpha$ is clearly **positive** by construction, so it has a unique equilibrium distribution π having all positive elements. This vector π has been called the \$25 billion eigenvector [6].

PageRank algorithm

- Generate adjacency matrix by web crawling. It is (extremely) sparse, so store accordingly
- Form the transition matrix P . Again, this is (extremely) sparse.
- Select value for **damping factor** α . The original paper by Sergei Brin and (UM alum) Larry Page used $\alpha = 0.85$, apparently based on how often users return to their bookmarks.
- Pick an initial distribution vector π_0 at random from unit simplex (nonnegative entries and sums to 1).
- Perform a modified version of the power iteration where we keep the sum equal to one instead of the Euclidean norm equal to one:

$$\pi_{k+1} = \tilde{P}_\alpha \pi_k = \alpha P \pi_k + \frac{1 - \alpha}{N} \mathbf{1}_N.$$

- Perform 40-50 iterations (according to original paper)
- Use the final π vector (sorting?) to quantify the “importance” of web pages.

Example. Applying this method to the preceding 4-state example with $\alpha = 0.85$ yields $\pi = \begin{bmatrix} 0.0634 \\ 0.7818 \\ 0.0914 \\ 0.0634 \end{bmatrix}$.

State 2 is the most important and state 3 is the next, and states 1 and 4 are tied for last. In this particular example, we could have ranked states based on the number of incoming links. However, Brin and Page argue in their **patent** that link counting is suboptimal in general; it is more informative if your web page is linked by other important web pages, not just by many (possibly spam) web pages.

PageRank computation

(Read)

A key step in the algorithm is multiplying the $N \times N$ sparse matrix \mathbf{P} by a (non-sparse) vector: $\mathbf{P}\pi_k$. Using ordinary matrix multiplication this would require N^2 operations.

In 2016, N was over 130 trillion says [this article](#). Google's page says [hundreds of billions](#).

Fortunately, \mathbf{P} is extremely sparse; most web pages link to just a few other pages, and the average number of outgoing links is probably roughly a constant that does not really scale with N . Using the fact that \mathbf{P} is sparse, with an average of, say, $L \ll N$ links per page, the product $\mathbf{P}\pi_k$ requires "just" $O(NL)$ operations. This is still enormous, requiring massive computation and considerable energy.

If N is about 100 trillion, then $1/N$ is about 10^{-14} which is smaller than machine precision (about 10^{-7}) of 32-bit floating point numbers, so apparently π_k must be stored as a 64-bit (double precision) vector. The number of bytes just to stored π is then $8N$ which is over 700 petabytes. Surely Google must have some tricks to deal with this.

PageRank is independent of a user's query so it can be done relatively infrequently (hourly? daily? weekly?) instead of for every individual search.

Numerous practical issues arise to deal with web sites trying to boost scores artificially.

PageRank summary

One of the most influential algorithms of modern times has its roots in Markov chains, directed graphs, positive matrices, and eigenvector computation using the power method.

For more reading, see the [article](#) by Mathworks founder **Cleve Moler** and the [related MATLAB demo](#).

7.5 Summary

This chapter has introduced just a few of the many important special matrices used in practice.

Another application of the Perron-Frobenius theorem is graph matching [7].

The following table summarizes the relationship between the properties of transition matrix of a Markov chain and its corresponding graph.

Transition matrix	Markov Chain graph	Key property
Nonnegative	All	Equilibrium distribution(s) exist with eigenvalue 1.
Irreducible	Strongly connected	" Unique, positive equilibrium distribution π
Primitive	Strongly connected (and more?)	" $P^k \rightarrow \pi \mathbf{1}'$
Positive	“Fully” connected	"

Other properties to consider when reviewing:

- $\rho \geq 0$ vs $\rho > 0$
- $\lambda_1 = \rho$ vs $|\lambda_1| = \rho$
- uniqueness of λ_1 (can any other $\lambda_i = \lambda_1$?)
- uniqueness of $|\lambda_1|$ (can any other $|\lambda_i| = |\lambda_1|$?)
- existence of $v_1 \geq 0$ vs existence of $v_1 > 0$

For any Markov chain with a $N \times N$ circulant transition matrix, the vector $\pi = \mathbf{1}_N/N$ is an equilibrium distribution of the chain. (?)

A: True

B: False

??

For any Markov chain with a $N \times N$ circulant transition matrix, the vector $\pi = \mathbf{1}_N/N$ is the unique equilibrium distribution of the chain. (?)

A: True

B: False

??

For any Markov chain with a $N \times N$ circulant transition matrix, exactly one eigenvalue of that transition matrix has magnitude equal to 1. (?)

A: True

B: False

??

If a monic polynomial has all negative coefficients (except for the leading coefficient), then the corresponding companion matrix has an eigenvector with all positive elements. (?)

A: True

B: False

??

The companion matrix for the polynomial $p(x) = x^2 - 2x - 3$ has spectral radius:

A: 1

B: 2

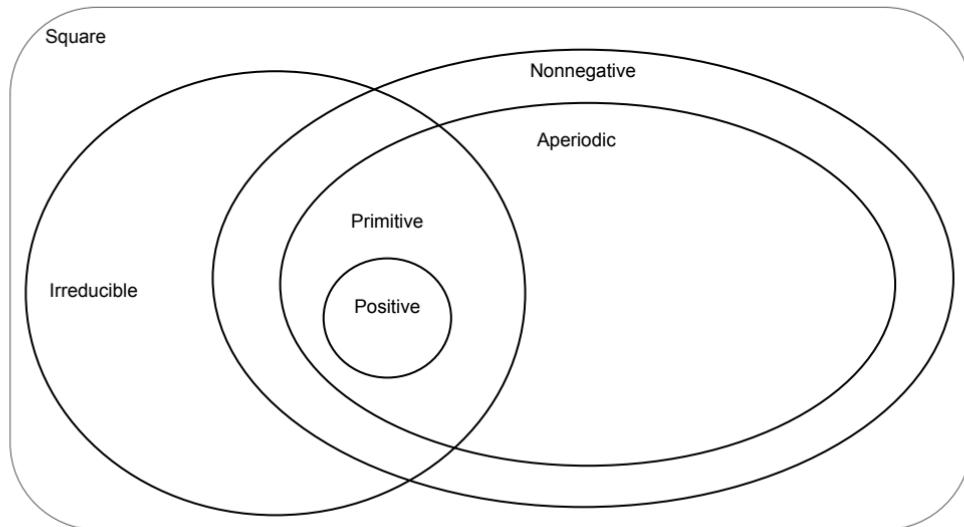
C: 3

D: 4

E: 5

??

This diagram summarizes the relationships between many of the matrix types in this chapter.



An important category that is missing from this diagram is **transition** matrices.

Exercise. Complete the Venn diagram by adding **transition** matrices.

(in class) ??

Bibliography

- [1] M. Akcakaya and V. Tarokh. “A frame construction and a universal distortion bound for sparse representations”. In: *IEEE Trans. Sig. Proc.* 56.6 (June 2008), 2443–50.
- [2] A. J. Laub. *Matrix analysis for scientists and engineers*. Soc. Indust. Appl. Math., 2005.
- [3] J. W. Cooley and J. W. Tukey. “An algorithm for machine calculation of complex Fourier series”. In: *Mathematics of Computation* 19.90 (Apr. 1965), 297–301.
- [4] S. U. Pillai, T. Suel, and S. Cha. “The Perron-Frobenius theorem: some of its applications”. In: *IEEE Sig. Proc. Mag.* 22.2 (Mar. 2005), 62–75.
- [5] C. D. Meyer. *Matrix analysis and applied linear algebra*. Soc. Indust. Appl. Math., 2000.
- [6] K. Bryan and T. Leise. “The \$25,000,000,000 eigenvector: The linear algebra behind Google”. In: *SIAM Review* 48.3 (Sept. 2006), 569–81.
- [7] S. Feizi, G. Quon, M. Recamonde-Mendoza, M. Medard, M. Kellis, and A. Jadbabaie. *Spectral alignment of graphs*. 2017.

Chapter 8

Optimization basics

Contents (final version)

8.0 Introduction	8.2
8.1 Preconditioned gradient descent (PGD) for LS	8.3
Tool: Matrix square root	8.4
Convergence rate analysis of PGD: first steps	8.7
Tool: Matrix powers	8.8
Classical GD: step size bounds	8.10
Optimal step size for GD	8.11
Practical step size for GD	8.12
Ideal preconditioner for PGD	8.13
Tool: Positive (semi)definiteness properties	8.14
General preconditioners for PGD	8.16
Diagonal majorizer	8.17
Tool: commuting (square) matrices	8.21
8.2 Preconditioned steepest descent	8.25
8.3 Gradient descent for smooth convex functions	8.26

8.4 Machine learning via logistic regression for binary classification	8.33
8.5 Summary	8.39

8.0 Introduction

Many of the previous topics have involved **optimization** formulations: linear LS, Procrustes, low-rank approximation, multidimensional scaling. In all these cases we derived analytical solutions, like the pseudo-inverse for minimum-norm LS problems and the truncated SVD for low-rank approximation.

But often we need iterative optimization algorithms, *e.g.*,

- if no closed-form minimizer exists, or
- if the analytical solution requires too much computation and/or memory, *e.g.*, SVD for large problems.

To solve a problem like $\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} f(\mathbf{x})$ via an iterative method, we start with some initial guess \mathbf{x}_0 , and then the algorithm produces a sequence $\{\mathbf{x}_k\}$ where hopefully the sequence **converges** to $\hat{\mathbf{x}}$, meaning $\|\mathbf{x}_k - \hat{\mathbf{x}}\| \rightarrow 0$ for some norm $\|\cdot\|$ as $k \rightarrow \infty$, as discussed in Ch. 5.

The homework has introduced a few such optimization algorithms. This chapter elaborates on those. Along the way we introduce several new matrix concepts: **matrix square root**, **matrix powers**, more about **positive (semi)definite matrices**, **commuting matrices**, and **majorization**.

8.1 Preconditioned gradient descent (PGD) for LS

For the (possibly regularized) LS problem with $\mathbf{A} \in \mathbb{F}^{M \times N}$ the **cost function** is **quadratic**:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} f(\mathbf{x}), \quad f(\mathbf{x}) = \underbrace{\frac{1}{2} \|\mathbf{Ax} - \mathbf{y}\|_2^2}_{f : \mathbb{F}^N \mapsto \mathbb{R}}, \quad \nabla f(\mathbf{x}) = \underbrace{\mathbf{A}' (\mathbf{Ax} - \mathbf{y})}_{\nabla f : \mathbb{F}^N \mapsto \mathbb{F}^N}.$$

The homework problems focused on the classical **gradient descent (GD)** iterative method:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k) = \mathbf{x}_k - \alpha \mathbf{A}' (\mathbf{Ax}_k - \mathbf{y}),$$

where convergence of the sequence $\{\mathbf{x}_k\}$ is ensured (according to HW) by choosing the **step size** α to satisfy

$$0 < \alpha < \frac{2}{\sigma_1(\mathbf{A}' \mathbf{A})} = \frac{2}{\sigma_1^2(\mathbf{A})}. \quad (8.1)$$

Now consider a generalization called **preconditioned gradient descent (PGD)**:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{P} \nabla f(\mathbf{x}_k) = \mathbf{x}_k - \mathbf{P} \mathbf{A}' (\mathbf{Ax}_k - \mathbf{y}), \quad (8.2)$$

where \mathbf{P} is a $N \times N$ **preconditioning matrix**. Classical gradient descent simply uses $\mathbf{P} = \alpha \mathbf{I}_N$. But a single step size α may be suboptimal (and hard to choose), especially if different elements of \mathbf{x} have different units.

What conditions on \mathbf{P} ensure convergence? Will some other $\mathbf{P} \neq \alpha \mathbf{I}$ provide faster convergence? Why does (8.1) suffice? Answering these questions will use many matrix methods!

Tool: Matrix square root

We need another linear algebra tool first. (And more will come before we finish this topic.)

Define. We call S a **(matrix) square root** of a square matrix P iff $SS = P$.

It is not unique: if S is a square root of P , then $-S$ is also square root of P .

If P is **positive semidefinite**, i.e., $P \succeq 0$, then P has a positive semidefinite square root.

Proof: $P \succeq 0 \implies P = V\Lambda V'$, where the eigenvalues are all real and nonnegative, so let $S = V\Lambda^{1/2}V'$ where $\Lambda^{1/2} \triangleq \text{diag}\{\sqrt{\lambda_i}\}$. Clearly $SS = P$ and $S \succeq 0$.

If P is **positive definite**, $P \succ 0$, then P has a positive definite (and hence invertible) square root.

Proof: Same as above only now the eigenvalues are all positive.

Define. We write $P^{1/2}$ to denote the positive (semi)definite square root of P ; we use this notation only if P is positive (semi)definite. When being careful, we call $P^{1/2}$ the **principal square root**, but often we just call it “the square root” of P , just like people say “the square root of 9 is 3.”

Every **diagonalizable** matrix has a **matrix square root**.

A: True

B: False

??

Example. If $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ then $S = \begin{bmatrix} 1 & 1/2 \\ 0 & 1 \end{bmatrix}$ is a matrix square root of A .

If a square matrix has a **matrix square root**, then it is **diagonalizable**.

A: True

B: False

??

Example.

For $x \neq 0$, consider $A = xx' \succeq 0$. Which of the following is a matrix square root of A ?

- A: xx' B: $xx'/\|x\|_2$ C: $xx'/\|x\|_2^2$ D: $xx'\|x\|_2$ E: $xx'\|x\|_2^2$

??

Example.

If Q is a unitary matrix, then Q has a matrix square root.

- A: True B: False

??

??

Practical implementation of matrix square root _____

For a square matrix, the command for finding a **matrix square root** is:

`sqrt(A)`

This operation differs *completely* from element-wise root:

`sqrt.(A)`



In MATLAB (and old JULIA versions) the operation is

`sqrtm(A)`

Caution. If A is square but not diagonalizable then the output of `sqrt` can be meaningless.

