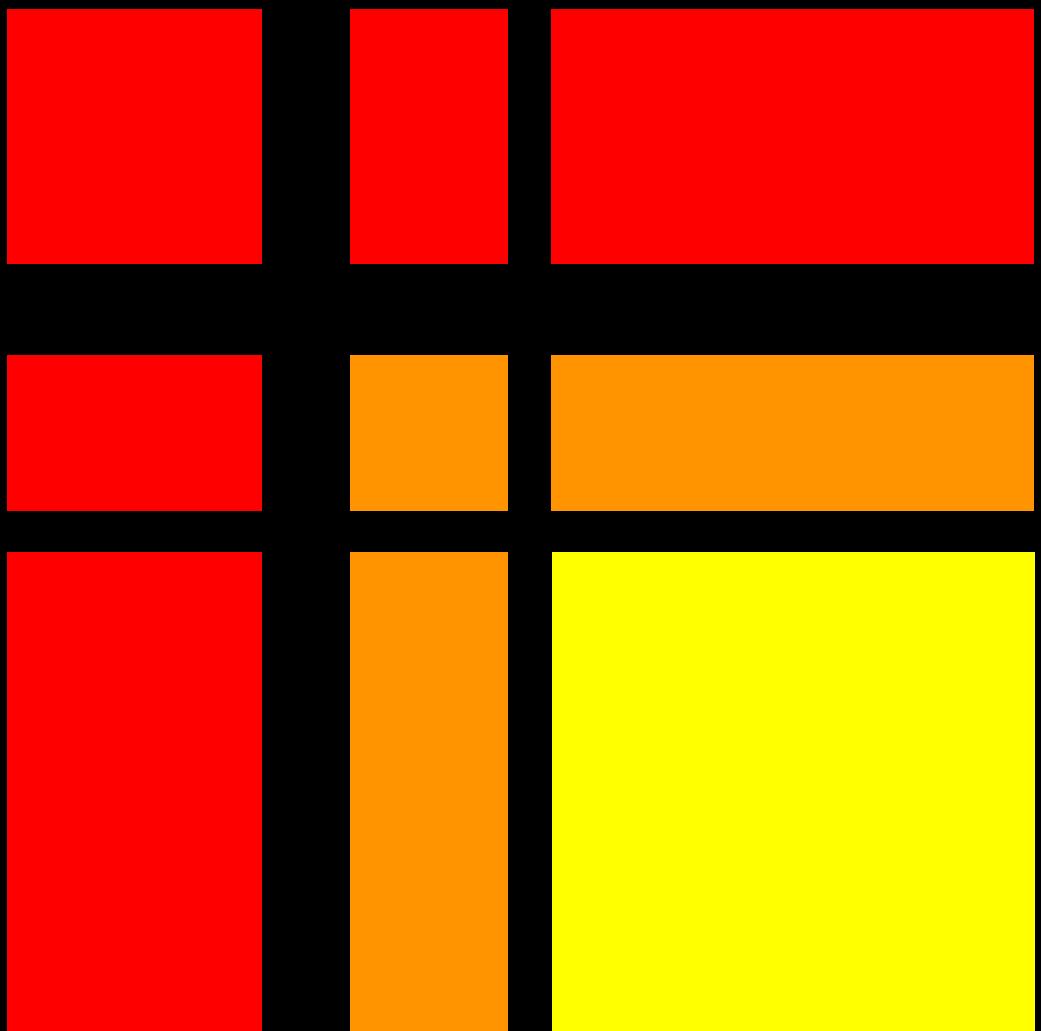


Advanced Linear Algebra

Foundations to Frontiers



Robert A. van de Geijn
Margaret E. Myers

Advanced Linear Algebra

Foundations to Frontiers

Advanced Linear Algebra

Foundations to Frontiers

Robert van de Geijn
The University of Texas at Austin

Margaret Myers
The University of Texas at Austin

August 17, 2020

Edition: Draft Edition 2019–2020

Website: ulaff.net

©2019–2020 Robert van de Geijn and Margaret Myers

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the appendix entitled “GNU Free Documentation License.” All trademarksTM are the registered® marks of their respective owners.

Acknowledgements

We would like to thank the people who created PreTeXt, the authoring system used to typeset these materials. We applaud you!

Preface

Robert van de Geijn
Maggie Myers
Austin, 2020

Contents

Acknowledgements	vii
Preface	viii
0 Getting Started	1
I Orthogonality	
1 Norms	12
2 The Singular Value Decomposition	89
3 The QR Decomposition	151
4 Linear Least Squares	212
II Solving Linear Systems	
5 The LU and Cholesky Factorizations	255
6 Numerical Stability	331
7 Solving Sparse Linear Systems	382

8 Descent Methods	416
III The Algebraic Eigenvalue Problem	
9 Eigenvalues and Eigenvectors	460
10 Practical Solution of the Hermitian Eigenvalue Problem	514
11 Computing the SVD	556
12 Attaining High Performance	585
A Are you ready?	621
B Notation	622
C Knowledge from Numerical Analysis	623
D GNU Free Documentation License	625
References	633
Index	637

Week 0

Getting Started

0.1 Opening Remarks

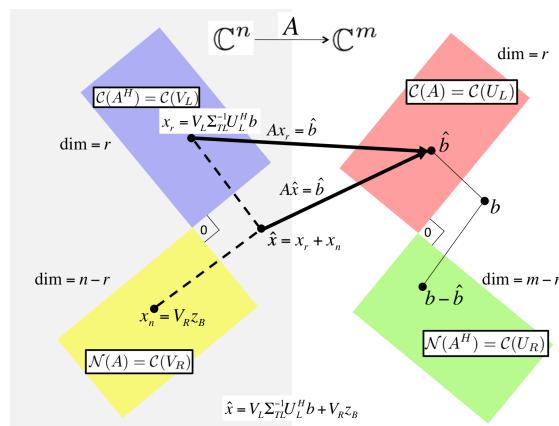
0.1.1 Welcome



YouTube: <https://www.youtube.com/watch?v=KzCTMLvxtQA>

Linear algebra is one of the fundamental tools for computational and data scientists. In Advanced Linear Algebra: Foundations to Frontiers (ALAFF), you build your knowledge, understanding, and skills in linear algebra, practical algorithms for matrix computations, and how floating-point arithmetic, as performed by computers, affects correctness.

The materials are organized into Weeks that correspond to a chunk of information that is covered in a typical on-campus week. These weeks are arranged into three parts:



Part I: Orthogonality

The Singular Value Decomposition (SVD) is possibly the most important result in linear algebra, yet too advanced to cover in an introductory undergraduate course. To be able to get to this topic as quickly as possible, we start by focusing on orthogonality, which is at the heart of image compression, Google's page rank algorithm, and linear least-squares approximation.

Part II: Solving Linear Systems

Solving linear systems, via direct or iterative methods, is at the core of applications in computational science and machine learning. We also leverage these topics to introduce numerical stability of algorithms: the classical study that qualifies and quantifies the "correctness" of an algorithm in the presence of floating point computation and approximation. Along the way, we discuss how to restructure algorithms so that they can attain high performance on modern CPUs.

Algorithm: Compute LU factorization with partial pivoting of A , overwriting A with factors L and U . The pivot vector is returned in p .

$$\text{Partition } A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ p_B \end{array} \right).$$

where A_{TL} is 0×0 and p_T is 0×1

while $n(A_{TL}) < n(A)$ do

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \pi_1 \\ p_2 \end{array} \right)$$

where $\alpha_{11}, \lambda_{11}, \pi_1$ are 1×1

$$\pi_1 = \max_i \left(\frac{\alpha_{11}}{\alpha_{21}} \right)$$

$$\left(\begin{array}{c|c|c} a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right) := P(\pi_1) \left(\begin{array}{c|c|c} a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

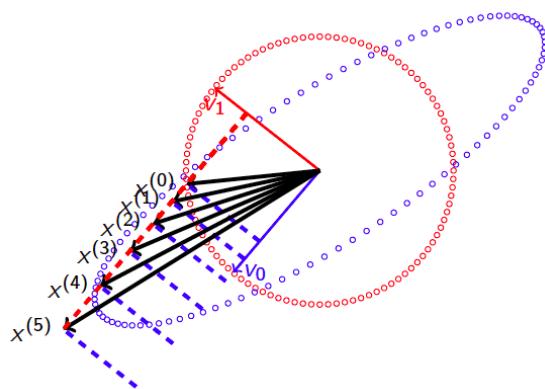
$$a_{21} := a_{21}/\alpha_{11}$$

$$A_{22} := A_{22} - a_{21}a_{12}^T$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ p_B \end{array} \right) \leftarrow \left(\begin{array}{c} p_0 \\ \pi_1 \\ p_2 \end{array} \right)$$

endwhile



Part III: Eigenvalues and Eigenvectors

Many problems in science have the property that if one looks at them in just the right way (in the right basis), they greatly simplify and/or decouple into simpler subproblems. Eigenvalue and eigenvectors are the key to discovering how to view a linear transformation, represented by a matrix, in that special way. Algorithms for computing them also are the key to practical algorithms for computing the SVD

In this week (Week 0), we walk you through some of the basic course information and help you set up for learning. The week itself is structured like future weeks, so that you become familiar with that structure.

0.1.2 Outline Week 0

Each week is structured so that we give the outline for the week immediately after the "launch:"

- 0.1 Opening Remarks
 - 0.1.1 Welcome
 - 0.1.2 Outline Week 0
 - 0.1.3 What you will learn
- 0.2 Setting Up For ALAFF

- 0.2.1 Accessing these notes
- 0.2.2 Cloning the ALAFF repository
- 0.2.3 MATLAB
- 0.2.4 Setting up to implement in C (optional)
- 0.3 Enrichments
 - 0.3.1 Ten surprises from numerical linear algebra
 - 0.3.2 Best algorithms of the 20th century
- 0.4 Wrap Up
 - 0.4.1 Additional Homework
 - 0.4.2 Summary

0.1.3 What you will learn

The third unit of each week informs you of what you will learn. This describes the knowledge and skills that you can expect to acquire. If you return to this unit after you complete the week, you will be able to use the below to self-assess.

Upon completion of this week, you should be able to

- Navigate the materials.
- Access additional materials from GitHub.
- Track your homework and progress.
- Register for MATLAB online.
- Recognize the structure of a typical week.

0.2 Setting Up For ALAFF

0.2.1 Accessing these notes

For information regarding these and our other materials, visit ulaff.net.

These notes are available in a number of formats:

- As an online book authored with PreTeXt at <http://www.cs.utexas.edu/users/flame/laff/alaff/>.

- As a PDF at <http://www.cs.utexas.edu/users/flame/laff/alaff/ALAFF.pdf>.

If you download this PDF and place it in just the right folder of the materials you will clone from GitHub (see next unit), the links in the PDF to the downloaded material will work.

We will be updating the materials frequently as people report typos and we receive feedback from learners. Please consider the environment before you print a copy...

- Eventually, if we perceive there is demand, we may offer a printed copy of these notes from [Lulu.com](#), a self-publishing service. This will not happen until Summer 2020, at the earliest.

0.2.2 Cloning the ALAFF repository

We have placed all materials on GitHub, a development environment for software projects. In our case, we use it to disseminate the various activities associated with this course.

On the computer on which you have chosen to work, "clone" the GitHub repository for this course:

- Visit <https://github.com/ULAFF/ALAFF>
- Click on



and copy <https://github.com/ULAFF/ALAFF.git>.

- On the computer where you intend to work, in a terminal session on the command line in the directory where you would like to place the materials, execute

```
git clone https://github.com/ULAFF/ALAFF.git
```

This will create a local copy (clone) of the materials.

- Sometimes we will update some of the files from the repository. When this happens you will want to execute, in the cloned directory,

```
git stash save
```

which saves any local changes you have made, followed by

```
git pull
```

which updates your local copy of the repository, followed by

```
git stash pop
```

which restores local changes you made. This last step may require you to "merge" files that were changed in the repository that conflict with local changes.

Upon completion of the cloning, you will have a directory structure similar to that given in Figure 0.2.2.1.

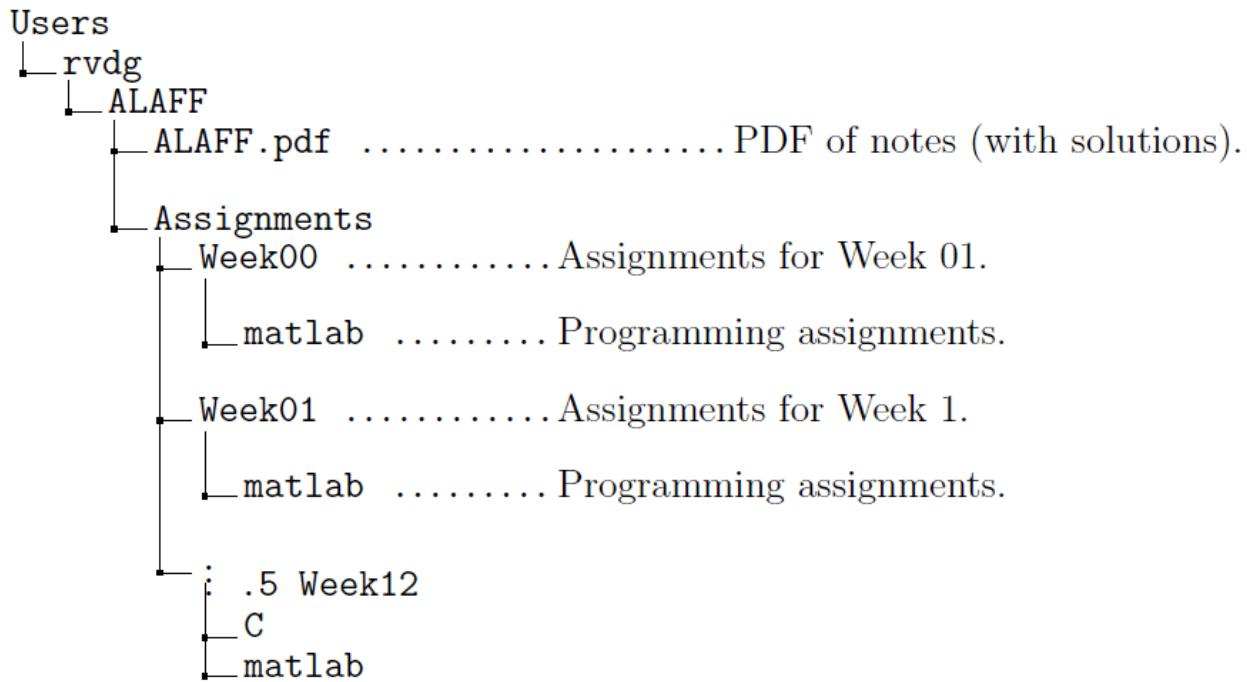


Figure 0.2.2.1 Directory structure for your ALAFF materials. In this example, we cloned the repository in Robert's home directory, rvdg.

0.2.3 MATLAB

We will use Matlab to translate algorithms into code and to experiment with linear algebra.

There are a number of ways in which you can use Matlab:

- Via MATLAB that is installed on the same computer as you will execute your performance experiments. This is usually called a "desktop installation of Matlab."
- Via [MATLAB Online](#). You will have to transfer files from the computer where you are performing your experiments to MATLAB Online. You could try to set up [MATLAB Drive](#), which allows you to share files easily between computers and with MATLAB Online. Be warned that there may be a delay in when files show up, and as a result you may be using old data to plot if you aren't careful!

If you are using these materials as part of an offering of the Massive Open Online Course (MOOC) titled "Advanced Linear Algebra: Foundations to Frontiers," you will be given a temporary license to Matlab, courtesy of MathWorks. In this case, there will be additional

instructions on how to set up MATLAB Online, in the Unit on edX that corresponds to this section.

You need relatively little familiarity with MATLAB in order to learn what we want you to learn in this course. So, you could just skip these tutorials altogether, and come back to them if you find you want to know more about MATLAB and its programming language (M-script).

Below you find a few short videos that introduce you to MATLAB. For a more comprehensive tutorial, you may want to visit [MATLAB Tutorials](#) at MathWorks and click "Launch Tutorial".

What is MATLAB?



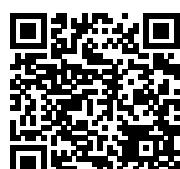
<https://www.youtube.com/watch?v=2sB-NMD9Qhk>

Getting Started with MATLAB Online



<https://www.youtube.com/watch?v=4shp284pGc8>

MATLAB Variables



<https://www.youtube.com/watch?v=gPIsIzHJA9I>

MATLAB as a Calculator



<https://www.youtube.com/watch?v=K9xy5kQHDBo>

Managing Files with MATLAB Online



<https://www.youtube.com/watch?v=mqYwMnM-x5Q>

Remark 0.2.3.1 Some of you may choose to use MATLAB on your personal computer while others may choose to use MATLAB Online. Those who use MATLAB Online will need to transfer some of the downloaded materials to that platform.

0.2.4 Setting up to implement in C (optional)

You may want to return to this unit later in the course. We are still working on adding programming exercises that require C implementation.

In some of the enrichments in these notes and the final week on how to attain performance, we suggest implementing algorithms that are encountered in C. Those who intend to pursue these activities will want to install a Basic Linear Algebra Subprograms (BLAS) library and our libflame library (which not only provides higher level linear algebra functionality, but also allows one to program in a manner that mirrors how we present algorithms.)

0.2.4.1 Installing the BLAS

The Basic Linear Algebra Subprograms (BLAS) are an interface to fundamental linear algebra operations. The idea is that if we write our software in terms of calls to these routines and vendors optimize an implementation of the BLAS, then our software can be easily ported to different computer architectures while achieving reasonable performance.

A popular and high-performing open source implementation of the BLAS is provided by our BLAS-like Library Instantiation Software (BLIS). The following steps will install BLIS if you are using the Linux OS (on a Mac, there may be a few more steps, which are discussed later in this unit.)

- Visit the [BLIS Github repository](#).
- Click on

[Clone or download ▾](#)

and copy <https://github.com/flame/blis.git>.

- In a terminal session, in your home directory, enter

```
git clone https://github.com/flame/blis.git
```

(to make sure you get the address right, you will want to paste the address you copied in the last step.)

- Change directory to blis:

```
cd blis
```

- Indicate a specific version of BLIS so that we all are using the same release:

```
git checkout pfhp
```

- Configure, build, and install with OpenMP turned on.

```
./configure -p ~/blis auto
make -j8
make check -j8
make install
```

The `-p ~/blis` installs the library in the subdirectory `~/blis` of your home directory, which is where the various exercises in the course expect it to reside.

- If you run into a problem while installing BLIS, you may want to consult <https://github.com/flame/blis/blob/master/docs/BuildSystem.md>.

On Mac OS-X

- You may need to install Homebrew, a program that helps you install various software on your mac. Warning: you may need "root" privileges to do so.

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/ma...)"
```

Keep an eye on the output to see if the "Command Line Tools" get installed. This may not be installed if you already have Xcode Command line tools installed. If this happens, post in the "Discussion" for this unit, and see if someone can help you out.

- Use Homebrew to install the gcc compiler:

```
$ brew install gcc
```

Check if gcc installation overrides clang:

```
$ which gcc
```

The output should be `/usr/local/bin`. If it isn't, you may want to add `/usr/local/bin` to "the path." I did so by inserting

```
export PATH="/usr/local/bin:$PATH"
```

into the file `.bash_profile` in my home directory. (Notice the "period" before `.bash_profile`)

- Now you can go back to the beginning of this unit, and follow the instructions to install BLIS.

0.2.4.2 Installing libflame

Higher level linear algebra functionality, such as the various decompositions we will discuss in this course, are supported by the LAPACK library [1]. Our libflame library is an implementation of LAPACK that also exports an API for representing algorithms in code in a way that closely reflects the FLAME notation to which you will be introduced in the course.

The libflame library can be cloned from

- <https://github.com/flame/libflame>.

by executing

```
git clone https://github.com/flame/libflame.git
```

in the command window.

Instructions on how to install it are at

- <https://github.com/flame/libflame/blob/master/INSTALL>.

Here is what I had to do on my MacBook Pro (OSX Catalina):

```
./configure --disable-autodetect-f77-ldflags --disable-autodetect-f77-name-mangling --pref:  
make -j8  
make install
```

This will take a while!

0.3 Enrichments

In each week, we include "enrichments" that allow the participant to go beyond.

0.3.1 Ten surprises from numerical linear algebra

You may find the following list of insights regarding numerical linear algebra, compiled by John D. Cook, interesting:

- John D. Cook. [Ten surprises from numerical linear algebra](#). 2010.

0.3.2 Best algorithms of the 20th century

An article published in SIAM News, a publication of the Society for Industrial and Applied Mathematics, lists the ten most important algorithms of the 20th century [10]:

1. *1946*: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the *Metropolis algorithm*, also known as the Monte Carlo method.
2. *1947*: George Dantzig, at the RAND Corporation, creates the *simplex method for linear programming*.
3. *1950*: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis at the National Bureau of Standards, initiate the development of *Krylov subspace iteration methods*.
4. *1951*: Alston Householder of Oak Ridge National Laboratory formalizes the *decompositional approach to matrix computations*.

5. 1957: John Backus leads a team at IBM in developing the *Fortran optimizing compiler*.
6. 1959–61: J.G.F. Francis of Ferranti Ltd., London, finds a stable method for computing eigenvalues, known as the *QR algorithm*.
7. 1962: Tony Hoare of Elliott Brothers, Ltd., London, presents *Quicksort*.
8. 1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories unveil the *fast Fourier transform*.
9. 1977: Helaman Ferguson and Rodney Forcade of Brigham Young University advance an *integer relation detection algorithm*.
10. 1987: Leslie Greengard and Vladimir Rokhlin of Yale University invent the *fast multipole algorithm*.

Of these, we will explicitly cover three: the decomposition method to matrix computations, Krylov subspace methods, and the QR algorithm. Although not explicitly covered, your understanding of numerical linear algebra will also be a first step towards understanding some of the other numerical algorithms listed.

0.4 Wrap Up

0.4.1 Additional Homework

For a typical week, additional assignments may be given in this unit.

0.4.2 Summary

In a typical week, we provide a quick summary of the highlights in this unit.

Part I

Orthogonality

Week 1

Norms

1.1 Opening

1.1.1 Why norms?



YouTube: <https://www.youtube.com/watch?v=DKX3TdQWQ90>

The following exercises expose some of the issues that we encounter when computing.
We start by computing $b = Ux$, where U is upper triangular.

Homework 1.1.1.1 Compute

$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & -1 & -1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} =$$

Solution.

$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & -1 & -1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -4 \\ -3 \\ 2 \end{pmatrix}$$

Next, let's examine the slightly more difficult problem of finding a vector x that satisfies $Ux = b$.

Homework 1.1.1.2 Solve

$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & -1 & -1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} -4 \\ -3 \\ 2 \end{pmatrix}$$

Solution. We can recognize the relation between this problem and [Homework 1.1.1.1](#) and hence deduce the answer without computation:

$$\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}$$

The point of these two homework exercises is that if one creates a (nonsingular) $n \times n$ matrix U and vector x of size n , then computing $b = Ux$ followed by solving $U\hat{x} = b$ should leave us with a vector \hat{x} such that $x = \hat{x}$.

Remark 1.1.1.1 We don't "teach" Matlab in this course. Instead, we think that Matlab is intuitive enough that we can figure out what the various commands mean. We can always investigate them by typing

`help <command>`

in the command window. For example, for this unit you may want to execute

```
help format
help rng
help rand
help triu
help *
help \
help diag
help abs
help min
help max
```

If you want to learn more about Matlab, you may want to take some of the tutorials offered by Mathworks at <https://www.mathworks.com/support/learn-with-matlab-tutorials.html>.

Let us see if Matlab can compute the solution of a triangular matrix correctly.

Homework 1.1.1.3 In Matlab's command window, create a random upper triangular matrix U :

```
format long
rng( 0 );
n = 3
U = triu( rand( n,n ) )
x = rand( n,1 )
```

Report results in long format. Seed the random number generator so that we all create the same random matrix U and vector x .

<code>b = U * x;</code>	Compute right-hand side b from known solution x .
<code>xhat = U \ b;</code>	Solve $U\hat{x} = b$.
<code>xhat - x</code>	Report the difference between \hat{x} and x .

What do we notice?

Next, check how close $U\hat{x}$ is to $b = Ux$:

`b - U * xhat`

This is known as the residual.

What do we notice?

Solution. A script with the described commands can be found in [Assignments/Week01/matlab/Test_Upper_triangular_solve_3.m](#).

Some things we observe:

- $\hat{x} - x$ does not equal zero. This is due to the fact that the computer stores floating point numbers and computes with floating point arithmetic, and as a result roundoff error happens.
- The difference is small (notice the $1.0e-15*$ before the vector, which shows that each entry in $\hat{x} - x$ is around 10^{-15}).
- The residual $b - U\hat{x}$ is small.
- Repeating this with a much larger n make things cumbersome since very long vectors are then printed.

To be able to compare more easily, we will compute the Euclidean length of $\hat{x} - x$ instead using the Matlab command `norm(xhat - x)`. By adding a semicolon at the end of Matlab commands, we suppress output.

Homework 1.1.1.4 Execute

`format long`

```

rng( 0 );
n = 100;
U = triu( rand( n,n ) );
x = rand( n,1 );