Try `sqrt([0 1; 0 0])`

Hereafter we assume the **preconditioner** \mathbf{P} in PGD (8.2) is **positive definite**, i.e., $\mathbf{P} \succ \mathbf{0}$, so that $\mathbf{P}^{1/2}$ exists and is invertible. We denote its inverse by $\mathbf{P}^{-1/2}$.

Convergence rate analysis of PGD: first steps

Let $\hat{\mathbf{x}}$ denote any minimizer of $f(\mathbf{x})$. That $\hat{\mathbf{x}}$ must satisfy the normal equations: $\mathbf{A}'\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}'\mathbf{y}$. For analysis purposes, replace $\mathbf{A}'\mathbf{y}$ in (8.2) with $\mathbf{A}'\mathbf{A}\hat{\mathbf{x}}$:

$$\begin{aligned}
 \mathbf{x}_{k+1} &= \mathbf{x}_k - \mathbf{P}\mathbf{A}'(\mathbf{A}\mathbf{x}_k - \mathbf{y}) = \mathbf{x}_k - \mathbf{P}(\mathbf{A}'\mathbf{A}\mathbf{x}_k - \mathbf{A}'\mathbf{y}) \\
 &= \mathbf{x}_k - \mathbf{P}(\mathbf{A}'\mathbf{A}\mathbf{x}_k - \mathbf{A}'\mathbf{A}\hat{\mathbf{x}}) = \mathbf{x}_k - \mathbf{P}\mathbf{A}'\mathbf{A}(\mathbf{x}_k - \hat{\mathbf{x}}) \\
 \implies \mathbf{x}_{k+1} - \hat{\mathbf{x}} &= \mathbf{x}_k - \hat{\mathbf{x}} - \mathbf{P}\mathbf{A}'\mathbf{A}(\mathbf{x}_k - \hat{\mathbf{x}}) \\
 &= (\mathbf{I} - \mathbf{P}\mathbf{A}'\mathbf{A})(\mathbf{x}_k - \hat{\mathbf{x}}) \\
 \implies \underbrace{\mathbf{P}^{-1/2}(\mathbf{x}_{k+1} - \hat{\mathbf{x}})}_{\delta_{k+1}} &= \mathbf{P}^{-1/2}(\mathbf{I} - \mathbf{P}\mathbf{A}'\mathbf{A})(\mathbf{x}_k - \hat{\mathbf{x}}) \\
 &= (\mathbf{I} - \mathbf{P}^{1/2}\mathbf{A}'\mathbf{A}\mathbf{P}^{1/2}) \underbrace{\mathbf{P}^{-1/2}(\mathbf{x}_k - \hat{\mathbf{x}})}_{\delta_k} \\
 \implies \delta_{k+1} &= (\mathbf{I} - \mathbf{P}^{1/2}\mathbf{A}'\mathbf{A}\mathbf{P}^{1/2}) \delta_k = \mathbf{G}\delta_k, \quad \delta_k \triangleq \mathbf{P}^{-1/2}(\mathbf{x}_k - \hat{\mathbf{x}}) \text{ (mod. error vector)} \\
 &\quad \mathbf{G} \triangleq \mathbf{I} - \mathbf{P}^{1/2}\mathbf{A}'\mathbf{A}\mathbf{P}^{1/2} \\
 \implies \delta_k &= \mathbf{G}^k \delta_0.
 \end{aligned}$$

The matrix \mathbf{G} “governs” the convergence of PGD.

Tool: Matrix powers

Now we need **matrix powers** to proceed, another easy byproduct of **eigendecomposition**.

Here $G = G'$, so $G = V\Lambda V'$ for some unitary V and Λ is real (but *not* nonnegative in general).

So $G^2 = GG = V\Lambda V'V\Lambda V' = V\Lambda^2 V'$ and more generally $G^k = V\Lambda^k V'$, $\forall k \in \mathbb{N}$.

If A is **diagonalizable** and **invertible**, is A^k diagonalizable and invertible for all $k \in \mathbb{N}$?

A: Yes

B: No

C: Depends

??

PGD convergence condition using eigenvalues

So we can express the modified error vector δ_k in terms of the initial error:

$$\delta_k = \mathbf{V} \Lambda^k \mathbf{V}' \delta_0.$$

Define $\tilde{\delta}_k \triangleq \mathbf{V}' \delta_k$ to be the error coordinates in the \mathbf{V} basis, then

$$\tilde{\delta}_k = \Lambda^k \tilde{\delta}_0 = \text{diag}\{\lambda_i^k\} \tilde{\delta}_0.$$

We have reduced the question of convergence of PGD down to seeing whether $\{\lambda_i^k\}$ is a decreasing geometric series. We need $-1 < \lambda_i < 1$ for all $i = 1, \dots, N$ to ensure that all error components diminish to zero.

Put concisely, a necessary and sufficient condition to ensure convergence of PGD from any initializer x_0 is:

$$\rho(\mathbf{G}) = \rho(\mathbf{I} - \mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2}) < 1, \quad i.e., \quad -1 < \text{eig}\{\mathbf{I} - \mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2}\} < 1. \quad (8.3)$$

Classical GD: step size bounds

Consider the classical case where $\mathbf{P} = \alpha\mathbf{I}$. Then the above condition (8.3) simplifies to

$$-1 < \text{eig}\{\mathbf{I} - \alpha\mathbf{A}'\mathbf{A}\} < 1, \quad \iff \quad -1 < 1 - \alpha\sigma_i^2 < 1, \quad \iff \quad 0 < \alpha\sigma_i^2 < 2,$$

where $\{\sigma_i\}$ denotes the singular values of \mathbf{A} .

The lower bound condition $0 < \alpha\sigma_i^2$ holds for all i if $0 < \alpha$ and if \mathbf{A} has full column rank (e.g., when Tikhonov regularization is used). So even though the analysis did not assume full rank at the start, if we want to make sure that all error modes diminish to zero, we must have full (column) rank matrix \mathbf{A} .

The upper bound condition $\alpha\sigma_i^2 < 2$ holds for all i if $\alpha\sigma_1^2 < 2$ because σ_1 is the largest, so we need $\alpha < 2/\sigma_1^2$.

In summary, we derived the step-size condition (8.1) for convergence of classical GD for LS problem:

$$0 < \alpha < \frac{2}{\sigma_1^2(\mathbf{A})} = \frac{2}{\sigma_1(\mathbf{A}'\mathbf{A})}.$$

Optimal step size for GD

For classical GD, where $\mathbf{P} = \alpha\mathbf{I}$, the *optimal* step size (for fastest convergence) is: (HW)

$$\alpha_* = \frac{2}{\sigma_1^2(\mathbf{A}) + \sigma_N^2(\mathbf{A})}. \quad (8.4)$$

This step size makes $\rho(\mathbf{I} - \alpha\mathbf{A}'\mathbf{A})$ as small as possible, *i.e.*, so that its largest absolute eigenvalue is as close to zero as possible. That largest eigenvalue will be the mode that converges to zero the slowest, so it will ultimately govern the convergence rate.

However, the step-size conditions (8.1) and (8.4) are unsatisfying practically because if a LS problem is so large that we cannot use the SVD to compute the pseudo-inverse, then probably we do not really want to use an SVD (or `svds`) to find $\sigma_1(\mathbf{A})$ and perhaps also $\sigma_N(\mathbf{A})$.

If \mathbf{A} is unitary, what is the best step size α when $\mathbf{P} = \alpha\mathbf{I}$?

- A: 0 B: 1 C: 1.99 D: 2 E: None of these

??

Suppose $\mathbf{P} = \alpha(\mathbf{A}'\mathbf{A})^{-1}$ for some $\alpha \geq 0$. What is the best value of α , so that the eigenvalues of \mathbf{G} are as small (in magnitude) as possible, so PGD converges as fast as possible?

- A: 0 B: 1/2 C: 1 D: $\sqrt{2}$ E: $2 - \epsilon$

??

Practical step size for GD

One option for avoiding the SVD is to use the bound from HW5:

$$\sigma_1(\mathbf{A}) = \|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_{\text{F}}.$$

So choosing $0 < \alpha < 2/\|\mathbf{A}\|_{\text{F}}^2$ always provides a valid step size. This is useful because it is easy to compute $\|\cdot\|_{\text{F}}$ because no SVD is needed. However, the upper bound above is often quite loose. If $\mathbf{A} = \mathbf{I}_N$, then $\|\mathbf{A}\|_2 = 1$ but $\|\mathbf{A}\|_{\text{F}} = \sqrt{N}$. So using this bound, the step size α would be much smaller than necessary.

Another option is to recall that because $\mathbf{A}'\mathbf{A}$ is symmetric:

$$\|\mathbf{A}'\mathbf{A}\|_2 = \sigma_1(\mathbf{A}'\mathbf{A}) = \rho(\mathbf{A}'\mathbf{A}) \leq \|\mathbf{A}'\mathbf{A}\|,$$

for any matrix norm $\|\cdot\|$. In particular, as discuss in Ch. 5, $\|\mathbf{A}'\mathbf{A}\|_1 \leq \|\mathbf{A}'\|_1 \|\mathbf{A}\|_1 = \|\mathbf{A}\|_{\infty} \|\mathbf{A}\|_1$. So convergence is always ensured by choosing:

$$0 < \alpha < \frac{2}{\|\mathbf{A}'\mathbf{A}\|_1} \quad \text{or} \quad 0 < \alpha < \frac{2}{\|\mathbf{A}\|_{\infty} \|\mathbf{A}\|_1}.$$

Why did I choose $\|\mathbf{A}'\mathbf{A}\|_1$ instead of $\|\mathbf{A}'\mathbf{A}\|_{\infty}$?

Which norm is bigger:

- A: $\|\mathbf{A}'\mathbf{A}\|_1$ usually B: $\|\mathbf{A}'\mathbf{A}\|_1$ always C: $\|\mathbf{A}'\mathbf{A}\|_{\infty}$ usually D: $\|\mathbf{A}'\mathbf{A}\|_{\infty}$ always E: They are the same.

??

Ideal preconditioner for PGD

Before looking at more general choices for the preconditioner \mathbf{P} , we first explore the “ideal” \mathbf{P} .

The ideal preconditioner is $\mathbf{P}_{\text{ideal}} = (\mathbf{A}' \mathbf{A})^{-1}$,
because for that choice:

$$\mathbf{G} = \mathbf{I} - \mathbf{P}_{\text{ideal}}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}_{\text{ideal}}^{1/2} = \mathbf{I} - (\mathbf{A}' \mathbf{A})^{-1/2} \mathbf{A}' \mathbf{A} (\mathbf{A}' \mathbf{A})^{-1/2} = \mathbf{I} - \mathbf{I} = \mathbf{0},$$

so $\boldsymbol{\delta}_{(1)} = \mathbf{G}^1 \boldsymbol{\delta}_0 = \mathbf{0} \boldsymbol{\delta}_0 = \mathbf{0}$, i.e., PGD converges in one iteration:

$$\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{P}_{\text{ideal}} \mathbf{A}' (\mathbf{A} \mathbf{x}_0 - \mathbf{y}) = \mathbf{x}_0 - (\mathbf{A}' \mathbf{A})^{-1} \mathbf{A}' (\mathbf{A} \mathbf{x}_0 - \mathbf{y}) = (\mathbf{A}' \mathbf{A})^{-1} \mathbf{A}' \mathbf{y} = \mathbf{A}^+ \mathbf{y}.$$

Why not always use this ideal preconditioner?

Inverting $\mathbf{A}' \mathbf{A}$ is expensive: it is as hard as the original LS problem. So we look for other preconditioners. Specifically we would like to find a preconditioner $\mathbf{P} \approx (\mathbf{A}' \mathbf{A})^{-1}$ or $\mathbf{P}^{-1} \approx \mathbf{A}' \mathbf{A}$ but where the inverse is much easier to compute.

For that we need another tool.

Tool: Positive (semi)definiteness properties

Recall that we define positive (semi)definiteness only for Hermitian matrices and:

- **positive semidefinite:** $\mathbf{A} \succeq \mathbf{0} \iff \mathbf{x}' \mathbf{A} \mathbf{x} \geq 0, \forall \mathbf{x}$
- **positive definite:** $\mathbf{A} \succ \mathbf{0} \iff \mathbf{x}' \mathbf{A} \mathbf{x} > 0, \forall \mathbf{x} \neq \mathbf{0}$

Properties (\mathbf{A} and \mathbf{B} are all Hermitian here):

- $\mathbf{X}' \mathbf{X} \succeq \mathbf{0}$ for *any* matrix \mathbf{X} , even non-square (shown previously)
- $\mathbf{A} \succeq \mathbf{0} \implies \mathbf{X}' \mathbf{A} \mathbf{X} \succeq \mathbf{0}$, for any matrix \mathbf{X} with `size(A, 2) == size(X, 1)`
- $\mathbf{A} \succeq \mathbf{0}$ and $\alpha \geq 0 \implies \alpha \mathbf{A} \succeq \mathbf{0}$
- $\mathbf{A} \succeq \mathbf{0} \implies$ all eigenvalues of \mathbf{A} are real and nonnegative

Proof 1: if \mathbf{x} is an eigenvector of \mathbf{A} then $0 \leq \mathbf{x}' (\mathbf{A} \mathbf{x}) = \mathbf{x}' \lambda \mathbf{x} = \lambda \|\mathbf{x}\|_2^2 \implies \lambda \geq 0$

Proof 2: \mathbf{A} Hermitian implies it has an orthogonal eigendecomposition: $\mathbf{A} = \mathbf{V} \Lambda \mathbf{V}'$ so $\mathbf{V} \Lambda \mathbf{V}' \succeq \mathbf{0}$.

Using 2nd property above: $\implies \mathbf{V}' (\mathbf{V} \Lambda \mathbf{V}') \mathbf{V} \succeq \mathbf{0} \implies \Lambda \succeq \mathbf{0} \implies \mathbf{e}_j' \Lambda \mathbf{e}_j \geq 0 \implies \lambda_j \geq 0$.

- $\mathbf{A} \succ \mathbf{0} \implies \mathbf{A}$ invertible and $\mathbf{A}^{-1} \succ \mathbf{0}$
- $\mathbf{B} \succeq \mathbf{A} \iff \mathbf{B} - \mathbf{A} \succeq \mathbf{0}$ (trivial from definition of \succeq)
- $\mathbf{B} \succ \mathbf{A} \iff \mathbf{B} - \mathbf{A} \succ \mathbf{0}$
- $\mathbf{B} \succeq \mathbf{A} \succ \mathbf{0} \implies \mathbf{A}^{-1} \succeq \mathbf{B}^{-1} \succ \mathbf{0}$
- $\mathbf{B} \succeq \mathbf{A} \succ \mathbf{0} \implies \gamma \mathbf{B} \succ \mathbf{A}, \forall \gamma > 1$
- $\mathbf{A} \succ \mathbf{0}, \mathbf{B} \succ \mathbf{0} \implies \mathbf{B} \mathbf{A} \mathbf{B} \succ \mathbf{0}$ (HW)

Notice the pattern: start with a definition, then develop properties (especially addition and multiplication).

The above properties relate to multiplication; now consider **matrix addition**.

If $A \succeq 0$ and $B \succeq 0$ have the same size, then $A + B \succeq 0$. (?)

A: True

B: False

??

If $A \succ 0$ and $B \succeq 0$ have the same size, then $A + B \succ 0$. (?)

A: True

B: False

??

The above properties are all useful and important for GD for LS because:

- we choose $P \succ 0$
- and we assume $A'A \succ 0$.

General preconditioners for PGD

Repeating (8.3), for convergence we want

$$-1 < \text{eig}\{\mathbf{I} - \mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2}\} < 1. \quad (8.5)$$

Now the RHS of (8.5) is easy; assuming $\mathbf{A}'\mathbf{A}$ and \mathbf{P} are positive definite:

$$\text{eig}\{\mathbf{I} - \mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2}\} = 1 - \underbrace{\text{eig}\{\mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2}\}}_{> 0} < 1.$$

For the LHS of (8.5), we want $-1 < \text{eig}\{\mathbf{I} - \mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2}\} = 1 - \text{eig}\{\mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2}\}$, i.e., (HW)

$$\text{eig}\{\mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2}\} < 2 \iff 2\mathbf{I} \succ \mathbf{P}^{1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{1/2} \iff 2\mathbf{P}^{-1} \succ \mathbf{A}' \mathbf{A}.$$

Matrix majorizers

Define. We say that a (Hermitian) symmetric matrix \mathbf{A} is a **majorizer** of (or **majorizes**) a (Hermitian) symmetric matrix \mathbf{B} iff $\mathbf{A} \succeq \mathbf{B}$, i.e., iff $\mathbf{A} - \mathbf{B}$ is positive semidefinite.

Summarizing: if $\beta\mathbf{P}^{-1}$ **majorizes** $\mathbf{A}'\mathbf{A}$ for some $0 < \beta < 2$, then convergence of PGD is ensured, because $\beta\mathbf{P}^{-1} \succeq \mathbf{A}'\mathbf{A} \implies 2\mathbf{P}^{-1} \succ \mathbf{A}'\mathbf{A}$.

Diagonal majorizer

Fact (diagonal majorizer). If \mathbf{B} is (Hermitian) symmetric and $N \times N$, then: (HW)

$$\text{diag}\{|\mathbf{B}| \mathbf{1}_N\} \succeq \mathbf{B}, \quad (8.6)$$

where here $|\mathbf{B}|$ denotes the element-wise absolute value of \mathbf{B} , i.e., `abs.(B)` in JULIA.

Thus, a valid preconditioner for PGD is $\mathbf{P}^{-1} = \alpha^{-1} \text{diag}\{|\mathbf{A}'\mathbf{A}| \mathbf{1}_N\}$ for any $0 < \alpha < 2$, leading to the **diagonally preconditioned GD** iteration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{D} \mathbf{A}' (\mathbf{A} \mathbf{x}_k - \mathbf{y}), \quad \mathbf{D} \triangleq \text{diag}\{|\mathbf{A}'\mathbf{A}| \mathbf{1}_N\}^{-1}, \quad 0 < \alpha < 2. \quad (8.7)$$

Typically we use $1 \leq \alpha \leq 1.99$ and it is easy to adjust α in this range to find good value.

If $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 1 & -1 \end{bmatrix}$ then what is the diagonal preconditioner \mathbf{D} in (8.7) ?

- | | | | |
|---|---|---|---|
| A: $\begin{bmatrix} 1/3 & 0 \\ 0 & 1/6 \end{bmatrix}$ | B: $\begin{bmatrix} 1/2 & 0 \\ 0 & 1/3 \end{bmatrix}$ | C: $\begin{bmatrix} 1/2 & 0 \\ 0 & 1/5 \end{bmatrix}$ | D: $\begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix}$ |
| E: None of these. | | | |

??

The “normalization” by the inverse matrix \mathbf{P} in (8.7) makes finding α much easier than in classical GD where changing the units of the problem (units of \mathbf{x} or \mathbf{y} and hence \mathbf{A}) necessitate finding a new α value.

The following code compares $\text{eig}\{\mathbf{I} - \mathbf{P}^{-1/2} \mathbf{A}' \mathbf{A} \mathbf{P}^{-1/2}\}$ for

- classical GD with $\mathbf{P} = \alpha_* \mathbf{I}$ for the optimal step size α_* ,
- versus diagonally preconditioned GD with $\alpha = 1.3$ chosen empirically.

The former has eigenvalues ± 0.5151 , and the latter has eigenvalues $(-0.3, 0.35)$ so at least in this example the diagonal preconditioner accelerates convergence, without requiring an eigendecomposition to find \mathbf{P} .

```
using LinearAlgebra
A = [1 0; 0 2; 1 -1]
@show A'A
D = Diagonal(1 ./ (abs.(A'A) * ones(2))) # diagonal preconditioner
alphal = 1.3
P1 = alphal * D
G1 = I - sqrt(P1) * (A' * A) * sqrt(P1)
@show eigvals(G1)

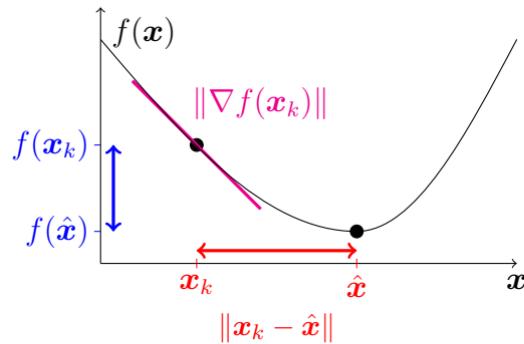
alphabest = 2 / (sum(svdvals(A'A)[[1, end]])) # optimal GD step size
G2 = I - alphabest * (A' * A)
@show eigvals(G2)
```

We have focused on the quadratic problem here, but the principles of diagonal majorization apply to using GD (and accelerated GD) in general for convex optimization of cost functions having Lipschitz gradients [1].

Convergence rates

There are many ways to assess the convergence rate of an iterative algorithm like PGD. Researchers study:

- $f(\mathbf{x}_k) \rightarrow f(\hat{\mathbf{x}})$
- $\|\nabla f(\mathbf{x}_k)\| \rightarrow 0$
- $\|\mathbf{x}_k - \hat{\mathbf{x}}\| \rightarrow 0$
- $\|\delta_k\| \rightarrow 0, \quad \delta_k \triangleq \mathbf{P}^{-1/2} (\mathbf{x}_k - \hat{\mathbf{x}})$
- $\|\tilde{\delta}_k\| \rightarrow 0, \quad \tilde{\delta}_k \triangleq \mathbf{V}' \delta_k$



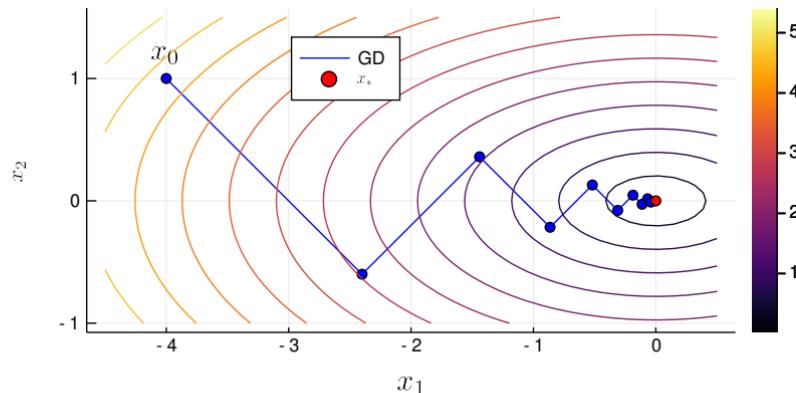
Quantifying bounds on the rates of decrease of these quantities is an active research area. Even classical GD has relatively recent results [2] that tighten up the traditional bounds. The tightest possible worst-case bound for GD for the decrease of the cost function (with a fixed step size $\alpha = 1/L$) is $O(1/k)$:

$$f(\mathbf{x}_k) - f(\hat{\mathbf{x}}) \leq \frac{\|\mathbf{x}_0 - \hat{\mathbf{x}}\|_2^2}{L(4k + 2)},$$

where L is the Lipschitz constant of the gradient $\nabla f(\mathbf{x})$.

In contrast, Nesterov's fast gradient method has a worst-case cost function decrease at rate at least $O(1/k^2)$, which can be improved (and has) by only a constant factor [3].

Example. The following figure illustrates how slow GD can converge for a simple LS problem with $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ and $y = 0$. This case used the optimal step size α_* for illustration. This slow convergence has been the impetus of thousands of papers on faster algorithms!



The ellipses show the contours of the LS cost function $\|Ax - y\|$.

What is the optimal step size α_* here?

A: 1/5

B: 2/5

C: 3/5

D: 1/3

E: 2/3

??

Tool: commuting (square) matrices

Let X and Y denote square matrices of the same size.

Which of the following conditions guarantee that X and Y commute, i.e., $XY = YX$.

Choose the most general correct combination of condition(s).

- 1 X diagonalizable
- 2 Y diagonalizable
- 3 X and Y have same eigenvectors (i.e., are simultaneously diagonalizable)
- 4 X (Hermitian) symmetric
- 5 Y (Hermitian) symmetric

A: 1 & 2

B: 1 & 2 & 3

C: 1 & 2 & 3 & (4 or 5)

D: 1 & 2 & 3 & 4 & 5

E: None of these suffice.

??

All $N \times N$ **companion** matrices commute. (?)

A: True

B: False

??

??

All $N \times N$ **circulant** matrices commute. (?)

A: True

B: False

??

If \mathbf{X} and \mathbf{Y} are positive semidefinite and are simultaneously diagonalizable then

$$\mathbf{XY} = \mathbf{YX} = \mathbf{X}^{1/2}\mathbf{Y}\mathbf{X}^{1/2} = \mathbf{Y}^{1/2}\mathbf{X}\mathbf{Y}^{1/2}.$$

(?)

A: True

B: False

??

Being simultaneously diagonalizable is a sufficient condition. A necessary (but not sufficient) condition for **commuting matrices** is that they are **simultaneously triangularizable**.

Example. The matrices $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ commute, but the first is not diagonalizable.

Monotonicity

Some algorithms get closer to the solution every iteration, and/or decrease the cost function every iteration, whereas other algorithms have no such guarantee.

For PGD, there are many quantities we can evaluate to see if they do (or do not) **decrease monotonically**, where to be precise we really mean “non-increasing” but usually we say “decreasing” for brevity:

- $f(\mathbf{x}_{k+1}) \stackrel{?}{\leq} f(\mathbf{x}_k)$ cost function
- $\|\nabla f(\mathbf{x}_{k+1})\| \stackrel{?}{\leq} \|\nabla f(\mathbf{x}_k)\|$ gradient norm
- $\|\mathbf{x}_{k+1} - \hat{\mathbf{x}}\| \stackrel{?}{\leq} \|\mathbf{x}_k - \hat{\mathbf{x}}\|$ distance to solution in natural coordinates
- $\|\boldsymbol{\delta}_{k+1}\| \stackrel{?}{\leq} \|\boldsymbol{\delta}_k\|, \quad \boldsymbol{\delta}_k \triangleq \mathbf{P}^{-1/2}(\mathbf{x}_k - \hat{\mathbf{x}})$ distance to solution with a \mathbf{P} -related coordinate transform
- $\|\tilde{\boldsymbol{\delta}}_{k+1}\| \stackrel{?}{\leq} \|\tilde{\boldsymbol{\delta}}_k\|, \quad \tilde{\boldsymbol{\delta}}_k \triangleq \mathbf{V}'\boldsymbol{\delta}_k$ distance to solution under the \mathbf{V} transform

When one of the above inequalities hold (typically with strict inequality for $\mathbf{x}_k \neq \hat{\mathbf{x}}$) we say the algorithm is **monotonic** or **monotone** but to be complete we should also qualify that by saying in which sense it is monotone (*e.g.*, cost function, gradient norm, distance to solution).

The easiest of the above list to analyze for PGD is the last one.

Recall $\tilde{\delta}_{k+1} = \Lambda \tilde{\delta}_k$. If $\tilde{\delta}_k \neq 0$, then because we choose P so that $-1 < \lambda_n < 1$ per (8.5), the distance to the solution diminishes monotonically in these coordinates:

$$\left\| \tilde{\delta}_{k+1} \right\|_2 = \left\| \Lambda \tilde{\delta}_k \right\|_2 \leq \| \Lambda \|_2 \left\| \tilde{\delta}_k \right\|_2 < \left\| \tilde{\delta}_k \right\|_2.$$

If $P \succ 0$ and $2P^{-1} \succ A'A \succ 0$, does $\|\delta_k\|_2$ decrease monotonically?

A: Yes.

B: Not necessarily.

??

Under the same conditions, when $x_k \neq \hat{x}$ does the distance $\|x_k - \hat{x}\|_2$ decrease monotonically?

Choose most general correct answer.

A: Yes.

B: Yes, if the right singular vectors of A are eigenvectors of P .