b = U * x;

```

`xhat = U \ b;`

`norm(xhat - x)`

Report results in long format.

Seed the random number generator so that we all create the same random matrix U and vector x .

Compute right-hand side b from known solution x .

Solve $U\hat{x} = b$

Report the Euclidean length of the difference between \hat{x} and x .

What do we notice?

Next, check how close $U\hat{x}$ is to $b = Ux$, again using the Euclidean length:

`norm(b - U * xhat)`

What do we notice?

Solution. A script with the described commands can be found in [Assignments/Week01/matlab/Test_Upper_triangular_solve_100.m](#).

Some things we observe:

- $\text{norm}(\hat{x} - x)$, the Euclidean length of $\hat{x} - x$, is huge. Matlab computed the wrong answer!
- However, the computed \hat{x} solves a problem that corresponds to a slightly different right-hand side. Thus, \hat{x} appears to be the solution to an only slightly changed problem.

The next exercise helps us gain insight into what is going on.

Homework 1.1.1.5 Continuing with the U , x , b , and \hat{x} from [Homework 1.1.1.4](#), consider

- When is an upper triangular matrix singular?
- How large is the smallest element on the diagonal of the U from [Homework 1.1.1.4](#)? ($\text{min}(\text{abs}(\text{diag}(U)))$ returns it!)
- If U were singular, how many solutions to $U\hat{x} = b$ would there be? How can we characterize them?
- What is the relationship between $\hat{x} - x$ and U ?

What have we learned?

Solution.

- When is an upper triangular matrix singular?

Answer:

If and only if there is a zero on its diagonal.

- How large is the smallest element on the diagonal of the U from [Homework 1.1.1.4](#)? ($\text{min}(\text{abs}(\text{diag}(U)))$ returns it!)

Answer:

It is small in magnitude. This is not surprising, since it is a random number and hence as the matrix size increases, the chance of placing a small entry (in magnitude) on the diagonal increases.

- If U were singular, how many solutions to $U\hat{x} = b$ would there be? How can we characterize them?

Answer:

An infinite number. Any vector in the null space can be added to a specific solution to create another solution.

- What is the relationship between $\hat{x} - x$ and U ?

Answer:

It maps almost to the zero vector. In other words, it is close to a vector in the null space of the matrix U that has its smallest entry (in magnitude) on the diagonal changed to a zero.

What have we learned? The :"wrong" answer that Matlab computed was due to the fact that matrix U was almost singular.

To mathematically qualify and quantify all this, we need to be able to talk about "small" and "large" vectors, and "small" and "large" matrices. For that, we need to generalize the notion of length. By the end of this week, this will give us some of the tools to more fully understand what we have observed.



YouTube: <https://www.youtube.com/watch?v=2ZEtcnaynnM>

1.1.2 Overview

- 1.1 Opening
 - 1.1.1 Why norms?
 - 1.1.2 Overview
 - 1.1.3 What you will learn
- 1.2 Vector Norms
 - 1.2.1 Absolute value
 - 1.2.2 What is a vector norm?
 - 1.2.3 The vector 2-norm (Euclidean length)
 - 1.2.4 The vector p-norms
 - 1.2.5 Unit ball
 - 1.2.6 Equivalence of vector norms
- 1.3 Matrix Norms
 - 1.3.1 Of linear transformations and matrices
 - 1.3.2 What is a matrix norm?

- 1.3.3 The Frobenius norm
- 1.3.4 Induced matrix norms
- 1.3.5 The matrix 2-norm
- 1.3.6 Computing the matrix 1-norm and ∞ -norm
- 1.3.7 Equivalence of matrix norms
- 1.3.8 Submultiplicative norms
- 1.3.9 Summary
- 1.4 Condition Number of a Matrix
 - 1.4.1 Conditioning of a linear system
 - 1.4.2 Loss of digits of accuracy
 - 1.4.3 The conditioning of an upper triangular matrix
- 1.5 Enrichments
 - 1.5.1 Condition number estimation
- 1.6 Wrap Up
 - 1.6.1 Additional homework
 - 1.6.2 Summary

1.1.3 What you will learn

Numerical analysis is the study of how the perturbation of a problem or data affects the accuracy of computation. This inherently means that you have to be able to measure whether changes are large or small. That, in turn, means we need to be able to quantify whether vectors or matrices are large or small. Norms are a tool for measuring magnitude.

Upon completion of this week, you should be able to

- Prove or disprove that a function is a norm.
- Connect linear transformations to matrices.
- Recognize, compute, and employ different measures of length, which differ and yet are equivalent.
- Exploit the benefits of examining vectors on the unit ball.
- Categorize different matrix norms based on their properties.
- Describe, in words and mathematically, how the condition number of a matrix affects how a relative change in the right-hand side can amplify into relative change in the solution of a linear system.
- Use norms to quantify the conditioning of solving linear systems.

1.2 Vector Norms

1.2.1 Absolute value

Remark 1.2.1.1 Don't Panic!

In this course, we mostly allow scalars, vectors, and matrices to be complex-valued. This means we will use terms like "conjugate" and "Hermitian" quite liberally. You will think this is a big deal, but actually, if you just focus on the real case, you will notice that the complex case is just a natural extension of the real case.

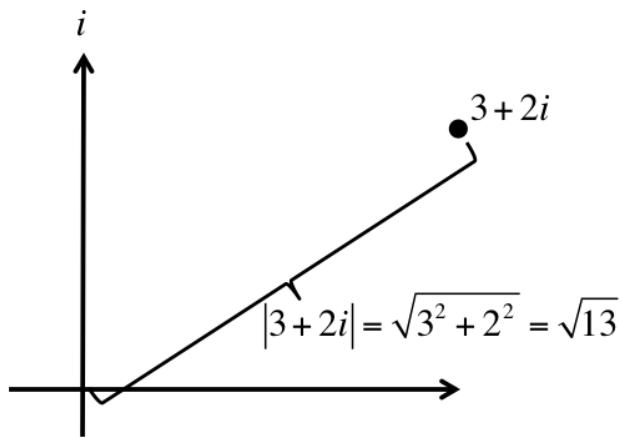


YouTube: <https://www.youtube.com/watch?v=V5ZQmR4zTeU>

Recall that $|\cdot| : \mathbb{C} \rightarrow \mathbb{R}$ is the function that returns the absolute value of the input. In other words, if $\alpha = \alpha_r + \alpha_c i$, where α_r and α_c are the real and imaginary parts of α , respectively, then

$$|\alpha| = \sqrt{\alpha_r^2 + \alpha_c^2}.$$

The absolute value (magnitude) of a complex number can also be thought of as the (Euclidean) distance from the point in the complex plane to the origin of that plane, as illustrated below for the number $3 + 2i$.



Alternatively, we can compute the absolute value as

$$\begin{aligned}
 |\alpha| &= \\
 &= \sqrt{\alpha_r^2 + \alpha_c^2} \\
 &= \sqrt{\alpha_r^2 - \alpha_c \alpha_r i + \alpha_r \alpha_c i + \alpha_c^2} \\
 &= \sqrt{(\alpha_r - \alpha_c i)(\alpha_r + \alpha_c i)} \\
 &= \sqrt{\bar{\alpha}\alpha} ,
 \end{aligned}$$

where $\bar{\alpha}$ denotes the complex conjugate of α :

$$\bar{\alpha} = \overline{\alpha_r + \alpha_c i} = \alpha_r - \alpha_c i.$$

The absolute value function has the following properties:

- $\alpha \neq 0 \Rightarrow |\alpha| > 0$ ($|\cdot|$ is positive definite),
- $|\alpha\beta| = |\alpha||\beta|$ ($|\cdot|$ is homogeneous), and
- $|\alpha + \beta| \leq |\alpha| + |\beta|$ ($|\cdot|$ obeys the triangle inequality).

Norms are functions from a domain to the real numbers that are positive definite, homogeneous, and obey the triangle inequality. This makes the absolute value function an example of a norm.

The below exercises help refresh your fluency with complex arithmetic.

Homework 1.2.1.1

$$1. (1+i)(2-i) =$$

$$2. (2-i)(1+i) =$$

$$3. \overline{(1-i)}(2-i) =$$

$$4. \overline{\overline{(1-i)}}(2-i) =$$

$$5. \overline{(2-i)}(1-i) =$$

$$6. (1-i)\overline{(2-i)} =$$

Solution.

$$1. (1+i)(2-i) = 2 + 2i - i - i^2 = 2 + i + 1 = 3 + i$$

$$2. (2-i)(1+i) = 2 - i + 2i - i^2 = 2 + i + 1 = 3 + i$$

$$3. \overline{(1-i)}(2-i) = (1+i)(2-i) = 2 - i + 2i - i^2 = 3 + i$$

4. $\overline{\overline{(1-i)}(2-i)} = \overline{(1+i)(2-i)} = \overline{2-i+2i-i^2} = \overline{2+i+1} = \overline{3+i} = 3-i$
5. $\overline{(2-i)}(1-i) = (2+i)(1-i) = 2-2i+i-i^2 = 2-i+1 = 3-i$
6. $(1-i)\overline{(2-i)} = (1-i)(2+i) = 2+i-2i-i^2 = 2-i+1 = 3-i$

Homework 1.2.1.2 Let $\alpha, \beta \in \mathbb{C}$.

1. ALWAYS/SOMETIMES/NEVER: $\alpha\beta = \beta\alpha$.
2. ALWAYS/SOMETIMES/NEVER: $\overline{\alpha}\beta = \overline{\beta}\alpha$.

Hint. Let $\alpha = \alpha_r + \alpha_c i$ and $\beta = \beta_r + \beta_c i$, where $\alpha_r, \alpha_c, \beta_r, \beta_c \in \mathbb{R}$.

Answer.

1. ALWAYS: $\alpha\beta = \beta\alpha$.
2. SOMETIMES: $\overline{\alpha}\beta = \overline{\beta}\alpha$.

Solution.

1. ALWAYS: $\alpha\beta = \beta\alpha$.

Proof:

$$\begin{aligned}
 \alpha\beta &= <\text{substitute}> \\
 (\alpha_r + \alpha_c i)(\beta_r + \beta_c i) &= <\text{multiply out}> \\
 \alpha_r\beta_r + \alpha_r\beta_c i + \alpha_c\beta_r i - \alpha_c\beta_c &= <\text{commutativity of real multiplication}> \\
 \beta_r\alpha_r + \beta_r\alpha_c i + \beta_c\alpha_r i - \beta_c\alpha_c &= <\text{factor}> \\
 (\beta_r + \beta_c i)(\alpha_r + \alpha_c i) &= <\text{substitute}> \\
 \beta\alpha.
 \end{aligned}$$

2. SOMETIMES: $\overline{\alpha}\beta = \overline{\beta}\alpha$.

An example where it is true: $\alpha = \beta = 0$.

An example where it is false: $\alpha = 1$ and $\beta = i$. Then $\overline{\alpha}\beta = 1 \times i = i$ and $\overline{\beta}\alpha = -i \times 1 = -i$.

Homework 1.2.1.3 Let $\alpha, \beta \in \mathbb{C}$.

ALWAYS/SOMETIMES/NEVER: $\overline{\alpha\beta} = \overline{\alpha}\beta$.

Hint. Let $\alpha = \alpha_r + \alpha_c i$ and $\beta = \beta_r + \beta_c i$, where $\alpha_r, \alpha_c, \beta_r, \beta_c \in \mathbb{R}$.

Answer. ALWAYS

Now prove it!

Solution 1.

$$\begin{aligned}
 & \overline{\alpha\beta} \\
 &= <\alpha = \alpha_r + \alpha_c i; \beta = \beta_r + \beta_c i> \\
 & \overline{(\alpha_r + \alpha_c i)(\beta_r + \beta_c i)} \\
 &= <\text{conjugate } \alpha> \\
 & \overline{(\alpha_r - \alpha_c i)(\beta_r + \beta_c i)} \\
 &= <\text{multiply out}> \\
 & \overline{(\alpha_r \beta_r - \alpha_c \beta_r i + \alpha_r \beta_c i + \alpha_c \beta_c)} \\
 &= <\text{conjugate}> \\
 & \alpha_r \beta_r + \alpha_c \beta_r i - \alpha_r \beta_c i + \alpha_c \beta_c \\
 &= <\text{rearrange}> \\
 & \beta_r \alpha_r + \beta_r \alpha_c i - \beta_c \alpha_r i + \beta_c \alpha_c \\
 &= <\text{factor}> \\
 & (\beta_r - \beta_c i)(\alpha_r + \alpha_c i) \\
 &= <\text{definition of conjugation}> \\
 & \overline{(\beta_r + \beta_c i)}(\alpha_r + \alpha_c i) \\
 &= <\alpha = \alpha_r + \alpha_c i; \beta = \beta_r + \beta_c i> \\
 & \overline{\beta\alpha}
 \end{aligned}$$

Solution 2. Proofs in mathematical textbooks seem to always be wonderfully smooth arguments that lead from the left-hand side of an equivalence to the right-hand side. In practice, you may want to start on the left-hand side, and apply a few rules:

$$\begin{aligned}
 & \overline{\alpha\beta} \\
 &= <\alpha = \alpha_r + \alpha_c i; \beta = \beta_r + \beta_c i> \\
 & \overline{(\alpha_r + \alpha_c i)(\beta_r + \beta_c i)} \\
 &= <\text{conjugate } \alpha> \\
 & \overline{(\alpha_r - \alpha_c i)(\beta_r + \beta_c i)} \\
 &= <\text{multiply out}> \\
 & \overline{(\alpha_r \beta_r - \alpha_c \beta_r i + \alpha_r \beta_c i + \alpha_c \beta_c)} \\
 &= <\text{conjugate}> \\
 & \alpha_r \beta_r + \alpha_c \beta_r i - \alpha_r \beta_c i + \alpha_c \beta_c
 \end{aligned}$$

and then move on to the right-hand side, applying a few rules:

$$\begin{aligned}
 & \overline{\beta\alpha} \\
 &= <\alpha = \alpha_r + \alpha_c i; \beta = \beta_r + \beta_c i> \\
 & \overline{(\beta_r + \beta_c i)(\alpha_r + \alpha_c i)} \\
 &= <\text{conjugate } \beta> \\
 & (\beta_r - \beta_c i)(\alpha_r + \alpha_c i) \\
 &= <\text{multiply out}> \\
 & \beta_r \alpha_r + \beta_r \alpha_c i - \beta_c \alpha_r i + \beta_c \alpha_c.
 \end{aligned}$$

At that point, you recognize that

$$\alpha_r \beta_r + \alpha_c \beta_r i - \alpha_r \beta_c i + \alpha_c \beta_c = \beta_r \alpha_r + \beta_r \alpha_c i - \beta_c \alpha_r i + \beta_c \alpha_c$$

since the second is a rearrangement of the terms of the first. Optionally, you then go back and presents these insights as a smooth argument that leads from the expression on the left-hand side to the one on the right-hand side:

$$\begin{aligned}
 & \overline{\alpha\beta} \\
 &= <\alpha = \alpha_r + \alpha_ci; \beta = \beta_r + \beta_ci> \\
 & \overline{(\alpha_r + \alpha_ci)(\beta_r + \beta_ci)} \\
 &= <\text{conjugate } \alpha> \\
 & \overline{(\alpha_r - \alpha_ci)(\beta_r + \beta_ci)} \\
 &= <\text{multiply out}> \\
 & \overline{(\alpha_r\beta_r - \alpha_c\beta_r i + \alpha_r\beta_ci + \alpha_c\beta_c)} \\
 &= <\text{conjugate}> \\
 & \alpha_r\beta_r + \alpha_c\beta_r i - \alpha_r\beta_ci + \alpha_c\beta_c \\
 &= <\text{rearrange}> \\
 & \beta_r\alpha_r + \beta_r\alpha_ci - \beta_c\alpha_r i + \beta_c\alpha_c \\
 &= <\text{factor}> \\
 & (\beta_r - \beta_ci)(\alpha_r + \alpha_ci) \\
 &= <\text{definition of conjugation}> \\
 & \overline{(\beta_r + \beta_ci)(\alpha_r + \alpha_ci)} \\
 &= <\alpha = \alpha_r + \alpha_ci; \beta = \beta_r + \beta_ci> \\
 & \overline{\beta}\alpha.
 \end{aligned}$$

Solution 3. Yet another way of presenting the proof uses an "equivalence style proof." The idea is to start with the equivalence you wish to prove correct:

$$\overline{\alpha\beta} = \overline{\beta}\alpha$$

and through a sequence of equivalent statements argue that this evaluates to TRUE:

$$\begin{aligned}
 & \overline{\alpha\beta} = \overline{\beta}\alpha \\
 & \Leftrightarrow <\alpha = \alpha_r + \alpha_ci; \beta = \beta_r + \beta_ci> \\
 & \overline{(\alpha_r + \alpha_ci)(\beta_r + \beta_ci)} = \overline{(\beta_r + \beta_ci)(\alpha_r + \alpha_ci)} \\
 & \Leftrightarrow <\text{conjugate } \times 2> \\
 & \overline{(\alpha_r - \alpha_ci)(\beta_r + \beta_ci)} = (\beta_r - \beta_ci)(\alpha_r + \alpha_ci) \\
 & \Leftrightarrow <\text{multiply out } \times 2> \\
 & \alpha_r\beta_r + \alpha_r\beta_ci - \alpha_c\beta_r i + \alpha_c\beta_c = \beta_r\alpha_r + \beta_r\alpha_ci - \beta_c\alpha_r i + \beta_c\alpha_c \\
 & \Leftrightarrow <\text{conjugate}> \\
 & \alpha_r\beta_r - \alpha_r\beta_ci + \alpha_c\beta_r i + \alpha_c\beta_c = \beta_r\alpha_r + \beta_r\alpha_ci - \beta_c\alpha_r i + \beta_c\alpha_c \\
 & \Leftrightarrow <\text{subtract equivalent terms from left-hand side and right-hand side}> \\
 & 0 = 0 \\
 & \Leftrightarrow <\text{algebra}> \\
 & \text{TRUE.}
 \end{aligned}$$

By transitivity of equivalence, we conclude that $\overline{\alpha\beta} = \overline{\beta}\alpha$ is TRUE.

Homework 1.2.1.4 Let $\alpha \in \mathbb{C}$.

ALWAYS/SOMETIMES/NEVER: $\bar{\alpha}\alpha \in \mathbb{R}$

Answer. ALWAYS.

Now prove it!

Solution. Let $\alpha = \alpha_r + \alpha_c i$. Then

$$\begin{aligned} \bar{\alpha}\alpha &= <\text{ instantiate}> \\ (\alpha_r + \alpha_c i)(\alpha_r + \alpha_c i) &= <\text{conjugate}> \\ (\alpha_r - \alpha_c i)(\alpha_r + \alpha_c i) &= <\text{multiply out}> \\ \alpha_r^2 + \alpha_c^2, & \end{aligned}$$

which is a real number.

Homework 1.2.1.5 Prove that the absolute value function is homogeneous: $|\alpha\beta| = |\alpha||\beta|$ for all $\alpha, \beta \in \mathbb{C}$.

Solution.

$$\begin{aligned} |\alpha\beta| &= |\alpha||\beta| \\ \Leftrightarrow & <\text{squaring both sides simplifies}> \\ |\alpha\beta|^2 &= |\alpha|^2|\beta|^2 \\ \Leftrightarrow & <\text{ instantiate}> \\ |(\alpha_r + \alpha_c i)(\beta_r + \beta_c i)|^2 &= |\alpha_r + \alpha_c i|^2|\beta_r + \beta_c i|^2 \\ \Leftrightarrow & <\text{algebra}> \\ |(\alpha_r\beta_r - \alpha_c\beta_c) + (\alpha_r\beta_c + \alpha_c\beta_r)i|^2 &= (\alpha_r^2 + \alpha_c^2)(\beta_r^2 + \beta_c^2) \\ \Leftrightarrow & <\text{algebra}> \\ (\alpha_r\beta_r - \alpha_c\beta_c)^2 + (\alpha_r\beta_c + \alpha_c\beta_r)^2 &= (\alpha_r^2 + \alpha_c^2)(\beta_r^2 + \beta_c^2) \\ \Leftrightarrow & <\text{algebra}> \\ \alpha_r^2\beta_r^2 - 2\alpha_r\alpha_c\beta_r\beta_c + \alpha_c^2\beta_c^2 + \alpha_r^2\beta_c^2 + 2\alpha_r\alpha_c\beta_r\beta_c + \alpha_c^2\beta_r^2 &= \alpha_r^2\beta_r^2 + \alpha_r^2\beta_c^2 + \alpha_c^2\beta_r^2 + \alpha_c^2\beta_c^2 \\ &= \alpha_r^2\beta_r^2 + \alpha_r^2\beta_c^2 + \alpha_c^2\beta_r^2 + \alpha_c^2\beta_c^2 \\ \Leftrightarrow & <\text{subtract equivalent terms from both sides}> \\ 0 &= 0 \\ \Leftrightarrow & <\text{algebra}> \\ T & \end{aligned}$$

Homework 1.2.1.6 Let $\alpha \in \mathbb{C}$.

ALWAYS/SOMETIMES/NEVER: $|\bar{\alpha}| = |\alpha|$.

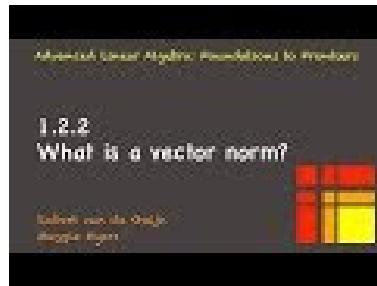
Answer. ALWAYS

Now prove it!

Solution. Let $\alpha = \alpha_r + \alpha_c i$.

$$\begin{aligned}
 |\bar{\alpha}| &= <\text{ instantiate}> \\
 \overline{|\alpha_r + \alpha_c i|} &= <\text{conjugate}> \\
 |\alpha_r - \alpha_c i| &= <\text{definition of } |\cdot|> \\
 \sqrt{\alpha_r^2 + \alpha_c^2} &= <\text{definition of } |\cdot|> \\
 |\alpha_r + \alpha_c i| &= <\text{instantiate}> \\
 |\alpha|
 \end{aligned}$$

1.2.2 What is a vector norm?



YouTube: <https://www.youtube.com/watch?v=CTrUVfLGcNM>

A vector norm extends the notion of an absolute value to vectors. It allows us to measure the magnitude (or length) of a vector. In different situations, a different measure may be more appropriate.

Definition 1.2.2.1 Vector norm. Let $\nu : \mathbb{C}^m \rightarrow \mathbb{R}$. Then ν is a (vector) norm if for all $x, y \in \mathbb{C}^m$ and all $\alpha \in \mathbb{C}$

- $x \neq 0 \Rightarrow \nu(x) > 0$ (ν is positive definite),
- $\nu(\alpha x) = |\alpha| \nu(x)$ (ν is homogeneous), and
- $\nu(x + y) \leq \nu(x) + \nu(y)$ (ν obeys the triangle inequality).

◊

Homework 1.2.2.1 TRUE/FALSE: If $\nu : \mathbb{C}^m \rightarrow \mathbb{R}$ is a norm, then $\nu(0) = 0$.

Hint. From context, you should be able to tell which of these 0's denotes the zero vector of a given size and which is the scalar 0.

$0x = 0$ (multiplying any vector x by the scalar 0 results in a vector of zeroes).

Answer. TRUE.

Now prove it.

Solution. Let $x \in \mathbb{C}^m$ and, just for clarity this first time, $\vec{0}$ be the zero vector of size m so that 0 is the scalar zero. Then

$$\begin{aligned}\nu(\vec{0}) &= < 0 \cdot x = \vec{0} > \\ \nu(0 \cdot x) &= < \nu(\cdots) \text{ is homogeneous} > \\ 0\nu(x) &= < \text{algebra} > \\ 0 &\end{aligned}$$

Remark 1.2.2.2 We typically use $\|\cdot\|$ instead of $\nu(\cdot)$ for a function that is a norm.

1.2.3 The vector 2-norm (Euclidean length)



YouTube: <https://www.youtube.com/watch?v=bxDpUZEqBs>

The length of a vector is most commonly measured by the "square root of the sum of the squares of the elements," also known as the Euclidean norm. It is called the 2-norm because it is a member of a class of norms known as p -norms, discussed in the next unit.

Definition 1.2.3.1 Vector 2-norm. The vector 2-norm $\|\cdot\|_2 : \mathbb{C}^m \rightarrow \mathbb{R}$ is defined for $x \in \mathbb{C}^m$ by

$$\|x\|_2 = \sqrt{|\chi_0|^2 + \cdots + |\chi_{m-1}|^2} = \sqrt{\sum_{i=0}^{m-1} |\chi_i|^2}.$$

Equivalently, it can be defined by

$$\|x\|_2 = \sqrt{x^H x}$$

or

$$\|x\|_2 = \sqrt{\bar{\chi}_0 \chi_0 + \cdots + \bar{\chi}_{m-1} \chi_{m-1}} = \sqrt{\sum_{i=0}^{m-1} \bar{\chi}_i \chi_i}.$$

◇

Remark 1.2.3.2 The notation x^H requires a bit of explanation. If

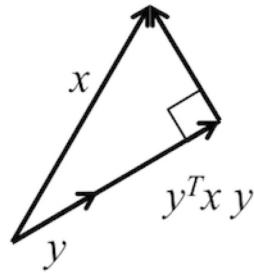
$$x = \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_m \end{pmatrix}$$

then the row vector

$$x^H = \begin{pmatrix} \bar{x}_0 & \cdots & \bar{x}_m \end{pmatrix}$$

is the Hermitian transpose of x (or, equivalently, the Hermitian transpose of the vector x that is viewed as a matrix) and $x^H y$ can be thought of as the dot product of x and y or, equivalently, as the matrix-vector multiplication of the matrix x^H times the vector y .

To prove that the 2-norm is a norm (just calling it a norm doesn't mean it is, after all), we need a result known as the Cauchy-Schwarz inequality. This inequality relates the magnitude of the dot product of two vectors to the product of their 2-norms: if $x, y \in \mathbb{R}^m$, then $|x^T y| \leq \|x\|_2 \|y\|_2$. To motivate this result before we rigorously prove it, recall from your undergraduate studies that the component of x in the direction of a vector y of unit length is given by $(y^T x)y$, as illustrated by



The length of the component of x in the direction of y then equals

$$\begin{aligned} & \|(y^T x)y\|_2 \\ &= <\text{definition}> \\ &= \sqrt{(y^T x)^T y^T (y^T x)y} \\ &= <z\alpha = \alpha z> \\ &= \sqrt{(x^T y)^2 y^T y} \\ &= <y \text{ has unit length}> \\ |y^T x| &= <\text{definition}> \\ &= |x^T y|. \end{aligned}$$

Thus $|x^T y| \leq \|x\|_2$ (since a component should be shorter than the whole). If y is not of unit length (but a nonzero vector), then $|x^T \frac{y}{\|y\|_2}| \leq \|x\|_2$ or, equivalently, $|x^T y| \leq \|x\|_2 \|y\|_2$.

We now state this result as a theorem, generalized to complex valued vectors:

Theorem 1.2.3.3 Cauchy-Schwarz inequality. *Let $x, y \in \mathbb{C}^m$. Then $|x^H y| \leq \|x\|_2 \|y\|_2$.*

Proof. Assume that $x \neq 0$ and $y \neq 0$, since otherwise the inequality is trivially true. We can then choose $\hat{x} = x/\|x\|_2$ and $\hat{y} = y/\|y\|_2$. This leaves us to prove that $|\hat{x}^H \hat{y}| \leq 1$ since $\|\hat{x}\|_2 = \|\hat{y}\|_2 = 1$.

Pick

$$\alpha = \begin{cases} 1 & \text{if } x^H y = 0 \\ \hat{y}^H \hat{x} / |\hat{x}^H \hat{y}| & \text{otherwise.} \end{cases}$$

so that $|\alpha| = 1$ and $\alpha \hat{x}^H \hat{y}$ is real and nonnegative. Note that since it is real we also know

that

$$\begin{aligned} \alpha \hat{x}^H \hat{y} \\ = & \quad \langle \beta = \bar{\beta} \text{ if } \beta \text{ is real} \rangle \\ \frac{\alpha \hat{x}^H \hat{y}}{\alpha \hat{x}^H \hat{y}} &= \langle \text{property of complex conjugation} \rangle \\ &= \bar{\alpha} \hat{y}^H \hat{x} \end{aligned}$$

Now,

$$\begin{aligned} 0 &\leq \langle \|\cdot\|_2 \text{ is nonnegative definite} \rangle \\ \|\hat{x} - \alpha \hat{y}\|_2^2 &= \langle \|z\|_2^2 = z^H z \rangle \\ (\hat{x} - \alpha \hat{y})^H (\hat{x} - \alpha \hat{y}) &= \langle \text{multiplying out} \rangle \\ \hat{x}^H \hat{x} - \bar{\alpha} \hat{y}^H \hat{x} - \alpha \hat{x}^H \hat{y} + \bar{\alpha} \alpha \hat{y}^H \hat{y} &= \langle \text{above assumptions and observations} \rangle \\ 1 - 2\alpha \hat{x}^H \hat{y} + |\alpha|^2 &= \langle \alpha \hat{x}^H \hat{y} = |\hat{x}^H \hat{y}|; |\alpha| = 1 \rangle \\ 2 - 2|\hat{x}^H \hat{y}|. & \end{aligned}$$

Thus $|\hat{x}^H \hat{y}| \leq 1$ and therefore $|x^H y| \leq \|x\|_2 \|y\|_2$. ■

The proof of [Theorem 1.2.3.3](#) does not employ any of the intuition we used to motivate it in the real valued case just before its statement. We leave it to the reader to prove the Cauchy-Schartz inequality for real-valued vectors by modifying (simplifying) the proof of [Theorem 1.2.3.3](#).

Ponder This 1.2.3.1 Let $x, y \in \mathbb{R}^m$. Prove that $|x^T y| \leq \|x\|_2 \|y\|_2$ by specializing the proof of [Theorem 1.2.3.3](#).

The following theorem states that the 2-norm is indeed a norm:

Theorem 1.2.3.4 *The vector 2-norm is a norm.*

We leave its proof as an exercise.

Homework 1.2.3.2 Prove [Theorem 1.2.3.4](#).

Solution. To prove this, we merely check whether the three conditions are met:

Let $x, y \in \mathbb{C}^m$ and $\alpha \in \mathbb{C}$ be arbitrarily chosen. Then

- $x \neq 0 \Rightarrow \|x\|_2 > 0$ ($\|\cdot\|_2$ is positive definite):

Notice that $x \neq 0$ means that at least one of its components is nonzero. Let's assume that $\chi_j \neq 0$. Then

$$\|x\|_2 = \sqrt{|\chi_0|^2 + \cdots + |\chi_{m-1}|^2} \geq \sqrt{|\chi_j|^2} = |\chi_j| > 0.$$

- $\|\alpha x\|_2 = |\alpha| \|x\|_2$ ($\|\cdot\|_2$ is homogeneous):

$$\begin{aligned}
 \|\alpha x\|_2 &= \sqrt{|\alpha \chi_0|^2 + \dots + |\alpha \chi_{m-1}|^2} && \text{scaling a vector scales its components; definition} \\
 &= \sqrt{|\alpha|^2 |\chi_0|^2 + \dots + |\alpha|^2 |\chi_{m-1}|^2} && \text{algebra} \\
 &= \sqrt{|\alpha|^2 (|\chi_0|^2 + \dots + |\chi_{m-1}|^2)} && \text{algebra} \\
 &= |\alpha| \sqrt{|\chi_0|^2 + \dots + |\chi_{m-1}|^2} && \text{algebra} \\
 &= |\alpha| \|x\|_2. && \text{definition}
 \end{aligned}$$

- $\|x + y\|_2 \leq \|x\|_2 + \|y\|_2$ ($\|\cdot\|_2$ obeys the triangle inequality):

$$\begin{aligned}
 \|x + y\|_2^2 &= \langle \|z\|_2^2 = z^H z \rangle \\
 (x + y)^H (x + y) &= \langle \text{distribute} \rangle \\
 x^H x + y^H x + x^H y + y^H y &= \langle \bar{\beta} + \beta = 2\text{Real}(\beta) \rangle \\
 x^H x + 2\text{Real}(x^H y) + y^H y &\leq \langle \text{algebra} \rangle \\
 x^H x + 2|\text{Real}(x^H y)| + y^H y &\leq \langle \text{algebra} \rangle \\
 x^H x + 2|x^H y| + y^H y &\leq \langle \text{algebra; Cauchy-Schwarz} \rangle \\
 \|x\|_2^2 + 2\|x\|_2\|y\|_2 + \|y\|_2^2 &= \langle \text{algebra} \rangle \\
 (\|x\|_2 + \|y\|_2)^2.
 \end{aligned}$$

Taking the square root (an increasing function that hence maintains the inequality) of both sides yields the desired result.

Throughout this course, we will reason about subvectors and submatrices. Let's get some practice:

Homework 1.2.3.3 Partition $x \in \mathbb{C}^m$ into subvectors:

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{M-1} \end{pmatrix}.$$

ALWAYS/SOMETIMES/NEVER: $\|x_i\|_2 \leq \|x\|_2$.

Answer. ALWAYS

Now prove it!

Solution.

$$\begin{aligned}
 \|x\|_2^2 &= \left\| \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{M-1} \end{pmatrix} \right\|_2^2 \\
 &= \left\| \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{M-1} \end{pmatrix} \right\|_2^H \left\| \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{M-1} \end{pmatrix} \right\|_2 \\
 &= \left\langle \text{dot product of partitioned vectors} \right\rangle \\
 x_0^H x_0 + x_1^H x_1 + \cdots + x_{M-1}^H x_{M-1} &= \left\langle \text{equivalent definition} \right\rangle \\
 \|x_0\|_2^2 + \|x_1\|_2^2 + \cdots + \|x_{M-1}\|_2^2 &\geq \left\langle \text{algebra} \right\rangle \\
 \|x_i\|_2^2
 \end{aligned}$$

so that $\|x_i\|_2^2 \leq \|x\|_2^2$. Taking the square root of both sides shows that $\|x_i\|_2 \leq \|x\|_2$.

1.2.4 The vector p -norms



YouTube: <https://www.youtube.com/watch?v=WGBMnmgJek8>

A vector norm is a measure of the magnitude of a vector. The Euclidean norm (length) is merely the best known such measure. There are others. A simple alternative is the 1-norm.

Definition 1.2.4.1 Vector 1-norm. The vector 1-norm, $\|\cdot\|_1 : \mathbb{C}^m \rightarrow \mathbb{R}$, is defined for $x \in \mathbb{C}^m$ by

$$\|x\|_1 = |\chi_0| + |\chi_1| + \cdots + |\chi_{m-1}| = \sum_{i=0}^{m-1} |\chi_i|.$$

◇

Homework 1.2.4.1 Prove that the vector 1-norm is a norm.

Solution. We show that the three conditions are met:

Let $x, y \in \mathbb{C}^m$ and $\alpha \in \mathbb{C}$ be arbitrarily chosen. Then

- $x \neq 0 \Rightarrow \|x\|_1 > 0$ ($\|\cdot\|_1$ is positive definite):

Notice that $x \neq 0$ means that at least one of its components is nonzero. Let's assume that $\chi_j \neq 0$. Then

$$\|x\|_1 = |\chi_0| + \dots + |\chi_{m-1}| \geq |\chi_j| > 0.$$

- $\|\alpha x\|_1 = |\alpha| \|x\|_1$ ($\|\cdot\|_1$ is homogeneous):

$$\begin{aligned} \|\alpha x\|_1 &= < \text{scaling a vector-scales-its-components; definition} > \\ |\alpha \chi_0| + \dots + |\alpha \chi_{m-1}| &= < \text{algebra} > \\ |\alpha| |\chi_0| + \dots + |\alpha| |\chi_{m-1}| &= < \text{algebra} > \\ |\alpha| (|\chi_0| + \dots + |\chi_{m-1}|) &= < \text{definition} > \\ |\alpha| \|x\|_1. & \end{aligned}$$

- $\|x + y\|_1 \leq \|x\|_1 + \|y\|_1$ ($\|\cdot\|_1$ obeys the triangle inequality):

$$\begin{aligned} \|x + y\|_1 &= < \text{vector addition; definition of 1-norm} > \\ |\chi_0 + \psi_0| + |\chi_1 + \psi_1| + \dots + |\chi_{m-1} + \psi_{m-1}| &\leq < \text{algebra} > \\ |\chi_0| + |\psi_0| + |\chi_1| + |\psi_1| + \dots + |\chi_{m-1}| + |\psi_{m-1}| &= < \text{commutivity} > \\ |\chi_0| + |\chi_1| + \dots + |\chi_{m-1}| + |\psi_0| + |\psi_1| + \dots + |\psi_{m-1}| &= < \text{associativity; definition} > \\ \|x\|_1 + \|y\|_1. & \end{aligned}$$

The vector 1-norm is sometimes referred to as the "taxi-cab norm". It is the distance that a taxi travels, from one point on a street to another such point, along the streets of a city that has square city blocks.

Another alternative is the infinity norm.

Definition 1.2.4.2 Vector ∞ -norm. The vector ∞ -norm, $\|\cdot\|_\infty : \mathbb{C}^m \rightarrow \mathbb{R}$, is defined for $x \in \mathbb{C}^m$ by

$$\|x\|_\infty = \max(|\chi_0|, \dots, |\chi_{m-1}|) = \max_{i=0}^{m-1} |\chi_i|.$$

◊

The infinity norm simply measures how large the vector is by the magnitude of its largest entry.

Homework 1.2.4.2 Prove that the vector ∞ -norm is a norm.

Solution. We show that the three conditions are met:

Let $x, y \in \mathbb{C}^m$ and $\alpha \in \mathbb{C}$ be arbitrarily chosen. Then

- $x \neq 0 \Rightarrow \|x\|_\infty > 0$ ($\|\cdot\|_\infty$ is positive definite):

Notice that $x \neq 0$ means that at least one of its components is nonzero. Let's assume that $\chi_j \neq 0$. Then

$$\|x\|_\infty = \max_{i=0}^{m-1} |\chi_i| \geq |\chi_j| > 0.$$

- $\|\alpha x\|_\infty = |\alpha| \|x\|_\infty$ ($\|\cdot\|_\infty$ is homogeneous):

$$\begin{aligned} \|\alpha x\|_\infty &= \max_{i=0}^{m-1} |\alpha \chi_i| \\ &= \max_{i=0}^{m-1} |\alpha| |\chi_i| \\ &= |\alpha| \max_{i=0}^{m-1} |\chi_i| \\ &= |\alpha| \|x\|_\infty. \end{aligned}$$

- $\|x + y\|_\infty \leq \|x\|_\infty + \|y\|_\infty$ ($\|\cdot\|_\infty$ obeys the triangle inequality):

$$\begin{aligned} \|x + y\|_\infty &= \max_{i=0}^{m-1} |\chi_i + \psi_i| \\ &\leq \max_{i=0}^{m-1} (|\chi_i| + |\psi_i|) \\ &\leq \max_{i=0}^{m-1} |\chi_i| + \max_{i=0}^{m-1} |\psi_i| \\ &= \|x\|_\infty + \|y\|_\infty. \end{aligned}$$

In this course, we will primarily use the vector 1-norm, 2-norm, and ∞ -norms. For completeness, we briefly discuss their generalization: the vector p -norm.

Definition 1.2.4.3 Vector p -norm. Given $p \geq 1$, the vector p -norm $\|\cdot\|_p : \mathbb{C}^m \rightarrow \mathbb{R}$ is defined for $x \in \mathbb{C}^m$ by

$$\|x\|_p = \sqrt[p]{|\chi_0|^p + \cdots + |\chi_{m-1}|^p} = \left(\sum_{i=0}^{m-1} |\chi_i|^p \right)^{1/p}.$$

◊

Theorem 1.2.4.4 *The vector p -norm is a norm.*

The proof of this result is very similar to the proof of the fact that the 2-norm is a norm. It depends on Hölder's inequality, which is a generalization of the Cauchy-Schwarz inequality:

Theorem 1.2.4.5 Hölder's inequality. *Let $1 \leq p, q \leq \infty$ with $\frac{1}{p} + \frac{1}{q} = 1$. If $x, y \in \mathbb{C}^m$ then $|x^H y| \leq \|x\|_p \|y\|_q$.*

We skip the proof of Hölder's inequality and [Theorem 1.2.4.4](#). You can easily find proofs for these results, should you be interested.

Remark 1.2.4.6 The vector 1-norm and 2-norm are obviously special cases of the vector p -norm. It can be easily shown that the vector ∞ -norm is also related:

$$\lim_{p \rightarrow \infty} \|x\|_p = \|x\|_\infty.$$

Ponder This 1.2.4.3 Consider Homework 1.2.3.3. Try to elegantly formulate this question in the most general way you can think of. How do you prove the result?

Ponder This 1.2.4.4 Consider the vector norm $\|\cdot\| : \mathbb{C}^m \rightarrow \mathbb{R}$, the matrix $A \in \mathbb{C}^{m \times n}$ and the function $f : \mathbb{C}^n \rightarrow \mathbb{R}$ defined by $f(x) = \|Ax\|$. For what matrices A is the function f a norm?

1.2.5 Unit ball



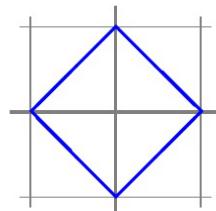
YouTube: <https://www.youtube.com/watch?v=aJgrpp7uscw>

In 3-dimensional space, the notion of the unit ball is intuitive: the set of all points that are a (Euclidean) distance of one from the origin. Vectors have no position and can have more than three components. Still the unit ball for the 2-norm is a straight forward extension to the set of all vectors with length (2-norm) one. More generally, the unit ball for any norm can be defined:

Definition 1.2.5.1 Unit ball. Given norm $\|\cdot\| : \mathbb{C}^m \rightarrow \mathbb{R}$, the unit ball with respect to $\|\cdot\|$ is the set $\{x \mid \|x\| = 1\}$ (the set of all vectors with norm equal to one). We will use $\|x\| = 1$ as shorthand for $\{x \mid \|x\| = 1\}$. \diamond

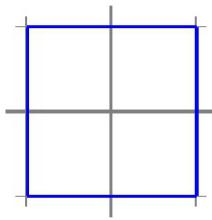
Homework 1.2.5.1 Although vectors have no position, it is convenient to visualize a vector $x \in \mathbb{R}^2$ by the point in the plane to which it extends when rooted at the origin. For example, the vector $x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ can be so visualized with the point $(2, 1)$. With this in mind, match the pictures on the right corresponding to the sets on the left:

(a) $\|x\|_2 = 1$. (1)



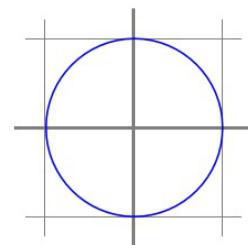
(b) $\|x\|_1 = 1$.

(2)



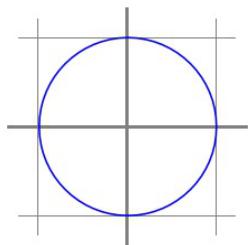
(c) $\|x\|_\infty = 1$.

(3)

**Solution.**

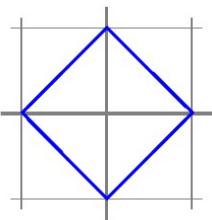
(a) $\|x\|_2 = 1$.

(3)



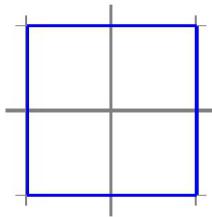
(b) $\|x\|_1 = 1$.

(1)



(c) $\|x\|_\infty = 1$.

(2)





YouTube: <https://www.youtube.com/watch?v=0v77sE90P58>

1.2.6 Equivalence of vector norms



YouTube: <https://www.youtube.com/watch?v=qjZyKHvL13E>

Homework 1.2.6.1 Fill out the following table:

x	$\ x\ _1$	$\ x\ _\infty$	$\ x\ _2$
$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$			
$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$			
$\begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix}$			

Solution.

x	$\ x\ _1$	$\ x\ _\infty$	$\ x\ _2$
$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$	1	1	1
$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	3	1	$\sqrt{3}$
$\begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix}$	4	2	$\sqrt{1^2 + (-2)^2 + (-1)^2} = \sqrt{6}$

In this course, norms are going to be used to reason that vectors are "small" or "large". It would be unfortunate if a vector were small in one norm yet large in another norm. Fortunately, the following theorem excludes this possibility:

Theorem 1.2.6.1 Equivalence of vector norms. *Let $\|\cdot\| : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\|\cdot\||| : \mathbb{C}^m \rightarrow \mathbb{R}$ both be vector norms. Then there exist positive scalars σ and τ such that for all $x \in \mathbb{C}^m$*

$$\sigma\|x\| \leq \|\|x\||| \leq \tau\|x\|.$$

Proof. The proof depends on a result from real analysis (sometimes called "advanced calculus") that states that $\sup_{x \in S} f(x)$ is attained for some vector $x \in S$ as long as f is continuous and S is a compact (closed and bounded) set. For any norm $\|\cdot\|$, the unit ball $\|x\| = 1$ is a compact set. When a supremum is an element in S , it is called the maximum instead and $\sup_{x \in S} f(x)$ can be restated as $\max_{x \in S} f(x)$.

Those who have not studied real analysis (which is not a prerequisite for this course) have to take this on faith. It is a result that we will use a few times in our discussion.

We prove that there exists a τ such that for all $x \in \mathbb{C}^m$

$$\|\|x\||| \leq \tau\|x\|,$$

leaving the rest of the proof as an exercise.

Let $x \in \mathbb{C}^m$ be an arbitrary vector. W.l.o.g. assume that $x \neq 0$. Then

$$\begin{aligned} \|\|x\||| &= <\text{algebra}> \\ \frac{\|\|x\|||}{\|x\|} \|x\| &\leq <\text{algebra}> \\ \left(\sup_{z \neq 0} \frac{\|\|z\|||}{\|z\|}\right) \|x\| &= <\text{change of variables: } y = z/\|z\|> \\ \left(\sup_{\|y\|=1} \|\|y\|||\right) \|x\| &= <\text{the set } \|y\|=1 \text{ is compact}> \\ \left(\max_{\|y\|=1} \|\|y\|||\right) \|x\| \end{aligned}$$

The desired τ can now be chosen to equal $\max_{\|y\|=1} \|\|y\|||$. ■



YouTube: <https://www.youtube.com/watch?v=I1W6ErdEyoc>

Homework 1.2.6.2 Complete the proof of Theorem 1.2.6.1.

Solution. We need to prove that

$$\sigma\|x\| \leq |||x|||.$$

From the first part of the proof of Theorem 1.2.6.1, we know that there exists a $\rho > 0$ such that

$$\|x\| \leq \rho|||x|||$$

and hence

$$\frac{1}{\rho}\|x\| \leq |||x|||.$$

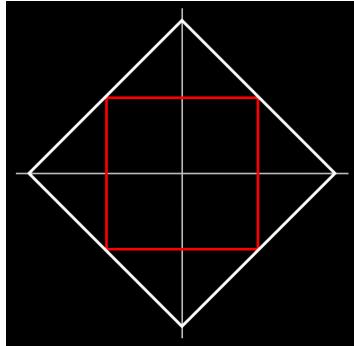
We conclude that

$$\sigma\|x\| \leq |||x|||$$

where $\sigma = 1/\rho$.

Example 1.2.6.2

- Let $x \in \mathbb{R}^2$. Use the picture

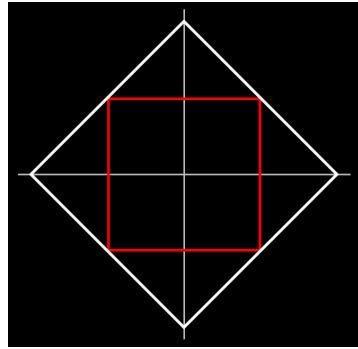


to determine the constant C such that $\|x\|_1 \leq C\|x\|_\infty$. Give a vector x for which $\|x\|_1 = C\|x\|_\infty$.

- For $x \in \mathbb{R}^2$ and the C you determined in the first part of this problem, prove that $\|x\|_1 \leq C\|x\|_\infty$.
- Let $x \in \mathbb{C}^m$. Extrapolate from the last part the constant C such that $\|x\|_1 \leq C\|x\|_\infty$ and then prove the inequality. Give a vector x for which $\|x\|_1 = C\|x\|_\infty$.

Solution.

- Consider the picture



- The red square represents all vectors such that $\|x\|_\infty = 1$ and the white square represents all vectors such that $\|x\|_1 = 2$.
- All points on or outside the red square represent vectors y such that $\|y\|_\infty \geq 1$. Hence if $\|y\|_1 = 2$ then $\|y\|_\infty \geq 1$.
- Now, pick any $z \neq 0$. Then $\|2z/\|z\|_1\|_1 = 2$. Hence

$$\|2z/\|z\|_1\|_\infty \geq 1$$

which can be rewritten as

$$\|z\|_1 \leq 2\|z\|_\infty.$$

Thus, $C = 2$ works.

- Now, from the picture it is clear that $x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ has the property that $\|x\|_1 = 2\|x\|_\infty$. Thus, the inequality is "tight."
- We now prove that $\|x\|_1 \leq 2\|x\|_\infty$ for $x \in \mathbb{R}^2$:

$$\begin{aligned}
 \|x\|_1 &= <\text{definition}> \\
 |\chi_0| + |\chi_1| &\leq <\text{algebra}> \\
 \max(|\chi_0|, |\chi_1|) + \max(|\chi_0|, |\chi_1|) &= <\text{algebra}> \\
 2 \max(|\chi_0|, |\chi_1|) &= <\text{definition}> \\
 2\|x\|_\infty.
 \end{aligned}$$

- From the last part we extrapolate that $\|x\|_1 \leq m\|x\|_\infty$.

$$\begin{aligned}
\|x\|_1 &= \sum_{i=0}^{m-1} |\chi_i| < \text{definition} > \\
&\leq \sum_{i=0}^{m-1} (\max_{j=0}^{m-1} |\chi_j|) < \text{algebra} > \\
&= m \max_{j=0}^{m-1} |\chi_j| < \text{algebra} > \\
&= m \|x\|_\infty. < \text{definition} >
\end{aligned}$$

Equality holds (i.e., $\|x\|_1 = m\|x\|_\infty$) for $x = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$.

Some will be able to go straight for the general result, while others will want to seek inspiration from the picture and/or the specialized case where $x \in \mathbb{R}^2$. \square

Homework 1.2.6.3 Let $x \in \mathbb{C}^m$. The following table organizes the various bounds:

	$\ x\ _1 \leq C_{1,2}\ x\ _2$	$\ x\ _1 \leq C_{1,\infty}\ x\ _\infty$
$\ x\ _2 \leq C_{2,1}\ x\ _1$		$\ x\ _2 \leq C_{2,\infty}\ x\ _\infty$
$\ x\ _\infty \leq C_{\infty,1}\ x\ _1$	$\ x\ _\infty \leq C_{\infty,2}\ x\ _2$	

For each, determine the constant $C_{x,y}$ and prove the inequality, including that it is a tight inequality.

Hint: look at the hint!

Hint. $\|x\|_1 \leq \sqrt{m}\|x\|_2$:

This is the hardest one to prove. Do it last and use the following hint:

Consider $y = \begin{pmatrix} \chi_0/|\chi_0| \\ \vdots \\ \chi_{m-1}/|\chi_{m-1}| \end{pmatrix}$ and employ the Cauchy-Schwarz inequality.

Solution 1 ($\|x\|_1 \leq C_{1,2}\|x\|_2$). $\|x\|_1 \leq \sqrt{m}\|x\|_2$:

Consider $y = \begin{pmatrix} \chi_0/|\chi_0| \\ \vdots \\ \chi_{m-1}/|\chi_{m-1}| \end{pmatrix}$. Then

$$|x^H y| = \left| \sum_{i=0}^{m-1} \overline{\chi_i} \chi_i / |\chi_i| \right| = \left| \sum_{i=0}^{m-1} |\chi_i|^2 / |\chi_i| \right| = \left| \sum_{i=0}^{m-1} |\chi_i| \right| = \|x\|_1.$$

We also notice that $\|y\|_2 = \sqrt{m}$.

From the Cauchy-Swartz inequality we know that

$$\|x\|_1 = |x^H y| \leq \|x\|_2 \|y\|_2 = \sqrt{m} \|x\|_2.$$

If we now choose

$$x = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

then $\|x\|_1 = m$ and $\|x\|_2 = \sqrt{m}$ so that $\|x\|_1 = \sqrt{m}\|x\|_2$.

Solution 2 ($\|x\|_1 \leq C_{1,\infty}\|x\|_\infty$). $\|x\|_1 \leq m\|x\|_\infty$:

See [Example 1.2.6.2](#).

Solution 3 ($\|x\|_2 \leq C_{2,1}\|x\|_1$). $\|x\|_2 \leq \|x\|_1$:

$$\begin{aligned} \|x\|_2^2 &= <\text{definition}> \\ \sum_{i=0}^{m-1} |\chi_i|^2 &\leq <\text{algebra}> \\ \left(\sum_{i=0}^{m-1} |\chi_i|\right)^2 &= <\text{definition}> \\ \|x\|_1^2. \end{aligned}$$

Taking the square root of both sides yields $\|x\|_2 \leq \|x\|_1$.

If we now choose

$$x = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

then $\|x\|_2 = \|x\|_1$.

Solution 4 ($\|x\|_2 \leq C_{2,\infty}\|x\|_\infty$). $\|x\|_2 \leq \sqrt{m}\|x\|_\infty$:

$$\begin{aligned} \|x\|_2^2 &= <\text{definition}> \\ \sum_{i=0}^{m-1} |\chi_i|^2 &\leq <\text{algebra}> \\ \sum_{i=0}^{m-1} \left(\max_{j=0}^{m-1} |\chi_j|\right)^2 &= <\text{definition}> \\ \sum_{i=0}^{m-1} \|x\|_\infty^2 &= <\text{algebra}> \\ m\|x\|_\infty^2. \end{aligned}$$

Taking the square root of both sides yields $\|x\|_2 \leq \sqrt{m}\|x\|_\infty$.

Consider

$$x = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

then $\|x\|_2 = \sqrt{m}$ and $\|x\|_\infty = 1$ so that $\|x\|_2 = \sqrt{m}\|x\|_\infty$.

Solution 5 ($\|x\|_\infty \leq C_{\infty,1}\|x\|_1$): $\|x\|_\infty \leq \|x\|_1$:

$$\begin{aligned} & \|x\|_\infty \\ &= < \text{definition} > \\ & \max_{i=0}^{m-1} |\chi_i| \\ &\leq < \text{algebra} > \\ & \sum_{i=0}^{m-1} |\chi_i| \\ &= < \text{definition} > \\ & \|x\|_1. \end{aligned}$$

Consider

$$x = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Then $\|x\|_\infty = 1 = \|x\|_1$.

Solution 6 ($\|x\|_\infty \leq C_{\infty,2}\|x\|_2$): $\|x\|_\infty \leq \|x\|_2$:

$$\begin{aligned} & \|x\|_\infty^2 \\ &= < \text{definition} > \\ & \left(\max_{i=0}^{m-1} |\chi_i| \right)^2 \\ &= < \text{algebra} > \\ & \max_{i=0}^{m-1} |\chi_i|^2 \\ &\leq < \text{algebra} > \\ & \sum_{i=0}^{m-1} |\chi_i|^2 \\ &= < \text{definition} > \\ & \|x\|_2^2. \end{aligned}$$

Taking the square root of both sides yields $\|x\|_\infty \leq \|x\|_2$.

Consider

$$x = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Then $\|x\|_\infty = 1 = \|x\|_2$.

Solution 7 (Table of constants).

	$\ x\ _1 \leq \sqrt{m}\ x\ _2$	$\ x\ _1 \leq m\ x\ _\infty$
$\ x\ _2 \leq \ x\ _1$		$\ x\ _2 \leq \sqrt{m}\ x\ _\infty$
$\ x\ _\infty \leq \ x\ _1$	$\ x\ _\infty \leq \ x\ _2$	

Remark 1.2.6.3 The bottom line is that, modulo a constant factor, if a vector is "small" in one norm, it is "small" in all other norms. If it is "large" in one norm, it is "large" in all other norms.

1.3 Matrix Norms

1.3.1 Of linear transformations and matrices



YouTube: <https://www.youtube.com/watch?v=x1kiZEbYh38>

We briefly review the relationship between linear transformations and matrices, which is key to understanding why linear algebra is all about matrices and vectors.

Definition 1.3.1.1 Linear transformations and matrices. Let $L : \mathbb{C}^n \rightarrow \mathbb{C}^m$. Then L is said to be a linear transformation if for all $\alpha \in \mathbb{C}$ and $x, y \in \mathbb{C}^n$

- $L(\alpha x) = \alpha L(x)$. That is, scaling first and then transforming yields the same result as transforming first and then scaling.
- $L(x + y) = L(x) + L(y)$. That is, adding first and then transforming yields the same result as transforming first and then adding.



The importance of linear transformations comes in part from the fact that many problems in science boil down to, given a function $F : \mathbb{C}^n \rightarrow \mathbb{C}^m$ and vector $y \in \mathbb{C}^m$, find x such that $F(x) = y$. This is known as an inverse problem. Under mild conditions, F can be locally approximated with a linear transformation L and then, as part of a solution method, one would want to solve $Lx = y$.

The following theorem provides the link between linear transformations and matrices:

Theorem 1.3.1.2 *Let $L : \mathbb{C}^n \rightarrow \mathbb{C}^m$ be a linear transformation, $v_0, v_1, \dots, v_{k-1} \in \mathbb{C}^n$, and $x \in \mathbb{C}^k$. Then*

$$L(\chi_0 v_0 + \chi_1 v_1 + \dots + \chi_{k-1} v_{k-1}) = \chi_0 L(v_0) + \chi_1 L(v_1) + \dots + \chi_{k-1} L(v_{k-1}),$$

where

$$x = \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{k-1} \end{pmatrix}.$$

Proof. A simple inductive proof yields the result. For details, see Week 2 of Linear Algebra: Foundations to Frontiers (LAFF) [26]. ■

The following set of vectors ends up playing a crucial role throughout this course:

Definition 1.3.1.3 Standard basis vector. In this course, we will use $e_j \in \mathbb{C}^m$ to denote the standard basis vector with a "1" in the position indexed with j . So,

$$e_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow j$$

◇

Key is the fact that any vector $x \in \mathbb{C}^n$ can be written as a linear combination of the standard basis vectors of \mathbb{C}^n :

$$\begin{aligned} x &= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} = \chi_0 \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \chi_1 \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \dots + \chi_{n-1} \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \\ &= \chi_0 e_0 + \chi_1 e_1 + \dots + \chi_{n-1} e_{n-1}. \end{aligned}$$

Hence, if L is a linear transformation,

$$\begin{aligned} L(x) &= L(\chi_0 e_0 + \chi_1 e_1 + \dots + \chi_{n-1} e_{n-1}) \\ &= \chi_0 \underbrace{L(e_0)}_{a_0} + \chi_1 \underbrace{L(e_1)}_{a_1} + \dots + \chi_{n-1} \underbrace{L(e_{n-1})}_{a_{n-1}}. \end{aligned}$$

If we now let $a_j = L(e_j)$ (the vector a_j is the transformation of the standard basis vector e_j) and collect these vectors into a two-dimensional array of numbers:

$$A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) \quad (1.3.1)$$

then we notice that information for evaluating $L(x)$ can be found in this array, since L can then alternatively be computed by

$$L(x) = \chi_0 a_0 + \chi_1 a_1 + \cdots + \chi_{n-1} a_{n-1}.$$

The array A in (1.3.1) we call a **matrix** and the operation $Ax = \chi_0 a_0 + \chi_1 a_1 + \cdots + \chi_{n-1} a_{n-1}$ we call **matrix-vector multiplication**. Clearly

$$Ax = L(x).$$

Remark 1.3.1.4 Notation. In these notes, as a rule,

- Roman upper case letters are used to denote matrices.
- Roman lower case letters are used to denote vectors.
- Greek lower case letters are used to denote scalars.

Corresponding letters from these three sets are used to refer to a matrix, the row or columns of that matrix, and the elements of that matrix. If $A \in \mathbb{C}^{m \times n}$ then

$$\begin{aligned} A &= <\text{partition } A \text{ by columns and rows}> \\ \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) &= \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} \\ &= <\text{expose the elements of } A> \\ &\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \hline \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} \end{aligned}$$

We now notice that the standard basis vector $e_j \in \mathbb{C}^m$ equals the column of the $m \times m$ **identity matrix** indexed with j :

$$I = \left(\begin{array}{c|c|c|c} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{array} \right) = \left(\begin{array}{c|c|c|c} e_0 & e_1 & \cdots & e_{m-1} \end{array} \right) = \begin{pmatrix} \tilde{e}_0^T \\ \tilde{e}_1^T \\ \vdots \\ \tilde{e}_{m-1}^T \end{pmatrix}.$$

Remark 1.3.1.5 The important thing to note is that a matrix is a convenient representation of a linear transformation and matrix-vector multiplication is an alternative way for evaluating that linear transformation.



YouTube: <https://www.youtube.com/watch?v=cCFAnQmwwIw>

Let's investigate matrix-matrix multiplication and its relationship to linear transformations. Consider two linear transformations

$$\begin{aligned} L_A : \mathbb{C}^k &\rightarrow \mathbb{C}^m \quad \text{represented by matrix } A \\ L_B : \mathbb{C}^n &\rightarrow \mathbb{C}^k \quad \text{represented by matrix } B \end{aligned}$$

and define

$$L_C(x) = L_A(L_B(x)),$$

as the composition of L_A and L_B . Then it can be easily shown that L_C is also a linear transformation. Let $m \times n$ matrix C represent L_C . How are A , B , and C related? If we let c_j equal the column of C indexed with j , then because of the link between matrices, linear transformations, and standard basis vectors

$$c_j = L_C(e_j) = L_A(L_B(e_j)) = L_A(b_j) = Ab_j,$$

where b_j equals the column of B indexed with j . Now, we say that $C = AB$ is the product of A and B defined by

$$\left(\begin{array}{c|c|c|c} c_0 & c_1 & \cdots & c_{n-1} \end{array} \right) = A \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} Ab_0 & Ab_1 & \cdots & Ab_{n-1} \end{array} \right)$$

and define the matrix-matrix multiplication as the operation that computes

$$C := AB,$$

which you will want to pronounce "C becomes A times B" to distinguish assignment from equality. If you think carefully how individual elements of C are computed, you will realize that they equal the usual "dot product of rows of A with columns of B ."



YouTube: https://www.youtube.com/watch?v=g_9RbA5EOIc

As already mentioned, throughout this course, it will be important that you can think about matrices in terms of their columns and rows, and matrix-matrix multiplication (and other operations with matrices and vectors) in terms of columns and rows. It is also important to be able to think about matrix-matrix multiplication in three different ways. If we partition each matrix by rows and by columns:

$$C = \left(\begin{array}{c|c|c} c_0 & \cdots & c_{n-1} \end{array} \right) = \left(\begin{array}{c} \tilde{c}_0^T \\ \vdots \\ \tilde{c}_{m-1}^T \end{array} \right), A = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{k-1} \end{array} \right) = \left(\begin{array}{c} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{array} \right),$$

and

$$B = \left(\begin{array}{c|c|c} b_0 & \cdots & b_{n-1} \end{array} \right) = \left(\begin{array}{c} \tilde{b}_0^T \\ \vdots \\ \tilde{b}_{k-1}^T \end{array} \right),$$

then $C := AB$ can be computed in the following ways:

1. By columns:

$$\left(\begin{array}{c|c|c} c_0 & \cdots & c_{n-1} \end{array} \right) := A \left(\begin{array}{c|c|c} b_0 & \cdots & b_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c} Ab_0 & \cdots & Ab_{n-1} \end{array} \right).$$

In other words, $c_j := Ab_j$ for all columns of C .

2. By rows:

$$\left(\begin{array}{c} \tilde{c}_0^T \\ \vdots \\ \tilde{c}_{m-1}^T \end{array} \right) := \left(\begin{array}{c} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{array} \right) B = \left(\begin{array}{c} \tilde{a}_0^T B \\ \vdots \\ \tilde{a}_{m-1}^T B \end{array} \right).$$

In other words, $\tilde{c}_i^T = \tilde{a}_i^T B$ for all rows of C .

3. One you may not have thought about much before:

$$C := \left(\begin{array}{c|c|c} a_0 & \cdots & a_{k-1} \end{array} \right) \left(\begin{array}{c} \tilde{b}_0^T \\ \vdots \\ \tilde{b}_{k-1}^T \end{array} \right) = a_0 \tilde{b}_0^T + \cdots + a_{k-1} \tilde{b}_{k-1}^T,$$

which should be thought of as a sequence of rank-1 updates, since each term is an outer product and an outer product has rank of at most one.

These three cases are special cases of the more general observation that, if we can partition C , A , and B by blocks (submatrices),

$$C = \left(\begin{array}{c|c|c} C_{0,0} & \cdots & C_{0,N-1} \\ \hline \vdots & & \vdots \\ \hline C_{M-1,0} & \cdots & C_{M-1,N-1} \end{array} \right), \left(\begin{array}{c|c|c} A_{0,0} & \cdots & A_{0,K-1} \\ \hline \vdots & & \vdots \\ \hline A_{M-1,0} & \cdots & A_{M-1,K-1} \end{array} \right),$$

and

$$\left(\begin{array}{c|c|c} B_{0,0} & \cdots & B_{0,N-1} \\ \hline \vdots & & \vdots \\ \hline B_{K-1,0} & \cdots & B_{K-1,N-1} \end{array} \right),$$

where the partitionings are "conformal", then

$$C_{i,j} = \sum_{p=0}^{K-1} A_{i,p} B_{p,j}.$$

Remark 1.3.1.6 If the above review of linear transformations, matrices, matrix-vector multiplication, and matrix-matrix multiplication makes you exclaim "That is all a bit too fast for me!" then it is time for you to take a break and review Weeks 2-5 of our introductory linear algebra course "Linear Algebra: Foundations to Frontiers." Information, including notes [26] (optionally downloadable for free) and a link to the course on edX [27] (which can be audited for free) can be found at <http://ulaff.net>.

1.3.2 What is a matrix norm?



YouTube: <https://www.youtube.com/watch?v=6DsBTz1eU7E>

A matrix norm extends the notions of an absolute value and vector norm to matrices:

Definition 1.3.2.1 **Matrix norm.** Let $\nu : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$. Then ν is a (matrix) norm if for all $A, B \in \mathbb{C}^{m \times n}$ and all $\alpha \in \mathbb{C}$

- $A \neq 0 \Rightarrow \nu(A) > 0$ (ν is positive definite),
- $\nu(\alpha A) = |\alpha|\nu(A)$ (ν is homogeneous), and
- $\nu(A + B) \leq \nu(A) + \nu(B)$ (ν obeys the triangle inequality).

◊

Homework 1.3.2.1 Let $\nu : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ be a matrix norm.

ALWAYS/SOMETIMES/NEVER: $\nu(0) = 0$.

Hint. Review the proof on [Homework 1.2.2.1](#).

Answer. ALWAYS.

Now prove it.

Solution. Let $A \in \mathbb{C}^{m \times n}$. Then

$$\begin{aligned}\nu(0) &= \langle 0 \cdot A = 0 \rangle \\ \nu(0 \cdot A) &= \langle \|\cdot\|_\nu \text{ is homogeneous} \rangle \\ 0\nu(A) &= \langle \text{algebra} \rangle \\ 0 &\end{aligned}$$

Remark 1.3.2.2 As we do with vector norms, we will typically use $\|\cdot\|$ instead of $\nu(\cdot)$ for a function that is a matrix norm.

1.3.3 The Frobenius norm



YouTube: <https://www.youtube.com/watch?v=0ZHnGgrJXa4>

Definition 1.3.3.1 The Frobenius norm. The Frobenius norm $\|\cdot\|_F : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is defined for $A \in \mathbb{C}^{m \times n}$ by

$$\|A\|_F = \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2} = \sqrt{|\alpha_{0,0}|^2 + \cdots + |\alpha_{0,n-1}|^2 + \cdots + |\alpha_{m-1,0}|^2 + \cdots + |\alpha_{m-1,n-1}|^2}.$$

◇

One can think of the Frobenius norm as taking the columns of the matrix, stacking them on top of each other to create a vector of size $m \times n$, and then taking the vector 2-norm of the result.

Homework 1.3.3.1 Partition $m \times n$ matrix A by columns:

$$A = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right).$$

Show that

$$\|A\|_F^2 = \sum_{j=0}^{n-1} \|a_j\|_2^2.$$

Solution.

$$\begin{aligned}
 \|A\|_F &= <\text{definition}> \\
 &= \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2} \\
 &= <\text{commutativity of addition}> \\
 &= \sqrt{\sum_{j=0}^{n-1} \sum_{i=0}^{m-1} |\alpha_{i,j}|^2} \\
 &= <\text{definition of vector 2-norm}> \\
 &= \sqrt{\sum_{j=0}^{n-1} \|a_j\|_2^2}
 \end{aligned}$$

Homework 1.3.3.2 Prove that the Frobenius norm is a norm.

Solution. Establishing that this function is positive definite and homogeneous is straight forward. To show that the triangle inequality holds it helps to realize that if $A = (a_0 \mid a_1 \mid \cdots \mid a_{n-1})$ then

$$\begin{aligned}
 \|A\|_F &= <\text{definition}> \\
 &= \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2} \\
 &= <\text{commutativity of addition}> \\
 &= \sqrt{\sum_{j=0}^{n-1} \sum_{i=0}^{m-1} |\alpha_{i,j}|^2} \\
 &= <\text{definition of vector 2-norm}> \\
 &= \sqrt{\sum_{j=0}^{n-1} \|a_j\|_2^2} \\
 &= <\text{definition of vector 2-norm}> \\
 &= \sqrt{\left\| \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \right\|_2^2}.
 \end{aligned}$$

In other words, it equals the vector 2-norm of the vector that is created by stacking the columns of A on top of each other. One can then exploit the fact that the vector 2-norm obeys the triangle inequality.

Homework 1.3.3.3 Partition $m \times n$ matrix A by rows:

$$A = \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix}.$$

Show that

$$\|A\|_F^2 = \sum_{i=0}^{m-1} \|\tilde{a}_i\|_2^2,$$

where $\tilde{a}_i = \tilde{a}_i^T$.

Solution.

$$\begin{aligned}
 \|A\|_F &= <\text{definition}> \\
 &= \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2} \\
 &= <\text{definition of vector 2-norm}> \\
 &= \sqrt{\sum_{i=0}^{m-1} \|\tilde{a}_i\|_2^2}.
 \end{aligned}$$

Let us review the definition of the transpose of a matrix (which we have already used when defining the dot product of two real-valued vectors and when identifying a row in a matrix):

Definition 1.3.3.2 Transpose. If $A \in \mathbb{C}^{m \times n}$ and

$$A = \left(\begin{array}{c|c|c|c} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \hline \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{array} \right)$$

then its **transpose** is defined by

$$A^T = \left(\begin{array}{c|c|c|c} \alpha_{0,0} & \alpha_{1,0} & \cdots & \alpha_{m-1,0} \\ \hline \alpha_{0,1} & \alpha_{1,1} & \cdots & \alpha_{m-1,1} \\ \hline \vdots & \vdots & & \vdots \\ \hline \alpha_{0,n-1} & \alpha_{1,n-1} & \cdots & \alpha_{m-1,n-1} \end{array} \right).$$

◊

For complex-valued matrices, it is important to also define the **Hermitian transpose** of a matrix:

Definition 1.3.3.3 Hermitian transpose. If $A \in \mathbb{C}^{m \times n}$ and

$$A = \left(\begin{array}{c|c|c|c} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \hline \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{array} \right)$$