C: Yes, if the left singular vectors of A are eigenvectors of P .

D: Yes, if $P = \alpha I$ for suitably chosen α .

E: No.

??

Hint: $\|x_{k+1} - \hat{x}\|_2 = \|(I - PA'A)(x_k - \hat{x})\|_2 \leq \|I - PA'A\|_2 \|x_k - \hat{x}\|_2$

8.2 Preconditioned steepest descent

(Read)

Instead of using GD with a fixed step size α , an alternative is to do a **line search** to find the best step size *at each iteration*. This variation is called **steepest descent** (or GD with a line search). Here is how **preconditioned steepest descent** for a linear LS problem works:

$$\mathbf{d}_k = -\mathbf{P}\nabla f(\mathbf{x}_k) = -\mathbf{P}\mathbf{A}'(\mathbf{A}\mathbf{x}_k - \mathbf{y}) \quad \text{search direction (negative preconditioned gradient)}$$

$$\alpha_k = \arg \min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{d}_k) \quad \text{step size}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \quad \text{update.}$$

- Finding α_k is a HW problem
- By construction, this iteration is guaranteed to decrease the cost function monotonically: $f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k)$ with strict decrease unless \mathbf{x}_k is already a minimizer, provided the preconditioner \mathbf{P} is positive semidefinite.
- Computing α_k takes some extra work, especially for non-quadratic problems. Often Nesterov's fast gradient method or the **optimized gradient method (OGM)** [3] are preferable because they do not require a line search (if the Lipschitz constant is available).

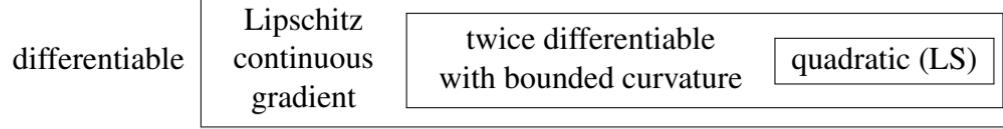
8.3 Gradient descent for smooth convex functions

So far we used (preconditioned) **gradient descent (PGD)** only for LS problems.
We often need to solve more general (non LS) unconstrained minimization problems:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathbb{R}^N} f(\mathbf{x}).$$

What algorithm we use depends in part on the properties of the cost function $f : \mathbb{R}^N \mapsto \mathbb{R}$.

Venn diagram for
convex functions:



GD is applicable to the broader family of **convex** functions having **gradients** that are **Lipschitz continuous**. We call these **smooth convex** functions.

Define. A differentiable function $f(\mathbf{x})$ has a **Lipschitz continuous gradient** if there exists $L < \infty$ such that

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{z})\|_2 \leq L \|\mathbf{x} - \mathbf{z}\|_2, \quad \forall \mathbf{x}, \mathbf{z} \in \mathbb{R}^N.$$

Example. For $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{y}\|_2^2$ we have $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{z})\|_2 = \|\mathbf{A}'(\mathbf{Ax} - \mathbf{y}) - \mathbf{A}'(\mathbf{Az} - \mathbf{y})\|_2 = \|\mathbf{A}'\mathbf{A}(\mathbf{x} - \mathbf{z})\|_2 \leq \|\mathbf{A}'\mathbf{A}\|_2 \|\mathbf{x} - \mathbf{z}\|_2$.

So the Lipschitz constant of ∇f is $L = \|\mathbf{A}'\mathbf{A}\|_2 = \|\mathbf{A}\|_2^2 = \sigma^2(\mathbf{A}) = \rho(\mathbf{A}'\mathbf{A})$.

Fact. If $f(\mathbf{x})$ is **twice differentiable** and if there exists $L < \infty$ such that its **Hessian matrix** has a bounded **spectral norm**:

$$\|\nabla^2 f(\mathbf{x})\|_2 \leq L, \quad \forall \mathbf{x} \in \mathbb{R}^N,$$

then $f(\mathbf{x})$ has a **Lipschitz continuous gradient** with Lipschitz constant L .

So twice differentiability with **bounded curvature** is sufficient, but not necessary, for a function to have Lipschitz continuous gradient.

Proof. Using **Taylor's theorem** and the **triangle inequality** and the definition of **spectral norm**:

$$\begin{aligned}\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{z})\|_2 &= \left\| \int_0^1 \nabla^2 f(\mathbf{x} + \tau(\mathbf{z} - \mathbf{x})) d\tau (\mathbf{x} - \mathbf{z}) \right\|_2 \\ &\leq \left(\int_0^1 \|\nabla^2 f(\mathbf{x} + \tau(\mathbf{z} - \mathbf{x}))\|_2 d\tau \right) \|\mathbf{z} - \mathbf{z}\|_2 \\ &\leq \left(\int_0^1 L d\tau \right) \|\mathbf{x} - \mathbf{z}\|_2 = L \|\mathbf{x} - \mathbf{z}\|_2.\end{aligned}$$

Example. $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 \implies \nabla^2 f = \mathbf{A}'\mathbf{A}$ so $\|\nabla^2 f\|_2 = \|\mathbf{A}'\mathbf{A}\|_2$.

The Lipschitz constant for the gradient of $f(\mathbf{x}) \triangleq \mathbf{x}' \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \mathbf{x}$ is:

A: 1

B: 2

C: 4

D: 5

E: 10

??

Convexity and Hessian

If $f(\mathbf{x})$ is twice differentiable, then $f(\mathbf{x})$ is **convex** iff its Hessian $\nabla^2 f(\mathbf{x})$ is **positive semidefinite** for all \mathbf{x} .

Example. $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{y}\|_2^2 \implies \nabla^2 f(\mathbf{x}) = \mathbf{A}'\mathbf{A} \succeq \mathbf{0} \implies f(\mathbf{x})$ is convex for any \mathbf{A} .

One can also show convexity of this $f(\mathbf{x})$ directly from the definition of convex functions.

If $f(\mathbf{x})$ is twice differentiable, then $f(\mathbf{x})$ is **strictly convex** if its Hessian $\nabla^2 f(\mathbf{x})$ is **positive definite** for all \mathbf{x} .

Why not only if? $f(x) = x^4$ is strictly convex but $\ddot{f}(0) = 0$.

Example. If \mathbf{A} has full column rank, then $\mathbf{A}'\mathbf{A}$ is positive definite so $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{y}\|_2^2$ is strictly convex.

Suppose \mathbf{A} is a wide matrix and consider the regularized LS cost function $f(\mathbf{x}) \triangleq \frac{1}{2} \|\mathbf{Ax} - \mathbf{y}\|_2^2 + \beta \|\mathbf{x}\|_2^2$ where $\beta > 0$. Then f is a **strictly convex** function. (?)

A: True

B: False

??

Example. The **Fair potential** used in many imaging applications [4] [5] is

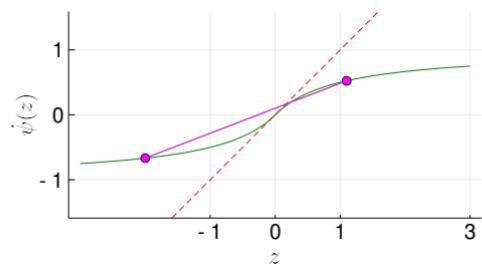
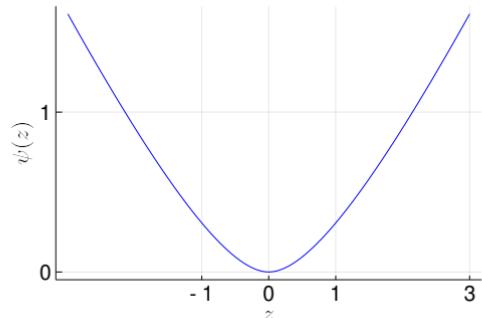
$$\psi(x) = |x| - \log(1 + |x|),$$

and has the property of being roughly quadratic for $x \approx 0$ and roughly like $|x|$ for $|x| \gg 0$.

For this function:

$$\dot{\psi}(x) = \frac{x}{1 + |x|} \text{ and } \ddot{\psi}(x) = \frac{1}{(1 + |x|)^2} \leq 1,$$

so the Lipschitz constant of the derivative of $\psi(\cdot)$ is 1. Furthermore, its second derivative is nonnegative so it is a convex function.



Is the Fair potential ψ itself **Lipschitz continuous**?

- A: No B: Yes with $L = 1/2$ C: Yes with $L = 1$ D: Yes with $L = 2$

??

GD convergence theorem

- If **convex** function $f(\mathbf{x})$ has a (not necessarily unique) minimizer $\hat{\mathbf{x}}$ for which

$$-\infty < f(\hat{\mathbf{x}}) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbb{R}^N,$$

- the gradient of $f(\mathbf{x})$ is **Lipschitz continuous**, i.e.,

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{z})\|_2 \leq L \|\mathbf{x} - \mathbf{z}\|_2, \quad \forall \mathbf{x}, \mathbf{z} \in \mathbb{R}^N,$$

- the **step size** α is chosen such that

$$0 < \alpha < 2/L,$$

then the GD iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)$$

has the following convergence properties [6, p. 207]:

- the cost function is non-increasing: $f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k)$
- for any minimizer $\hat{\mathbf{x}}$, the distance to that minimizer is non-increasing: $\|\mathbf{x}_{k+1} - \hat{\mathbf{x}}\| \leq \|\mathbf{x}_k - \hat{\mathbf{x}}\|$
- the sequence $\{\mathbf{x}_k\}$ **converges** to a minimizer of $f(\cdot)$.
- For $0 < \alpha \leq 1/L$, the cost function decrease is bounded by [2]:

$$f(\mathbf{x}_k) - f(\hat{\mathbf{x}}) \leq \frac{L \|\mathbf{x}_0 - \hat{\mathbf{x}}\|_2^2}{2} \max \left(\frac{1}{2k\alpha + 1}, (1 - \alpha)^{2k} \right).$$

This upper bound is conjectured to also hold for $1/L < \alpha < 2/L$ [7].

Convex sets

Define. A nonempty set \mathcal{C} in a vector space \mathcal{V} is a **convex set** iff

$$\mathbf{x}, \mathbf{z} \in \mathcal{C} \implies \alpha\mathbf{x} + (1 - \alpha)\mathbf{z} \in \mathcal{C}, \quad \forall 0 \leq \alpha \leq 1.$$

The linear combination $\alpha\mathbf{x} + (1 - \alpha)\mathbf{z}$ for any $0 \leq \alpha \leq 1$ is a **convex combination**.

Example. Any subspace \mathcal{S} in a vector space \mathcal{V} is convex.

Example. The **nonnegative orthant** $\mathbb{R}_+^N \triangleq \{\mathbf{x} \in \mathbb{R}^N : \mathbf{x} \geq \mathbf{0}\}$ is convex.

Example. For any norm, the ball of radius $r > 0$ $\{\mathbf{x} : \|\mathbf{x}\| \leq r\}$ is convex. (HW)

Finding the nearest point in a convex set is an important operation. $\mathcal{P}_{\mathcal{C}}(\cdot)$ denotes the “**projection**” of its argument onto the closest point in \mathcal{C} :

$$\mathcal{P}_{\mathcal{C}}(\mathbf{z}) \triangleq \min_{\mathbf{x} \in \mathcal{C}} \|\mathbf{x} - \mathbf{z}\|$$

For $\mathcal{C} = \{\mathbf{x} \in \mathbb{R}^N : \|\mathbf{x}\|_\infty \leq 5\}$, which of these is the projection of a point $\mathbf{z} \in \mathbb{R}^N$ onto \mathcal{C} ?

- A: `min.(x, 5)` B: `min.(abs.(x), 5)` C: `min.(abs.(x), 5).*sign(x)` D: None of these

??

Gradient projection method

In many applications, we seek the minimizer of a convex function $f(\mathbf{x})$ over a convex set \mathcal{C} :

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x} \in \mathcal{C}} f(\mathbf{x}).$$

This is called **constrained optimization**.

Fact. The conclusions about convergence for GD given above also hold for the more **gradient projection (GP)** method

$$\mathbf{x}_{k+1} = \mathcal{P}_{\mathcal{C}}(\mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)).$$

Example. (HW) NNLS, where $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$ and $\mathcal{C} = \mathbb{R}_+^N$.

Example. If $f(\mathbf{x}) = \frac{1}{2} \left\| \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \mathbf{x} - \begin{bmatrix} 4 \\ -2 \end{bmatrix} \right\|_2^2$ and we apply GP with $\alpha = 1/L$ and $\mathbf{x}_0 = \mathbf{0}$ then $\mathbf{x}_1 = ?$

A: (0, 0)

B: (1, 0)

C: (1, -1)

D: (0, -1)

E: (1, 1)

??

8.4 Machine learning via logistic regression for binary classification

To learn weights $\mathbf{x} \in \mathbb{R}^N$ of a binary classifier given feature vectors $\{\mathbf{v}_i\} \in \mathbb{R}^N$ (training data) and labels $\{y_i = \pm 1 : i = 1, \dots, M\}$, we can minimize a cost function with a regularization parameter $\beta > 0$:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} f(\mathbf{x}), \quad f(\mathbf{x}) = \sum_{i=1}^M \psi(y_i \langle \mathbf{x}, \mathbf{v}_i \rangle) + \beta \frac{1}{2} \|\mathbf{x}\|_2^2.$$

Want:

- $\langle \mathbf{x}, \mathbf{v}_i \rangle > 0$ if $y_i = +1$ and
- $\langle \mathbf{x}, \mathbf{v}_i \rangle < 0$ if $y_i = -1$,
- i.e., $\langle \mathbf{x}, y_i \mathbf{v}_i \rangle > 0$,

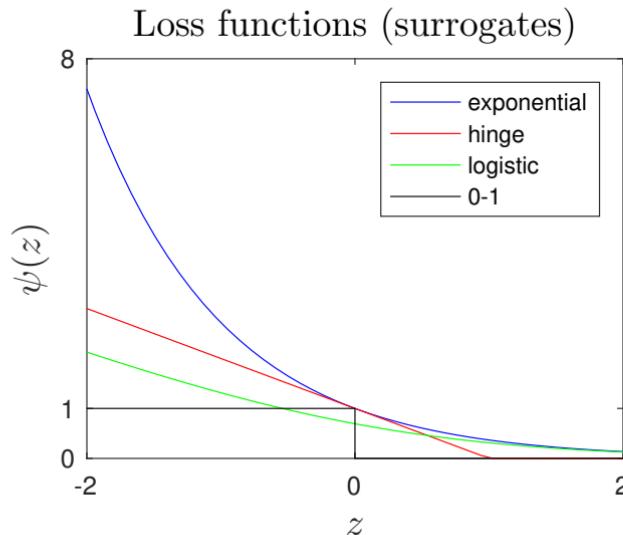
so that $\text{sign}(\langle \mathbf{x}, \mathbf{v}_i \rangle)$ is a reasonable classifier.