then its **Hermitian transpose** is defined by

$$A^H = \overline{A}^T \left(\begin{array}{c|c|c|c} \overline{\alpha}_{0,0} & \overline{\alpha}_{1,0} & \cdots & \overline{\alpha}_{m-1,0} \\ \hline \overline{\alpha}_{0,1} & \overline{\alpha}_{1,1} & \cdots & \overline{\alpha}_{m-1,1} \\ \vdots & \vdots & & \vdots \\ \vdots & & & \vdots \\ \hline \overline{\alpha}_{0,n-1} & \overline{\alpha}_{1,n-1} & \cdots & \overline{\alpha}_{m-1,n-1} \end{array} \right),$$

where \overline{A} denotes the **conjugate of a matrix**, in which each element of the matrix is conjugated. \diamond

We note that

- $\overline{A}^T = \overline{A^T}$.
- If $A \in \mathbb{R}^{m \times n}$, then $A^H = A^T$.
- If $x \in \mathbb{C}^m$, then x^H is defined consistent with how we have used it before.
- If $\alpha \in \mathbb{C}$, then $\alpha^H = \overline{\alpha}$.

(If you view the scalar as a matrix and then Hermitian transpose it, you get the matrix with as only element $\overline{\alpha}$.)

Don't Panic! While working with complex-valued scalars, vectors, and matrices may appear a bit scary at first, you will soon notice that it is not really much more complicated than working with their real-valued counterparts.

Homework 1.3.3.4 Let $A \in \mathbb{C}^{m \times k}$ and $B \in \mathbb{C}^{k \times n}$. Using what you once learned about matrix transposition and matrix-matrix multiplication, reason that $(AB)^H = B^H A^H$.

Solution.

$$\begin{aligned} (AB)^H &= < X^H = \overline{X^T} > \\ \overline{(AB)^T} &= < \text{you once discovered that } (AB)^T = B^T A^T > \\ \overline{B^T A^T} &= < \text{you may check separately that } \overline{XY} = \overline{X}\overline{Y} > \\ \overline{B^T} \overline{A^T} &= < \overline{X^T} = \overline{X}^T > \\ B^H A^H & \end{aligned}$$

Definition 1.3.3.4 Hermitian. A matrix $A \in \mathbb{C}^{m \times m}$ is **Hermitian** if and only if $A = A^H$. \diamond

Obviously, if $A \in \mathbb{R}^{m \times m}$, then A is a Hermitian matrix if and only if A is a symmetric matrix.

Homework 1.3.3.5 Let $A \in \mathbb{C}^{m \times n}$.

ALWAYS/SOMETIMES/NEVER: $\|A^H\|_F = \|A\|_F$.

Answer. ALWAYS

Solution.

$$\begin{aligned}
 \|A\|_F &= <\text{definition}> \\
 &= \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2} \\
 &= <\text{commutativity of addition}> \\
 &= \sqrt{\sum_{j=0}^{n-1} \sum_{i=0}^{m-1} |\alpha_{i,j}|^2} \\
 &= <\text{change of variables}> \\
 &= \sqrt{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} |\alpha_{j,i}|^2} \\
 &= <\text{algebra}> \\
 &= \sqrt{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} |\bar{\alpha}_{j,i}|^2} \\
 &= <\text{definition}> \\
 \|A^H\|_F
 \end{aligned}$$

Similarly, other matrix norms can be created from vector norms by viewing the matrix as a vector. It turns out that, other than the Frobenius norm, these aren't particularly interesting in practice. An example can be found in [Homework 1.6.1.6](#).

Remark 1.3.3.5 The Frobenius norm of a $m \times n$ matrix is easy to compute (requiring $O(mn)$ computations). The functions $f(A) = \|A\|_F$ and $f(A) = \|A\|_F^2$ are also differentiable. However, you'd be hard-pressed to find a meaningful way of linking the definition of the Frobenius norm to a measure of an underlying linear transformation (other than by first transforming that linear transformation into a matrix).

1.3.4 Induced matrix norms



YouTube: <https://www.youtube.com/watch?v=M6ZVBRFnYcU>

Recall from [Subsection 1.3.1](#) that a matrix, $A \in \mathbb{C}^{m \times n}$, is a 2-dimensional array of numbers that represents a linear transformation, $L : \mathbb{C}^n \rightarrow \mathbb{C}^m$, such that for all $x \in \mathbb{C}^n$ the matrix-vector multiplication Ax yields the same result as does $L(x)$.

The question "What is the norm of matrix A ?" or, equivalently, "How 'large' is A ?" is the same as asking the question "How 'large' is L ?" What does this mean? It suggests that what we really want is a measure of how much linear transformation L or, equivalently, matrix A "stretches" (magnifies) the "length" of a vector. This observation motivates a class of matrix norms known as induced matrix norms.

Definition 1.3.4.1 Induced matrix norm. Let $\|\cdot\|_\mu : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ be vector norms. Define $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

$$\|A\|_{\mu,\nu} = \sup_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_\mu}{\|x\|_\nu}.$$

◊

Matrix norms that are defined in this way are said to be **induced** matrix norms.

Remark 1.3.4.2 In context, it is obvious (from the column size of the matrix) what the size of vector x is. For this reason, we will write

$$\|A\|_{\mu,\nu} = \sup_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_\mu}{\|x\|_\nu} \quad \text{as} \quad \|A\|_{\mu,\nu} = \sup_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\nu}.$$

Let us start by interpreting this. How "large" A is, as measured by $\|A\|_{\mu,\nu}$, is defined as the most that A magnifies the length of nonzero vectors, where the length of the vector, x , is measured with norm $\|\cdot\|_\nu$ and the length of the transformed vector, Ax , is measured with norm $\|\cdot\|_\mu$.

Two comments are in order. First,

$$\sup_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\nu} = \sup_{\|x\|_\nu=1} \|Ax\|_\mu.$$

This follows from the following sequence of equivalences:

$$\begin{aligned} & \sup_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\nu} \\ &= \quad < \text{homogeneity} > \\ & \sup_{x \neq 0} \left\| \frac{Ax}{\|x\|_\nu} \right\|_\mu \\ &= \quad < \text{norms are associative} > \\ & \sup_{x \neq 0} \left\| A \frac{x}{\|x\|_\nu} \right\|_\mu \\ &= \quad < \text{substitute } y = x/\|x\|_\nu > \\ & \sup_{\|y\|_\nu=1} \|Ay\|_\mu. \end{aligned}$$

Second, the "sup" (which stands for supremum) is used because we can't claim yet that there is a nonzero vector x for which

$$\sup_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\nu}$$

is attained or, alternatively, a vector, x , with $\|x\|_\nu = 1$ for which

$$\sup_{\|x\|_\nu=1} \|Ax\|_\mu$$

is attained. In words, it is not immediately obvious that there is a vector for which the supremum is attained. The fact is that there is always such a vector x . The proof again depends on a result from real analysis, also employed in [Proof 1.2.6.1](#), that states that $\sup_{x \in S} f(x)$ is attained for some vector $x \in S$ as long as f is continuous and S is a compact set. For any norm, $\|x\| = 1$ is a compact set. Thus, we can replace sup by max from here on in our discussion.

We conclude that the following two definitions are equivalent definitions to the one we already gave:

Definition 1.3.4.3 Let $\|\cdot\|_\mu : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ be vector norms. Define $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

$$\|A\|_{\mu,\nu} = \max_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\nu}.$$

or, equivalently,

$$\|A\|_{\mu,\nu} = \max_{\|x\|_\nu=1} \|Ax\|_\mu.$$

◊

Remark 1.3.4.4 In this course, we will often encounter proofs involving norms. Such proofs are much cleaner if one starts by strategically picking the most convenient of these two definitions. Until you gain the intuition needed to pick which one is better, you may have to start your proof using one of them and then switch to the other one if the proof becomes unwieldy.

Theorem 1.3.4.5 $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is a norm.

Proof. To prove this, we merely check whether the three conditions are met:

Let $A, B \in \mathbb{C}^{m \times n}$ and $\alpha \in \mathbb{C}$ be arbitrarily chosen. Then

- $A \neq 0 \Rightarrow \|A\|_{\mu,\nu} > 0$ ($\|\cdot\|_{\mu,\nu}$ is positive definite):

Notice that $A \neq 0$ means that at least one of its columns is not a zero vector (since at least one element is nonzero). Let us assume it is the j th column, a_j , that is nonzero. Let e_j equal the column of I (the identity matrix) indexed with j . Then

$$\begin{aligned} \|A\|_{\mu,\nu} &= <\text{definition}> \\ &= \max_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\nu} \\ &\geq <\text{ } e_j \text{ is a specific vector}> \\ &= \frac{\|Ae_j\|_\mu}{\|e_j\|_\nu} \\ &= <\text{ } Ae_j = a_j > \\ &= \frac{\|a_j\|_\mu}{\|e_j\|_\nu} \\ &> <\text{we assumed that } a_j \neq 0> \\ &= 0. \end{aligned}$$

- $\|\alpha A\|_{\mu,\nu} = |\alpha| \|A\|_{\mu,\nu}$ ($\|\cdot\|_{\mu,\nu}$ is homogeneous):

$$\begin{aligned}
& \|\alpha A\|_{\mu,\nu} \\
&= <\text{definition}> \\
& \max_{x \neq 0} \frac{\|\alpha Ax\|_\mu}{\|x\|_\nu} \\
&= <\text{homogeneity}> \\
& \max_{x \neq 0} |\alpha| \frac{\|Ax\|_\mu}{\|x\|_\nu} \\
&= <\text{algebra}> \\
& |\alpha| \max_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\nu} \\
&= <\text{definition}> \\
& |\alpha| \|A\|_{\mu,\nu}.
\end{aligned}$$

- $\|A + B\|_{\mu,\nu} \leq \|A\|_{\mu,\nu} + \|B\|_{\mu,\nu}$ ($\|\cdot\|_{\mu,\nu}$ obeys the triangle inequality).

$$\begin{aligned}
& \|A + B\|_{\mu,\nu} \\
&= <\text{definition}> \\
& \max_{x \neq 0} \frac{\|(A+B)x\|_\mu}{\|x\|_\nu} \\
&= <\text{distribute}> \\
& \max_{x \neq 0} \frac{\|Ax+Bx\|_\mu}{\|x\|_\nu} \\
&\leq <\text{triangle inequality}> \\
& \max_{x \neq 0} \frac{\|Ax\|_\mu + \|Bx\|_\mu}{\|x\|_\nu} \\
&\leq <\text{algebra}> \\
& \max_{x \neq 0} \left(\frac{\|Ax\|_\mu}{\|x\|_\nu} + \frac{\|Bx\|_\mu}{\|x\|_\nu} \right) \\
&\leq <\text{algebra}> \\
& \max_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\nu} + \max_{x \neq 0} \frac{\|Bx\|_\mu}{\|x\|_\nu} \\
&= <\text{definition}> \\
& \|A\|_{\mu,\nu} + \|B\|_{\mu,\nu}.
\end{aligned}$$

■

When $\|\cdot\|_\mu$ and $\|\cdot\|_\nu$ are the same norm (but possibly for different sizes of vectors), the induced norm becomes

Definition 1.3.4.6 Define $\|\cdot\|_\mu : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

$$\|A\|_\mu = \max_{x \neq 0} \frac{\|Ax\|_\mu}{\|x\|_\mu}$$

or, equivalently,

$$\|A\|_\mu = \max_{\|x\|_\mu=1} \|Ax\|_\mu.$$

◇

Homework 1.3.4.1 Consider the vector p -norm $\|\cdot\|_p : \mathbb{C}^n \rightarrow \mathbb{R}$ and let us denote the induced matrix norm by $\|\|\cdot\||| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ for this exercise: $\|\|A\||| = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$.

ALWAYS/SOMETIMES/NEVER: $\|\|y\||| = \|y\|_p$ for $y \in \mathbb{C}^m$.

Answer. ALWAYS

Solution.

$$\begin{aligned}
 & \||y|\| \\
 &= <\text{definition}> \\
 & \max_{x \neq 0} \frac{\|yx\|_p}{\|x\|_p} \\
 &= <x \text{ is a scalar since } y \text{ is a matrix with one column. Then } \|x\|_p = \|(\chi_0)\|_p = \sqrt[p]{|\chi_0|^p} = |\chi_0|> \\
 & \max_{\chi_0 \neq 0} |\chi_0| \frac{\|y\|_p}{|\chi_0|} \\
 &= <\text{algebra}> \\
 & \max_{\chi_0 \neq 0} \|y\|_p \\
 &= <\text{algebra}> \\
 & \|y\|_p
 \end{aligned}$$

This last exercise is important. One can view a vector $x \in \mathbb{C}^m$ as an $m \times 1$ matrix. What this last exercise tells us is that regardless of whether we view x as a matrix or a vector, $\|x\|_p$ is the same.

We already encountered the vector p -norms as an important class of vector norms. The matrix p -norm is induced by the corresponding vector norm, as defined by

Definition 1.3.4.7 Matrix p -norm. For any vector p -norm, define the corresponding matrix p -norm $\|\cdot\|_p : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

$$\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \quad \text{or, equivalently,} \quad \|A\|_p = \max_{\|x\|_p=1} \|Ax\|_p.$$

◇

Remark 1.3.4.8 The matrix p -norms with $p \in \{1, 2, \infty\}$ will play an important role in our course, as will the Frobenius norm. As the course unfolds, we will realize that in practice the matrix 2-norm is of great theoretical importance but difficult to evaluate, except for special matrices. The 1-norm, ∞ -norm, and Frobenius norms are straightforward and relatively cheap to compute (for an $m \times n$ matrix, computing these costs $O(mn)$ computation).

1.3.5 The matrix 2-norm



YouTube: https://www.youtube.com/watch?v=wZAlH_K9XeI

Let us instantiate the definition of the vector p norm for the case where $p = 2$, giving us a matrix norm induced by the vector 2-norm or Euclidean norm:

Definition 1.3.5.1 Matrix 2-norm. Define the matrix 2-norm $\|\cdot\|_2 : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2=1} \|Ax\|_2.$$

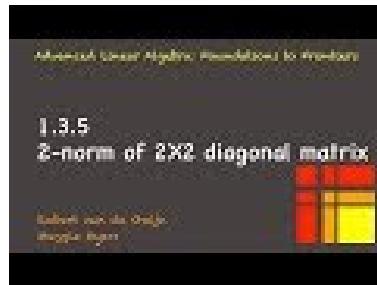
◊

Remark 1.3.5.2 The problem with the matrix 2-norm is that it is hard to compute. At some point later in this course, you will find out that if A is a Hermitian matrix ($A = A^H$), then $\|A\|_2 = |\lambda_0|$, where λ_0 equals the eigenvalue of A that is largest in magnitude. You may recall from your prior linear algebra experience that computing eigenvalues involves computing the roots of polynomials, and for polynomials of degree three or greater, this is a nontrivial task. We will see that the matrix 2-norm plays an important role in the theory of linear algebra, but less so in practical computation.

Example 1.3.5.3 Show that

$$\left\| \begin{pmatrix} \delta_0 & 0 \\ 0 & \delta_1 \end{pmatrix} \right\|_2 = \max(|\delta_0|, |\delta_1|).$$

Solution.



YouTube: <https://www.youtube.com/watch?v=B2rz0i5BB3A>

[slides (PDF)] [LaTeX source] □

Remark 1.3.5.4 The proof of the last example builds on a general principle: Showing that $\max_{x \in D} f(x) = \alpha$ for some function $f : D \rightarrow R$ can be broken down into showing that both

$$\max_{x \in D} f(x) \leq \alpha$$

and

$$\max_{x \in D} f(x) \geq \alpha.$$

In turn, showing that $\max_{x \in D} f(x) \geq \alpha$ can often be accomplished by showing that there exists a vector $y \in D$ such that $f(y) = \alpha$ since then

$$\alpha = f(y) \leq \max_{x \in D} f(x).$$

We will use this technique in future proofs involving matrix norms.

Homework 1.3.5.1 Let $D \in \mathbb{C}^{m \times m}$ be a diagonal matrix with diagonal entries $\delta_0, \dots, \delta_{m-1}$. Show that

$$\|D\|_2 = \max_{j=0}^{m-1} |\delta_j|.$$

Solution. First, we show that $\|D\|_2 = \max_{\|x\|_2=1} \|Dx\|_2 \leq \max_{i=0}^{m-1} |\delta_i|$:

$$\begin{aligned} & \|D\|_2^2 \\ &= <\text{definition}> \\ &= \max_{\|x\|_2=1} \|Dx\|_2^2 \\ &= <\text{diagonal vector multiplication}> \\ &= \max_{\|x\|_2=1} \left\| \begin{pmatrix} \delta_0 \chi_0 \\ \vdots \\ \delta_{m-1} \chi_{m-1} \end{pmatrix} \right\|_2^2 \\ &= <\text{definition}> \\ &= \max_{\|x\|_2=1} \sum_{i=0}^{m-1} |\delta_i \chi_i|^2 \\ &= <\text{homogeneity}> \\ &= \max_{\|x\|_2=1} \sum_{i=0}^{m-1} |\delta_i|^2 |\chi_i|^2 \\ &\leq <\text{algebra}> \\ &= \max_{\|x\|_2=1} \sum_{i=0}^{m-1} \left[\max_{j=0}^{m-1} |\delta_j| \right]^2 |\chi_i|^2 \\ &= <\text{algebra}> \\ &= \left[\max_{j=0}^{m-1} |\delta_j| \right]^2 \max_{\|x\|_2=1} \sum_{i=0}^{m-1} |\chi_i|^2 \\ &= <\|x\|_2 = 1> \\ &= \left[\max_{j=0}^{m-1} |\delta_j| \right]^2. \end{aligned}$$

Next, we show that there is a vector y with $\|y\|_2 = 1$ such that $\|Dy\|_2 = \max_{i=0}^{m-1} |\delta_i|$: Let j be such that $|\delta_j| = \max_{i=0}^{m-1} |\delta_i|$ and choose $y = e_j$. Then

$$\begin{aligned} & \|Dy\|_2 \\ &= <y = e_j> \\ &= \|De_j\|_2 \\ &= < D = \text{diag}(\delta_0, \dots, \delta_{m-1})> \\ &= \|\delta_j e_j\|_2 \\ &= <\text{homogeneity}> \\ &= |\delta_j| \|e_j\|_2 \\ &= <\|e_j\|_2 = 1> \\ &= |\delta_j| \\ &= <\text{choice of } j> \\ &= \max_{i=0}^{m-1} |\delta_i| \end{aligned}$$

Hence $\|D\|_2 = \max_{j=0}^{m-1} |\delta_j|$.

Homework 1.3.5.2 Let $y \in \mathbb{C}^m$ and $x \in \mathbb{C}^n$.

ALWAYS/SOMETIMES/NEVER: $\|yx^H\|_2 = \|y\|_2\|x\|_2$.

Hint. Prove that $\|yx^H\|_2 \leq \|y\|_2\|x\|_2$ and that there exists a vector z so that $\frac{\|yx^H z\|_2}{\|z\|_2} = \|y\|_2\|x\|_2$.

Answer. ALWAYS

Now prove it!

Solution. W.l.o.g. assume that $x \neq 0$.

We know by the Cauchy-Schwarz inequality that $|x^H z| \leq \|x\|_2\|z\|_2$. Hence

$$\begin{aligned} \|yx^H\|_2 &= <\text{definition}> \\ &= \max_{\|z\|_2=1} \|yx^H z\|_2 \\ &= <\|\cdot\|_2 \text{ is homogenous}> \\ &= \max_{\|z\|_2=1} |x^H z| \|y\|_2 \\ &\leq <\text{Cauchy-Schwarz inequality}> \\ &= \max_{\|z\|_2=1} \|x\|_2\|z\|_2\|y\|_2 \\ &= <\|z\|_2 = 1> \\ &= \|x\|_2\|y\|_2. \end{aligned}$$

But also

$$\begin{aligned} \|yx^H\|_2 &= <\text{definition}> \\ &= \max_{z \neq 0} \|yx^H z\|_2/\|z\|_2 \\ &\geq <\text{specific } z> \\ &= \|yx^H x\|_2/\|x\|_2 \\ &= <x^H x = \|x\|_2^2; \text{ homogeneity}> \\ &= \|x\|_2^2\|y\|_2/\|x\|_2 \\ &= <\text{algebra}> \\ &= \|y\|_2\|x\|_2. \end{aligned}$$

Hence

$$\|yx^H\|_2 = \|y\|_2\|x\|_2.$$

Homework 1.3.5.3 Let $A \in \mathbb{C}^{m \times n}$ and a_j its column indexed with j . ALWAYS/SOMETIMES/NEVER: $\|a_j\|_2 \leq \|A\|_2$.

Hint. What vector has the property that $a_j = Ax$?

Answer. ALWAYS.

Now prove it!

Solution.

$$\begin{aligned} \|a_j\|_2 &= \\ \|Ae_j\|_2 &\leq \\ \max_{\|x\|_2=1} \|Ax\|_2 &= \\ \|A\|_2. & \end{aligned}$$

Homework 1.3.5.4 Let $A \in \mathbb{C}^{m \times n}$. Prove that

- $\|A\|_2 = \max_{\|x\|_2=\|y\|_2=1} |y^H Ax|$.
- $\|A^H\|_2 = \|A\|_2$.
- $\|A^H A\|_2 = \|A\|_2^2$.

Hint. Proving $\|A\|_2 = \max_{\|x\|_2=\|y\|_2=1} |y^H Ax|$ requires you to invoke the Cauchy-Schwarz inequality from [Theorem 1.2.3.3](#).

Solution.

- $\|A\|_2 = \max_{\|x\|_2=\|y\|_2=1} |y^H Ax|$:

$$\begin{aligned} \max_{\|x\|_2=\|y\|_2=1} |y^H Ax| &\leq < \text{Cauchy-Schwarz} > \\ \max_{\|x\|_2=\|y\|_2=1} \|y\|_2 \|Ax\|_2 &= < \|y\|_2 = 1 > \\ \max_{\|x\|_2=1} \|Ax\|_2 &= < \text{definition} > \\ \|A\|_2. & \end{aligned}$$

Also, we know there exists x with $\|x\|_2 = 1$ such that $\|A\|_2 = \|Ax\|_2$. Let $y = Ax/\|Ax\|_2$. Then

$$\begin{aligned} |y^H Ax| &= < \text{instantiate} > \\ \left| \frac{(Ax)^H (Ax)}{\|Ax\|_2} \right| &= < z^H z = \|z\|_2^2 > \\ \left| \frac{\|Ax\|_2^2}{\|Ax\|_2} \right| &= < \text{algebra} > \\ \|Ax\|_2 &= < x \text{ was chosen so that } \|Ax\|_2 = \|A\|_2 > \\ \|A\|_2 & \end{aligned}$$

Hence the bound is attained. We conclude that $\|A\|_2 = \max_{\|x\|_2=\|y\|_2=1} |y^H Ax|$.

- $\|A^H\|_2 = \|A\|_2$:

$$\begin{aligned}
 & \|A^H\|_2 \\
 &= <\text{first part of homework}> \\
 & \max_{\|x\|_2=\|y\|_2=1} |y^H A^H x| \\
 &= <|\bar{\alpha}| = |\alpha|> \\
 & \max_{\|x\|_2=\|y\|_2=1} |x^H A y| \\
 &= <\text{first part of homework}> \\
 & \|A\|_2.
 \end{aligned}$$

- $\|A^H A\|_2 = \|A\|_2^2$:

$$\begin{aligned}
 & \|A^H A\|_2 \\
 &= <\text{first part of homework}> \\
 & \max_{\|x\|_2=\|y\|_2=1} |y^H A^H A x| \\
 &\geq <\text{restricts choices of } y> \\
 & \max_{\|x\|_2=1} |x^H A^H A x| \\
 &= <z^H z = \|z\|_2^2> \\
 & \max_{\|x\|_2=1} \|Ax\|_2^2 \\
 &= <\text{algebra}> \\
 & \left(\max_{\|x\|_2=1} \|Ax\|_2 \right)^2 \\
 &= <\text{definition}> \\
 & \|A\|_2^2.
 \end{aligned}$$

So, $\|A^H A\|_2 \geq \|A\|_2^2$.

Now, let's show that $\|A^H A\|_2 \leq \|A\|_2^2$. This would be trivial if we had already discussed the fact that $\|\cdot\|_2$ is a submultiplicative norm (which we will in a future unit). But let's do it from scratch. First, we show that $\|Ax\|_2 \leq \|A\|_2 \|x\|_2$ for all (appropriately sized) matrices A and x :

$$\begin{aligned}
 & \|Ax\|_2 \\
 &= <\text{norms are homogeneous}> \\
 & \|A \frac{x}{\|x\|_2}\|_2 \|x\|_2 \\
 &\leq <\text{algebra}> \\
 & \max_{\|y\|_2=1} \|Ay\|_2 \|x\|_2 \\
 &= <\text{definition of 2-norm}> \\
 & \|A\|_2 \|x\|_2.
 \end{aligned}$$

With this, we can then show that

$$\begin{aligned}
 & \|A^H A\|_2 \\
 &= \quad < \text{definition of 2-norm} > \\
 & \max_{\|x\|_2=1} \|A^H A x\|_2 \\
 &\leq \quad < \|Az\|_2 \leq \|A\|_2 \|z\|_2 > \\
 & \max_{\|x\|_2=1} (\|A^H\|_2 \|Ax\|_2) \\
 &= \quad < \text{algebra} > \\
 & \|A^H\|_2 \max_{\|x\|_2=1} \|Ax\|_2 \\
 &= \quad < \text{definition of 2-norm} > \\
 & \|A^H\|_2 \|A\|_2 \\
 &= \quad < \|A^H\|_2 = \|A\| > \\
 & \|A\|_2^2
 \end{aligned}$$

Alternatively, as suggested by one of the learners in the course, we can use the Cauchy-Schwarz inequality:

$$\begin{aligned}
 & \|A^H A\|_2 \\
 &= \quad < \text{part (a) of this homework} > \\
 & \max_{\|x\|_2=\|y\|_2=1} |x^H A^H A y| \\
 &= \quad < \text{simple manipulation} > \\
 & \max_{\|x\|_2=\|y\|_2=1} |(Ax)^H A y| \\
 &\leq \quad < \text{Cauchy-Schwarz inequality} > \\
 & \max_{\|x\|_2=\|y\|_2=1} \|Ax\|_2 \|Ay\|_2 \\
 &= \quad < \text{algebra} > \\
 & \max_{\|x\|_2=1} \|Ax\|_2 \max_{\|y\|_2=1} \|Ay\|_2 \\
 &= \quad < \text{definition} > \\
 & \|A\|_2 \|A\|_2 \\
 &= \quad < \text{algebra} > \\
 & \|A\|_2^2
 \end{aligned}$$

Homework 1.3.5.5 Partition $A = \left(\begin{array}{c|c|c} A_{0,0} & \cdots & A_{0,N-1} \\ \hline \vdots & & \vdots \\ \hline A_{M-1,0} & \cdots & A_{M-1,N-1} \end{array} \right)$.

ALWAYS/SOMETIMES/NEVER: $\|A_{i,j}\|_2 \leq \|A\|_2$.

Hint. Using [Homework 1.3.5.4](#) choose v_j and w_i such that $\|A_{i,j}\|_2 = |w_i^H A_{i,j} v_j|$.

Solution. Choose v_j and w_i such that $\|A_{i,j}\|_2 = |w_i^H A_{i,j} v_j|$. Next, choose v and w such

that

$$v = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ v_j \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad w = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ w_i \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

You can check (using partitioned multiplication and the last homework) that $w^H A v = w_i^H A_{i,j} v_j$. Then, by [Homework 1.3.5.4](#)

$$\begin{aligned} \|A\|_2 &= <\text{last homework}> \\ &\max_{\|x\|_2=\|y\|_2=1} |y^H A x| \\ &\geq <\text{w and v are specific vectors}> \\ &|w^H A v| \\ &= <\text{partitioned multiplication}> \\ &|w_i^H A_{i,j} v_j| \\ &= <\text{how } w_i \text{ and } v_j \text{ were chosen}> \\ &\|A_{i,j}\|_2. \end{aligned}$$

1.3.6 Computing the matrix 1-norm and ∞ -norm



YouTube: <https://www.youtube.com/watch?v=QTKZdGQ2C6w>

The matrix 1-norm and matrix ∞ -norm are of great importance because, unlike the matrix 2-norm, they are easy and relatively cheap to compute.. The following exercises show how to practically compute the matrix 1-norm and ∞ -norm.

Homework 1.3.6.1 Let $A \in \mathbb{C}^{m \times n}$ and partition $A = (a_0 \mid a_1 \mid \cdots \mid a_{n-1})$. ALWAYS/SOMETIMES/NEVER: $\|A\|_1 = \max_{0 \leq j < n} \|a_j\|_1$.

Hint. Prove it for the real valued case first.

Answer. ALWAYS

Solution. Let J be chosen so that $\max_{0 \leq j < n} \|a_j\|_1 = \|a_J\|_1$. Then

$$\begin{aligned}
 & \|A\|_1 \\
 &= < \text{definition} > \\
 & \max_{\|x\|_1=1} \|Ax\|_1 \\
 &= < \text{expose the columns of } A \text{ and elements of } x > \\
 & \max_{\|x\|_1=1} \left\| \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \right\|_1 \\
 &= < \text{definition of matrix-vector multiplication} > \\
 & \max_{\|x\|_1=1} \|\chi_0 a_0 + \chi_1 a_1 + \cdots + \chi_{n-1} a_{n-1}\|_1 \\
 &\leq < \text{triangle inequality} > \\
 & \max_{\|x\|_1=1} (\|\chi_0 a_0\|_1 + \|\chi_1 a_1\|_1 + \cdots + \|\chi_{n-1} a_{n-1}\|_1) \\
 &= < \text{homogeneity} > \\
 & \max_{\|x\|_1=1} (|\chi_0| \|a_0\|_1 + |\chi_1| \|a_1\|_1 + \cdots + |\chi_{n-1}| \|a_{n-1}\|_1) \\
 &\leq < \text{choice of } a_J > \\
 & \max_{\|x\|_1=1} (|\chi_0| \|a_J\|_1 + |\chi_1| \|a_J\|_1 + \cdots + |\chi_{n-1}| \|a_J\|_1) \\
 &= < \text{factor out } \|a_J\|_1 > \\
 & \max_{\|x\|_1=1} (|\chi_0| + |\chi_1| + \cdots + |\chi_{n-1}|) \|a_J\|_1 \\
 &= < \text{algebra} > \\
 & \|a_J\|_1.
 \end{aligned}$$

Also,

$$\begin{aligned}
 & \|a_J\|_1 \\
 &= < e_J \text{ picks out column } J > \\
 & \|Ae_J\|_1 \\
 &\leq < e_J \text{ is a specific choice of } x > \\
 & \max_{\|x\|_1=1} \|Ax\|_1.
 \end{aligned}$$

Hence

$$\|a_J\|_1 \leq \max_{\|x\|_1=1} \|Ax\|_1 \leq \|a_J\|_1$$

which implies that

$$\max_{\|x\|_1=1} \|Ax\|_1 = \|a_J\|_1 = \max_{0 \leq j < n} \|a_j\|_1.$$

Homework 1.3.6.2 Let $A \in \mathbb{C}^{m \times n}$ and partition $A = \begin{pmatrix} \frac{\tilde{a}_0^T}{\tilde{a}_1^T} \\ \vdots \\ \frac{\tilde{a}_{m-1}^T}{} \end{pmatrix}$.

ALWAYS/SOMETIMES/NEVER:

$$\|A\|_\infty = \max_{0 \leq i < m} \|\tilde{a}_i\|_1 (= \max_{0 \leq i < m} (|\alpha_{i,0}| + |\alpha_{i,1}| + \cdots + |\alpha_{i,n-1}|))$$

Notice that in this exercise \tilde{a}_i is really $(\tilde{a}_i^T)^T$ since \tilde{a}_i^T is the label for the i th row of matrix

A.

Hint. Prove it for the real valued case first.

Answer. ALWAYS

Solution. Partition $A = \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix}$. Then

$$\begin{aligned}
& \|A\|_\infty \\
&= <\text{definition}> \\
&\max_{\|x\|_\infty=1} \|Ax\|_\infty \\
&= <\text{expose rows}> \\
&\max_{\|x\|_\infty=1} \left\| \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} x \right\|_\infty \\
&= <\text{matrix-vector multiplication}> \\
&\max_{\|x\|_\infty=1} \left\| \begin{pmatrix} \tilde{a}_0^T x \\ \vdots \\ \tilde{a}_{m-1}^T x \end{pmatrix} \right\|_\infty \\
&= <\text{definition of } \|\cdot\|_\infty> \\
&\max_{\|x\|_\infty=1} \left(\max_{0 \leq i < m} |\tilde{a}_i^T x| \right) \\
&= <\text{expose } \tilde{a}_i^T x> \\
&\max_{\|x\|_\infty=1} \max_{0 \leq i < m} \left| \sum_{p=0}^{n-1} \alpha_{i,p} \chi_p \right| \\
&\leq <\text{triangle inequality}> \\
&\max_{\|x\|_\infty=1} \max_{0 \leq i < m} \sum_{p=0}^{n-1} |\alpha_{i,p} \chi_p| \\
&= <\text{algebra}> \\
&\max_{\|x\|_\infty=1} \max_{0 \leq i < m} \sum_{p=0}^{n-1} (|\alpha_{i,p}| |\chi_p|) \\
&\leq <\text{algebra}> \\
&\max_{\|x\|_\infty=1} \max_{0 \leq i < m} \sum_{p=0}^{n-1} (|\alpha_{i,p}| (\max_k |\chi_k|)) \\
&= <\text{definition of } \|\cdot\|_\infty> \\
&\max_{\|x\|_\infty=1} \max_{0 \leq i < m} \sum_{p=0}^{n-1} (|\alpha_{i,p}| \|x\|_\infty) \\
&= <\|x\|_\infty = 1> \\
&\max_{0 \leq i < m} \sum_{p=0}^{n-1} |\alpha_{i,p}| \\
&= <\text{definition of } \|\cdot\|_1> \\
&\max_{0 \leq i < m} \|\tilde{a}_i\|_1
\end{aligned}$$

so that $\|A\|_\infty \leq \max_{0 \leq i < m} \|\tilde{a}_i\|_1$.

We also want to show that $\|A\|_\infty \geq \max_{0 \leq i < m} \|\tilde{a}_i\|_1$. Let k be such that $\max_{0 \leq i < m} \|\tilde{a}_i\|_1 = \|\tilde{a}_k\|_1$ and pick $y = \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{n-1} \end{pmatrix}$ so that $\tilde{a}_k^T y = |\alpha_{k,0}| + |\alpha_{k,1}| + \cdots + |\alpha_{k,n-1}| = \|\tilde{a}_k\|_1$. (This is a matter of picking $\psi_j = |\alpha_{k,j}|/\alpha_{k,j}$ if $\alpha_{k,j} \neq 0$ and $\psi_j = 1$ otherwise. Then $|\psi_j| = 1$, and

hence $\|y\|_\infty = 1$ and $\psi_j \alpha_{k,j} = |\alpha_{k,j}|$.) Then

$$\begin{aligned}
 & \|A\|_\infty \\
 &= <\text{definition}> \\
 & \max_{\|x\|_1=1} \|Ax\|_\infty \\
 &= <\text{expose rows}> \\
 & \max_{\|x\|_1=1} \left\| \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} x \right\|_\infty \\
 &\geq <\text{y is a specific } x> \\
 & \left\| \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} y \right\|_\infty \\
 &= <\text{matrix-vector multiplication}> \\
 & \left\| \begin{pmatrix} \tilde{a}_0^T y \\ \vdots \\ \tilde{a}_{m-1}^T y \end{pmatrix} \right\|_\infty \\
 &\geq <\text{algebra}> \\
 & |\tilde{a}_k^T y| \\
 &= <\text{choice of } y> \\
 & \|\tilde{a}_k\|_1 \\
 &= <\text{choice of } k> \\
 & \max_{0 \leq i < m} \|\tilde{a}_i\|_1
 \end{aligned}$$

Remark 1.3.6.1 The last homework provides a hint as to how to remember how to compute the matrix 1-norm and ∞ -norm: Since $\|x\|_1$ must result in the same value whether x is considered as a vector or as a matrix, we can remember that the matrix 1-norm equals the maximum of the 1-norms of the columns of the matrix: Similarly, considering $\|x\|_\infty$ as a vector norm or as matrix norm reminds us that the matrix ∞ -norm equals the maximum of the 1-norms of vectors that become the rows of the matrix.

1.3.7 Equivalence of matrix norms



YouTube: <https://www.youtube.com/watch?v=Csqd4AnH7ws>

Homework 1.3.7.1 Fill out the following table:

A	$\ A\ _1$	$\ A\ _\infty$	$\ A\ _F$	$\ A\ _2$
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$				
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$				
$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$				

Hint. For the second and third, you may want to use [Homework 1.3.5.2](#) when computing the 2-norm.

Solution.

A	$\ A\ _1$	$\ A\ _\infty$	$\ A\ _F$	$\ A\ _2$
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	1	1	$\sqrt{3}$	1
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	4	3	$2\sqrt{3}$	$2\sqrt{3}$
$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	3	1	$\sqrt{3}$	$\sqrt{3}$

To compute the 2-norm of I , notice that

$$\|I\|_2 = \max_{\|x\|_2=1} \|Ix\|_2 = \max_{\|x\|_2=1} \|x\|_2 = 1.$$

Next, notice that

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}.$$

and

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}.$$

which allows us to invoke the result from [Homework 1.3.5.2](#).

We saw that vector norms are equivalent in the sense that if a vector is "small" in one

norm, it is "small" in all other norms, and if it is "large" in one norm, it is "large" in all other norms. The same is true for matrix norms.

Theorem 1.3.7.1 Equivalence of matrix norms. *Let $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ and $\|\|\cdot\|\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ both be matrix norms. Then there exist positive scalars σ and τ such that for all $A \in \mathbb{C}^{m \times n}$*

$$\sigma\|A\| \leq \|\|A\|\| \leq \tau\|A\|.$$

Proof. The proof again builds on the fact that the supremum over a compact set is achieved and can be replaced by the maximum.

We will prove that there exists a τ such that for all $A \in \mathbb{C}^{m \times n}$

$$\|\|A\|\| \leq \tau\|A\|$$

leaving the rest of the proof to the reader.

Let $A \in \mathbb{C}^{m \times n}$ be an arbitrary matrix. W.l.o.g. assume that $A \neq 0$ (the zero matrix). Then

$$\begin{aligned} \|\|A\|\| &= \text{algebra} \\ \frac{\|\|A\|\|}{\|A\|} \|A\| &\leq \text{definition of supremum} \\ \left(\sup_{Z \neq 0} \frac{\|\|Z\|\|}{\|Z\|} \right) \|A\| &= \text{homogeneity} \\ \left(\sup_{Z \neq 0} \left| \frac{Z}{\|Z\|} \right| \right) \|A\| &= \text{change of variables } B = Z/\|Z\| \\ \left(\sup_{\|B\|=1} \|\|B\|\| \right) \|A\| &= \text{the set } \|B\| = 1 \text{ is compact} \\ \left(\max_{\|B\|=1} \|\|B\|\| \right) \|A\| \end{aligned}$$

The desired τ can now be chosen to equal $\max_{\|B\|=1} \|\|B\|\|$. ■

Remark 1.3.7.2 The bottom line is that, modulo a constant factor, if a matrix is "small" in one norm, it is "small" in any other norm.

Homework 1.3.7.2 Given $A \in \mathbb{C}^{m \times n}$ show that $\|A\|_2 \leq \|A\|_F$. For what matrix is equality attained?

Hmmm, actually, this is really easy to prove once we know about the SVD... Hard to prove without it. So, this problem will be moved...

Solution. Next week, we will learn about the SVD. Let us go ahead and insert that proof here, for future reference.

Let $A = U\Sigma V^H$ be the Singular Value Decomposition of A , where U and V are unitary and $\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{\min(m,n)})$ with $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{\min(m,n)} \geq 0$. Then

$$\|A\|_2 = \|U\Sigma V^H\|_2 = \sigma_0$$

and

$$\|A\|_F = \|U\Sigma V^H\|_F = \|\Sigma\|_F = \sqrt{\sigma_0^2 + \dots + \sigma_{\min(m,n)}^2}.$$

Hence, $\|A\|_2 \leq \|A\|_F$.

Homework 1.3.7.3 Let $A \in \mathbb{C}^{m \times n}$. The following table summarizes the equivalence of various matrix norms:

	$\ A\ _1 \leq \sqrt{m}\ A\ _2$	$\ A\ _1 \leq m\ A\ _\infty$	$\ A\ _1 \leq \sqrt{m}\ A\ _F$
$\ A\ _2 \leq \sqrt{n}\ A\ _1$		$\ A\ _2 \leq \sqrt{m}\ A\ _\infty$	$\ A\ _2 \leq \ A\ _F$
$\ A\ _\infty \leq n\ A\ _1$	$\ A\ _\infty \leq \sqrt{n}\ A\ _2$		$\ A\ _\infty \leq \sqrt{n}\ A\ _F$
$\ A\ _F \leq \sqrt{n}\ A\ _1$	$\ A\ _F \leq ?\ A\ _2$	$\ A\ _F \leq \sqrt{m}\ A\ _\infty$	

For each, prove the inequality, including that it is a tight inequality for some nonzero A .

(Skip $\|A\|_F \leq ?\|A\|_2$: we will revisit it in Week 2.)

Solution.

- $\|A\|_1 \leq \sqrt{m}\|A\|_2$:

$$\begin{aligned}
 \|A\|_1 &= <\text{definition}> \\
 \max_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} &\leq <\|z\|_1 \leq \sqrt{m}\|z\|_2> \\
 \max_{x \neq 0} \frac{\sqrt{m}\|Ax\|_2}{\|x\|_1} &\leq <\|z\|_1 \geq \|z\|_2> \\
 \max_{x \neq 0} \frac{\sqrt{m}\|Ax\|_2}{\|x\|_2} &= <\text{algebra; definition}> \\
 \sqrt{m}\|A\|_2 &
 \end{aligned}$$

Equality is attained for $A = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$.

- $\|A\|_1 \leq m\|A\|_\infty$:

$$\begin{aligned}
 \|A\|_1 &= <\text{definition}> \\
 \max_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} &\leq <\|z\|_1 \leq m\|z\|_\infty> \\
 \max_{x \neq 0} \frac{m\|Ax\|_\infty}{\|x\|_1} &\leq <\|z\|_1 \geq \|z\|_\infty> \\
 \max_{x \neq 0} \frac{m\|Ax\|_\infty}{\|x\|_\infty} &= <\text{algebra; definition}> \\
 m\|A\|_\infty &
 \end{aligned}$$

Equality is attained for $A = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$.

- $\|A\|_1 \leq \sqrt{m}\|A\|_F$:

It pays to show that $\|A\|_2 \leq \|A\|_F$ first. Then

$$\begin{aligned} & \|A\|_1 \\ & \leq \frac{\|A\|_1}{\sqrt{m}\|A\|_2} < \text{last part} > \\ & \leq \frac{\|A\|_2}{\sqrt{m}\|A\|_F} < \text{some other part: } \|A\|_2 \leq \|A\|_F > \\ & \leq \sqrt{m}\|A\|_F. \end{aligned}$$

Equality is attained for $A = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$.

- $\|A\|_2 \leq \sqrt{n}\|A\|_1$:

$$\begin{aligned} & \|A\|_2 \\ & = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} < \text{definition} > \\ & \leq \max_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_2} < \|z\|_2 \leq \|z\|_1 > \\ & \leq \max_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} < \sqrt{m}\|z\|_2 \geq \|z\|_1 \text{ when } z \text{ is of size } n > \\ & = \sqrt{n}\|A\|_1. < \text{algebra; definition} > \end{aligned}$$

Equality is attained for $A = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}$.

- $\|A\|_2 \leq \sqrt{m}\|A\|_\infty$:

$$\begin{aligned}
 \|A\|_2 &= <\text{definition}> \\
 \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} &\leq <\|z\|_2 \leq \sqrt{m}\|z\|_\infty> \\
 \max_{x \neq 0} \frac{\sqrt{m}\|Ax\|_\infty}{\|x\|_2} &\leq <\|z\|_2 \geq \|z\|_\infty> \\
 \max_{x \neq 0} \frac{\sqrt{m}\|Ax\|_\infty}{\|x\|_\infty} &= <\text{algebra; definition}> \\
 &= \sqrt{m}\|A\|_\infty.
 \end{aligned}$$

Equality is attained for $A = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$.

- $\|A\|_2 \leq \|A\|_F$:
(See Homework 1.3.7.2, which requires the SVD, as mentioned...)
- Please share more solutions!

1.3.8 Submultiplicative norms



YouTube: <https://www.youtube.com/watch?v=TvthvYGt9x8>

There are a number of properties that we would like for a matrix norm to have (but not all norms do have). Recalling that we would like for a matrix norm to measure by how much a vector is "stretched," it would be good if for a given matrix norm, $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$, there are vector norms $\|\cdot\|_\mu : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ such that, for arbitrary nonzero $x \in \mathbb{C}^n$, the matrix norm bounds by how much the vector is stretched:

$$\frac{\|Ax\|_\mu}{\|x\|_\nu} \leq \|A\|$$

or, equivalently,

$$\|Ax\|_\mu \leq \|A\| \|x\|_\nu$$

where this second formulation has the benefit that it also holds if $x = 0$. When this relationship between the involved norms holds, the matrix norm is said to be subordinate to the

vector norms:

Definition 1.3.8.1 Subordinate matrix norm. A matrix norm $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is said to be subordinate to vector norms $\|\cdot\|_\mu : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ if, for all $x \in \mathbb{C}^n$,

$$\|Ax\|_\mu \leq \|A\| \|x\|_\nu.$$

If $\|\cdot\|_\mu$ and $\|\cdot\|_\nu$ are the same norm (but perhaps for different m and n), then $\|\cdot\|$ is said to be subordinate to the given vector norm. \diamond

Fortunately, all the norms that we will employ in this course are subordinate matrix norms.

Homework 1.3.8.1 ALWAYS/SOMETIMES/NEVER: The Frobenius norm is subordinate to the vector 2-norm.

Answer. TRUE

Now prove it.

Solution. W.l.o.g., assume $x \neq 0$.

$$\|Ax\|_2 = \frac{\|Ax\|_2}{\|x\|_2} \|x\|_2 \leq \max_{y \neq 0} \frac{\|Ay\|_2}{\|y\|_2} \|x\|_2 = \max_{\|y\|_2=1} \|Ay\|_2 \|x\|_2 = \|A\|_2 \|x\|_2.$$

So, it suffices to show that $\|A\|_2 \leq \|A\|_F$. But we showed that in [Homework 1.3.7.2](#).

Theorem 1.3.8.2 *Induced matrix norms, $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$, are subordinate to the norms, $\|\cdot\|_\mu$ and $\|\cdot\|_\nu$, that induce them.*

Proof. W.l.o.g. assume $x \neq 0$. Then

$$\|Ax\|_\mu = \frac{\|Ax\|_\mu}{\|x\|_\nu} \|x\|_\nu \leq \max_{y \neq 0} \frac{\|Ay\|_\mu}{\|y\|_\nu} \|x\|_\nu = \|A\|_{\mu,\nu} \|x\|_\nu.$$

■

Corollary 1.3.8.3 *Any matrix p -norm is subordinate to the corresponding vector p -norm.*

Another desirable property that not all norms have is that

$$\|AB\| \leq \|A\| \|B\|.$$

This requires the given norm to be defined for all matrix sizes..

Definition 1.3.8.4 Consistent matrix norm. A matrix norm $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is said to be a consistent matrix norm if it is defined for all m and n , using the same formula for all m and n . \diamond

Obviously, this definition is a bit vague. Fortunately, it is pretty clear that all the matrix norms we will use in this course, the Frobenius norm and the p -norms, are all consistently defined for all matrix sizes.

Definition 1.3.8.5 Submultiplicative matrix norm. A consistent matrix norm $\|\cdot\| :$

$\mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is said to be submultiplicative if it satisfies

$$\|AB\| \leq \|A\|\|B\|.$$

◊

Theorem 1.3.8.6 Let $\|\cdot\| : \mathbb{C}^n \rightarrow \mathbb{R}$ be a vector norm defined for all n . Define the corresponding induced matrix norm as

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|.$$

Then for any $A \in \mathbb{C}^{m \times k}$ and $B \in \mathbb{C}^{k \times n}$ the inequality $\|AB\| \leq \|A\|\|B\|$ holds.

In other words, induced matrix norms are submultiplicative. To prove this theorem, it helps to first prove a simpler result:

Lemma 1.3.8.7 Let $\|\cdot\| : \mathbb{C}^n \rightarrow \mathbb{R}$ be a vector norm defined for all n and let $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ be the matrix norm it induces. Then $\|Ax\| \leq \|A\|\|x\|$.

Proof. If $x = 0$, the result obviously holds since then $\|Ax\| = 0$ and $\|x\| = 0$. Let $x \neq 0$. Then

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} \geq \frac{\|Ax\|}{\|x\|}.$$

Rearranging this yields $\|Ax\| \leq \|A\|\|x\|$. ■

We can now prove the theorem:

Proof.

$$\begin{aligned} & \|AB\| \\ &= < \text{definition of induced matrix norm} > \\ & \max_{\|x\|=1} \|ABx\| \\ &= < \text{associativity} > \\ & \max_{\|x\|=1} \|A(Bx)\| \\ &\leq < \text{lemma} > \\ & \max_{\|x\|=1} (\|A\|\|Bx\|) \\ &\leq < \text{lemma} > \\ & \max_{\|x\|=1} (\|A\|\|B\|\|x\|) \\ &= < \|x\| = 1 > \\ & \|A\|\|B\|. \end{aligned}$$

■

Homework 1.3.8.2 Show that $\|Ax\|_\mu \leq \|A\|_{\mu,\nu}\|x\|_\nu$.

Solution. W.l.o.g. assume that $x \neq 0$.

$$\|A\|_{\mu,\nu} = \max_{y \neq 0} \frac{\|Ay\|_\mu}{\|y\|_\nu} \geq \frac{\|Ax\|_\mu}{\|x\|_\nu}.$$

Rearranging this establishes the result.

Homework 1.3.8.3 Show that $\|AB\|_\mu \leq \|A\|_{\mu,\nu} \|B\|_{\nu,\mu}$.

Solution.

$$\begin{aligned}
 & \|AB\|_\mu \\
 &= < \text{definition} > \\
 &\max_{\|x\|_\mu=1} \|ABx\|_\mu \\
 &\leq < \text{last homework} > \\
 &\max_{\|x\|_\mu=1} \|A\|_{\mu,\nu} \|Bx\|_\nu \\
 &= < \text{algebra} > \\
 &\|A\|_{\mu,\nu} \max_{\|x\|_\mu=1} \|Bx\|_\nu \\
 &= < \text{definition} > \\
 &\|A\|_{\mu,\nu} \|B\|_{\nu,\mu}
 \end{aligned}$$

Homework 1.3.8.4 Show that the Frobenius norm, $\|\cdot\|_F$, is submultiplicative.

Solution.

$$\begin{aligned}
 & \|AB\|_F^2 \\
 &= < \text{partition} > \\
 &\left\| \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} \left(b_0 \mid b_1 \mid \cdots \mid b_{n-1} \right) \right\|_F^2 \\
 &= < \text{partitioned matrix-matrix multiplication} > \\
 &\left\| \begin{pmatrix} \tilde{a}_0^T b_0 & \tilde{a}_0^T b_1 & \cdots & \tilde{a}_0^T b_{n-1} \\ \hline \tilde{a}_0^T b_0 & \tilde{a}_0^T b_1 & \cdots & \tilde{a}_0^T b_{n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline \tilde{a}_{m-1}^T b_0 & \tilde{a}_{m-1}^T b_1 & \cdots & \tilde{a}_{m-1}^T b_{n-1} \end{pmatrix} \right\|_F^2 \\
 &= < \text{definition of Frobenius norm} > \\
 &\sum_i \sum_j |\tilde{a}_i^T b_j|^2 \\
 &= < \text{definition of Hermitian transpose vs transpose} > \\
 &\sum_i \sum_j |\tilde{a}_i^H b_j|^2 \\
 &\leq < \text{Cauchy-Schwarz inequality} > \\
 &\sum_i \sum_j \|\tilde{a}_i\|_2^2 \|b_j\|_2^2 \\
 &= < \text{algebra and } \|\bar{x}\|_2 = \|x\|_2 > \\
 &(\sum_i \|\tilde{a}_i\|_2^2) (\sum_j \|b_j\|_2^2) \\
 &= < \text{previous observations about the Frobenius norm} > \\
 &\|A\|_F^2 \|B\|_F^2
 \end{aligned}$$

Hence $\|AB\|_F^2 \leq \|A\|_F^2 \|B\|_F^2$. Taking the square root of both sides leaves us with $\|AB\|_F \leq \|A\|_F \|B\|_F$.

This proof brings to the forefront that the notation \tilde{a}_i^T leads to some possible confusion. In this particular situation, it is best to think of \tilde{a}_i as a vector that, when transposed, becomes the row of A indexed with i . In this case, $\tilde{a}_i^T = \bar{\tilde{a}}_i^H$ and $(\tilde{a}_i^T)^H = \bar{\tilde{a}}_i$ (where, recall, \bar{x} equals the vector with all its entries conjugated). Perhaps it is best to just work through

this problem for the case where A and B are real-valued, and not worry too much about the details related to the complex-valued case...

Homework 1.3.8.5 For $A \in \mathbb{C}^{m \times n}$ define

$$\|A\| = \max_{i=0}^{m-1} \max_{j=0}^{n-1} |\alpha_{i,j}|.$$

1. TRUE/FALSE: This is a norm.
2. TRUE/FALSE: This is a consistent norm.

Answer.

1. TRUE
2. TRUE

Solution.

1. This is a norm. You can prove this by checking the three conditions.
2. It is a consistent norm since it is defined for all m and n .

Remark 1.3.8.8 The important take-away: The norms we tend to use in this course, the p -norms and the Frobenius norm, are all submultiplicative.

Homework 1.3.8.6 Let $A \in \mathbb{C}^{m \times n}$.

ALWAYS/SOMETIMES/NEVER: There exists a vector

$$x = \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \text{ with } |\chi_i| = 1 \text{ for } i = 0, \dots, n-1$$

such that $\|A\|_\infty = \|Ax\|_\infty$.

Answer. ALWAYS

Now prove it!

Solution. Partition A by rows:

$$A = \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix}.$$

We know that there exists k such that $\|\tilde{a}_k\|_1 = \|A\|_\infty$. Now

$$\begin{aligned} \|\tilde{a}_k\|_1 &= <\text{definition of 1-norm}> \\ |\alpha_{k,0}| + \cdots + |\alpha_{k,n-1}| &= <\text{algebra}> \\ \frac{|\alpha_{k,0}|}{\alpha_{k,0}} \alpha_{k,0} + \cdots + \frac{|\alpha_{k,n-1}|}{\alpha_{k,n-1}} \alpha_{k,n-1}. \end{aligned}$$

where we take $\frac{|\alpha_{k,j}|}{\alpha_{k,j}} = 1$ whenever $\alpha_{k,j} = 0$. Vector

$$x = \begin{pmatrix} \frac{|\alpha_{k,0}|}{\alpha_{k,0}} \\ \vdots \\ \frac{|\alpha_{k,n-1}|}{\alpha_{k,n-1}} \end{pmatrix}$$

has the desired property.

1.3.9 Summary



YouTube: <https://www.youtube.com/watch?v=DyoT2tJhxIs>

1.4 Condition Number of a Matrix

1.4.1 Conditioning of a linear system



YouTube: <https://www.youtube.com/watch?v=QwFQNAPKIwk>

A question we will run into later in the course asks how accurate we can expect the solution of a linear system to be if the right-hand side of the system has error in it.

Formally, this can be stated as follows: We wish to solve $Ax = b$, where $A \in \mathbb{C}^{m \times m}$ but the right-hand side has been perturbed by a small vector so that it becomes $b + \delta b$.

Remark 1.4.1.1 Notice how the δ touches the b . This is meant to convey that this is a symbol that represents a vector rather than the vector b that is multiplied by a scalar δ .

The question now is how a relative error in b is amplified into a relative error in the solution x .

This is summarized as follows:

$$\begin{array}{ll} Ax = b & \text{exact equation} \\ A(x + \delta x) = b + \delta b & \text{perturbed equation} \end{array}$$

We would like to determine a formula, $\kappa(A, b, \delta b)$, that gives us a bound on how much a relative error in b is potentially amplified into a relative error in the solution x :

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A, b, \delta b) \frac{\|\delta b\|}{\|b\|}.$$

We assume that A has an inverse since otherwise there may be no solution or there may be an infinite number of solutions. To find an expression for $\kappa(A, b, \delta b)$, we notice that

$$\begin{array}{rcl} Ax + A\delta x & = & b + \delta b \\ Ax & = & b - \\ \hline A\delta x & = & \delta b \end{array}$$

and from this we deduce that

$$\begin{array}{rcl} Ax & = & b \\ \delta x & = & A^{-1}\delta b. \end{array}$$

If we now use a vector norm $\|\cdot\|$ and its induced matrix norm $\|\cdot\|$, then

$$\begin{array}{rcl} \|b\| & = & \|Ax\| \leq \|A\|\|x\| \\ \|\delta x\| & = & \|A^{-1}\delta b\| \leq \|A^{-1}\|\|\delta b\| \end{array}$$

since induced matrix norms are subordinate.

From this we conclude that

$$\frac{1}{\|x\|} \leq \|A\| \frac{1}{\|b\|}$$

and

$$\|\delta x\| \leq \|A^{-1}\|\|\delta b\|$$

so that

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\|\|A^{-1}\| \frac{\|\delta b\|}{\|b\|}.$$

Thus, the desired expression $\kappa(A, b, \delta b)$ doesn't depend on anything but the matrix A :

$$\frac{\|\delta x\|}{\|x\|} \leq \underbrace{\|A\|\|A^{-1}\|}_{\kappa(A)} \frac{\|\delta b\|}{\|b\|}.$$

Definition 1.4.1.2 Condition number of a nonsingular matrix. The value $\kappa(A) = \|A\|\|A^{-1}\|$ is called the condition number of a nonsingular matrix A . \diamond

A question becomes whether this is a pessimistic result or whether there are examples of b and δb for which the relative error in b is amplified by exactly $\kappa(A)$. The answer is that, unfortunately, the bound is tight.

- There is an \hat{x} for which

$$\|A\| = \max_{\|x\|=1} \|Ax\| = \|A\hat{x}\|,$$

namely the x for which the maximum is attained. This is the direction of maximal magnification. Pick $\hat{b} = A\hat{x}$.

- There is an $\hat{\delta}$ for which

$$\|A^{-1}\| = \max_{\|x\|\neq 0} \frac{\|A^{-1}x\|}{\|x\|} = \frac{\|A^{-1}\hat{\delta}\|}{\|\hat{\delta}\|},$$

again, the x for which the maximum is attained.

It is when solving the perturbed system

$$A(x + \delta x) = \hat{b} + \hat{\delta}$$

that the maximal magnification by $\kappa(A)$ is observed.

Homework 1.4.1.1 Let $\|\cdot\|$ be a vector norm and corresponding induced matrix norm.

TRUE/FALSE: $\|I\| = 1$.

Answer. TRUE

Solution.

$$\|I\| = \max_{\|x\|=1} \|Ix\| = \max_{\|x\|=1} \|x\| = 1$$

Homework 1.4.1.2 Let $\|\cdot\|$ be a vector norm and corresponding induced matrix norm, and A a nonsingular matrix.

TRUE/FALSE: $\kappa(A) = \|A\|\|A^{-1}\| \geq 1$.

Answer. TRUE

Solution.

$$\begin{aligned} 1 &= <\text{last homework}> \\ \|I\| &= < A \text{ is invertible} > \\ \|AA^{-1}\| &\leq <\|\cdot\| \text{ is submultiplicative}> \\ \|A\|\|A^{-1}\|. \end{aligned}$$

Remark 1.4.1.3 This last exercise shows that there will always be choices for b and δ for which the relative error is at best directly translated into an equal relative error in the solution (if $\kappa(A) = 1$).

1.4.2 Loss of digits of accuracy



YouTube: <https://www.youtube.com/watch?v=-5l90v5RXYo>

Homework 1.4.2.1 Let $\alpha = -14.24123$ and $\hat{\alpha} = -14.24723$. Compute

- $|\alpha| =$
- $|\alpha - \hat{\alpha}| =$
- $\frac{|\alpha - \hat{\alpha}|}{|\alpha|} =$
- $\log_{10} \left(\frac{|\alpha - \hat{\alpha}|}{|\alpha|} \right) =$

Solution. Let $\alpha = -14.24123$ and $\hat{\alpha} = -14.24723$. Compute

- $|\alpha| = 14.24123$
- $|\alpha - \hat{\alpha}| = 0.006$
- $\frac{|\alpha - \hat{\alpha}|}{|\alpha|} \approx 0.00042$
- $\log_{10} \left(\frac{|\alpha - \hat{\alpha}|}{|\alpha|} \right) \approx -3.4$

The point of this exercise is as follows:

- If you compare $\alpha = -14.24123$ and $\hat{\alpha} = -14.24723$ and you consider $\hat{\alpha}$ to be an approximation of α , then $\hat{\alpha}$ is accurate to four digits: -14.24 is accurate.
- Computing $\log_{10} \left(\frac{|\alpha - \hat{\alpha}|}{|\alpha|} \right)$ tells you approximately how many decimal digits are accurate: 3.4 digits.

Be sure to read the solution to the last homework!

1.4.3 The conditioning of an upper triangular matrix



YouTube: <https://www.youtube.com/watch?v=LGBFyjhjt6U>

We now revisit the material from the launch for the semester. We understand that when solving $Lx = b$, even a small relative change to the right-hand side b can amplify into a large relative change in the solution \hat{x} if the condition number of the matrix is large.

Homework 1.4.3.1 Change the script [Assignments/Week01/matlab/Test_Upper_triangular_solve_100.m](#) to also compute the condition number of matrix U , $\kappa(U)$. Investigate what happens to the condition number as you change the problem size n .

Since in the example the upper triangular matrix is generated to have random values as its entries, chances are that at least one element on its diagonal is small. If that element were zero, then the triangular matrix would be singular. Even if it is not exactly zero, the condition number of U becomes very large if the element is small.

1.5 Enrichments

1.5.1 Condition number estimation

It has been observed that high-quality numerical software should not only provide routines for solving a given problem, but, when possible, should also (optionally) provide the user with feedback on the conditioning (sensitivity to changes in the input) of the problem. In this enrichment, we relate this to what you have learned this week.

Given a vector norm $\|\cdot\|$ and induced matrix norm $\|\cdot\|$, the condition number of matrix A using that norm is given by $\kappa(A) = \|A\| \|A^{-1}\|$. When trying to practically compute the condition number, this leads to two issues:

- Which norm should we use? A case has been made in this week that the 1-norm and ∞ -norm are candidates since they are easy and cheap to compute.
- It appears that A^{-1} needs to be computed. We will see in future weeks that this is costly: $O(m^3)$ computation when A is $m \times m$. This is generally considered to be expensive.

This leads to the question "Can a reliable estimate of the condition number be cheaply computed?" In this unit, we give a glimpse of how this can be achieved and then point the interested learner to related papers.

Partition $m \times m$ matrix A :

$$A = \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix}.$$

We recall that

- The ∞ -norm is defined by

$$\|A\|_\infty = \max_{\|x\|_\infty=1} \|Ax\|_\infty.$$

- From [Homework 1.3.6.2](#), we know that the ∞ -norm can be practically computed as

$$\|A\|_\infty = \max_{0 \leq i < m} \|\tilde{a}_i\|_1,$$

where $\tilde{a}_i = (\tilde{a}_i^T)^T$. This means that $\|A\|_\infty$ can be computed in $O(m^2)$ operations.

- From the solution to [Homework 1.3.6.2](#), we know that there is a vector x with $|\chi_j| = 1$ for $0 \leq j < m$ such that $\|A\|_\infty = \|Ax\|_\infty$. This x satisfies $\|x\|_\infty = 1$.

More precisely: $\|A\|_\infty = \|\tilde{a}_k^T\|_1$ for some k . For simplicity, assume A is real valued. Then

$$\begin{aligned} \|A\|_\infty &= |\alpha_{k,0}| + \cdots + |\alpha_{k,m-1}| \\ &= \alpha_{k,0}\chi_0 + \cdots + \alpha_{k,m-1}\chi_{m-1}, \end{aligned}$$

where each $\chi_j = \pm 1$ is chosen so that $\chi_j \alpha_{k,j} = |\alpha_{k,j}|$. That vector x then has the property that $\|A\|_\infty = \|\tilde{a}_k\|_1 = \|Ax\|_\infty$.

From this we conclude that

$$\|A\|_\infty = \max_{x \in \mathcal{S}} \|Ax\|_\infty,$$

where \mathcal{S} is the set of all vectors x with $|\chi_j| = 1$, $0 \leq j < n$.

We will illustrate the techniques that underly efficient condition number estimation by looking at the simpler case where we wish to estimate the condition number of a *real-valued* nonsingular upper triangular $m \times m$ matrix U , using the ∞ -norm. Since U is real-valued, $|\chi_i| = 1$ means $\chi_i = \pm 1$. The problem is that it appears we must compute $\|U^{-1}\|_\infty$. Computing U^{-1} when U is dense requires $O(m^3)$ operations (a topic we won't touch on until much later in the course).

Our observations tell us that

$$\|U^{-1}\|_\infty = \max_{x \in \mathcal{S}} \|U^{-1}x\|_\infty,$$

where \mathcal{S} is the set of all vectors x with elements $\chi_i \in \{-1, 1\}$. This is equivalent to

$$\|U^{-1}\|_\infty = \max_{z \in \mathcal{T}} \|z\|_\infty,$$

where \mathcal{T} is the set of all vectors z that satisfy $Uz = y$ for some y with elements $\psi_i \in \{-1, 1\}$. So, we could solve $Uz = y$ for all vectors $y \in \mathcal{S}$, compute the ∞ -norm for all those vectors z , and pick the maximum of those values. But that is not practical.

One simple solution is to try to construct a vector y that results in a large amplification (in the ∞ -norm) when solving $Uz = y$, and to then use that amplification as an estimate for $\|U^{-1}\|_\infty$. So how do we do this? Consider

$$\underbrace{\begin{pmatrix} \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & v_{m-2,m-2} & v_{m-2,m-1} \\ 0 & \cdots & 0 & v_{m-1,m-1} \end{pmatrix}}_U \underbrace{\begin{pmatrix} \vdots \\ \zeta_{m-2} \\ \zeta_{m-1} \end{pmatrix}}_z = \underbrace{\begin{pmatrix} \vdots \\ \psi_{m-2} \\ \psi_{m-1} \end{pmatrix}}_y.$$

Here is a *heuristic* for picking $y \in \mathcal{S}$:

- We want to pick $\psi_{m-1} \in \{-1, 1\}$ in order to construct a vector $y \in \mathcal{S}$. We can pick $\psi_{m-1} = 1$ since picking it equal to -1 will simply carry through negation in the appropriate way in the scheme we are describing.

From this ψ_{m-1} we can compute ζ_{m-1} .

- Now,

$$v_{m-2,m-2}\zeta_{m-2} + v_{m-2,m-1}\zeta_{m-1} = \psi_{m-2}$$

where ζ_{m-1} is known and ψ_{m-2} can be strategically chosen. We want z to have a large ∞ -norm and hence a *heuristic* is to now pick $\psi_{m-2} \in \{-1, 1\}$ in such a way that ζ_{m-2} is as large as possible in magnitude.

With this ψ_{m-2} we can compute ζ_{m-2} .

- And so forth!

When done, the magnification equals $\|z\|_\infty = |\zeta_k|$, where ζ_k is the element of z with largest magnitude. This approach provides an estimate for $\|U^{-1}\|_\infty$ with $O(m^2)$ operations.

The described method underlies the condition number estimator for LINPACK, developed in the 1970s [16], as described in [11]:

- A.K. Cline, C.B. Moler, G.W. Stewart, and J.H. Wilkinson, An estimate for the condition number of a matrix, SIAM J. Numer. Anal., 16 (1979).

The method discussed in that paper yields a lower bound on $\|A^{-1}\|_\infty$ and with that on $\kappa_\infty(A)$.

Remark 1.5.1.1 Alan Cline has his office on our floor at UT-Austin. G.W. (Pete) Stewart was Robert's Ph.D. advisor. Cleve Moler is the inventor of Matlab. John Wilkinson received the Turing Award for his contributions to numerical linear algebra.

More sophisticated methods are discussed in [21]:

- N. Higham, A Survey of Condition Number Estimates for Triangular Matrices, SIAM Review, 1987.

His methods underlie the LAPACK [1] condition number estimator and are remarkably accurate: most of the time they provide an almost exact estimate of the actual condition number.

1.6 Wrap Up

1.6.1 Additional homework

Homework 1.6.1.1 For $e_j \in \mathbb{R}^n$ (a standard basis vector), compute

- $\|e_j\|_2 =$
- $\|e_j\|_1 =$
- $\|e_j\|_\infty =$
- $\|e_j\|_p =$

Homework 1.6.1.2 For $I \in \mathbb{R}^{n \times n}$ (the identity matrix), compute

- $\|I\|_1 =$
- $\|I\|_\infty =$
- $\|I\|_2 =$
- $\|I\|_p =$
- $\|I\|_F =$

Homework 1.6.1.3 Let $D = \begin{pmatrix} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & \delta_{n-1} \end{pmatrix}$ (a diagonal matrix). Compute

- $\|D\|_1 =$
- $\|D\|_\infty =$
- $\|D\|_p =$
- $\|D\|_F =$

Homework 1.6.1.4 Let $x = \begin{pmatrix} \frac{x_0}{\cdot} \\ \frac{x_1}{\cdot} \\ \vdots \\ \frac{x_{N-1}}{\cdot} \end{pmatrix}$ and $1 \leq p < \infty$ or $p = \infty$.

ALWAYS/SOMETIMES/NEVER: $\|x_i\|_p \leq \|x\|_p$.

Homework 1.6.1.5 For

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

compute

- $\|A\|_1 =$
- $\|A\|_\infty =$
- $\|A\|_F =$

Homework 1.6.1.6 For $A \in \mathbb{C}^{m \times n}$ define