Logistic loss function has a Lipschitz derivative:

$$\psi(z) = \log(1 + e^{-z})$$

$$\dot{\psi}(z) = \frac{-1}{e^z + 1}$$

$$\ddot{\psi}(z) = \frac{e^z}{(e^z + 1)^2} \in \left(0, \frac{1}{4}\right].$$



A regularization term like $\frac{\beta}{2} \|\mathbf{x}\|_2^2$ is especially important in the typical case where the feature vector dimension N is large relative to the sample size M .

This cost function is not quadratic, but it does have a Lipschitz gradient.

For gradient-based optimization, we need the cost function gradient:

$$\underbrace{\nabla f(\mathbf{x})}_{N \times 1} = \sum_{m=1}^M y_m \mathbf{v}_m \dot{\psi}(\langle \mathbf{x}, y_m \mathbf{v}_m \rangle) + \beta \mathbf{x} = \mathbf{V} \begin{bmatrix} \dot{\psi}([\mathbf{V}' \mathbf{x}]_1) \\ \vdots \\ \dot{\psi}([\mathbf{V}' \mathbf{x}]_M) \end{bmatrix},$$

where the following $N \times M$ matrix has columns that are the products of the labels *and* the feature vectors:

$$\mathbf{V} = [y_1 \mathbf{v}_1 \quad \dots \quad y_M \mathbf{v}_M].$$

Cost function **Hessian matrix**:

$$\nabla^2 f(\mathbf{x}) = \sum_{m=1}^M (y_m \mathbf{v}_m) \ddot{\psi}(\langle \mathbf{x}, y_m \mathbf{v}_m \rangle) (y_m \mathbf{v}_m)' + \beta \mathbf{I}.$$

f is a **strictly convex** function when ψ is the **logistic** loss function. (?)

A: True

B: False

??

Rewriting the Hessian of f in matrix form:

$$\nabla^2 f(\mathbf{x}) = \underbrace{\mathbf{V} \mathbf{D} \mathbf{V}'}_{\succeq \mathbf{0}} + \underbrace{\beta \mathbf{I}}_{\succ \mathbf{0}} \succ \mathbf{0}, \quad \mathbf{D} \triangleq \text{diag}\left\{\ddot{\psi}(\langle \mathbf{x}, y_m \mathbf{v}_m \rangle)\right\} \succ \mathbf{0}.$$

A Hessian majorizer leads to Lipschitz constant of $\nabla f(\mathbf{x})$ that is useful for many iterative algorithms: (HW)

$$\nabla^2 f(\mathbf{x}) = \mathbf{V} \mathbf{D} \mathbf{V}' + \beta \mathbf{I} \preceq \mathbf{V} \left(\frac{1}{4} \mathbf{I} \right) \mathbf{V}' + \beta \mathbf{I} = \frac{1}{4} \mathbf{V} \mathbf{V}' + \beta \mathbf{I} \preceq \frac{1}{4} \|\mathbf{V} \mathbf{V}'\|_2 \mathbf{I} + \beta \mathbf{I} = \left(\frac{1}{4} \|\mathbf{V}\|_2^2 + \beta \right) \mathbf{I},$$

leading to a Lipschitz constant that depends on the training data \mathbf{V} and the regularization parameter:

$$L \triangleq \frac{1}{4} \|\mathbf{V}\|_2^2 + \beta.$$

Notice the process here: typically we do not try to find L numerically. Instead we analyze either ∇f or $\nabla^2 f$ and use properties and inequalities (analytically, on paper) to find a suitable L expressed in terms of the problem variables (in this case \mathbf{V} and β).

Practical implementation:

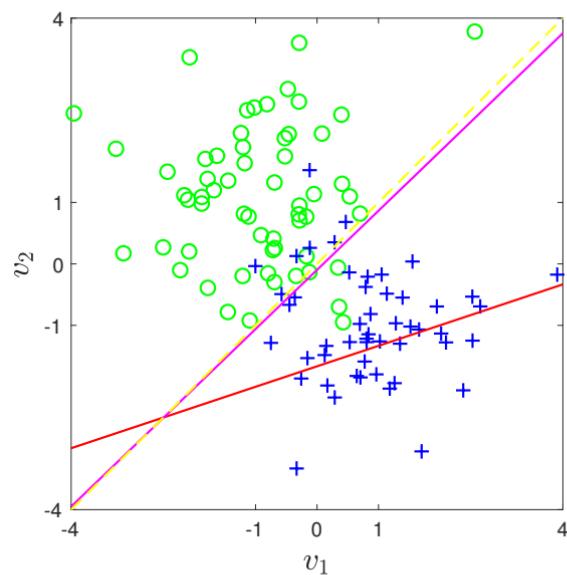
- Normalizing each column of \mathbf{V} to unit norm can help keep e^z from overflowing.
- Tuning β should use cross validation or other such tools from machine learning.
- The cost function is convex with Lipschitz gradient, so it is well-suited for Nesterov's fast gradient method (or OGM).

- When feature dimension N is very large, seeking a sparse weight vector \boldsymbol{x} may be preferable. For that, replace the Tikhonov regularizer $\|\boldsymbol{x}\|_2^2$ with $\|\boldsymbol{x}\|_1$ and then use FISTA (or POGM [8]) for optimization.

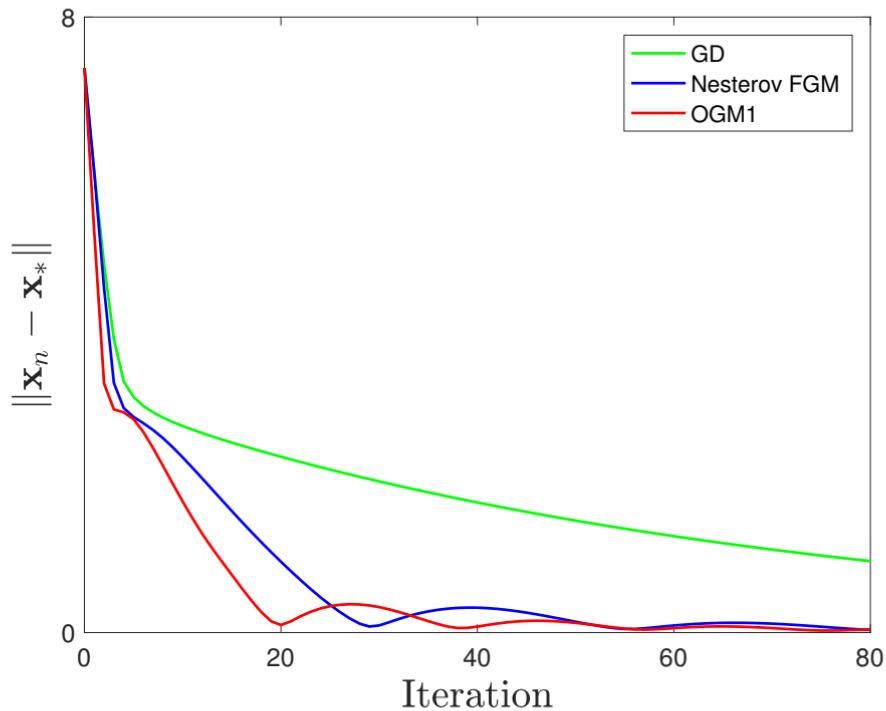
This **logistic regression** approach to classification will be explored in a task sheet.

Numerical Results: logistic regression

Labeled training data (green and blue points);
initial decision boundary (red);
final decision boundary (magenta);
ideal boundary (yellow).

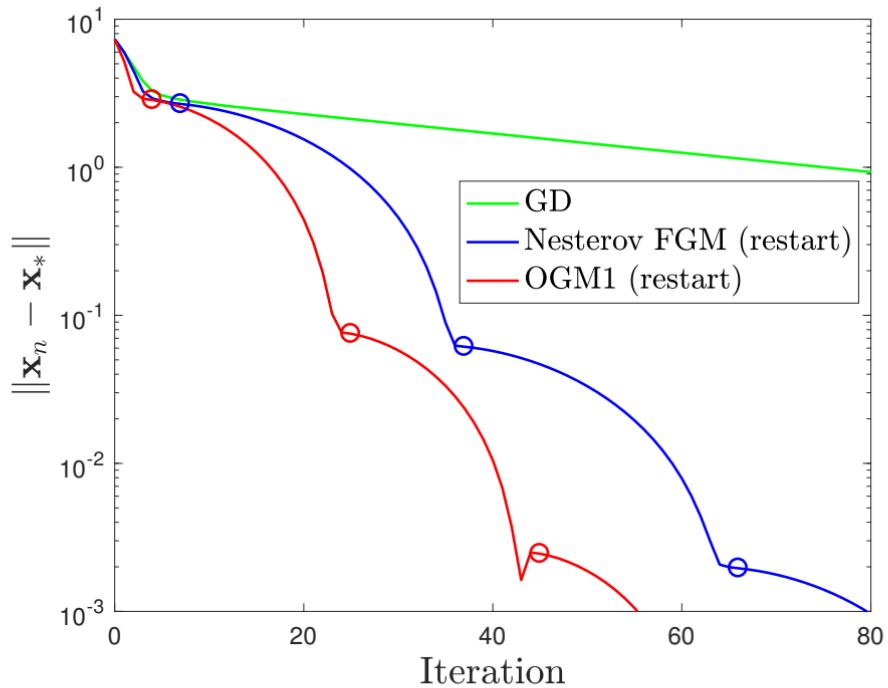


Numerical Results: convergence rates



OGM faster than FGM in early iterations...

Adaptive restart of accelerated GD



FGM restart, O'Donoghue & Candès, 2015. [9]
OGM1 restart [10]

8.5 Summary

This chapter barely scratches the surface of the field of optimization algorithms, but it illustrates how crucial matrix methods are to that field.

Bibliography

- [1] W. Zuo and Z. Lin. “A generalized accelerated proximal gradient approach for total-variation-based image restoration”. In: *IEEE Trans. Im. Proc.* 20.10 (Oct. 2011), 2748–59.
- [2] Y. Drori and M. Teboulle. “Performance of first-order methods for smooth convex minimization: A novel approach”. In: *Mathematical Programming* 145.1-2 (June 2014), 451–82.
- [3] D. Kim and J. A. Fessler. “Optimized first-order methods for smooth convex minimization”. In: *Mathematical Programming* 159.1 (Sept. 2016), 81–107.
- [4] R. C. Fair. “On the robust estimation of econometric models”. In: *Ann. Econ. Social Measurement* 2 (Oct. 1974), 667–77.
- [5] K. Lange. “Convergence of EM image reconstruction algorithms with Gibbs smoothing”. In: *IEEE Trans. Med. Imag.* 9.4 (Dec. 1990). Corrections, T-MI, 10:2(288), June 1991., 439–46.
- [6] B. T. Polyak. *Introduction to optimization*. New York: Optimization Software Inc, 1987.
- [7] A. B. Taylor, J. M. Hendrickx, and Francois Glineur. “Smooth strongly convex interpolation and exact worst-case performance of first- order methods”. In: *Mathematical Programming* 161.1 (Jan. 2017), 307–45.
- [8] A. B. Taylor, J. M. Hendrickx, and Francois Glineur. “Exact worst-case performance of first-order methods for composite convex optimization”. In: *SIAM J. Optim.* 27.3 (Jan. 2017), 1283–313.
- [9] B. O’Donoghue and E. Candes. “Adaptive restart for accelerated gradient schemes”. In: *Found. Comp. Math.* 15.3 (June 2015), 715–32.
- [10] D. Kim and J. A. Fessler. “Adaptive restart of the optimized gradient method for convex optimization”. In: *J. Optim. Theory Appl.* 178.1 (July 2018), 240–63.

Chapter 9

Matrix completion

Contents (final version)

9.0 Introduction (<code>s, intro</code>)	9.2
9.1 Measurement model (<code>s, model</code>)	9.3
9.2 LRMC: noiseless case (<code>s, lrmc</code>)	9.7
Noiseless problem statement	9.7
Alternating projection approach (<code>s, pocs</code>)	9.8
9.3 LRMC: noisy case (<code>s, noise</code>)	9.11
Noisy problem statement	9.11
Majorize-minimize (MM) iterations (<code>s, mm</code>)	9.13
MM methods for LRMC	9.14
LRMC by iterative low-rank approximation	9.16
LRMC by iterative singular value hard thresholding	9.17
LRMC by iterative singular value soft thresholding	9.18
Demo (<code>s, demo</code>)	9.21
9.4 Summary (<code>s, summ</code>)	9.26

9.0 Introduction

This chapter discusses the important problem of **matrix completion**, where we know some, but not all, elements of a matrix and want to “complete” the matrix by filling in the missing entries. Obviously this problem is **ill posed** in general because one could assign arbitrary values to the missing entries, unless one assumes some **model** for the matrix elements. The most common model is that the matrix is **low rank**.

A particularly famous application of **low-rank matrix completion (LRMC)** is the **Netflix problem**; this topic is also relevant to dynamic MR image reconstruction, and numerous other applications with missing data (incomplete observations).