$$\|A\| = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}| = \sum \begin{pmatrix} |\alpha_{0,0}|, & \cdots, & |\alpha_{0,n-1}|, \\ \vdots & & \vdots \\ |\alpha_{m-1,0}|, & \cdots, & |\alpha_{m-1,n-1}| \end{pmatrix}.$$

- TRUE/FALSE: This function is a matrix norm.
- How can you relate this norm to the vector 1-norm?
- TRUE/FALSE: For this norm, $\|A\| = \|A^H\|$.
- TRUE/FALSE: This norm is submultiplicative.

Homework 1.6.1.7 Let $A \in \mathbb{C}^{m \times n}$. Partition

$$A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) = \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix}.$$

Prove that

- $\|A\|_F = \|A^T\|_F$.
- $\|A\|_F = \sqrt{\|a_0\|_2^2 + \|a_1\|_2^2 + \cdots + \|a_{n-1}\|_2^2}$.
- $\|A\|_F = \sqrt{\|\tilde{a}_0\|_2^2 + \|\tilde{a}_1\|_2^2 + \cdots + \|\tilde{a}_{m-1}\|_2^2}$.

Note that here $\tilde{a}_i = (\tilde{a}_i^T)^T$.

Homework 1.6.1.8 Let $x \in \mathbb{R}^m$ with $\|x\|_1 = 1$.

TRUE/FALSE: $\|x\|_2 = 1$ if and only if $x = \pm e_j$ for some j .

Solution. Obviously, if $x = e_j$ then $\|x\|_1 = \|x\|_2 = 1$.

Assume $x \neq e_j$. Then $|\chi_i| < 1$ for all i . But then $\|x\|_2 = \sqrt{|\chi_0|^2 + \cdots + |\chi_{m-1}|^2} < \sqrt{|\chi_0| + \cdots + |\chi_{m-1}|} = \sqrt{1} = 1$.

Homework 1.6.1.9 Prove that if $\|x\|_\nu \leq \beta \|x\|_\mu$ is true for all x , then $\|A\|_\nu \leq \beta \|A\|_{\mu,\nu}$.

1.6.2 Summary

If $\alpha, \beta \in \mathbb{C}$ with $\alpha = \alpha_r + \alpha_c i$ and $\beta = \beta_r + i\beta_c$, where $\alpha_r, \alpha_c, \beta_r, \beta_c \in \mathbb{R}$, then

- Conjugate: $\bar{\alpha} = \alpha_r - \alpha_c i$.
- Product: $\alpha\beta = (\alpha_r\beta_r - \alpha_c\beta_c) + (\alpha_r\beta_c + \alpha_c\beta_r)i$.
- Absolute value: $|\alpha| = \sqrt{\alpha_r^2 + \alpha_c^2} = \sqrt{\bar{\alpha}\alpha}$.

Let $x, y \in \mathbb{C}^m$ with $x = \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{m-1} \end{pmatrix}$ and $y = \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{m-1} \end{pmatrix}$. Then

- Conjugate:

$$\bar{x} = \begin{pmatrix} \bar{\chi}_0 \\ \vdots \\ \bar{\chi}_{m-1} \end{pmatrix}.$$

- Transpose of vector:

$$x^T = \begin{pmatrix} \chi_0 & \cdots & \chi_{m-1} \end{pmatrix}$$

- Hermitian transpose (conjugate transpose) of vector:

$$x^H = \bar{x}^T = \overline{x^T} = \begin{pmatrix} \bar{\chi}_0 & \cdots & \bar{\chi}_{m-1} \end{pmatrix}.$$

- Dot product (inner product): $x^H y = \bar{x}^T y = \overline{x^T y} = \bar{\chi}_0\psi_0 + \cdots + \bar{\chi}_{m-1}\psi_{m-1} = \sum_{i=0}^{m-1} \bar{\chi}_i\psi_i$.

Definition 1.6.2.1 Vector norm. Let $\|\cdot\| : \mathbb{C}^m \rightarrow \mathbb{R}$. Then $\|\cdot\|$ is a (vector) norm if for all $x, y \in \mathbb{C}^m$ and all $\alpha \in \mathbb{C}$

- $x \neq 0 \Rightarrow \|x\| > 0$ ($\|\cdot\|$ is positive definite),
- $\|\alpha x\| = |\alpha|\|x\|$ ($\|\cdot\|$ is homogeneous), and
- $\|x + y\| \leq \|x\| + \|y\|$ ($\|\cdot\|$ obeys the triangle inequality).

◊

- 2-norm (Euclidean length): $\|x\|_2 = \sqrt{x^H x} = \sqrt{|\chi_0|^2 + \cdots + |\chi_{m-1}|^2} = \sqrt{\bar{\chi}_0\chi_0 + \cdots + \bar{\chi}_{m-1}\chi_{m-1}} = \sqrt{\sum_{i=0}^{m-1} |\chi_i|^2}$.
- p -norm: $\|x\|_p = \sqrt[p]{|\chi_0|^p + \cdots + |\chi_{m-1}|^p} = \sqrt[p]{\sum_{i=0}^{m-1} |\chi_i|^p}$.
- 1-norm: $\|x\|_1 = |\chi_0| + \cdots + |\chi_{m-1}| = \sum_{i=0}^{m-1} |\chi_i|$.
- ∞ -norm: $\|x\|_\infty = \max(|\chi_0|, \dots, |\chi_{m-1}|) = \max_{i=0}^{m-1} |\chi_i| = \lim_{p \rightarrow \infty} \|x\|_p$.
- Unit ball: Set of all vectors with norm equal to one. Notation: $\|x\| = 1$.

Theorem 1.6.2.2 Equivalence of vector norms. Let $\|\cdot\| : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\|\cdot\|\| : \mathbb{C}^m \rightarrow \mathbb{R}$ both be vector norms. Then there exist positive scalars σ and τ such that for all $x \in \mathbb{C}^m$

$$\begin{array}{c} \sigma\|x\| \leq \|\|x\|\| \leq \tau\|x\|. \\ \|x\|_1 \leq \sqrt{m}\|x\|_2 \quad \|x\|_1 \leq m\|x\|_\infty \\ \hline \|x\|_2 \leq \|x\|_1 \quad \|x\|_2 \leq \sqrt{m}\|x\|_\infty \\ \|x\|_\infty \leq \|x\|_1 \quad \|x\|_\infty \leq \|x\|_2 \end{array}$$

Definition 1.6.2.3 Linear transformations and matrices. Let $L : \mathbb{C}^n \rightarrow \mathbb{C}^m$. Then L is said to be a linear transformation if for all $\alpha \in \mathbb{C}$ and $x, y \in \mathbb{C}^n$

- $L(\alpha x) = \alpha L(x)$. That is, scaling first and then transforming yields the same result as transforming first and then scaling.
- $L(x + y) = L(x) + L(y)$. That is, adding first and then transforming yields the same result as transforming first and then adding.

◊

Definition 1.6.2.4 Standard basis vector. In this course, we will use $e_j \in \mathbb{C}^m$ to denote the standard basis vector with a "1" in the position indexed with j . So,

$$e_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow j$$

◊

If L is a linear transformation and we let $a_j = L(e_j)$ then

$$A = (a_0 \mid a_1 \mid \cdots \mid a_{n-1})$$

is the matrix that represents L in the sense that $Ax = L(x)$.

Partition C , A , and B by rows and columns

$$C = (c_0 \mid \cdots \mid c_{n-1}) = \begin{pmatrix} \tilde{c}_0^T \\ \vdots \\ \tilde{c}_{m-1}^T \end{pmatrix}, A = (a_0 \mid \cdots \mid a_{k-1}) = \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix},$$

and

$$B = (b_0 \mid \cdots \mid b_{n-1}) = \begin{pmatrix} \tilde{b}_0^T \\ \vdots \\ \tilde{b}_{k-1}^T \end{pmatrix},$$

then $C := AB$ can be computed in the following ways:

1. By columns:

$$\left(\begin{array}{c|c|c} c_0 & \cdots & c_{n-1} \end{array} \right) := A \left(\begin{array}{c|c|c} b_0 & \cdots & b_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c} Ab_0 & \cdots & Ab_{n-1} \end{array} \right).$$

In other words, $c_j := Ab_j$ for all columns of C .

2. By rows:

$$\left(\begin{array}{c} \tilde{c}_0^T \\ \vdots \\ \tilde{c}_{m-1}^T \end{array} \right) := \left(\begin{array}{c} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{array} \right) B = \left(\begin{array}{c} \tilde{a}_0^T B \\ \vdots \\ \tilde{a}_{m-1}^T B \end{array} \right).$$

In other words, $\tilde{c}_i^T = \tilde{a}_i^T B$ for all rows of C .

3. As the sum of outer products:

$$C := \left(\begin{array}{c|c|c} a_0 & \cdots & a_{k-1} \end{array} \right) \left(\begin{array}{c} \tilde{b}_0^T \\ \vdots \\ \tilde{b}_{k-1}^T \end{array} \right) = a_0 \tilde{b}_0^T + \cdots + a_{k-1} \tilde{b}_{k-1}^T,$$

which should be thought of as a sequence of rank-1 updates, since each term is an outer product and an outer product has rank of at most one.

Partition C , A , and B by blocks (submatrices),

$$C = \left(\begin{array}{c|c|c} C_{0,0} & \cdots & C_{0,N-1} \\ \vdots & & \vdots \\ C_{M-1,0} & \cdots & C_{M-1,N-1} \end{array} \right), \quad \left(\begin{array}{c|c|c} A_{0,0} & \cdots & A_{0,K-1} \\ \vdots & & \vdots \\ A_{M-1,0} & \cdots & A_{M-1,K-1} \end{array} \right),$$

and

$$\left(\begin{array}{c|c|c} B_{0,0} & \cdots & B_{0,N-1} \\ \vdots & & \vdots \\ B_{K-1,0} & \cdots & B_{K-1,N-1} \end{array} \right),$$

where the partitionings are "conformal." Then

$$C_{i,j} = \sum_{p=0}^{K-1} A_{i,p} B_{p,j}.$$

Definition 1.6.2.5 Matrix norm. Let $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$. Then $\|\cdot\|$ is a (matrix) norm if for all $A, B \in \mathbb{C}^{m \times n}$ and all $\alpha \in \mathbb{C}$

- $A \neq 0 \Rightarrow \|A\| > 0$ ($\|\cdot\|$ is positive definite),
- $\|\alpha A\| = |\alpha| \|A\|$ ($\|\cdot\|$ is homogeneous), and
- $\|A + B\| \leq \|A\| + \|B\|$ ($\|\cdot\|$ obeys the triangle inequality).



Let $A \in \mathbb{C}^{m \times n}$ and

$$A = \begin{pmatrix} \alpha_{0,0} & \cdots & \alpha_{0,n-1} \\ \vdots & & \vdots \\ \alpha_{m-1,0} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right) = \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix}.$$

Then

- Conjugate of matrix:

$$\bar{A} = \begin{pmatrix} \bar{\alpha}_{0,0} & \cdots & \bar{\alpha}_{0,n-1} \\ \vdots & \vdots & \vdots \\ \bar{\alpha}_{m-1,0} & \cdots & \bar{\alpha}_{m-1,n-1} \end{pmatrix}.$$

- Transpose of matrix:

$$A^T = \begin{pmatrix} \alpha_{0,0} & \cdots & \alpha_{m-1,0} \\ \vdots & \vdots & \vdots \\ \alpha_{0,n-1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix}.$$

- Conjugate transpose (Hermitian transpose) of matrix:

$$A^H = \bar{A}^T = \bar{A}^T = \begin{pmatrix} \bar{\alpha}_{0,0} & \cdots & \bar{\alpha}_{m-1,0} \\ \vdots & \vdots & \vdots \\ \bar{\alpha}_{0,n-1} & \cdots & \bar{\alpha}_{m-1,n-1} \end{pmatrix}.$$

- Frobenius norm: $\|A\|_F = \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2} = \sqrt{\sum_{j=0}^{n-1} \|a_j\|_2^2} = \sqrt{\sum_{i=0}^{m-1} \|\tilde{a}_i\|_2^2}$
- matrix p-norm: $\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p=1} \|Ax\|_p$.
- matrix 2-norm: $\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2=1} \|Ax\|_2 = \|A^H\|_2$.
- matrix 1-norm: $\|A\|_1 = \max_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{\|x\|_1=1} \|Ax\|_1 = \max_{0 \leq j < n} \|a_j\|_1 = \|A^H\|_\infty$.
- matrix ∞ -norm: $\|A\|_\infty = \max_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{\|x\|_\infty=1} \|Ax\|_\infty = \max_{0 \leq i < m} \|\tilde{a}_i\|_1 = \|A^H\|_1$.

Theorem 1.6.2.6 Equivalence of matrix norms. Let $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ and $\|\|\cdot\|\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ both be matrix norms. Then there exist positive scalars σ and τ such that for all $A \in \mathbb{C}^{m \times n}$

$\sigma\ A\ \leq \ \ A\ \ \leq \tau\ A\ $.	$\ A\ _1 \leq \sqrt{m}\ A\ _2$	$\ A\ _1 \leq m\ A\ _\infty$	$\ A\ _1 \leq \sqrt{m}\ A\ _F$
$\ A\ _2 \leq \sqrt{n}\ A\ _1$		$\ A\ _2 \leq \sqrt{m}\ A\ _\infty$	$\ A\ _2 \leq \ A\ _F$
$\ A\ _\infty \leq n\ A\ _1$	$\ A\ _\infty \leq \sqrt{n}\ A\ _2$		$\ A\ _\infty \leq \sqrt{n}\ A\ _F$
$\ A\ _F \leq \sqrt{n}\ A\ _1$	$\ A\ _F \leq \text{rank}(A)\ A\ _2$	$\ A\ _F \leq \sqrt{m}\ A\ _\infty$	

Definition 1.6.2.7 Subordinate matrix norm. A matrix norm $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is said to be subordinate to vector norms $\|\cdot\|_\mu : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ if, for all $x \in \mathbb{C}^n$,

$$\|Ax\|_\mu \leq \|A\| \|x\|_\nu.$$

If $\|\cdot\|_\mu$ and $\|\cdot\|_\nu$ are the same norm (but perhaps for different m and n), then $\|\cdot\|$ is said to be subordinate to the given vector norm. \diamond

Definition 1.6.2.8 Consistent matrix norm. A matrix norm $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is said to be a consistent matrix norm if it is defined for all m and n , using the same formula for all m and n . \diamond

Definition 1.6.2.9 Submultiplicative matrix norm. A consistent matrix norm $\|\cdot\| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is said to be submultiplicative if it satisfies

$$\|AB\| \leq \|A\| \|B\|.$$

\diamond

Let $A, \Delta A \in \mathbb{C}^{m \times m}$, $x, \delta x, b, \delta b \in \mathbb{C}^m$, A be nonsingular, and $\|\cdot\|$ be a vector norm and corresponding subordinate matrix norm. Then

$$\frac{\|\delta x\|}{\|x\|} \leq \underbrace{\|A\| \|A^{-1}\|}_{\kappa(A)} \frac{\|\delta b\|}{\|b\|}.$$

Definition 1.6.2.10 Condition number of a nonsingular matrix. The value $\kappa(A) = \|A\| \|A^{-1}\|$ is called the condition number of a nonsingular matrix A . \diamond

Week 2

The Singular Value Decomposition

2.1 Opening Remarks

2.1.1 Low rank approximation



YouTube: <https://www.youtube.com/watch?v=12K5aydB9cQ>

Consider this picture of the Gates Dell Complex that houses our Department of Computer Science:



It consists of an $m \times n$ array of pixels, each of which is a numerical value. Think of the j th column of pixels as a vector of values, b_j , so that the whole picture is represented by columns as

$$B = (b_0 \mid b_1 \mid \cdots \mid b_{n-1}),$$

where we recognize that we can view the picture as a matrix. What if we want to store this picture with fewer than $m \times n$ data? In other words, what if we want to compress the picture? To do so, we might identify a few of the columns in the picture to be the "chosen ones" that are representative of the other columns in the following sense: All columns in the picture are approximately linear combinations of these chosen columns.

Let's let linear algebra do the heavy lifting: what if we choose k roughly equally spaced columns in the picture:

$$\begin{aligned} a_0 &= b_0 \\ a_1 &= b_{n/k-1} \\ \vdots &\quad \vdots \\ a_{k-1} &= b_{(k-1)n/k-1}, \end{aligned}$$

where for illustration purposes we assume that n is an integer multiple of k . (We could instead choose them randomly or via some other method. This detail is not important as we try to gain initial insight.) We could then approximate each column of the picture, b_j , as a linear combination of a_0, \dots, a_{k-1} :

$$b_j \approx \chi_{0,j}a_0 + \chi_{1,j}a_1 + \cdots + \chi_{k-1,j}a_{k-1} = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{k-1} \end{array} \right) \begin{pmatrix} \frac{\chi_{0,j}}{\chi_{k-1,j}} \\ \vdots \end{pmatrix}.$$

We can write this more concisely by viewing these chosen columns as the columns of matrix A so that

$$b_j \approx Ax_j, \quad \text{where } A = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{k-1} \end{array} \right) \text{ and } x_j = \begin{pmatrix} \frac{\chi_{0,j}}{\chi_{k-1,j}} \\ \vdots \end{pmatrix}.$$

If A has linearly independent columns, the best such approximation (in the linear least squares sense) is obtained by choosing

$$x_j = (A^T A)^{-1} A^T b_j,$$

where you may recognize $(A^T A)^{-1} A^T$ as the (left) pseudo-inverse of A , leaving us with

$$b_j \approx A(A^T A)^{-1} A^T b_j.$$

This approximates b_j with the orthogonal projection of b_j onto the column space of A . Doing this for every column b_j leaves us with the following approximation to the picture:

$$B \approx \left(\begin{array}{c|c|c} A \underbrace{(A^T A)^{-1} A^T b_0}_{x_0} & \cdots & A \underbrace{(A^T A)^{-1} A^T b_{n-1}}_{x_{n-1}} \end{array} \right),$$

which is equivalent to

$$B \approx A \underbrace{\left(A^T A \right)^{-1} A^T \begin{pmatrix} b_0 & | & \cdots & | & b_{n-1} \\ x_0 & | & \cdots & | & x_{n-1} \end{pmatrix}}_{X} = A \underbrace{\left(A^T A \right)^{-1} A^T B}_{X} = AX.$$

Importantly, instead of requiring $m \times n$ data to store B , we now need only store A and X .

Homework 2.1.1.1 If B is $m \times n$ and A is $m \times k$, how many entries are there in A and X ?

Solution.

- A is $m \times k$.
- X is $k \times n$.

A total of $(m + n)k$ entries are in A and X .

Homework 2.1.1.2 AX is called a rank- k approximation of B . Why?

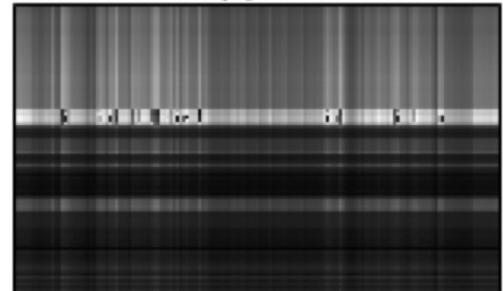
Solution. The matrix AX has rank at most equal to k (it is a rank- k matrix) since each of its columns can be written as a linear combinations of the columns of A and hence it has at most k linearly independent columns.

Let's have a look at how effective this approach is for our picture:

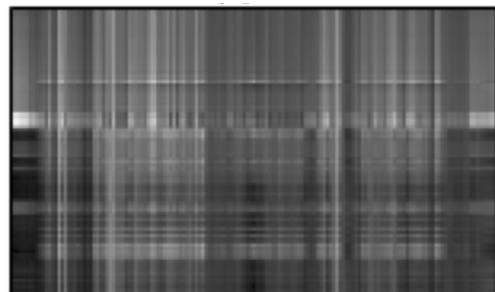
original:



$k = 1$



$k = 2$



$k = 10$



$k = 25$  $k = 50$ 

Now, there is no reason to believe that picking equally spaced columns (or restricting ourselves to columns in B) will yield the best rank- k approximation for the picture. It yields a pretty good result here in part because there is quite a bit of repetition in the picture, from column to column. So, the question can be asked: How do we find the best rank- k approximation for a picture or, more generally, a matrix? This would allow us to get the most from the data that needs to be stored. It is the Singular Value Decomposition (SVD), possibly the most important result in linear algebra, that provides the answer.

Remark 2.1.1.1 Those who need a refresher on this material may want to review Week 11 of Linear Algebra: Foundations to Frontiers [26]. We will discuss solving linear least squares problems further in [Week 4](#).

2.1.2 Overview

- 2.1 Opening Remarks
 - 2.1.1 Low rank approximation
 - 2.1.2 Overview
 - 2.1.3 What you will learn
- 2.2 Orthogonal Vectors and Matrices
 - 2.2.1 Orthogonal vectors
 - 2.2.2 Component in the direction of a vector
 - 2.2.3 Orthonormal vectors and matrices
 - 2.2.4 Unitary matrices
 - 2.2.5 Examples of unitary matrices
 - 2.2.6 Change of orthonormal basis
 - 2.2.7 Why we love unitary matrices choice
- 2.3 The Singular Value Decomposition
 - 2.3.1 The Singular Value Decomposition Theorem
 - 2.3.2 Geometric interpretation

- 2.3.3 An "algorithm" for computing the SVD
- 2.3.4 The Reduced Singular Value Decomposition
- 2.3.5 The SVD of nonsingular matrices
- 2.3.6 Best rank-k approximation
- 2.4 Enrichments
 - 2.4.1 Principle Component Analysis (PCA)
- 2.5 Wrap Up
 - 2.5.1 Additional homework
 - 2.5.2 Summary

2.1.3 What you will learn

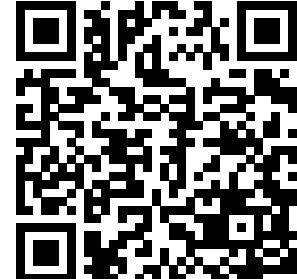
This week introduces two concepts that have theoretical and practical importance: unitary matrices and the Singular Value Decomposition (SVD).

Upon completion of this week, you should be able to

- Determine whether vectors are orthogonal.
- Compute the component of a vector in the direction of another vector.
- Relate sets of orthogonal vectors to orthogonal and unitary matrices.
- Connect unitary matrices to the changing of orthonormal basis.
- Identify transformations that can be represented by unitary matrices.
- Prove that multiplying with unitary matrices does not amplify relative error.
- Use norms to quantify the conditioning of solving linear systems.
- Prove and interpret the Singular Value Decomposition.
- Link the Reduced Singular Value Decomposition to the rank of the matrix and determine the best rank-k approximation to a matrix.
- Determine whether a matrix is close to being nonsingular by relating the Singular Value Decomposition to the condition number.

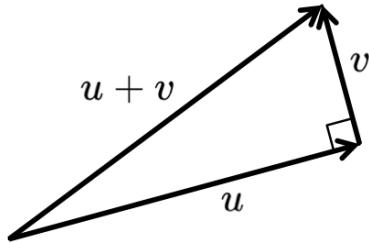
2.2 Orthogonal Vectors and Matrices

2.2.1 Orthogonal vectors

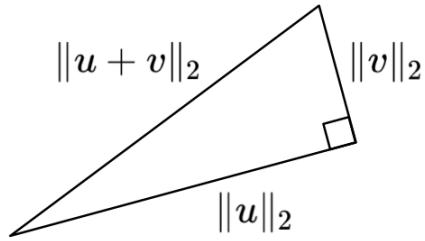


YouTube: <https://www.youtube.com/watch?v=3zpdTfwZSEo>

At some point in your education you were told that vectors are orthogonal (perpendicular) if and only if their dot product (inner product) equals zero. Let's review where this comes from. Given two vectors $u, v \in \mathbb{R}^m$, those two vectors, and their sum all exist in the same two dimensional (2D) subspace. So, they can be visualized as



where the page on which they are drawn is that 2D subspace. Now, if they are, as drawn, perpendicular and we consider the lengths of the sides of the triangle that they define



then we can employ the first theorem you were probably ever exposed to, the Pythagorean Theorem, to find that

$$\|u\|_2^2 + \|v\|_2^2 = \|u + v\|_2^2.$$

Using what we know about the relation between the two norm and the dot product, we find

that

$$\begin{aligned}
 u^T u + v^T v &= (u + v)^T (u + v) \\
 &\Leftrightarrow \quad < \text{multiply out} > \\
 u^T u + v^T v &= u^T u + u^T v + v^T u + v^T v \\
 &\Leftrightarrow \quad < u^T v = v^T u \text{ if } u \text{ and } v \text{ are real-valued} > \\
 u^T u + v^T v &= u^T u + 2u^T v + v^T v \\
 &\Leftrightarrow \quad < \text{delete common terms} > \\
 0 &= 2u^T v
 \end{aligned}$$

so that we can conclude that $u^T v = 0$.

While we already encountered the notation $x^H x$ as an alternative way of expressing the length of a vector, $\|x\|_2 = \sqrt{x^H x}$, we have not formally defined the inner product (dot product), for complex-valued vectors:

Definition 2.2.1.1 Dot product (Inner product). Given $x, y \in \mathbb{C}^m$ their dot product (inner product) is defined as

$$x^H y = \bar{x}^T y = \bar{x}^T y = \bar{\chi}_0 \psi_0 + \bar{\chi}_1 \psi_1 + \cdots + \bar{\chi}_{m-1} \psi_{m-1} = \sum_{i=0}^{m-1} \bar{\chi}_i \psi_i.$$

◊

The notation x^H is short for \bar{x}^T , where \bar{x} equals the vector x with all its entries conjugated. So,

$$\begin{aligned}
 x^H y &= < \text{expose the elements of the vectors} > \\
 \left(\begin{array}{c} \chi_0 \\ \vdots \\ \chi_{m-1} \end{array} \right)^H \left(\begin{array}{c} \psi_0 \\ \vdots \\ \psi_{m-1} \end{array} \right) &= < x^H = \bar{x}^T > \\
 \overline{\left(\begin{array}{c} \chi_0 \\ \vdots \\ \chi_{m-1} \end{array} \right)}^T \left(\begin{array}{c} \psi_0 \\ \vdots \\ \psi_{m-1} \end{array} \right) &= < \text{conjugate the elements of } x > \\
 \left(\begin{array}{c} \bar{\chi}_0 \\ \vdots \\ \bar{\chi}_{m-1} \end{array} \right)^T \left(\begin{array}{c} \psi_0 \\ \vdots \\ \psi_{m-1} \end{array} \right) &= < \text{view } x \text{ as a } m \times 1 \text{ matrix and transpose} > \\
 \left(\begin{array}{c|c|c} \bar{\chi}_0 & \cdots & \bar{\chi}_{m-1} \end{array} \right) \left(\begin{array}{c} \psi_0 \\ \vdots \\ \psi_{m-1} \end{array} \right) &= < \text{view } x^H \text{ as a matrix and perform matrix-vector multiplication} > \\
 \sum_{i=0}^{m-1} \bar{\chi}_i \psi_i. &
 \end{aligned}$$

Homework 2.2.1.1 Let $x, y \in \mathbb{C}^m$.

ALWAYS/SOMETIMES/NEVER: $\overline{x^H y} = y^H x$.

Answer. ALWAYS

Now prove it!

Solution.

$$\overline{x^H y} = \overline{\sum_{i=0}^{m-1} \bar{\chi}_i \psi_i} = \sum_{i=0}^{m-1} \overline{\bar{\chi}_i \psi_i} = \sum_{i=0}^{m-1} \bar{\psi}_i \bar{\chi}_i = y^H x.$$

Homework 2.2.1.2 Let $x \in \mathbb{C}^m$.

ALWAYS/SOMETIMES/NEVER: $x^H x$ is real-valued.

Answer. ALWAYS

Now prove it!

Solution. By the last homework,

$$\overline{x^H x} = x^H x,$$

A complex number is equal to its conjugate only if it is real-valued.

The following defines orthogonality of two vectors with complex-valued elements:

Definition 2.2.1.2 Orthogonal vectors. Let $x, y \in \mathbb{C}^m$. These vectors are said to be orthogonal (perpendicular) iff $x^H y = 0$. \diamond

2.2.2 Component in the direction of a vector



YouTube: <https://www.youtube.com/watch?v=CqcJ6Nh1QWg>

In a previous linear algebra course, you may have learned that if $a, b \in \mathbb{R}^m$ then

$$\hat{b} = \frac{a^T b}{a^T a} a = \frac{aa^T}{a^T a} b$$

equals the component of b in the direction of a and

$$b^\perp = b - \hat{b} = \left(I - \frac{aa^T}{a^T a} \right) b$$

equals the component of b orthogonal to a , since $b = \hat{b} + b^\perp$ and $\hat{b}^T b^\perp = 0$. Similarly, if $a, b \in \mathbb{C}^m$ then

$$\hat{b} = \frac{a^H b}{a^H a} a = \frac{aa^H}{a^H a} b$$

equals the component of b in the direction of a and

$$b^\perp = b - \hat{b} = \left(I - \frac{aa^H}{a^H a}\right)b$$

equals the component of b orthogonal to a .

Remark 2.2.2.1 The matrix that (orthogonally) projects the vector to which it is applied onto the vector a is given by

$$\frac{aa^H}{a^H a}$$

while

$$I - \frac{aa^H}{a^H a}$$

is the matrix that (orthogonally) projects the vector to which it is applied onto the space orthogonal to the vector a .

Homework 2.2.2.1 Let $a \in \mathbb{C}^m$.

ALWAYS/SOMETIMES/NEVER>:

$$\left(\frac{aa^H}{a^H a}\right) \left(\frac{aa^H}{a^H a}\right) = \frac{aa^H}{a^H a}.$$

Interpret what this means about a matrix that projects onto a vector.

Answer. ALWAYS.

Now prove it.

Solution.

$$\begin{aligned} & \left(\frac{aa^H}{a^H a}\right) \left(\frac{aa^H}{a^H a}\right) \\ &= < \text{multiply numerators and denominators} > \\ & \frac{aa^H aa^H}{(a^H a)(a^H a)} \\ &= < \text{associativity} > \\ & \frac{a(a^H a)a^H}{(a^H a)(a^H a)} \\ &= < a^H a \text{ is a scalar and hence commutes to front} > \\ & \frac{a^H aaa^H}{(a^H a)(a^H a)} \\ &= < \text{scalar division} > \\ & \frac{aa^H}{a^H a}. \end{aligned}$$

Interpretation: orthogonally projecting the orthogonal projection of a vector yields the orthogonal projection of the vector.

Homework 2.2.2.2 Let $a \in \mathbb{C}^m$.

ALWAYS/SOMETIMES/NEVER:

$$\left(\frac{aa^H}{a^H a}\right) \left(I - \frac{aa^H}{a^H a}\right) = 0$$

(the zero matrix). Interpret what this means.

Answer. ALWAYS.

Now prove it.

Solution.

$$\begin{aligned}
 & \left(\frac{aa^H}{a^H a} \right) \left(I - \frac{aa^H}{a^H a} \right) \\
 & = \quad < \text{distribute} > \\
 & \left(\frac{aa^H}{a^H a} \right) - \left(\frac{aa^H}{a^H a} \right) \left(\frac{aa^H}{a^H a} \right) \\
 & = \quad < \text{last homework} > \\
 & \left(\frac{aa^H}{a^H a} \right) - \left(\frac{aa^H}{a^H a} \right) \\
 & = \\
 & 0.
 \end{aligned}$$

Interpretation: first orthogonally projecting onto the space *orthogonal to* vector a and then orthogonally projecting the resulting vector onto that a leaves you with the zero vector.

Homework 2.2.2.3 Let $a, b \in \mathbb{C}^n$, $\hat{b} = \frac{aa^H}{a^H a}b$, and $b^\perp = b - \hat{b}$.

ALWAYS/SOMETIMES/NEVER: $\hat{b}^H b^\perp = 0$.

Answer. ALWAYS.

Now prove it.

Solution.

$$\begin{aligned}
 & \hat{b}^H b^\perp \\
 & = \quad < \text{substitute } \hat{b} \text{ and } b^\perp > \\
 & \left(\frac{aa^H}{a^H a} b \right)^H (b - \hat{b}) \\
 & = \quad < (Ax)^H = x^H A^H; \text{ substitute } b - \hat{b} > \\
 & b^H \left(\frac{aa^H}{a^H a} \right)^H \left(I - \frac{aa^H}{a^H a} \right) b \\
 & = \quad < (((xy^H)/\alpha)^H = yx^H/\alpha \text{ if } \alpha \text{ is real} > \\
 & b^H \frac{aa^H}{a^H a} \left(I - \frac{aa^H}{a^H a} \right) b \\
 & = \quad < \text{last homework} > \\
 & b^H 0b \\
 & = \quad < 0x = 0; y^H 0 = 0 > \\
 & 0.
 \end{aligned}$$

2.2.3 Orthonormal vectors and matrices



YouTube: <https://www.youtube.com/watch?v=GFFfvDpj5dzw>

A lot of the formulae in the last unit become simpler if the length of the vector equals

one: If $\|u\|_2 = 1$ then

- the component of v in the direction of u equals

$$\frac{u^H v}{u^H u} u = u^H v u.$$

- the matrix that projects a vector onto the vector u is given by

$$\frac{u u^H}{u^H u} = u u^H.$$

- the component of v orthogonal to u equals

$$v - \frac{u^H v}{u^H u} u = v - u^H v u.$$

- the matrix that projects a vector onto the space orthogonal to u is given by

$$I - \frac{u u^H}{u^H u} = I - u u^H.$$

Homework 2.2.3.1 Let $u \neq 0 \in \mathbb{C}^m$.

ALWAYS/SOMETIMES/NEVER $u/\|u\|_2$ has unit length.

Answer. ALWAYS.

Now prove it.

Solution.

$$\begin{aligned} \left\| \frac{u}{\|u\|_2} \right\|_2 &= < \text{homogeneity of norms} > \\ \frac{\|u\|_2}{\|u\|_2} &= < \text{algebra} > \\ 1 & \end{aligned}$$

This last exercise shows that any nonzero vector can be scaled (normalized) to have unit length.

Definition 2.2.3.1 Orthonormal vectors. Let $u_0, u_1, \dots, u_{n-1} \in \mathbb{C}^m$. These vectors are said to be mutually orthonormal if for all $0 \leq i, j < n$

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

◊

The definition implies that $\|u_i\|_2 = \sqrt{u_i^H u_i} = 1$ and hence each of the vectors is of unit length in addition to being orthogonal to each other.

The standard basis vectors ([Definition 1.3.1.3](#))

$$\{e_j\}_{j=0}^{m-1} \subset \mathbb{C}^m,$$

where

$$e_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow \text{entry indexed with } j$$

are mutually orthonormal since, clearly,

$$e_i^H e_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Naturally, any subset of the standard basis vectors is a set of mutually orthonormal vectors.

Remark 2.2.3.2 For n vectors of size m to be mutually orthonormal, n must be less than or equal to m . This is because n mutually orthonormal vectors are linearly independent and there can be at most m linearly independent vectors of size m .

A very concise way of indicating that a set of vectors are mutually orthonormal is to view them as the columns of a matrix, which then has a very special property:

Definition 2.2.3.3 Orthonormal matrix. Let $Q \in \mathbb{C}^{m \times n}$ (with $n \leq m$). Then Q is said to be an orthonormal matrix iff $Q^H Q = I$. \diamond

The subsequent exercise makes the connection between mutually orthonormal vectors and an orthonormal matrix.

Homework 2.2.3.2 Let $Q \in \mathbb{C}^{m \times n}$ (with $n \leq m$). Partition $Q = (q_0 | q_1 | \cdots | q_{n-1})$.

TRUE/FALSE: Q is an orthonormal matrix if and only if q_0, q_1, \dots, q_{n-1} are mutually orthonormal.

Answer. TRUE

Now prove it!

Solution. Let $Q \in \mathbb{C}^{m \times n}$ (with $n \leq m$). Partition $Q = (q_0 | q_1 | \cdots | q_{n-1})$. Then

$$\begin{aligned} Q^H Q &= (q_0 | q_1 | \cdots | q_{n-1})^H (q_0 | q_1 | \cdots | q_{n-1}) \\ &= \begin{pmatrix} q_0^H \\ q_1^H \\ \vdots \\ q_{n-1}^H \end{pmatrix} (q_0 | q_1 | \cdots | q_{n-1}) \\ &= \begin{pmatrix} q_0^H q_0 & q_0^H q_1 & \cdots & q_0^H q_{n-1} \\ \hline q_1^H q_0 & q_1^H q_1 & \cdots & q_1^H q_{n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline q_{n-1}^H q_0 & q_{n-1}^H q_1 & \cdots & q_{n-1}^H q_{n-1} \end{pmatrix}. \end{aligned}$$

Now consider that $Q^H Q = I$:

$$\left(\begin{array}{c|c|c|c} q_0^H q_0 & q_0^H q_1 & \cdots & q_0^H q_{n-1} \\ \hline q_1^H q_0 & q_1^H q_1 & \cdots & q_1^H q_{n-1} \\ \vdots & \vdots & & \vdots \\ \hline q_{n-1}^H q_0 & q_{n-1}^H q_1 & \cdots & q_{n-1}^H q_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} 1 & 0 & \cdots & 0 \\ \hline 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ \hline 0 & 0 & \cdots & 1 \end{array} \right).$$

Clearly Q is orthonormal if and only if q_0, q_1, \dots, q_{n-1} are mutually orthonormal.

Homework 2.2.3.3 Let $Q \in \mathbb{C}^{m \times n}$.

ALWAYS/SOMETIMES/NEVER: If $Q^H Q = I$ then $Q Q^H = I$.

Answer. SOMETIMES.

Now explain why.

Solution.

- If Q is a square matrix ($m = n$) then $Q^H Q = I$ means $Q^{-1} = Q^H$. But then $Q Q^{-1} = I$ and hence $Q Q^H = I$.
- If Q is not square, then $Q^H Q = I$ means $m > n$. Hence Q has rank equal to n which in turn means $Q Q^H$ is a matrix with rank at most equal to n . (Actually, its rank equals n). Since I has rank equal to m (it is an $m \times m$ matrix with linearly independent columns), $Q Q^H$ cannot equal I .

More concretely: let $m > 1$ and $n = 1$. Choose $Q = \begin{pmatrix} e_0 \end{pmatrix}$. Then $Q^H Q = e_0^H e_0 = 1 = I$. But

$$Q Q^H = e_0 e_0^H = \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}.$$

2.2.4 Unitary matrices



YouTube: <https://www.youtube.com/watch?v=izONEm09uqw>

Homework 2.2.4.1 Let $Q \in \mathbb{C}^{m \times n}$ be an orthonormal matrix.

ALWAYS/SOMETIMES/NEVER: $Q^{-1} = Q^H$ and $Q Q^H = I$.

Answer. SOMETIMES

Now explain it!

Solution. If Q is unitary, then it is an orthonormal matrix and square. Because it is an orthonormal matrix, $Q^H Q = I$. If $A, B \in \mathbb{C}^{m \times m}$, the matrix B such that $BA = I$ is the inverse of A . Hence $Q^{-1} = Q^H$. Also, if $BA = I$ then $AB = I$ and hence $QQ^H = I$.

However, an orthonormal matrix is not necessarily square. For example, the matrix $Q = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}$ is an orthonormal matrix: $Q^T Q = I$. However, it doesn't have an inverse because it is not square.

If an orthonormal matrix is square, then it is called a unitary matrix.

Definition 2.2.4.1 Unitary matrix. Let $U \in \mathbb{C}^{m \times m}$. Then U is said to be a unitary matrix if and only if $U^H U = I$ (the identity). \diamond

Remark 2.2.4.2 Unitary matrices are always square. Sometimes the term **orthogonal matrix** is used instead of unitary matrix, especially if the matrix is real valued.

Unitary matrices have some very nice properties, as captured by the following exercises.

Homework 2.2.4.2 Let $Q \in \mathbb{C}^{m \times m}$ be a unitary matrix.

ALWAYS/SOMETIMES/NEVER: $Q^{-1} = Q^H$ and $QQ^H = I$.

Answer. ALWAYS

Now explain it!

Solution. If Q is unitary, then it is square and $Q^H Q = I$. Hence $Q^{-1} = Q^H$ and $QQ^H = I$.

Homework 2.2.4.3 TRUE/FALSE: If U is unitary, so is U^H .

Answer. TRUE

Now prove it!

Solution. Clearly, U^H is square. Also, $(U^H)^H U^H = (UU^H)^H = I$ by the last homework.

Homework 2.2.4.4 Let $U_0, U_1 \in \mathbb{C}^{m \times m}$ both be unitary.

ALWAYS/SOMETIMES/NEVER: $U_0 U_1$, is unitary.

Answer. ALWAYS

Now prove it!

Solution. Obviously, $U_0 U_1$ is a square matrix.

Now,

$$(U_0 U_1)^H (U_0 U_1) = U_1^H \underbrace{U_0^H U_0}_I U_1 = \underbrace{U_1^H U_1}_I = I.$$

Hence $U_0 U_1$ is unitary.

Homework 2.2.4.5 Let $U_0, U_1, \dots, U_{k-1} \in \mathbb{C}^{m \times m}$ all be unitary.

ALWAYS/SOMETIMES/NEVER: Their product, $U_0 U_1 \cdots U_{k-1}$, is unitary.

Answer. ALWAYS

Now prove it!

Solution. Strictly speaking, we should do a proof by induction. But instead we will make

the more informal argument that

$$\begin{aligned}
 (U_0 U_1 \cdots U_{k-1})^H U_0 U_1 \cdots U_{k-1} &= U_{k-1}^H \cdots U_1^H U_0^H U_0 U_1 \cdots U_{k-1} \\
 &= U_{k-1}^H \cdots U_1^H \underbrace{U_0^H U_0}_I U_1 \cdots U_{k-1} = I.
 \end{aligned}$$

(When you see a proof that involved \cdots , it would be more rigorous to use a proof by induction.)

Remark 2.2.4.3 Many algorithms that we encounter in the future will involve the application of a sequence of unitary matrices, which is why the result in this last exercise is of great importance.

Perhaps the most important property of a unitary matrix is that it preserves length.

Homework 2.2.4.6 Let $U \in \mathbb{C}^{m \times m}$ be a unitary matrix and $x \in \mathbb{C}^m$. Prove that $\|Ux\|_2 = \|x\|_2$.

Solution.

$$\begin{aligned}
 \|Ux\|_2^2 &= <\text{alternative definition}> \\
 (Ux)^H Ux &= <(Az)^H = z^H A^H> \\
 x^H U^H Ux &= <U \text{ is unitary}> \\
 x^H x &= <\text{alternative definition}> \\
 \|x\|_2^2.
 \end{aligned}$$

The converse is true as well:

Theorem 2.2.4.4 If $A \in \mathbb{C}^{m \times m}$ preserves length ($\|Ax\|_2 = \|x\|_2$ for all $x \in \mathbb{C}^m$), then A is unitary.

Proof. We first prove that $(Ax)^H (Ay) = x^H y$ for all x, y by considering $\|x - y\|_2^2 = \|A(x - y)\|_2^2$. We then use that to evaluate $e_i^H A^H A e_j$.

Let $x, y \in \mathbb{C}^m$. Then

$$\begin{aligned}
& \|x - y\|_2^2 = \|A(x - y)\|_2^2 \\
& \Leftrightarrow <\text{alternative definition}> \\
& (x - y)^H(x - y) = (A(x - y))^H A(x - y) \\
& = <(Bz)^H = z^H B^H> \\
& (x - y)^H(x - y) = (x - y)^H A^H A(x - y) \\
& \Leftrightarrow <\text{multiply out}> \\
& x^H x - x^H y - y^H x + y^H y = x^H A^H A x - x^H A^H A y - y^H A^H A x + y^H A^H A y \\
& \Leftrightarrow <\text{alternative definition}; \overline{x^H y} = y^H x> \\
& \|x\|_2^2 - (x^H y + \overline{x^H y}) + \|y\|_2^2 = \|Ax\|_2^2 - (x^H A^H A y + \overline{x^H A^H A y}) + \|Ay\|_2^2 \\
& \Leftrightarrow <\|Ax\|_2 = \|x\|_2 \text{ and } \|Ay\|_2 = \|y\|_2; \alpha + \bar{\alpha} = 2\operatorname{Re}(\alpha)> \\
& \operatorname{Re}(x^H y) = \operatorname{Re}((Ax)^H A y)
\end{aligned}$$

One can similarly show that $\operatorname{Im}(x^H y) = \operatorname{Im}((Ax)^H A y)$ by considering $A(ix - y)$.

Conclude that $(Ax)^H(Ay) = x^H y$.

We now use this to show that $A^H A = I$ by using the fact that the standard basis vectors have the property that

$$e_i^H e_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

and that the i, j entry in $A^H A$ equals $e_i^H A^H A e_j$.

Note: I think the above can be made much more elegant by choosing α such that $\alpha x^H y$ is real and then looking at $\|x + \alpha y\|_2 = \|A(x + \alpha y)\|_2$ instead, much like we did in the proof of the Cauchy-Schwartz inequality. Try and see if you can work out the details. ■

Homework 2.2.4.7 Prove that if U is unitary then $\|U\|_2 = 1$.

Solution.

$$\begin{aligned}
& \|U\|_2 \\
& = <\text{definition}> \\
& \max_{\|x\|_2=1} \|Ux\|_2 \\
& = <\text{unitary matrices preserve length}> \\
& \max_{\|x\|_2=1} \|x\|_2 \\
& = <\text{algebra}> \\
& 1
\end{aligned}$$

(The above can be really easily proven with the SVD. Let's point that out later.)

Homework 2.2.4.8 Prove that if U is unitary then $\kappa_2(U) = 1$.

Solution.

$$\begin{aligned}
 & \kappa_2 U \\
 &= < \text{definition} > \\
 & \|U\|_2 \|U^{-1}\|_2 \\
 &= < \text{both } U \text{ and } U^{-1} \text{ are unitary ; last homework} > \\
 & 1 \times 1 \\
 &= < \text{arithmetic} > \\
 & 1
 \end{aligned}$$

The preservation of length extends to the preservation of norms that have a relation to the 2-norm:

Homework 2.2.4.9 Let $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ be unitary and $A \in \mathbb{C}^{m \times n}$. Show that

- $\|U^H A\|_2 = \|A\|_2$.
- $\|AV\|_2 = \|A\|_2$.
- $\|U^H AV\|_2 = \|A\|_2$.

Hint. Exploit the definition of the 2-norm:

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2.$$

Solution.

$$\begin{aligned}
 & \bullet \\
 & \|U^H A\|_2 \\
 &= < \text{definition of 2-norm} > \\
 & \max_{\|x\|_2=1} \|U^H Ax\|_2 \\
 &= < U \text{ is unitary and unitary matrices preserve length} > \\
 & \max_{\|x\|_2=1} \|Ax\|_2 \\
 &= < \text{definition of 2-norm} > \\
 & \|A\|_2. \\
 \\
 & \bullet \\
 & \|AV\|_2 \\
 &= < \text{definition of 2-norm} > \\
 & \max_{\|x\|_2=1} \|AVx\|_2 \\
 &= < V^H \text{ is unitary and unitary matrices preserve length} > \\
 & \max_{\|Vx\|_2=1} \|A(Vx)\|_2 \\
 &= < \text{substitute } y = Vx > \\
 & \max_{\|y\|_2=1} \|Ay\|_2 \\
 &= < \text{definition of 2-norm} > \\
 & \|A\|_2.
 \end{aligned}$$

- The last part follows immediately from the previous two:

$$\|U^H AV\|_2 = \|U^H(AV)\|_2 = \|AV\|_2 = \|A\|_2.$$

Homework 2.2.4.10 Let $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ be unitary and $A \in \mathbb{C}^{m \times n}$. Show that

- $\|U^H A\|_F = \|A\|_F$.
- $\|AV\|_F = \|A\|_F$.
- $\|U^H AV\|_F = \|A\|_F$.

Hint. How does $\|A\|_F$ relate to the 2-norms of its columns?

Solution.

- Partition

$$A = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right).$$

Then we saw in [Subsection 1.3.3](#) that $\|A\|_F^2 = \sum_{j=0}^{n-1} \|a_j\|_2^2$.

Now,

$$\begin{aligned} \|U^H A\|_F^2 &= \langle \text{partition } A \text{ by columns} \rangle \\ \|U^H \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right) \|_F^2 &= \langle \text{property of matrix-vector multiplication} \rangle \\ \left\| \left(\begin{array}{c|c|c} U^H a_0 & \cdots & U^H a_{n-1} \end{array} \right) \right\|_F^2 &= \langle \text{exercice in Chapter 1} \rangle \\ \sum_{j=0}^{n-1} \|U^H a_j\|_2^2 &= \langle \text{unitary matrices preserve length} \rangle \\ \sum_{j=0}^{n-1} \|a_j\|_2^2 &= \langle \text{exercice in Chapter 1} \rangle \\ \|A\|_F^2. \end{aligned}$$

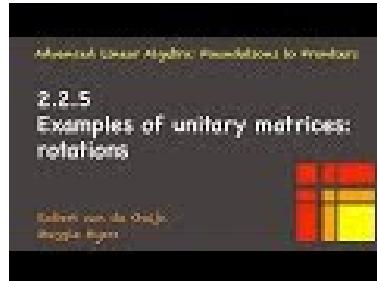
- To prove that $\|AV\|_F = \|A\|_F$ recall that $\|A^H\|_F = \|A\|_F$.
- The last part follows immediately from the first two parts.

In the last two exercises we consider $U^H AV$ rather than UAV because it sets us up better for future discussion.

2.2.5 Examples of unitary matrices

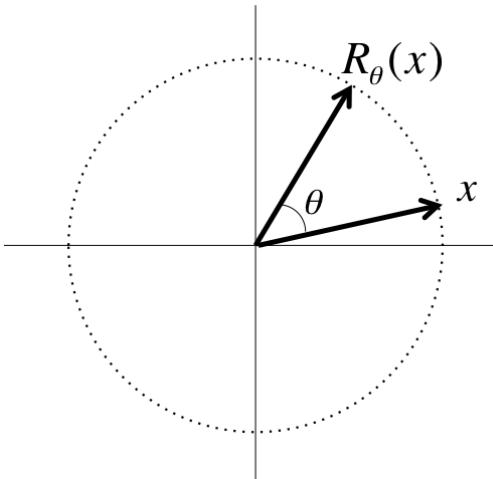
In this unit, we will discuss a few situations where you may have encountered unitary matrices without realizing. Since few of us walk around pointing out to each other "Look, another matrix!", we first consider if a transformation (function) might be a linear transformation. This allows us to then ask the question "What kind of transformations we see around us preserve length?" After that, we discuss how those transformations are represented as matrices. That leaves us to then check whether the resulting matrix is unitary.

2.2.5.1 Rotations



YouTube: <https://www.youtube.com/watch?v=C0mlDZ280hc>

A rotation in 2D, $R_\theta : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, takes a vector and rotates that vector through the angle θ :

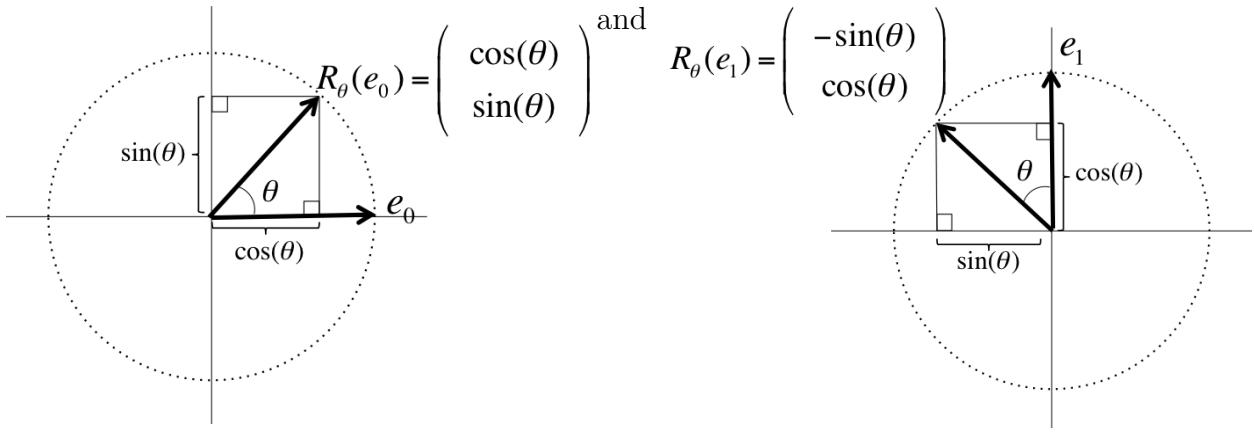


If you think about it,

- If you scale a vector first and then rotate it, you get the same result as if you rotate it first and then scale it.
- If you add two vectors first and then rotate, you get the same result as if you rotate them first and then add them.

Thus, a rotation is a linear transformation. Also, the above picture captures that a rotation preserves the length of the vector to which it is applied. We conclude that the matrix that represents a rotation should be a unitary matrix.

Let us compute the matrix that represents the rotation through an angle θ . Recall that if $L : \mathbb{C}^n \rightarrow \mathbb{C}^m$ is a linear transformation and A is the matrix that represents it, then the j th column of A , a_j , equals $L(e_j)$. The pictures



illustrate that

$$R_\theta(e_0) = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \quad \text{and} \quad R_\theta(e_1) = \begin{pmatrix} -\sin(\theta) \\ \cos(\theta) \end{pmatrix}.$$

Thus,

$$R_\theta(x) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}.$$

Homework 2.2.5.1 Show that

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

is a unitary matrix. (Since it is real valued, it is usually called an orthogonal matrix instead.)

Hint. Hint: use c for $\cos(\theta)$ and s for $\sin(\theta)$ to save yourself a lot of writing!

Solution.

$$\begin{aligned}
 & \left(\begin{array}{c|cc} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array} \right)^H \left(\begin{array}{c|cc} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array} \right) \\
 &= < \text{the matrix is real valued} > \\
 & \left(\begin{array}{c|cc} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array} \right)^T \left(\begin{array}{c|cc} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array} \right) \\
 &= < \text{transpose} > \\
 & \left(\begin{array}{cc} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{array} \right) \left(\begin{array}{c|cc} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array} \right) \\
 &= < \text{multiply} > \\
 & \left(\begin{array}{cc} \cos^2(\theta) + \sin^2(\theta) & -\cos(\theta)\sin(\theta) + \sin(\theta)\cos(\theta) \\ -\sin(\theta)\cos(\theta) + \cos(\theta)\sin(\theta) & \sin^2(\theta) + \cos^2(\theta) \end{array} \right) \\
 &= < \text{geometry; algebra} > \\
 & \left(\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right)
 \end{aligned}$$

Homework 2.2.5.2 Prove, without relying on geometry but using what you just discovered, that $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$

Solution. Undoing a rotation by an angle θ means rotating in the opposite direction through angle θ or, equivalently, rotating through angle $-\theta$. Thus, the inverse of R_θ is $R_{-\theta}$. The matrix that represents R_θ is given by

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

and hence the matrix that represents $R_{-\theta}$ is given by

$$\begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix}.$$

Since $R_{-\theta}$ is the inverse of R_θ we conclude that

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}^{-1} = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix}.$$

But we just discovered that

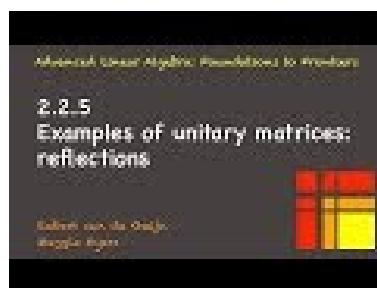
$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}^{-1} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}^T = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}.$$

Hence

$$\begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

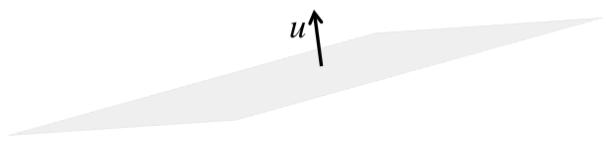
from which we conclude that $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$.

2.2.5.2 Reflections

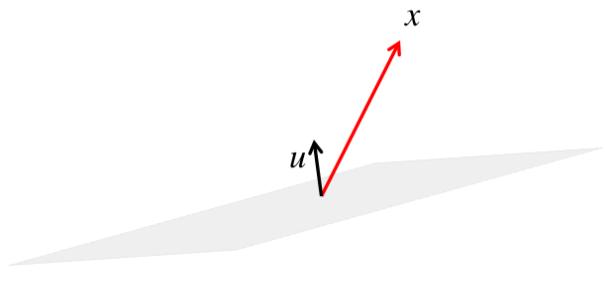


YouTube: <https://www.youtube.com/watch?v=r8S04qqcc-o>

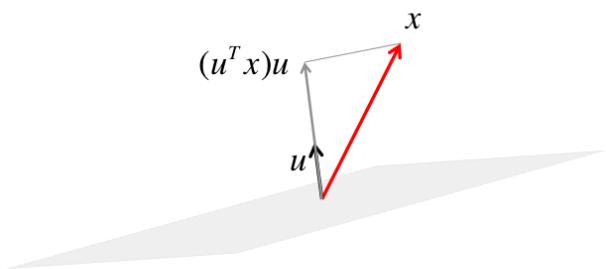
Picture a mirror with its orientation defined by a unit length vector, u , that is orthogonal to it.



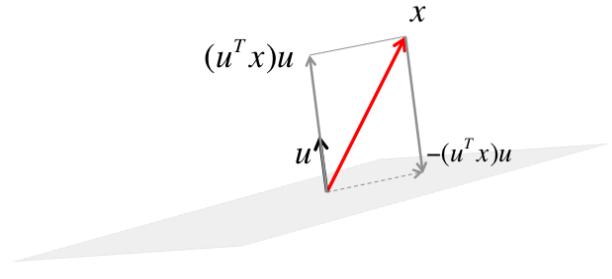
We will consider how a vector, x , is reflected by this mirror.



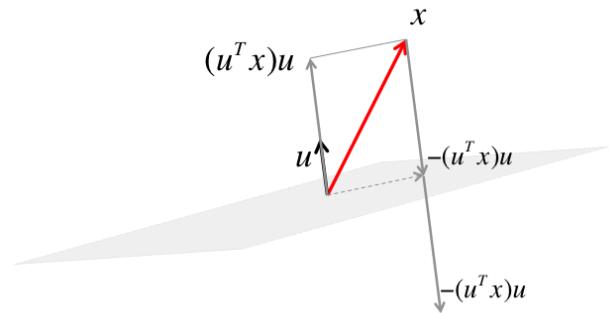
The component of x orthogonal to the mirror equals the component of x in the direction of u , which equals $(u^T x)u$.



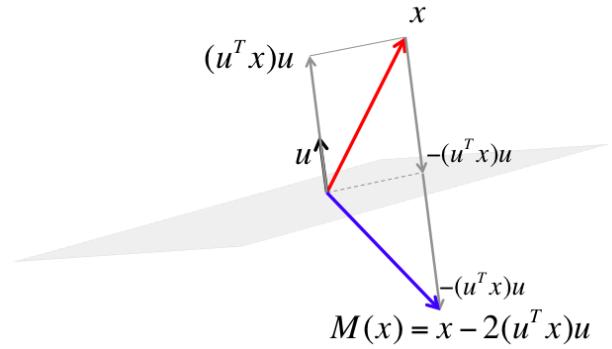
The orthogonal projection of x onto the mirror is then given by the dashed vector, which equals $x - (u^T x)u$.



To get to the reflection of x , we now need to go further yet by $-(u^T x)u$.



We conclude that the transformation that mirrors (reflects) x with respect to the mirror is given by $M(x) = x - 2(u^T x)u$.



The transformation described above preserves the length of the vector to which it is applied.

Homework 2.2.5.3 (Verbally) describe why reflecting a vector as described above is a linear transformation.

Solution.

- If you scale a vector first and then reflect it, you get the same result as if you reflect it first and then scale it.
- If you add two vectors first and then reflect, you get the same result as if you reflect them first and then add them.

Homework 2.2.5.4 Show that the matrix that represents $M : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ in the above example is given by $I - 2uu^T$.

Hint. Rearrange $x - 2(u^Tx)u$.

Solution. We notice that

$$\begin{aligned} x - 2(u^Tx)u &= \langle \alpha x = x\alpha \rangle \\ x - 2u(u^Tx) &= \quad \text{associativity} \quad > \\ Ix - 2uu^Tx &= \quad \text{distributivity} \quad > \\ (I - 2uu^T)x. & \end{aligned}$$

Hence $M(x) = (I - 2uu^T)x$ and the matrix that represents M is given by $I - 2uu^T$.

Homework 2.2.5.5 (Verbally) describe why $(I - 2uu^T)^{-1} = I - 2uu^T$ if $u \in \mathbb{R}^3$ and $\|u\|_2 = 1$.

Solution. If you take a vector, x , and reflect it with respect to the mirror defined by u , and you then reflect the result with respect to the same mirror, you should get the original vector x back. Hence, the matrix that represents the reflection should be its own inverse.

Homework 2.2.5.6 Let $M : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ be defined by $M(x) = (I - 2uu^T)x$, where $\|u\|_2 = 1$. Show that the matrix that represents it is unitary (or, rather, orthogonal since it is in $\mathbb{R}^{3 \times 3}$).

Solution. Pushing through the math we find that

$$\begin{aligned} (I - 2uu^T)^T(I - 2uu^T) &= \langle (A + B)^T = A^T + B^T \rangle \\ (I^T - (2uu^T)^T)(I - 2uu^T) &= \langle (\alpha AB^T)^T = \alpha BA^T \rangle \\ (I - 2uu^T)(I - 2uu^T) &= \quad \text{distributivity} \quad > \\ (I - 2uu^T) - (I - 2uu^T)(2uu^T) &= \quad \text{distributivity} \quad > \\ I - 2uu^T - 2uu^T + 2uu^T 2uu^T &= \quad \langle u^T u = 1 \rangle \\ I - 4uu^T + 4uu^T &= \quad \langle A - A = 0 \rangle \\ I. & \end{aligned}$$

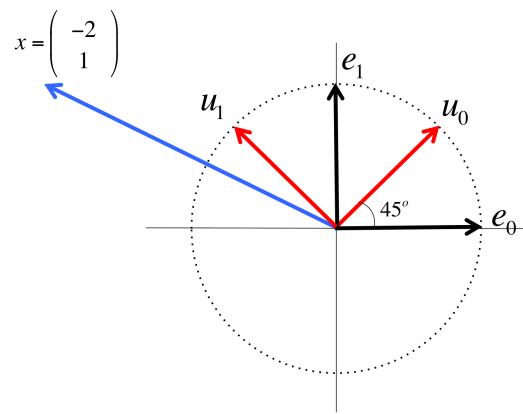
Remark 2.2.5.1 Unitary matrices in general, and rotations and reflections in particular, will play a key role in many of the practical algorithms we will develop in this course.

2.2.6 Change of orthonormal basis



YouTube: <https://www.youtube.com/watch?v=DwTVkdQKJK4>

Homework 2.2.6.1 Consider the vector $x = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$ and the following picture that depicts a rotated basis with basis vectors u_0 and u_1 .



What are the coordinates of the vector x in this rotated system? In other words, find $\hat{x} = \begin{pmatrix} \hat{x}_0 \\ \hat{x}_1 \end{pmatrix}$ such that $\hat{x}_0 u_0 + \hat{x}_1 u_1 = x$.

Solution. There are a number of approaches to this. One way is to try to remember the formula you may have learned in a pre-calculus course about change of coordinates. Let's instead start by recognizing (from geometry or by applying the Pythagorean Theorem) that

$$u_0 = \begin{pmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \end{pmatrix} = \frac{\sqrt{2}}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad u_1 = \begin{pmatrix} -\sqrt{2}/2 \\ \sqrt{2}/2 \end{pmatrix} = \frac{\sqrt{2}}{2} \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

Here are two ways in which you can employ what you have discovered in this course:

- Since u_0 and u_1 are orthonormal vectors, you know that

$$\begin{aligned}
 x &= \underbrace{\langle u_0 \text{ and } u_1 \text{ are orthonormal} \rangle}_{\substack{(u_0^T x)u_0 \\ \text{component in the direction of } u_0}} + \underbrace{\langle u_1^T x)u_1 \rangle}_{\substack{\text{component in the direction of } u_1}} \\
 &= \langle \text{instantiate } u_0 \text{ and } u_1 \rangle \\
 &= \left(\frac{\sqrt{2}}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -2 \\ 1 \end{pmatrix} \right) u_0 + \left(\frac{\sqrt{2}}{2} \begin{pmatrix} -1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -2 \\ 1 \end{pmatrix} \right) u_1 \\
 &= \langle \text{evaluate} \rangle \\
 &= -\frac{\sqrt{2}}{2}u_0 + \frac{3\sqrt{2}}{2}u_1.
 \end{aligned}$$

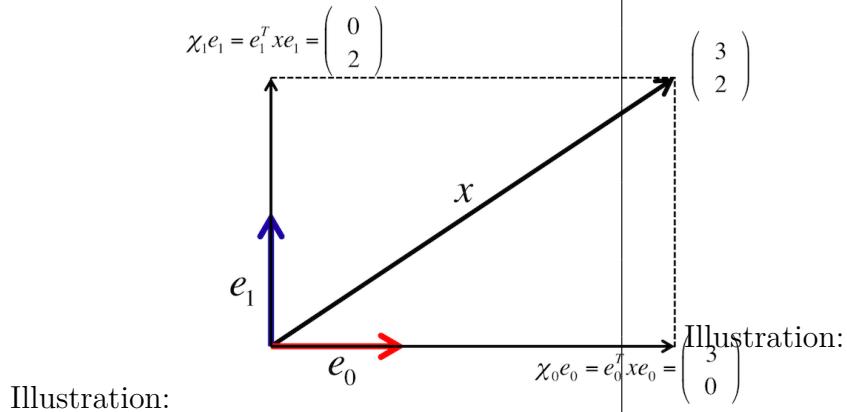
- An alternative way to arrive at the same answer that provides more insight. Let $U = (u_0 | u_1)$. Then

$$\begin{aligned}
 x &= \langle U \text{ is unitary (or orthogonal since it is real valued)} \rangle \\
 UU^T x &= \langle \text{instantiate } U \rangle \\
 (u_0 | u_1) \left(\frac{u_0^T}{u_1^T} \right) x &= \langle \text{matrix-vector multiplication} \rangle \\
 (u_0 | u_1) \left(\frac{u_0^T x}{u_1^T x} \right) &= \langle \text{instantiate} \rangle \\
 (u_0 | u_1) \left(\begin{array}{c} \frac{\sqrt{2}}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -2 \\ 1 \end{pmatrix} \\ \frac{\sqrt{2}}{2} \begin{pmatrix} -1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -2 \\ 1 \end{pmatrix} \end{array} \right) &= \langle \text{evaluate} \rangle \\
 (u_0 | u_1) \left(\begin{array}{c} -\frac{\sqrt{2}}{2} \\ \frac{3\sqrt{2}}{2} \end{array} \right) &= \langle \text{simplify} \rangle \\
 (u_0 | u_1) \left(\begin{array}{c} \frac{\sqrt{2}}{2} \\ 3 \end{array} \right)
 \end{aligned}$$

Below we compare side-by-side how to describe a vector x using the standard basis vectors e_0, \dots, e_{m-1} (on the left) and vectors u_0, \dots, u_{m-1} (on the right):

The vector $x = \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{m-1} \end{pmatrix}$ describes the vector x in terms of the standard basis vectors e_0, \dots, e_{m-1} :

$$\begin{aligned} x &= \langle x = Ix = IIx = II^T x \rangle \\ II^T x &= \langle \text{expose columns of } I \rangle \\ \left(e_0 \mid \dots \mid e_{m-1} \right) \begin{pmatrix} e_0^T \\ \vdots \\ e_{m-1}^T \end{pmatrix} x &= \langle \text{evaluate} \rangle \\ \left(e_0 \mid \dots \mid e_{m-1} \right) \begin{pmatrix} e_0^T x \\ \vdots \\ e_{m-1}^T x \end{pmatrix} &= \langle e_j^T x = \chi_j \rangle \\ \left(e_0 \mid \dots \mid e_{m-1} \right) \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{m-1} \end{pmatrix} &= \langle \text{evaluate} \rangle \\ \chi_0 e_0 + \chi_1 e_1 + \dots + \chi_{m-1} e_{m-1}. & \end{aligned}$$

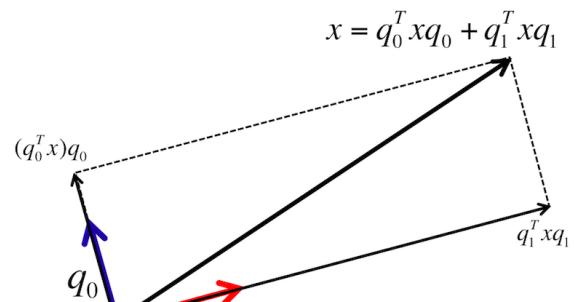


Another way of looking at this is that if u_0, u_1, \dots, u_{m-1} is an orthonormal basis for \mathbb{C}^m , then any $x \in \mathbb{C}^m$ can be written as a linear combination of these vectors:

$$x = \alpha_0 u_0 + \alpha_1 u_1 + \dots + \alpha_{m-1} u_{m-1}.$$

The vector $\hat{x} = \begin{pmatrix} u_0^T x \\ \vdots \\ u_{m-1}^T x \end{pmatrix}$ describes the vector x in terms of the orthonormal basis u_0, \dots, u_{m-1} :

$$\begin{aligned} x &= \langle x = Ix = UU^H x \rangle \\ UU^H x &= \langle \text{expose columns of } U \rangle \\ \left(u_0 \mid \dots \mid u_{m-1} \right) \begin{pmatrix} u_0^H \\ \vdots \\ u_{m-1}^H \end{pmatrix} x &= \langle \text{evaluate} \rangle \\ \left(u_0 \mid \dots \mid u_{m-1} \right) \begin{pmatrix} u_0^H x \\ \vdots \\ u_{m-1}^H x \end{pmatrix} &= \langle \text{evaluate} \rangle \\ u_0^H x u_0 + u_1^H x u_1 + \dots + u_{m-1}^H x u_{m-1}. & \end{aligned}$$



Now,

$$\begin{aligned}
 u_i^H x &= u_i^H (\alpha_0 u_0 + \alpha_1 u_1 + \cdots + \alpha_{i-1} u_{i-1} + \alpha_i u_i + \alpha_{i+1} u_{i+1} + \cdots + \alpha_{m-1} u_{m-1}) \\
 &= \underbrace{\alpha_0 u_i^H u_0}_0 + \underbrace{\alpha_1 u_i^H u_1}_0 + \cdots + \underbrace{\alpha_{i-1} u_i^H u_{i-1}}_0 \\
 &\quad + \underbrace{\alpha_i u_i^H u_i}_1 + \underbrace{\alpha_{i+1} u_i^H u_{i+1}}_0 + \cdots + \underbrace{\alpha_{m-1} u_i^H u_{m-1}}_0 \\
 &= \alpha_i.
 \end{aligned}$$

Thus $u_i^H x = \alpha_i$, the coefficient that multiplies u_i .

Remark 2.2.6.1 The point is that given vector x and unitary matrix U , $U^H x$ computes the coefficients for the orthonormal basis consisting of the columns of matrix U . Unitary matrices allow one to elegantly change between orthonormal bases.

2.2.7 Why we love unitary matrices



YouTube: <https://www.youtube.com/watch?v=d8-AeC3Q8Cw>

In Subsection 1.4.1, we looked at how sensitive solving

$$Ax = b$$

is to a change in the right-hand side

$$A(x + \delta x) = b + \delta b$$

when A is nonsingular. We concluded that

$$\frac{\|\delta x\|}{\|x\|} \leq \underbrace{\|A\| \|A^{-1}\|}_{\kappa(A)} \frac{\|\delta b\|}{\|b\|},$$

when an induced matrix norm is used. Let's look instead at how sensitive matrix-vector multiplication is.

Homework 2.2.7.1 Let $A \in \mathbb{C}^{n \times n}$ be nonsingular and $x \in \mathbb{C}^n$ a nonzero vector. Consider

$$y = Ax \quad \text{and} \quad y + \delta y = A(x + \delta x).$$

Show that

$$\frac{\|\delta y\|}{\|y\|} \leq \underbrace{\|A\| \|A^{-1}\|}_{\kappa(A)} \frac{\|\delta x\|}{\|x\|},$$

where $\|\cdot\|$ is an induced matrix norm.

Solution. Since $x = A^{-1}y$ we know that

$$\|x\| \leq \|A^{-1}\| \|y\|$$

and hence

$$\frac{1}{\|y\|} \leq \|A^{-1}\| \frac{1}{\|x\|}. \quad (2.2.1)$$

Subtracting $y = Ax$ from $y + \delta y = A(x + \delta x)$ yields

$$\delta y = A\delta x$$

and hence

$$\|\delta y\| \leq \|A\| \|\delta x\|. \quad (2.2.2)$$

Combining (2.2.1) and (2.2.2) yields the desired result.

There are choices of x and δx for which the bound is tight.

What does this mean? It means that if as part of an algorithm we use matrix-vector or matrix-matrix multiplication, we risk amplifying relative error by the condition number of the matrix by which we multiply. Now, we saw in Section 1.4 that $1 \leq \kappa(A)$. So, if we there are algorithms that only use matrices for which $\kappa(A) = 1$, then those algorithms don't amplify relative error.

Remark 2.2.7.1 We conclude that unitary matrices, which do not amplify the 2-norm of a vector or matrix, should be our tool of choice, whenever practical.

2.3 The Singular Value Decomposition

2.3.1 The Singular Value Decomposition Theorem



YouTube: <https://www.youtube.com/watch?v=uBo3XAGt24Q>

The following is probably the most important result in linear algebra:

Theorem 2.3.1.1 Singular Value Decomposition Theorem. Given $A \in \mathbb{C}^{m \times n}$ there exist unitary $U \in \mathbb{C}^{m \times m}$, unitary $V \in \mathbb{C}^{n \times n}$, and $\Sigma \in \mathbb{R}^{m \times n}$ such that $A = U\Sigma V^H$. Here

$$\Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right) \text{ with } \Sigma_{TL} = \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{r-1} \end{pmatrix} \quad (2.3.1)$$

and $\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > 0$. The values $\sigma_0, \dots, \sigma_{r-1}$ are called the singular values of matrix A . The columns of U and V are called the left and right singular vectors, respectively.

Recall that in our notation a 0 indicates a matrix "of appropriate size" and that in this setting the zero matrices in (2.3.1) may be 0×0 , $(m-r) \times 0$, and/or $0 \times (n-r)$.

Before proving this theorem, we are going to put some intermediate results in place.

Remark 2.3.1.2 As the course progresses, we will notice that there is a conflict between the notation that explicitly exposes indices, e.g.,

$$U = \begin{pmatrix} u_0 & u_1 & \cdots & u_{n-1} \end{pmatrix}$$

and the notation we use to hide such explicit indexing, which we call the FLAME notation, e.g.,

$$U = \begin{pmatrix} U_0 & | & u_1 & U_2 \end{pmatrix}.$$

The two linked by

$$\left(\begin{array}{cc|cc} u_0 & u_{k-1} & u_k & u_{k+1} & u_{n-1} \\ \hline U_0 & & u_1 & & U_2 \end{array} \right).$$

In algorithms that use explicit indexing, k often is the loop index that identifies where in the matrix or vector the algorithm currently has reached. In the FLAME notation, the index 1 identifies that place. This creates a conflict for the two distinct items that are both indexed with 1, e.g., u_1 in our example here. It is our experience that learners quickly adapt to this and hence have not tried to introduce even more notation that avoids this conflict. In other words: you will almost always be able to tell from context what is meant. The following lemma and its proof illustrate this further.

Lemma 2.3.1.3 Given $A \in \mathbb{C}^{m \times n}$, with $1 \leq n \leq m$ and $A \neq 0$ (the zero matrix), there exist unitary matrices $\tilde{U} \in \mathbb{C}^{m \times m}$ and $\tilde{V} \in \mathbb{C}^{n \times n}$ such that

$$A = \tilde{U} \begin{pmatrix} \sigma_1 & 0 \\ \hline 0 & B \end{pmatrix} \tilde{V}^H, \text{ where } \sigma_1 = \|A\|_2.$$