Successful applications of LRMC include:

- Recommender systems (netflix, spotify, amazon, ...)
- Imaging: denoising, reconstruction in medical, hyper-spectral imaging.
- Anomaly detection in network flows
- Source localization and target tracking in radar and sonar
- Computer vision: background subtraction, object tracking, and to represent a single scene under varying illuminations
- Environmental monitoring of soil and crop conditions, water contamination, and air pollution, also sensor calibration
- Seismological activity and modal estimation in materials and human-made structures
- ...

9.1 Measurement model

If \mathbf{X} is a **latent** $M \times N$ “low-rank” matrix with rank $r \leq \min(M, N)$, then

$$\mathbf{X} = \sum_{k=1}^r \sigma_k \mathbf{u}_k \mathbf{v}'_k.$$

Suppose we observe only *some* of the entries of \mathbf{X} in a sampling set Ω , *i.e.*,

$$Y_{ij} = \begin{cases} X_{ij}, & (i, j) \in \Omega \\ ?, & \text{otherwise,} \end{cases} \quad \Omega \subset \{1, \dots, M\} \times \{1, \dots, N\},$$

where “?” denotes a value is “missing,” *e.g.*, the i th user has not rated the j th movie.

Key questions:

- For a given (known) sampling set Ω , can we recover all of \mathbf{X} from just the observed entries \mathbf{Y} ?
The answer will depend on the rank r , where typically $r \ll \min(M, N)$, and on the sampling set.
- How do we do it?
- How well does it work (*e.g.*, in the presence of noise or if \mathbf{X} is not exactly low rank)?

Practical implementation

This topic is so important to modern data analysis problems that JULIA has its own data type, `Missing`, to represent missing values. Try this:

```
Y = [2 0; missing 4; 5 missing]
```

The type of `Y` is `3x2 Array{Union{Missing, Int64},2}`

because it is a 3×2 array of values that are either 64-bit integers or the special type `Missing`.

We cannot use any particular value like “0” to represent a missing value because 0 might be one of the observed values!

What is the minimum possible rank of `Y` in this example?

A: 0

B: 1

C: 2

D: 3

E: None of these.

??

Sampling conditions for LRMC

The first question to ask is whether it should even be possible to recover \mathbf{X} from \mathbf{Y} in the noiseless case.

If \mathbf{X} has rank r , then we can write \mathbf{X} as the product of a $M \times r$ matrix (think $\tilde{\mathbf{U}}_r = \mathbf{U}_r \boldsymbol{\Sigma}_r$ for example) with the transpose of a $N \times r$ matrix $\tilde{\mathbf{V}}_r$. The number of **degrees of freedom (DoF)** in this representation looks to be $Mr + Nr = (M + N)r$. However, actually there are fewer DoF because once we fix $\tilde{\mathbf{U}}_r$ there are constraints on the values that $\tilde{\mathbf{V}}_r$ can take. So $(M + N)r$ is an upper bound on the DoF. The total number of elements of \mathbf{X} is MN , so if the matrix is low rank, then $(M + N)r \ll MN$ so it seems plausible that having around $(M + N)r$ samples might suffice.

A more careful analysis of the DoF is

$$\text{DoF} = rM + (N - r)r = (M + N)r - r^2,$$

because

- the first r columns (of length M) are linearly independent,
- and the next $N - r$ columns depend linearly on those first r columns and we need r coefficients per column to describe the linear combination.

Note that if $r = N \leq M$ then $(M + N)r - r^2 = (M + N)N - N^2 = MN$, as expected.

When $M = N$ and $r \leq N/2$, a **lower bound on the number of samples** is $4Nr - 4r^2$ [1].

If $M \approx N$, then the DoF $\approx 2Nr$. So we need at least $O(Nr)$ samples to have any chance of recovery (unless we have additional information such as knowing that \mathbf{U}_r and/or \mathbf{V}_r is sparse).

Modern algorithms need $O(Nr \text{ polylog}(N))$ random samples under certain assumptions about \mathbf{X} .

Approximately how many observations *per column* are the minimum needed to recover a rank- r low-rank matrix of size $N \times N$?

A: r B: $2r$ C: r^2 D: N E: $2N$

??

Sampling mask

Define a $M \times N$ binary **sampling mask** matrix \mathbf{M} in terms of the sampling set Ω by:

$$M_{ij} \triangleq \begin{cases} 1, & (i, j) \in \Omega, \\ 0, & \text{otherwise.} \end{cases} \quad (9.1)$$

In words, \mathbf{M} is zero wherever we are missing a value in \mathbf{Y} .

Example. For the earlier 3×2 example we have $\mathbf{M} = \text{Int}(\text{.}\sim\text{ismissing}(\mathbf{Y}))$, i.e., $\mathbf{M} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$

Hereafter, let $\mathbf{Y} \in \mathbb{F}^{M \times N}$ be an matrix where

$$Y_{ij} = \begin{cases} X_{ij}, & (i, j) \in \Omega \\ 0, & \text{otherwise.} \end{cases} \quad (9.2)$$

Now we *can* store a 0 in the locations of missing values because we have the separate mask \mathbf{M} to keep track of which values are sampled and which are missing.

9.2 LRMC: noiseless case

Noiseless problem statement

The (noiseless) LRMC problem is to find a matrix $\hat{\mathbf{X}}$ that is **low-rank** and that agrees with the measurements \mathbf{Y} on Ω , *i.e.*, we want

$$\mathbf{M} \odot \hat{\mathbf{X}} = \mathbf{M} \odot \mathbf{Y},$$

where \odot denotes element-wise multiplication, *i.e.*, `M .* X` in JULIA and MATLAB, also known as the **Hadamard product** or **array multiplication** (as opposed to **matrix multiplication**) and is applicable to arrays of any dimension.

(In Python, `M * X` is the Hadamard product for `array` objects in NumPy, but is ordinary matrix multiplication for `matrix` objects.) 

One “ideal” (noiseless) optimization formulation for the LRMC problem is [2]:

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X}} \text{rank}(\mathbf{X}) \text{ s.t. } \mathbf{M} \odot \mathbf{X} = \mathbf{M} \odot \mathbf{Y}. \quad (9.3)$$

This is a **non-convex** problem and is **NP hard** (impractical). No fast algorithm exists that is guaranteed to solve this formulation of the problem. Nevertheless, there are practical algorithms that often work reasonably well.

Alternating projection approach

An alternative to (9.3) is to pick a rank $1 \leq K \ll \min(M, N)$ and seek a matrix $\hat{\mathbf{X}}$ that satisfies

$$\hat{\mathbf{X}} \in \mathcal{C} \cap \mathcal{D}, \quad \mathcal{C} \triangleq \left\{ \mathbf{X} \in \mathbb{F}^{M \times N} : \text{rank}(\mathbf{X}) \leq K \right\}, \quad \mathcal{D} \triangleq \left\{ \mathbf{X} \in \mathbb{F}^{M \times N} : \mathbf{M} \odot \mathbf{X} = \mathbf{M} \odot \mathbf{Y} \right\}.$$

In words, we seek a matrix $\hat{\mathbf{X}}$ having rank at most K and that agrees with the measurements \mathbf{Y} on Ω .

The **projections onto convex sets (POCS)** approach is a classic signal processing tool for finding points in the intersection of two or more convex sets. *Discuss method using pictures on wikipedia.*

Which (if any) of the two sets \mathcal{C} and \mathcal{D} above is **convex**?

- A: neither \mathcal{C} nor \mathcal{D} B: \mathcal{C} but not \mathcal{D} C: \mathcal{D} but not \mathcal{C} D: both \mathcal{C} and \mathcal{D}

??

The **alternating projection** method alternates between projecting onto \mathcal{C} and \mathcal{D} :

$$\mathbf{X}_{k+1} = \mathcal{P}_{\mathcal{C}}(\mathcal{P}_{\mathcal{D}}(\mathbf{X}_k)),$$

where $\mathcal{P}_{\mathcal{D}}(\mathbf{X})$ denotes the **projection** of its argument \mathbf{X} onto the set \mathcal{D} , *i.e.*, the closest point in \mathcal{D} :

$$\mathcal{P}_{\mathcal{D}}(\mathbf{X}) = \arg \min_{\mathbf{Z} \in \mathcal{D}} \|\mathbf{X} - \mathbf{Z}\|_F.$$

The algorithm designer gets to choose the norm for quantifying “closest.” Here we are using matrices so we work with the simple choice, the Frobenius norm.

For the “data” set \mathcal{D} above, the projection operation is very simple:

$$[\mathcal{P}_{\mathcal{D}}(\mathbf{X})]_{i,j} = \begin{cases} Y_{i,j}, & (i,j) \in \Omega \\ X_{i,j}, & \text{otherwise.} \end{cases}$$

Example.

Suppose $(M, N) = (2, 1)$ and $\mathbf{M} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ so $\mathbf{Y} = \begin{bmatrix} Y_1 \\ ? \end{bmatrix}$.

Then \mathcal{D} is just $\{(X_1, X_2) \in \mathbb{R}^2 : X_1 = Y_1\}$.

For any point (X_1, X_2) , the projection onto \mathcal{D} is simply (Y_1, X_2) .

If \mathbf{M} is a logical array, then we can implement this operation “in place” in JULIA as follows:

```
X [M] .= Y [M]
```

Even though \mathcal{C} is not convex, finding a projection onto \mathcal{C} is fairly easy:

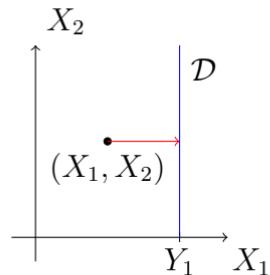
$$\mathcal{P}_{\mathcal{C}}(\mathbf{Z}) = \arg \min_{\mathbf{X} : \text{rank}(\mathbf{X}) \leq K} \|\mathbf{Z} - \mathbf{X}\|_{\text{F}} = \mathbf{U}_K \boldsymbol{\Sigma}_K \mathbf{V}'_K, \quad \mathbf{X} = \mathbf{U}_r \boldsymbol{\Sigma}_r \mathbf{V}'_r.$$

This is simply the low-rank approximation problem of Ch. 6. It requires an SVD of the input argument.

The projection $\mathcal{P}_{\mathcal{C}}(\mathbf{Z})$ is **unique** for any $M \times N$ input matrix \mathbf{Z} . (?)

A: True

B: False

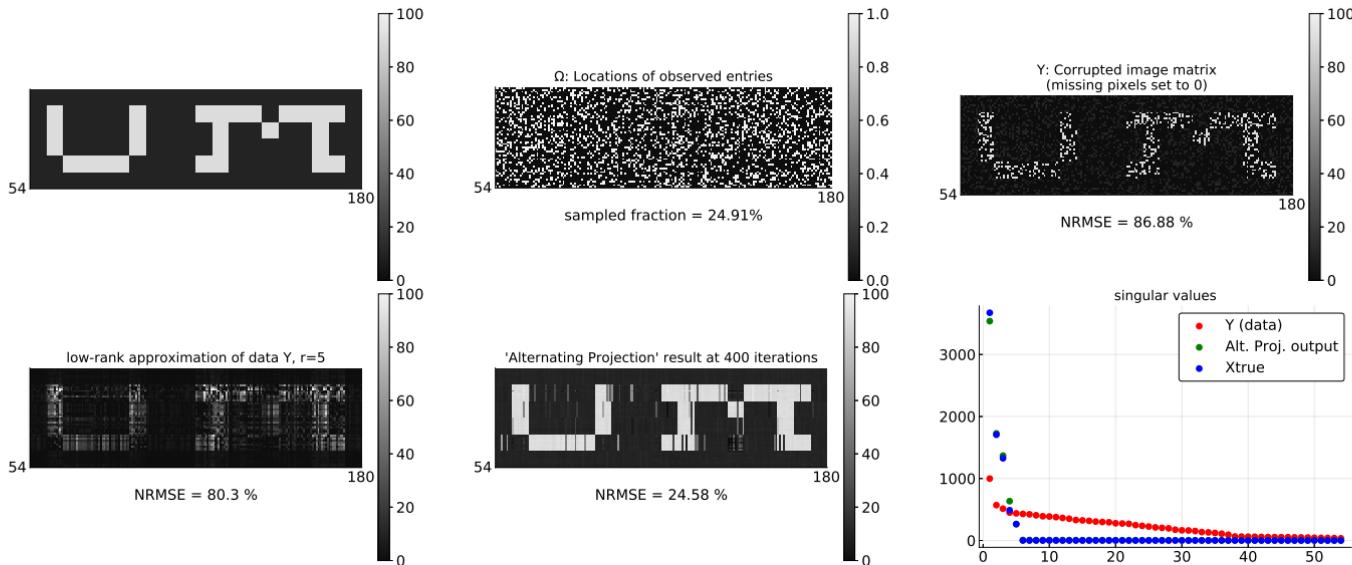


??

Example. This demo shows matrix completion via alternating projection with 25% sampling.

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/09_matcomp_altpro.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/09_matcomp_altpro.ipynb



The alternating projection method works better than simply making a low-rank approximation to \mathbf{Y} , but it does not account for noise in the data, so we can do better.

9.3 LRMC: noisy case

Noisy problem statement

In practice, often there is noise in the data, so the model (9.2) is often unrealistic. A more realistic model is

$$Y_{ij} = \begin{cases} X_{ij} + \varepsilon_{ij}, & (i, j) \in \Omega \\ 0, & \text{otherwise.} \end{cases}$$

Again, it is fine to store a 0 (or any other value) in the missing locations because we have the sampling mask \mathbf{M} available.

In this case, it would be unreasonable to insist on exact data equality $\mathbf{M} \odot \mathbf{X} = \mathbf{M} \odot \mathbf{Y}$.

So now we want to find $\hat{\mathbf{X}}$ where $\mathbf{M} \odot \hat{\mathbf{X}} \approx \mathbf{M} \odot \mathbf{Y}$ and also $\hat{\mathbf{X}}$ is low-rank.

One possible LRMC formulation uses a (non-convex) **rank constraint**:

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} : \text{rank}(\mathbf{X}) \leq K} \| \mathbf{M} \odot (\mathbf{X} - \mathbf{Y}) \|_{\text{F}}^2. \quad (9.4)$$

Another possible LRMC formulation uses a (non-convex) **rank regularizer**:

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathbb{R}^{M \times N}} \| \mathbf{M} \odot (\mathbf{X} - \mathbf{Y}) \|_{\text{F}}^2 + \beta \text{rank}(\mathbf{X}). \quad (9.5)$$

The following LRMC optimization formulation is the **convex relaxation** of the rank regularizer, using instead a **nuclear norm** regularizer:

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X}} \frac{1}{2} \|\mathbf{M} \odot (\mathbf{X} - \mathbf{Y})\|_F^2 + \beta \|\mathbf{X}\|_*.$$
 (9.6)

There are fast and practical algorithms for all three of these formulations, and there are convergence guarantees for the convex formulation.

Unfortunately, the optimization tools presented in Ch. 8 (GD, PGD, PSD) are inapplicable here because none of the cost functions above are differentiable! So we need something more than gradient-based methods.

Even the cost function that uses the (convex) nuclear norm $\|\mathbf{X}\|_*$ is not differentiable.

What is nuclear norm $\|x\|_*$ of a 1×1 “matrix” x ?

A: x

B: $|x|$

C: $\mathbb{I}_{\{x \geq 0\}}$

D: $\max(0, x)$

E: x^2

??

Majorize-minimize (MM) iterations

To solve optimization problems like (9.6), we first consider **majorize-minimize (MM)** algorithms.

To solve an optimization problem like

(Picture)

$$\arg \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x})$$

we first design a **majorizer** or **surrogate function** $\phi_k(\boldsymbol{x})$ that satisfies the following two conditions:

$$\begin{aligned} f(\boldsymbol{x}_k) &= \phi_k(\boldsymbol{x}_k) \\ f(\boldsymbol{x}) &\leq \phi_k(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathcal{X}. \end{aligned}$$

Then the MM algorithm update is simply:

$$\boldsymbol{x}_{k+1} = \arg \min_{\boldsymbol{x} \in \mathcal{X}} \phi_k(\boldsymbol{x}).$$

Any MM algorithm will monotonically decrease the cost function because

$$f(\boldsymbol{x}_{k+1}) \leq \phi_k(\boldsymbol{x}_{k+1}) \leq \phi_k(\boldsymbol{x}_k) = f(\boldsymbol{x}_k).$$

Next we design a MM algorithm for LRMC problems like (9.6).

MM methods for LRM

First define the complement of the mask matrix as follows:

$$\tilde{\mathbf{M}} \triangleq \mathbf{1}_M \mathbf{1}'_N - \mathbf{M}, \quad \tilde{M}_{i,j} = \begin{cases} 0, & (i,j) \in \Omega \\ 1, & (i,j) \notin \Omega. \end{cases}$$

We focus now on the quadratic data-fit term in (9.4) (9.5) (9.6):

$$q(\mathbf{X}) \triangleq \| \mathbf{M} \odot (\mathbf{X} - \mathbf{Y}) \|_{\text{F}}^2.$$

Now define the following function:

$$Q(\mathbf{X}; \mathbf{Z}) = \| \mathbf{X} - \mathbf{Z} + \mathbf{M} \odot (\mathbf{Z} - \mathbf{Y}) \|_{\text{F}}^2 = \| \mathbf{X} - (\tilde{\mathbf{M}} \odot \mathbf{Z} + \mathbf{M} \odot \mathbf{Y}) \|_{\text{F}}^2. \quad (9.7)$$

This function is a **majorizer** of $q(\mathbf{X})$ because

$$q(\mathbf{X}) = Q(\mathbf{X}; \mathbf{X})$$

$$q(\mathbf{X}) \leq Q(\mathbf{X}; \mathbf{Z}), \quad \forall \mathbf{Z} \in \mathbb{F}^{M \times N}.$$

Proof:

(Read)

$$\begin{aligned}
 Q(\mathbf{X}; \mathbf{Z}) &= \| \mathbf{X} - \mathbf{Z} + \mathbf{M} \odot (\mathbf{Z} - \mathbf{Y}) \|_{\text{F}}^2 = \| (\tilde{\mathbf{M}} + \mathbf{M}) \odot (\mathbf{X} - \mathbf{Z}) + \mathbf{M} \odot (\mathbf{Z} - \mathbf{Y}) \|_{\text{F}}^2 \\
 &= \| \tilde{\mathbf{M}} \odot (\mathbf{X} - \mathbf{Z}) + \mathbf{M} \odot (\mathbf{X} - \mathbf{Y}) \|_{\text{F}}^2 = \| \tilde{\mathbf{M}} \odot (\mathbf{X} - \mathbf{Z}) \|_{\text{F}}^2 + \| \mathbf{M} \odot (\mathbf{X} - \mathbf{Y}) \|_{\text{F}}^2 \\
 &\geq \| \tilde{\mathbf{M}} \odot (\mathbf{X} - \mathbf{Y}) \|_{\text{F}}^2 = q(\mathbf{X}),
 \end{aligned}$$

where the 4th equality holds because $\tilde{\mathbf{M}} \odot \mathbf{M} = \mathbf{0}$.

I designed the function (9.7) by making a 2nd-order Taylor expansion of $q(\mathbf{X})$ about \mathbf{Z} and then using the ♦♦ fact that $\mathbf{M} \preceq \mathbf{I}$. The above proof verifies that Q is a majorizer, which is all that is needed here.

Now we use the majorizer (9.7) to develop a few different LRMC algorithms.

LRMC by iterative low-rank approximation

First consider LRMC using the rank constraint (9.4):

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} : \text{rank}(\mathbf{X}) \leq K} \|\mathbf{M} \odot (\mathbf{X} - \mathbf{Y})\|_F^2.$$

The MM algorithm update for this formulation is simply

$$\begin{aligned}\mathbf{X}_{k+1} &= \arg \min_{\mathbf{X} : \text{rank}(\mathbf{X}) \leq K} Q(\mathbf{X}; \mathbf{X}_k) \\ &= \arg \min_{\mathbf{X} : \text{rank}(\mathbf{X}) \leq K} \left\| \mathbf{X} - (\tilde{\mathbf{M}} \odot \mathbf{X}_k + \mathbf{M} \odot \mathbf{Y}) \right\|_F^2 \\ &= \arg \min_{\mathbf{X} : \text{rank}(\mathbf{X}) \leq K} \left\| \mathbf{X} - \tilde{\mathbf{X}}_k \right\|_F^2, \quad \tilde{\mathbf{X}}_k \triangleq \tilde{\mathbf{M}} \odot \mathbf{X}_k + \mathbf{M} \odot \mathbf{Y} = \begin{cases} Y_{i,j}, & (i, j) \in \Omega \\ [\mathbf{X}_k]_{i,j}, & (i, j) \notin \Omega. \end{cases}\end{aligned}$$

This algorithm alternates between two steps:

- Take the current guess \mathbf{X}_k and replace all the values at sampled locations with the measurements from \mathbf{Y} to get $\tilde{\mathbf{X}}_k$. (As mentioned earlier, this can be done “in place” using $\mathbf{X}[\mathbf{M}] . = \mathbf{Y}[\mathbf{M}]$)
- Perform low-rank (rank at most K) approximation (using **SVD**) to $\tilde{\mathbf{X}}_k$ to get the next iterate \mathbf{X}_{k+1} .

This process is exactly the same as the alternating projection method described earlier. So now we know that it is a MM method that decreases the Frobenius norm cost function monotonically.

(There is still no guarantee of convergence of $\{\mathbf{X}_k\}$ to a global minimizer because the rank constraint set is nonconvex.)

LRMC by iterative singular value hard thresholding

Now consider LRMC using the rank regularizer (9.5):

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathbb{R}^{M \times N}} \| \mathbf{M} \odot (\mathbf{X} - \mathbf{Y}) \|_{\text{F}}^2 + \beta \operatorname{rank}(\mathbf{X}).$$

The MM algorithm update for this formulation is simply

$$\begin{aligned}\mathbf{X}_{k+1} &= \arg \min_{\mathbf{X} \in \mathbb{R}^{M \times N}} Q(\mathbf{X}; \mathbf{X}_k) + \beta \operatorname{rank}(\mathbf{X}) \\ &= \arg \min_{\mathbf{X} \in \mathbb{R}^{M \times N}} \| \mathbf{X} - (\tilde{\mathbf{M}} \odot \mathbf{X}_k + \mathbf{M} \odot \mathbf{Y}) \|_{\text{F}}^2 + \beta \operatorname{rank}(\mathbf{X}) \\ &= \arg \min_{\mathbf{X} \in \mathbb{R}^{M \times N}} \| \mathbf{X} - \tilde{\mathbf{X}}_k \|_{\text{F}}^2 + \beta \operatorname{rank}(\mathbf{X}).\end{aligned}$$

This algorithm alternates between two steps:

- Take the current guess \mathbf{X}_k and replace all the values at sampled locations with the measurements from \mathbf{Y} to get $\tilde{\mathbf{X}}_k$. (This can be done “in place.”)
- Apply **singular value hard thresholding** to $\tilde{\mathbf{X}}_k$ to get the next iterate \mathbf{X}_{k+1} .

This MM method decreases the rank-regularized Frobenius norm cost function monotonically.

LRMC by iterative singular value soft thresholding

Now consider LRMC using the **convex nuclear norm** regularizer (9.6):

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathbb{F}^{M \times N}} \frac{1}{2} \|\mathbf{M} \odot (\mathbf{X} - \mathbf{Y})\|_F^2 + \beta \|\mathbf{X}\|_*.$$

The MM algorithm update for this formulation is simply

$$\begin{aligned} \mathbf{X}_{k+1} &= \arg \min_{\mathbf{X} \in \mathbb{F}^{M \times N}} \frac{1}{2} Q(\mathbf{X}; \mathbf{X}_k) + \beta \|\mathbf{X}\|_* \\ &= \arg \min_{\mathbf{X} \in \mathbb{F}^{M \times N}} \frac{1}{2} \left\| \mathbf{X} - (\tilde{\mathbf{M}} \odot \mathbf{X}_k + \mathbf{M} \odot \mathbf{Y}) \right\|_F^2 + \beta \|\mathbf{X}\|_* \\ &= \arg \min_{\mathbf{X} \in \mathbb{F}^{M \times N}} \frac{1}{2} \left\| \mathbf{X} - \tilde{\mathbf{X}}_k \right\|_F^2 + \beta \|\mathbf{X}\|_*. \end{aligned}$$

This algorithm alternates between two steps:

- Take the current guess \mathbf{X}_k and replace all the values at sampled locations with the measurements from \mathbf{Y} to get $\tilde{\mathbf{X}}_k$. (This can be done “in place.”)
- Apply **singular value soft thresholding** to $\tilde{\mathbf{X}}_k$ to get the next iterate \mathbf{X}_{k+1} .

This MM method decreases its cost function monotonically. Because the cost function is convex, with some additional work one can show that the sequence $\{\mathbf{X}_k\}$ converges to a global minimizer.



Iterative soft-thresholding algorithm (ISTA) ---

In many modern applications, including LRMC, we have a **composite cost function** of the form:

$$\arg \min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{x}) \quad (9.8)$$

where $f(\mathbf{x})$ is convex and has a Lipschitz gradient (smooth) but $g(\mathbf{x})$ is convex but *not necessarily* smooth.

To develop an algorithm for such problems, we first reinterpret GD for minimizing $f(\mathbf{x})$ as follows:

$$\mathbf{x}_{k+1} = \arg \min_{\mathbf{x}} \left\{ f(\mathbf{x}_k) + \langle \nabla f(\mathbf{x}_k), \mathbf{x} - \mathbf{x}_k \rangle + \frac{1}{2\alpha} \|\mathbf{x} - \mathbf{x}_k\|_2^2 \right\}.$$

Proof.

$$\begin{aligned} \mathbf{x}_{k+1} &= \arg \min_{\mathbf{x}} \left\{ \langle \nabla f(\mathbf{x}_k), \mathbf{x} \rangle + \frac{1}{2\alpha} \|\mathbf{x} - \mathbf{x}_k\|_2^2 \right\} \\ &= \dots \text{ (a few steps shown in class)} \\ &= \arg \min_{\mathbf{x}} \frac{1}{2\alpha} \|\mathbf{x} - (\mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k))\|_2^2 = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k), \end{aligned}$$

by expanding the norm and completing the square and ignoring irrelevant constants independent of \mathbf{x} .

This alternate perspective on GD is the key to extending the GD approach to composite cost functions like (9.8). Consider the following iteration:

$$\begin{aligned}\mathbf{x}_{k+1} &= \arg \min_{\mathbf{x}} \left\{ f(\mathbf{x}_k) + \langle \nabla f(\mathbf{x}_k), \mathbf{x} - \mathbf{x}_k \rangle + \frac{1}{2\alpha} \|\mathbf{x} - \mathbf{x}_k\|_2^2 + g(\mathbf{x}) \right\} \\ &= \arg \min_{\mathbf{x}} \left\{ \frac{1}{2\alpha} \|\mathbf{x} - (\mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k))\|_2^2 + g(\mathbf{x}) \right\}.\end{aligned}$$

This algorithm is called **iterative soft-thresholding algorithm (ISTA)**, aka the **iterative shrinkage thresholding algorithm**, or more generally, the **proximal gradient method (PGM)**.

More concisely, ISTA uses the following two steps per iteration:

$$\begin{aligned}\tilde{\mathbf{x}}_k &\triangleq \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k) \text{ (usual GD step)} \\ \mathbf{x}_{k+1} &= \arg \min_{\mathbf{x}} \frac{1}{2\alpha} \|\mathbf{x} - \tilde{\mathbf{x}}_k\|_2^2 + g(\mathbf{x})\end{aligned}\tag{9.9}$$

The first step is a conventional GD step. The second step is called a **proximity operation**.

The method is convergent [3] (in the sense of p. 5.28 in Ch. 5) when the convex part $f(\mathbf{x})$ is smooth, *i.e.*, its gradient $\nabla f(\mathbf{x})$ has Lipschitz constant L , and we choose $0 < \alpha < 2/L$. We do not need $f(\mathbf{x})$ to be quadratic! Convex and smooth (Lipschitz gradient) is sufficient.

Demo

Recall that Nesterov's accelerated gradient method converges faster than GD for problems with smooth cost functions. There are also accelerated versions of ISTA including (**FISTA**) [4] and (**POGM**) [5].

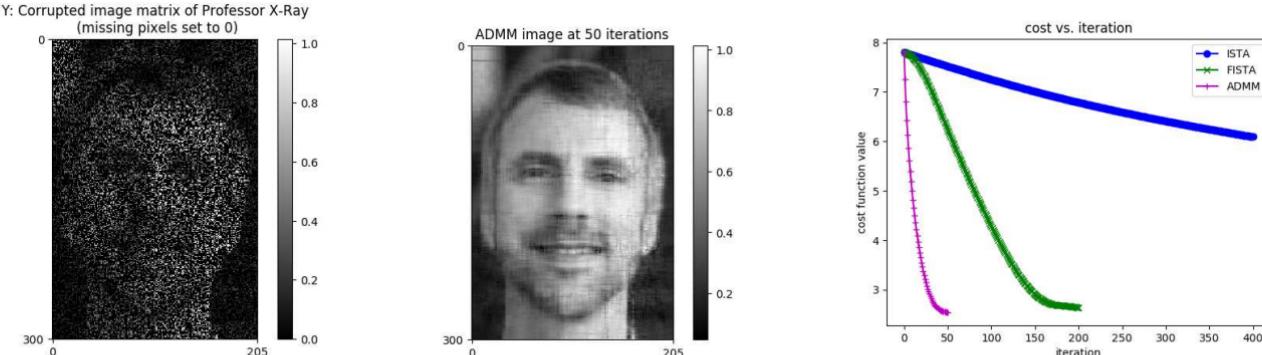
An alternative approach to the LRMC optimization problem is the ADMM method [6]. ADMM requires a tuning parameter for fast convergence. When well tuned, ADMM can be faster than FISTA.

Historically, the original work on this problem used an **EM algorithm** perspective [7].

See the demo notebook created by Dr. Greg Ongie:

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/09_lrmc_nuc.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/09_lrmc_nuc.ipynb



9.4 Summary

Matrix completion using low-rank models is a rich area with numerous applications.

Low-rank matrix *approximation* is an easier problem and Ch. 6 provided nice SVD-based solutions.

LRMC is a more challenging problem (due to the missing data) so algorithms for LRMC typically are iterative.

Interestingly, LRMC algorithm usually use ingredients from low-rank matrix approximation. This pattern of using methods for simpler problems as part of some iterative approach for solving more complicate problem is quite common.

For image processing applications, often low-rank models are applied to collections of image patches, rather than to the entire image, *e.g.*, [8–12].

Bibliography

- [1] Z. Xu. “The minimal measurement number for low-rank matrix recovery”. In: *Applied and Computational Harmonic Analysis* 44.2 (Mar. 2018), 497–508.
- [2] E. Candes and B. Recht. “Exact matrix completion via convex optimization”. In: *Found. Comp. Math.* 9.6 (2009), 717–72.
- [3] K. Bredies and D. A. Lorenz. “Linear convergence of iterative soft-thresholding”. In: *J. Fourier Anal. and Appl.* 14.5 (Dec. 2008), 813–37.
- [4] A. Beck and M. Teboulle. “A fast iterative shrinkage-thresholding algorithm for linear inverse problems”. In: *SIAM J. Imaging Sci.* 2.1 (2009), 183–202.
- [5] A. B. Taylor, J. M. Hendrickx, and Francois Glineur. “Exact worst-case performance of first-order methods for composite convex optimization”. In: *SIAM J. Optim.* 27.3 (Jan. 2017), 1283–313.
- [6] J. Cai, E. Candes, and Z. Shen. “A singular value thresholding algorithm for matrix completion”. In: *SIAM J. Optim.* 20.4 (2010), 1956–82.
- [7] N. Srebro and T. Jaakkola. “Weighted low-rank approximations”. In: *Proc. Intl. Conf. Mach. Learn.* 2003, 720–7.
- [8] H. Ji, C. Liu, Z. Shen, and Y. Xu. “Robust video denoising using low rank matrix completion”. In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2010, 1791–8.
- [9] J. D. Trzasko and A. Manduca. “Calibrationless parallel MRI using CLEAR”. In: *Proc., IEEE Asilomar Conf. on Signals, Systems, and Comp.* 2011, 75–9.
- [10] W. Dong, G. Shi, and X. Li. “Nonlocal image restoration with bilateral variance estimation: A low-rank approach”. In: *IEEE Trans. Im. Proc.* 22.2 (Feb. 2013), 700–11.
- [11] W. Dong, G. Shi, X. Li, Y. Ma, and F. Huang. “Compressive sensing via nonlocal low-rank regularization”. In: *IEEE Trans. Im. Proc.* 23.8 (Aug. 2014), 3618–32.
- [12] S. Ravishankar, B. E. Moore, R. R. Nadakuditi, and J. A. Fessler. “Low-rank and adaptive sparse signal (LASSI) models for highly accelerated dynamic imaging”. In: *IEEE Trans. Med. Imag.* 36.5 (May 2017), 1116–28.

Chapter 99

Miscellaneous topics

Contents (final version)

99.0 Review / practice questions	99.2
Ch01: Matrices	99.2
Ch02: Matrix decompositions	99.3
Ch03: Subspaces	99.4
Ch04: Linear least-squares	99.5
Ch05: Norms	99.6
Ch06: Low-rank approximation	99.7
Ch07: Special matrices	99.9
Ch08: Optimization	99.10
Ch09: Matrix completion	99.12

99.0 Review / practice questions

Disclaimer: this set of remarks and questions is far from being comprehensive!

The goal is to highlight some of the main take-aways, while also reiterating some subtle points.

Ch01: Matrices

shapes / transpose / symmetry / multiplication / orthogonality / determinant / eigenvalues / trace

If $\mathbf{u}_1, \dots, \mathbf{u}_N$ are **orthogonal** vectors in \mathbb{F}^N , then $\mathbf{U} = [\mathbf{u}_1 \ \dots \ \mathbf{u}_N]$ is an **orthogonal matrix**. (?)

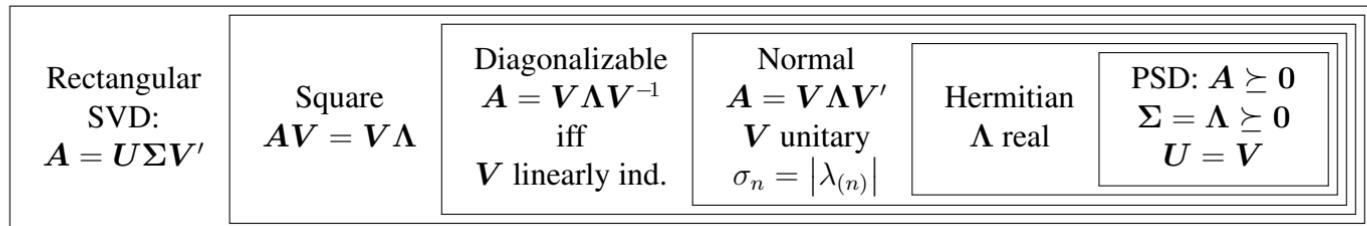
A: True

B: False

??

Ch02: Matrix decompositions

Venn diagram of matrices and decompositions



where $|\lambda_{(n)}|$ denotes the n th largest magnitude eigenvalue.

If A is positive semidefinite in $\mathbb{F}^{N \times N}$ with rank r , then there exists a set of orthonormal vectors $\{z_1, \dots, z_r\}$ in \mathbb{F}^N such that $A = \sum_{k=1}^r \sigma_k z_k z'_k$ is a **compact SVD** of A .

A: True

B: False

??

If a companion matrix has repeated eigenvalues, then it is not diagonalizable.

A: True

B: False

??

If a square matrix has repeated eigenvalues, then it is not diagonalizable.

A: True

B: False

??

Ch03: Subspaces

span / linear dependence / basis / dimension / direct sum / orthogonal complement / range / nullspace / rank / four fundamental spaces / orthogonal bases

If A is a $M \times N$ matrix with $M > N$, then $\dim(\mathcal{R}(A)) = N$.

A: True

B: False

??

If S is a subspace with basis vectors $\{b_1, \dots, b_K\}$ then $\dim(S) = K$ (?)

A: True

B: False

??

If S is a **subspace** with **basis** $\{b_1, \dots, b_K\}$ and we define $B = [b_1 \ \dots \ b_K]$, then P_S is (?)

A: BB'

B: $B'B$

C: B^+B

D: BB^+

E: None of these.

??

If S is a subspace with **orthogonal basis** $\{b_1, \dots, b_K\}$ then $P_S^\perp = I - BB'$. (?)

A: True

B: False

??

If $\{S_1, \dots, S_K\}$ are K subspaces, then an appropriate nearest-subspace classifier of x is:

A: $\arg \min_k \|P_{S_k}^\perp x\|$ B: $\arg \min_k \|P_{S_k} x\|$ C: $\arg \max_k \|P_{S_k}^\perp x\|$ D: $\arg \max_k \|P_{S_k} x\|$ E: None of these

??

Ch04: Linear least-squares

pseudo-inverse / Tikhonov regularization / truncated SVD / frames / projection / orthogonal projection

If $A \in \mathbb{F}^{M \times N}$ has compact SVD $U_r \Sigma_r V_r'$, then a solution to the **LS problem** $\arg \min_x \|Ax - y\|_2$ is $x_* = V_r \Sigma_r^{-1} U_r' y + (I - V_r V_r') z$ for any vector $z \in \mathbb{F}^N$. (?)

A: True

B: False

??

A solution to $\arg \min_x \|Ax - y\|_2^2 + \|Bx - z\|_2^2$ is:

- A: $A^+y + B^+z$ B: $(A + B)^+(y + z)$ C: $[A, B]^+ [y, z]$ D: $\begin{bmatrix} A \\ B \end{bmatrix}^+ \begin{bmatrix} y \\ z \end{bmatrix}$ E: None of these. ??

If $\Phi \in \mathbb{F}^{N \times M}$ is a **tight frame** and U is a $M \times M$ unitary matrix, then $B = \Phi U$ is a frame. (?)

A: True

B: False

??

Ch05: Norms

vector norms / inner products / Cauchy-Schwarz inequality / matrix norms / induced norms / unitary invariance / norm equivalence / spectral radius / orthogonal Procrustes problem / Stiefel manifold

If W is a **positive semidefinite** matrix, then $\|x\|_W \triangleq \sqrt{x'Wx}$ is a norm. (?)

A: True

B: False

??

If B and A are $M \times N$ then

$$\min_{Q : Q'Q = I_M} \|B - QA\|_F^2 = \|B\|_F^2 + \|B\|_F^2 - 2\|BA'\|_*.$$

(?)

A: True

B: False

??

??

Ch06: Low-rank approximation

Frobenius norm / generalizations to UI norms / hard thresholding / soft thresholding / choosing β or rank / OptShrink / Subspace learning

Determine $\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathbb{R}^{M \times N}} \|\mathbf{X} - \mathbf{Y}\|_{S,4}^4 + \beta \operatorname{rank}(\mathbf{X})$ and consider the *units* of β and \hat{w}_k .

??

Determine $\hat{\mathbf{X}} = \arg \min_{\mathbf{X} \in \mathbb{F}^{M \times N}} \frac{1}{4} \|\mathbf{X} - \mathbf{Y}\|_{S,4}^4 + \beta \|\mathbf{X}\|_*$. Think about units again.

??

Ch07: Special matrices

companion / Vandermonde / circulant / power iteration / primitive / irreducible / Markov chains / PageRank

If C is a **Hermitian circulant** matrix, then the vector x with elements $x_n = \cos(2\pi kn/N)$ is an **eigenvector** of C for any $k \in \mathbb{Z}$.

A: True

B: False

??

??

Ch08: Optimization

When solving a LS problem with $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix}$ via GD, what is the optimal step size?

A: 1/5

B: 2/5

C: 3/5

D: 1/3

E: 2/3

??

The Lipschitz constant for the gradient of $f(x) \triangleq x' \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} x$ is:

A: 1

B: 2

C: 4

D: 5

E: 10

??

Suppose A is a wide matrix and consider the regularized LS cost function $f(x) \triangleq \frac{1}{2} \|Ax - y\|_2^2 + \beta \|x\|_2^2$ where $\beta > 0$. Then f is a **strictly convex** function. (?)

A: True

B: False

??

Is the Fair potential $\psi(x) = |x| - \log(1 + |x|)$ itself **Lipschitz continuous?**

- A: No B: Yes with $L = 1/2$ C: Yes with $L = 1$ D: Yes with $L = 2$

??

For $\mathcal{C} = \{x \in \mathbb{R}^N : \|x\|_\infty \leq 5\}$, which of these is the projection of a point $z \in \mathbb{R}^N$ onto \mathcal{C} ?

- A: $\min.(x, 5)$ B: $\min.(\text{abs.}(x), 5)$ C: $\min.(\text{abs.}(x), 5) .* \text{sign}(x)$ D: None of these

??

Example. If $f(x) = \frac{1}{2} \left\| \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} x - \begin{bmatrix} 4 \\ -2 \end{bmatrix} \right\|_2^2$ and we apply GP with $\alpha = 1/L$ and $x_0 = 0$ then $x_1 = ?$

- A: (0, 0) B: (1, 0) C: (1, -1) D: (0, -1) E: (1, 1)

??

Ch09: Matrix completion

alternating projection / MM methods / iterative SVHT / SVST

If $\mathbf{Y} = \begin{bmatrix} 1 & ? \\ 0 & 4 \end{bmatrix}$ and we apply one iteration of the alternating projection method for rank $K = 1$

starting with $\mathbf{X}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$ then $\mathbf{X}_1 = ?$

- A: $\begin{bmatrix} 0 & 0 \\ 0 & 4 \end{bmatrix}$ B: $\begin{bmatrix} 4 & 0 \\ 0 & 0 \end{bmatrix}$ C: $\begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$ D: $\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ E: None of these. ??