Proof. In the below proof, it is really important to keep track of when a line is part of the partitioning of a matrix or vector, and when it denotes scalar division.

Choose σ_1 and $\tilde{v}_1 \in \mathbb{C}^n$ such that

- $\|\tilde{v}_1\|_2 = 1$; and
- $\sigma_1 = \|A\tilde{v}_1\|_2 = \|A\|_2$.

In other words, \tilde{v}_1 is the vector that maximizes $\max_{\|x\|_2=1} \|Ax\|_2$.

Let $\tilde{u}_1 = A\tilde{v}_1/\sigma_1$. Then

$$\|\tilde{u}_1\|_2 = \|A\tilde{v}_1\|_2/\sigma_1 = \|A\tilde{v}_1\|_2/\|A\|_2 = \|A\|_2/\|A\|_2 = 1.$$

Choose $\tilde{U}_2 \in \mathbb{C}^{m \times (m-1)}$ and $\tilde{V}_2 \in \mathbb{C}^{n \times (n-1)}$ so that

$$\tilde{U} = \left(\begin{array}{c|c} \tilde{u}_1 & \tilde{U}_2 \end{array} \right) \text{ and } \tilde{V} = \left(\begin{array}{c|c} \tilde{v}_1 & \tilde{V}_2 \end{array} \right)$$

are unitary. Then

$$\begin{aligned} \tilde{U}^H A \tilde{V} &= \left(\begin{array}{c|c} \tilde{u}_1 & \tilde{U}_2 \end{array} \right)^H A \left(\begin{array}{c|c} \tilde{v}_1 & \tilde{V}_2 \end{array} \right) \\ &= \left(\begin{array}{c|c} \tilde{u}_1^H A \tilde{v}_1 & \tilde{u}_1^H A \tilde{V}_2 \\ \hline \tilde{U}_2^H A \tilde{v}_1 & \tilde{U}_2^H A \tilde{V}_2 \end{array} \right) \\ &= \left(\begin{array}{c|c} \tilde{u}_1^H A \tilde{v}_1 & \tilde{u}_1^H A \tilde{V}_2 \\ \hline \sigma_1 \tilde{u}_1^H \tilde{u}_1 & \tilde{U}_2^H A \tilde{V}_2 \end{array} \right) \\ &= \left(\begin{array}{c|c} \tilde{u}_1^H A \tilde{v}_1 & \tilde{u}_1^H A \tilde{V}_2 \\ \hline 0 & B \end{array} \right), \end{aligned}$$

where $w = \tilde{V}_2^H A^H \tilde{u}_1$ and $B = \tilde{U}_2^H A \tilde{V}_2$.

We will now argue that $w = 0$, the zero vector of appropriate size:

$$\begin{aligned}
 \sigma_1^2 &= <\text{assumption}> \\
 \|A\|_2^2 &= <\text{2-norm is invariant under multiplication by unitary matrix}> \\
 \|\tilde{U}^H A \tilde{V}\|_2^2 &= <\text{definition of } \|\cdot\|_2> \\
 \max_{x \neq 0} \frac{\|\tilde{U}^H A \tilde{V} x\|_2^2}{\|x\|_2^2} &= <\text{see above}> \\
 \max_{x \neq 0} \frac{\left\| \begin{pmatrix} \sigma_1 & w^H \\ 0 & B \end{pmatrix} x \right\|_2^2}{\|x\|_2^2} &\geq <\text{x replaced by specific vector}> \\
 \left\| \begin{pmatrix} \sigma_1 & w^H \\ 0 & B \end{pmatrix} \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2^2 &= <\text{multiply out numerator}> \\
 \left\| \begin{pmatrix} \sigma_1^2 + w^H w \\ Bw \end{pmatrix} \right\|_2^2 &\geq <\left\| \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} \right\|_2^2 = \|\psi_1\|_2^2 + \|y_2\|_2^2 \geq \|\psi_1\|_2^2; \left\| \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2^2 = \sigma_1^2 + w^H w > \\
 (\sigma_1^2 + w^H w)^2 / (\sigma_1^2 + w^H w) &= <\text{algebra}> \\
 \sigma_1^2 + w^H w. &
 \end{aligned}$$

Thus $\sigma_1^2 \geq \sigma_1^2 + w^H w$ which means that $w = 0$ (the zero vector) and $\tilde{U}^H A \tilde{V} = \begin{pmatrix} \sigma_1 & 0 \\ 0 & B \end{pmatrix}$ so that $A = \tilde{U} \begin{pmatrix} \sigma_1 & 0 \\ 0 & B \end{pmatrix} \tilde{V}^H$. ■

Hopefully you can see where this is going: If one can recursively find that $B = U_B \Sigma_B V_B^H$,

then

$$\begin{aligned}
 A &= \tilde{U} \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array} \right) \tilde{V}^H \\
 &= \tilde{U} \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & U_B \Sigma_B V_B^H \end{array} \right) \tilde{V}^H \\
 &= \underbrace{\tilde{U} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & U_B \end{array} \right)}_{U} \underbrace{\left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & \Sigma_B \end{array} \right)}_{\Sigma} \underbrace{\left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & V_B^H \end{array} \right) \tilde{V}^H}_{V^H} \\
 &= \underbrace{\tilde{U} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & U_B \end{array} \right)}_{U} \underbrace{\left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & \Sigma_B \end{array} \right)}_{\Sigma} \underbrace{\left(\tilde{V} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & V_B \end{array} \right) \right)^H}_{V^H}.
 \end{aligned}$$

The next exercise provides the insight that the values on the diagonal of Σ will be ordered from largest to smallest.

Homework 2.3.1.1 Let $A \in \mathbb{C}^{m \times n}$ with $A = \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array} \right)$ and assume that $\|A\|_2 = \sigma_1$.

ALWAYS/SOMETIMES/NEVER: $\|B\|_2 \leq \sigma_1$.

Solution. We will employ a proof by contradiction. Assume that $\|B\|_2 > \sigma_1$. Then there exists a vector z with $\|z\|_2 = 1$ such that $\|B\|_2 = \|Bz\|_2 = \max_{\|x\|_2=1} \|Bx\|_2$. But then

$$\begin{aligned}
 \|A\|_2 &= <\text{definition}> \\
 \max_{\|x\|_2=1} \|Ax\|_2 &= <\text{pick a specific vector with } 2-\text{norm equal to one}> \\
 \left\| A \left(\begin{array}{c} 0 \\ z \end{array} \right) \right\|_2 &= <\text{instantiate } A> \\
 \left\| \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array} \right) \left(\begin{array}{c} 0 \\ z \end{array} \right) \right\|_2 &= <\text{partitioned matrix-vector multiplication}> \\
 \left\| \left(\begin{array}{c} 0 \\ Bz \end{array} \right) \right\|_2 &= <\left\| \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} \right\|_2^2 = \|y_0\|_2^2 + \|y_1\|_2^2> \\
 \|Bz\|_2 &= <\text{assumption about } z> \\
 \|B\|_2 &> <\text{assumption}> \\
 &\sigma_1.
 \end{aligned}$$

which is a contradiction.

Hence $\|B\|_2 \leq \sigma_1$.

We are now ready to prove the Singular Value Decomposition Theorem.

Proof of Singular Value Decomposition Theorem for $n \leq m$. We will prove this for $m \geq n$, leaving the case where $m \leq n$ as an exercise.

Proof by induction: Since $m \geq n$, we select m to be arbitrary and induct on n .

- Base case: $n = 1$.

In this case $A = \begin{pmatrix} a_1 \end{pmatrix}$ where $a_1 \in \mathbb{C}^m$ is its only column.

Case 1: $a_1 = 0$ (the zero vector).

Then

$$A = \begin{pmatrix} 0 \end{pmatrix} = \underbrace{I_{m \times m}}_U \begin{pmatrix} \vdots \\ 0 \end{pmatrix} \underbrace{I_{1 \times 1}}_{V^H}$$

so that $U = I_{m \times m}$, $V = I_{1 \times 1}$, and Σ_{TL} is an empty matrix.

Case 2: $a_1 \neq 0$.

Then

$$A = \begin{pmatrix} a_1 \end{pmatrix} = \begin{pmatrix} u_1 \end{pmatrix} (\|a_1\|_2)$$

where $u_1 = a_1 / \|a_1\|_2$. Choose $U_2 \in \mathbb{C}^{m \times (m-1)}$ so that $U = \begin{pmatrix} u_1 & | & U_2 \end{pmatrix}$ is unitary. Then

$$\begin{aligned} A &= \begin{pmatrix} a_1 \end{pmatrix} \\ &= \begin{pmatrix} u_1 \end{pmatrix} (\|a_1\|_2) \\ &= \begin{pmatrix} u_1 & | & U_2 \end{pmatrix} \begin{pmatrix} \|a_1\|_2 & | \\ 0 & \vdots \end{pmatrix} \begin{pmatrix} 1 \end{pmatrix}^H \\ &= U \Sigma V^H, \end{aligned}$$

where

- $U = \begin{pmatrix} u_0 & | & U_1 \end{pmatrix}$,
- $\Sigma = \begin{pmatrix} \Sigma_{TL} & | \\ 0 & \vdots \end{pmatrix}$ with $\Sigma_{TL} = \begin{pmatrix} \sigma_1 \end{pmatrix}$ and $\sigma_1 = \|a_1\|_2 = \|A\|_2$
- $V = \begin{pmatrix} 1 \end{pmatrix}$.

- Inductive step:

Assume the result is true for matrices with $1 \leq k$ columns. Show that it is true for matrices with $k+1$ columns.

Let $A \in \mathbb{C}^{m \times (k+1)}$ with $1 \leq k < n$.

Case 1: $A = 0$ (the zero matrix)

Then

$$A = I_{m \times m} \begin{pmatrix} & & | \\ & \vdots & \vdots \\ 0_{m \times (k+1)} & & \vdots \end{pmatrix} I_{(k+1) \times (k+1)}$$

so that $U = I_{m \times m}$, $V = I_{(k+1) \times (k+1)}$, and Σ_{TL} is an empty matrix.

Case 2: $A \neq 0$.

Then $\|A\|_2 \neq 0$. By Lemma 2.3.1.3, we know that there exist unitary $\tilde{U} \in \mathbb{C}^{m \times m}$ and $\tilde{V} \in \mathbb{C}^{(k+1) \times (k+1)}$ such that $A = \tilde{U} \begin{pmatrix} \sigma_1 & 0 \\ 0 & B \end{pmatrix} \tilde{V}$ with $\sigma_1 = \|A\|_2$.

By the inductive hypothesis, there exist unitary $\check{U}_B \in \mathbb{C}^{(m-1) \times (m-1)}$, unitary $\check{V}_B \in \mathbb{C}^{k \times k}$, and $\check{\Sigma}_B \in \mathbb{R}^{(m-1) \times k}$ such that $B = \check{U}_B \check{\Sigma}_B \check{V}_B^H$ where $\check{\Sigma}_B = \begin{pmatrix} \check{\Sigma}_{TL} & 0 \\ 0 & 0 \end{pmatrix}$, $\check{\Sigma}_{TL} = \text{diag}(\sigma_2, \dots, \sigma_{r-1})$, and $\sigma_2 \geq \dots \geq \sigma_{r-1} > 0$.

Now, let

$$U = \tilde{U} \begin{pmatrix} 1 & 0 \\ 0 & \check{U}_B \end{pmatrix}, V = \tilde{V} \begin{pmatrix} 1 & 0 \\ 0 & \check{V}_B \end{pmatrix}, \text{ and } \Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \check{\Sigma}_B \end{pmatrix}.$$

(There are some really tough to see "checks" in the definition of U , V , and Σ !) Then $A = U\Sigma V^H$ where U , V , and Σ have the desired properties. Key here is that $\sigma_1 = \|A\|_2 \geq \|B\|_2$ which means that $\sigma_1 \geq \sigma_2$.

- By the Principle of Mathematical Induction the result holds for all matrices $A \in \mathbb{C}^{m \times n}$ with $m \geq n$.

■

Homework 2.3.1.2 Let $\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{n-1})$. ALWAYS/SOMETIMES/NEVER: $\|\Sigma\|_2 = \max_{i=0}^{n-1} |\sigma_i|$.

Answer. ALWAYS

Now prove it.

Solution. Yes, you have seen this before, in Homework 1.3.5.1. We repeat it here because of its importance to this topic.

$$\begin{aligned}
\|\Sigma\|_2^2 &= \max_{\|x\|_2=1} \|\Sigma x\|_2^2 \\
&= \max_{\|x\|_2=1} \left\| \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{n-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \right\|_2^2 \\
&= \max_{\|x\|_2=1} \left\| \begin{pmatrix} \sigma_0 \chi_0 \\ \sigma_1 \chi_1 \\ \vdots \\ \sigma_{n-1} \chi_{n-1} \end{pmatrix} \right\|_2^2 \\
&= \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} |\sigma_j \chi_j|^2 \right] \\
&= \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} [|\sigma_j|^2 |\chi_j|^2] \right] \\
&\leq \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} \left[\max_{i=0}^{n-1} |\sigma_i|^2 |\chi_j|^2 \right] \right] \\
&= \max_{\|x\|_2=1} \left[\max_{i=0}^{n-1} |\sigma_i|^2 \sum_{j=0}^{n-1} |\chi_j|^2 \right] \\
&= \left(\max_{i=0}^{n-1} |\sigma_i| \right)^2 \max_{\|x\|_2=1} \|x\|_2^2 \\
&= \left(\max_{i=0}^{n-1} |\sigma_i| \right)^2.
\end{aligned}$$

so that $\|\Sigma\|_2 \leq \max_{i=0}^{n-1} |\sigma_i|$.

Also, choose j so that $|\sigma_j| = \max_{i=0}^{n-1} |\sigma_i|$. Then

$$\|\Sigma\|_2 = \max_{\|x\|_2=1} \|\Sigma x\|_2 \geq \|\Sigma e_j\|_2 = \|\sigma_j e_j\|_2 = |\sigma_j| \|e_j\|_2 = |\sigma_j| = \max_{i=0}^{n-1} |\sigma_i|.$$

so that $\max_{i=0}^{n-1} |\sigma_i| \leq \|\Sigma\|_2 \leq \max_{i=0}^{n-1} |\sigma_i|$, which implies that $\|\Sigma\|_2 = \max_{i=0}^{n-1} |\sigma_i|$.

Homework 2.3.1.3 Assume that $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ are unitary matrices. Let $A, B \in \mathbb{C}^{m \times n}$ with $B = U A V^H$. Show that the singular values of A equal the singular values of B .

Solution. Let $A = U_A \Sigma_A V_A^H$ be the SVD of A . Then $B = U U_A \Sigma_A V_A^H V^H = (U U_A) \Sigma_A (V V_A)^H$ where both $U U_A$ and $V V_A$ are unitary. This gives us the SVD for B and it shows that the singular values of B equal the singular values of A .

Homework 2.3.1.4 Let $A \in \mathbb{C}^{m \times n}$ with $n \leq m$ and $A = U \Sigma V^H$ be its SVD.

ALWAYS/SOMETIMES/NEVER: $A^H = V \Sigma^T U^H$.

Answer. ALWAYS

Solution.

$$A^H = (U \Sigma V^H)^H = (V^H)^H \Sigma^T U^H = V \Sigma^T U^H$$

since Σ is real valued. Notice that Σ is only "sort of diagonal" (it is possibly rectangular) which is why $\Sigma^T \neq \Sigma$.

Homework 2.3.1.5 Prove the Singular Value Decomposition Theorem for $m \leq n$.

Hint. Consider the SVD of $B = A^H$

Solution. Let $B = A^H$. Since it is $n \times m$ with $n \geq m$ its SVD exists: $B = U_B \Sigma_B V_B^H$.

Then $A = B^H = V_B \Sigma_B^T U_B^H$ and hence $A = U \Sigma V^H$ with $U = V_B$, $\Sigma = \Sigma_B^T$, and $V = U_B$.

I believe the following video has material that is better presented in second video of 2.3.2.



YouTube: <https://www.youtube.com/watch?v=ZYzqTC5LeLs>

2.3.2 Geometric interpretation



YouTube: <https://www.youtube.com/watch?v=XKhCTtX1z6A>

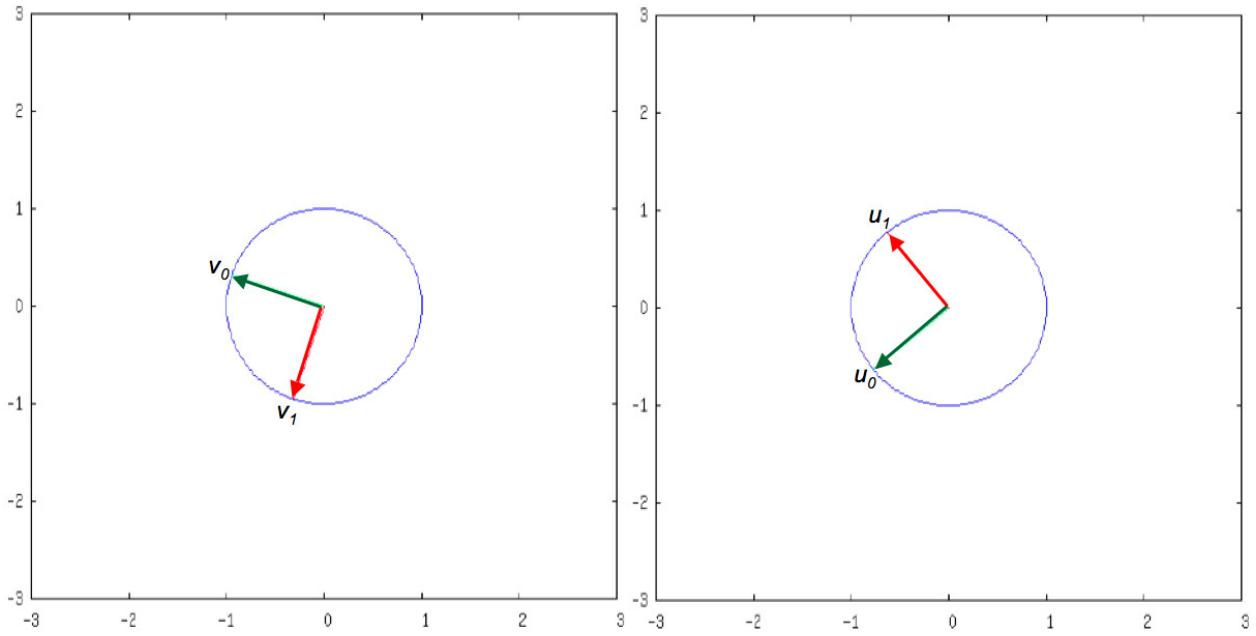
We will now illustrate what the SVD Theorem tells us about matrix-vector multiplication (linear transformations) by examining the case where $A \in \mathbb{R}^{2 \times 2}$. Let $A = U \Sigma V^T$ be its SVD. (Notice that all matrices are now real valued, and hence $V^H = V^T$.) Partition

$$A = \left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} \sigma_0 & 0 \\ 0 & \sigma_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T.$$

Since U and V are unitary matrices, $\{u_0, u_1\}$ and $\{v_0, v_1\}$ form orthonormal bases for the range and domain of A , respectively:

\mathbb{R}^2 : Domain of A :

\mathbb{R}^2 : Range (codomain) of A :



Let us manipulate the decomposition a little:

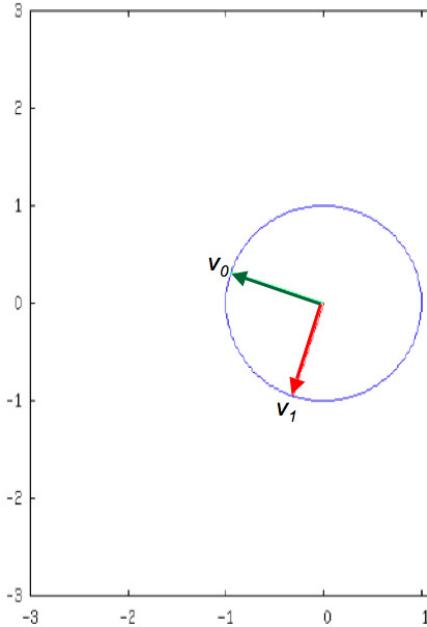
$$\begin{aligned}
 A &= \left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} \sigma_0 & 0 \\ 0 & \sigma_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T \\
 &= \left[\left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} \sigma_0 & 0 \\ 0 & \sigma_1 \end{array} \right) \right] \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T \\
 &= \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T.
 \end{aligned}$$

Now let us look at how A transforms v_0 and v_1 :

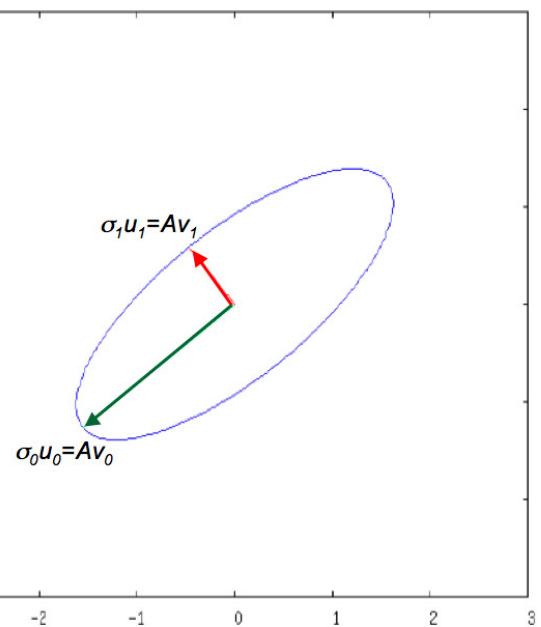
$$Av_0 = \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T v_0 = \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c} 1 \\ 0 \end{array} \right) = \sigma_0 u_0$$

and similarly $Av_1 = \sigma_1 u_1$. This motivates the pictures in [Figure 2.3.2.1](#).

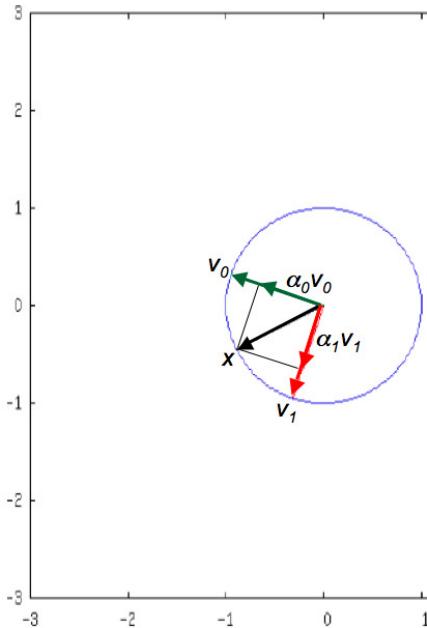
\mathbb{R}^2 : Domain of A :



\mathbb{R}^2 : Range (codomain) of A :



\mathbb{R}^2 : Domain of A :



\mathbb{R}^2 : Range (codomain) of A :

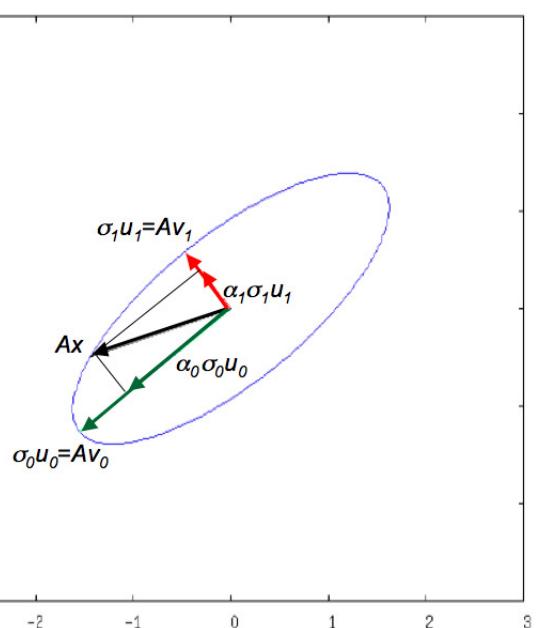


Figure 2.3.2.1 Illustration of how orthonormal vectors v_0 and v_1 are transformed by matrix $A = U\Sigma V$.

Next, let us look at how A transforms any vector with (Euclidean) unit length. Notice that $x = \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}$ means that

$$x = \chi_0 e_0 + \chi_1 e_1,$$

where e_0 and e_1 are the unit basis vectors. Thus, χ_0 and χ_1 are the coefficients when x is expressed using e_0 and e_1 as basis. However, we can also express x in the basis given by v_0 and v_1 :

$$\begin{aligned} x &= \underbrace{VV^T}_{I} x = \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T x = \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \left(\begin{array}{c} \frac{v_0^T x}{v_1^T x} \\ \frac{v_1^T x}{v_1^T x} \end{array} \right) \\ &= \underbrace{v_0^T x}_{\alpha_0} v_0 + \underbrace{v_1^T x}_{\alpha_1} v_1 = \alpha_0 v_0 + \alpha_1 v_1 = \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \left(\begin{array}{c} \alpha_0 \\ \alpha_1 \end{array} \right). \end{aligned}$$

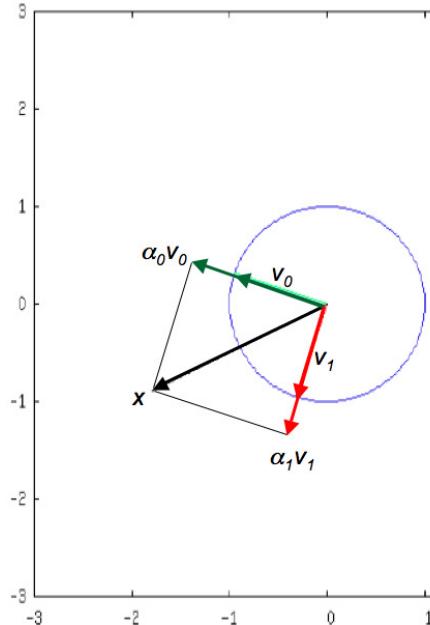
Thus, in the basis formed by v_0 and v_1 , its coefficients are α_0 and α_1 . Now,

$$\begin{aligned} Ax &= \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T x \\ &= \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \left(\begin{array}{c} \alpha_0 \\ \alpha_1 \end{array} \right) \\ &= \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c} \alpha_0 \\ \alpha_1 \end{array} \right) = \alpha_0 \sigma_0 u_0 + \alpha_1 \sigma_1 u_1. \end{aligned}$$

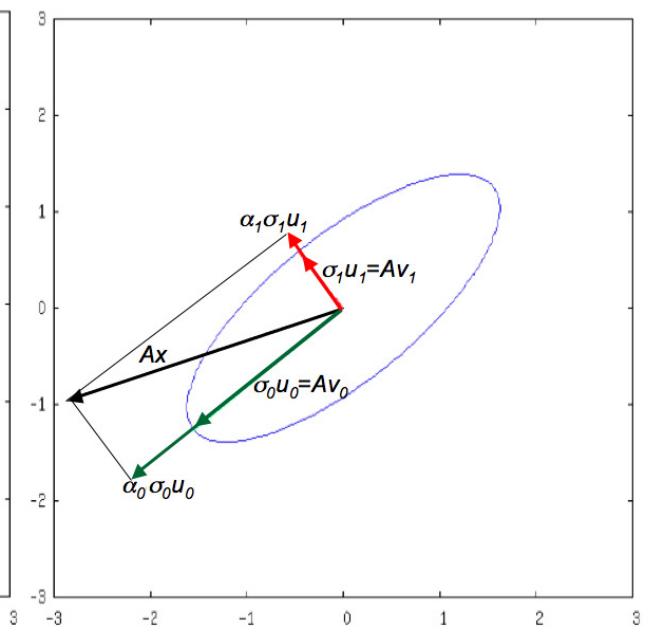
This is illustrated by the following picture, which also captures the fact that the unit ball is mapped to an oval with major axis equal to $\sigma_0 = \|A\|_2$ and minor axis equal to σ_1 , as illustrated in [Figure 2.3.2.1](#) (bottom).

Finally, we show the same insights for general vector x (not necessarily of unit length):

\mathbb{R}^2 : Domain of A :



\mathbb{R}^2 : Range (codomain) of A :



Another observation is that if one picks the right basis for the domain and codomain, then the computation Ax simplifies to a matrix multiplication with a diagonal matrix. Let

us again illustrate this for nonsingular $A \in \mathbb{R}^{2 \times 2}$ with

$$A = \underbrace{\begin{pmatrix} u_0 & u_1 \end{pmatrix}}_U \underbrace{\begin{pmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} v_0 & v_1 \end{pmatrix}}_V^T.$$

Now, if we chose to express y using u_0 and u_1 as the basis and express x using v_0 and v_1 as the basis, then

$$\begin{aligned} \underbrace{UU^T}_I y &= U \underbrace{U^T y}_{\hat{y}} = (u_0^T y)u_0 + (u_1^T y)u_1 \\ &= \begin{pmatrix} u_0 & u_1 \end{pmatrix} \begin{pmatrix} u_0^T y \\ u_1^T y \end{pmatrix} = U \underbrace{\begin{pmatrix} \hat{\psi}_0 \\ \hat{\psi}_1 \end{pmatrix}}_{\hat{y}} \\ \underbrace{VV^T}_I x &= V \underbrace{V^T x}_{\hat{x}} = (v_0^T x)v_0 + (v_1^T x)v_1 \\ &= \begin{pmatrix} v_0 & v_1 \end{pmatrix} \begin{pmatrix} v_0^T x \\ v_1^T x \end{pmatrix} = V \underbrace{\begin{pmatrix} \hat{\chi}_0 \\ \hat{\chi}_1 \end{pmatrix}}_{\hat{x}}. \end{aligned}$$

If $y = Ax$ then

$$U \underbrace{U^T y}_{\hat{y}} = \underbrace{U\Sigma V^T x}_{Ax} = U\Sigma \hat{x}$$

so that

$$\hat{y} = \Sigma \hat{x}$$

and

$$\begin{pmatrix} \hat{\psi}_0 \\ \hat{\psi}_1 \end{pmatrix} = \begin{pmatrix} \sigma_0 \hat{\chi}_0 \\ \sigma_1 \hat{\chi}_1 \end{pmatrix}.$$

Remark 2.3.2.2 The above discussion shows that if one transforms the input vector x and output vector y into the right bases, then the computation $y := Ax$ can be computed with a diagonal matrix instead: $\hat{y} := \Sigma \hat{x}$. Also, solving $Ax = y$ for x can be computed by multiplying with the inverse of the diagonal matrix: $\hat{x} := \Sigma^{-1} \hat{y}$.

These observations generalize to $A \in \mathbb{C}^{m \times n}$: If

$$y = Ax$$

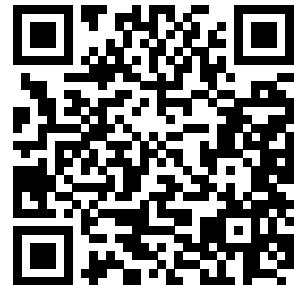
then

$$U^H y = U^H A \underbrace{VV^H}_I x$$

so that

$$\underbrace{U^H y}_{\hat{y}} = \Sigma \underbrace{V^H x}_{\hat{x}}$$

(Σ is a rectangular "diagonal" matrix.)



YouTube: <https://www.youtube.com/watch?v=1LpK0dbFX1g>

2.3.3 An "algorithm" for computing the SVD

We really should have created a video for this section. Those who have taken our "Programming for Correctness" course will recognize what we are trying to describe here. Regardless, you can safely skip this unit without permanent (or even temporary) damage to your linear algebra understanding.

In this unit, we show how the insights from the last unit can be molded into an "algorithm" for computing the SVD. We put algorithm in quotes because while the details of the algorithm mathematically exist, they are actually very difficult to compute in practice. So, this is *not* a practical algorithm. We will not discuss a practical algorithm until the very end of the course, in (((section to be determined))).

We observed that, starting with matrix A , we can compute one step towards the SVD. If we overwrite A with the intermediate results, this means that after one step

$$\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} = (\tilde{u}_1 \ \tilde{U}_2)^H \begin{pmatrix} \hat{\alpha}_{11} & \hat{a}_{12}^T \\ \hat{a}_{21} & \hat{A}_{22} \end{pmatrix} (\tilde{v}_1 \ \tilde{V}_2) = \begin{pmatrix} \sigma_{11} & 0 \\ 0 & B \end{pmatrix},$$

where \hat{A} allows us to refer to the original contents of A .

In our proof of [Theorem 2.3.1.1](#), we then said that the SVD of B , $B = U_B \Sigma_B V_B^H$ could be computed, and the desired U and V can then be created by computing $U = \tilde{U} U_B$ and $V = \tilde{V} V_B$.

Alternatively, one can accumulate U and V every time a new singular value is exposed. In this approach, you start by setting $U = I_{m \times m}$ and $V = I_{n \times n}$. Upon completing the first step (which computes the first singular value), one multiplies U and V from the right with the computed \tilde{U} and \tilde{V} :

$$\begin{aligned} U &:= U \tilde{U} \\ V &:= V \tilde{V}. \end{aligned}$$

Now, every time another singular value is computed in future steps, the corresponding unitary matrices are similarly accumulated into U and V .

To explain this more completely, assume that the process has proceeded for k steps to

the point where

$$\begin{aligned} U &= \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \in \mathbb{C}^{m \times m} && \text{with } U_L \in \mathbb{C}^{m \times k} \\ V &= \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \in \mathbb{C}^{n \times m} && \text{with } V_L \in \mathbb{C}^{n \times k} \\ A &= \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) && \text{with } A_{TL} \in \mathbb{C}^{k \times k}, \end{aligned}$$

where the current contents of A are

$$\begin{aligned} \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) &= \left(\begin{array}{c|c} U_L & U_R \end{array} \right)^H \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \\ &= \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array} \right). \end{aligned}$$

This means that in the current step we need to update the contents of A_{BR} with

$$\tilde{U}^H A \tilde{V} = \left(\begin{array}{c|c} \sigma_{11} & 0 \\ \hline 0 & \tilde{B} \end{array} \right)$$

and update

$$\begin{aligned} \left(\begin{array}{c|c} U_L & U_R \end{array} \right) &:= \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} I_{k \times k} & 0 \\ \hline 0 & \tilde{U} \end{array} \right) \\ \left(\begin{array}{c|c} V_L & V_R \end{array} \right) &:= \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c|c} I_{k \times k} & 0 \\ \hline 0 & \tilde{V} \end{array} \right), \end{aligned}$$

which simplify to

$$U_{BR} := U_{BR} \tilde{U} \text{ and } V_{BR} := V_{BR} \tilde{V}.$$

At that point, A_{TL} is expanded by one row and column, and the left-most columns of U_R and V_R are moved to U_L and V_L , respectively. If A_{BR} ever contains a zero matrix, the process completes with A overwritten with $\Sigma = U^H \tilde{V}$. These observations, with all details, are captured in [Figure 2.3.3.1](#). In that figure, the boxes in yellow are assertions that capture the current contents of the variables. Those familiar with proving loops correct will recognize the first and last such box as the precondition and postcondition for the operation and

$$\begin{aligned} \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) &= \left(\begin{array}{c|c} U_L & U_R \end{array} \right)^H \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \\ &= \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array} \right) \end{aligned}$$

as the loop-invariant that can be used to prove the correctness of the loop via a proof by induction.

$A = \widehat{A}$
$U := I_{m \times m}; V := I_{n \times n}$ $A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), U \rightarrow \left(\begin{array}{c c} U_L & U_R \end{array} \right), V \rightarrow \left(\begin{array}{c c} V_L & V_R \end{array} \right)$ where A_{TL} is 0×0 , U_L is $m \times 0$, V_L is $n \times 0$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} U_L & U_R \end{array} \right)^H \left(\begin{array}{c c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) \left(\begin{array}{c c} V_L & V_R \end{array} \right) = \left(\begin{array}{c c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array} \right)$ (here Σ_{TL} is 0×0)
while $\ B\ _2 \neq 0$ $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} U_L & U_R \end{array} \right)^H \left(\begin{array}{c c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) \left(\begin{array}{c c} V_L & V_R \end{array} \right) = \left(\begin{array}{c c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array} \right) \wedge \ B\ _2 \neq 0$
$\sigma_{11} = \ A_{BR}\ _2$ if $\sigma_{11} = 0$ break (exit loop) pick \tilde{v}_1 s.t. $\ \tilde{v}_1\ _2 = 1$ and $\ A_{BR}\tilde{v}_1\ = \ A_{BR}\ _2 (= \sigma_{11})$ $\tilde{u}_1 := A_{BR}\tilde{v}_1/\sigma_{11}$ pick \tilde{V}_2 and \tilde{U}_2 s.t. $\tilde{V} = \left(\begin{array}{c c} \tilde{v}_1 & \tilde{V}_2 \end{array} \right)$ and $\tilde{U} = \left(\begin{array}{c c} \tilde{u}_1 & \tilde{U}_2 \end{array} \right)$ are unitary $V_R := V_R\tilde{V}; U_R := U_R\tilde{U}$ $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c c} U_L & U_R \end{array} \right) \rightarrow \left(\begin{array}{c cc} U_0 & u_1 & U_2 \end{array} \right), \left(\begin{array}{c c} V_L & V_R \end{array} \right) \rightarrow \left(\begin{array}{c cc} V_0 & v_1 & V_2 \end{array} \right)$ $\left(\begin{array}{cc} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{cc} \sigma_{11} & 0 \\ 0 & \tilde{U}_2^H A_{BR} \tilde{V}_2 \end{array} \right)$ $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c c} U_L & U_R \end{array} \right) \leftarrow \left(\begin{array}{c cc} U_0 & u_1 & U_2 \end{array} \right), \left(\begin{array}{c c} V_L & V_R \end{array} \right) \leftarrow \left(\begin{array}{c cc} V_0 & v_1 & V_2 \end{array} \right)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} U_L & U_R \end{array} \right)^H \left(\begin{array}{c c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) \left(\begin{array}{c c} V_L & V_R \end{array} \right) = \left(\begin{array}{c c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array} \right)$
endwhile
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} U_L & U_R \end{array} \right)^H \left(\begin{array}{c c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) \left(\begin{array}{c c} V_L & V_R \end{array} \right) = \left(\begin{array}{c c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array} \right) \wedge \ B\ _2 = 0$
$\underbrace{\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)}_A = \underbrace{\left(\begin{array}{c c} U_L & U_R \end{array} \right)}_{U^H} \underbrace{\left(\begin{array}{c c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right)}_{\widehat{A}} \underbrace{\left(\begin{array}{c c} V_L & V_R \end{array} \right)}_V = \underbrace{\left(\begin{array}{c c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right)}_{\Sigma}$

Figure 2.3.3.1 Algorithm for computing the SVD of A , overwriting A with Σ . In the yellow boxes are assertions regarding the contents of the various matrices.

The reason this algorithm is not practical is that many of the steps are easy to state mathematically, but difficult (computationally expensive) to compute in practice. In particular:

- Computing $\|A_{BR}\|_2$ is tricky and as a result, so is computing \tilde{v}_1 .
- Given a vector, determining a unitary matrix with that vector as its first column is computationally expensive.
- Assuming for simplicity that $m = n$, even if all other computations were free, computing the product $A_{22} := \tilde{U}_2^H A_{BR} \tilde{V}_2$ requires $O((m - k)^3)$ operations. This means that the entire algorithm requires $O(m^4)$ computations, which is prohibitively expensive when n gets large. (We will see that most practical algorithms discussed in this course cost $O(m^3)$ operations or less.)

Later in this course, we will discuss an algorithm that has an effective cost of $O(m^3)$ (when $m = n$).

Ponder This 2.3.3.1 An implementation of the "algorithm" in Figure 2.3.3.1, using our FLAME API for Matlab (FLAME@lab) [5] that allows the code to closely resemble the algorithm as we present it, is given in mySVD.m (Assignments/Week02/matlab/mySVD.m). This implementation depends on routines in subdirectory Assignments/flameatlab being in the path. Examine this code. What do you notice? Execute it with

```
m = 5;
n = 4;
A = rand( m, n );      % create m x n random matrix
[ U, Sigma, V ] = mySVD( A )
```

Then check whether the resulting matrices form the SVD:

```
norm( A - U * Sigma * V' )
```

and whether U and V are unitary

```
norm( eye( n,n ) - V' * V )
norm( eye( m,m ) - U' * U )
```

2.3.4 The Reduced Singular Value Decomposition



YouTube: <https://www.youtube.com/watch?v=HAAh4IsIdsY>

Corollary 2.3.4.1 Reduced Singular Value Decomposition. Let $A \in \mathbb{C}^{m \times n}$ and $r = \text{rank}(A)$. There exist orthonormal matrix $U_L \in \mathbb{C}^{m \times r}$, orthonormal matrix $V_L \in \mathbb{C}^{n \times r}$, and matrix $\Sigma_{TL} \in \mathbb{R}^{r \times r}$ with $\Sigma_{TL} = \text{diag}(\sigma_0, \dots, \sigma_{r-1})$ and $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{r-1} > 0$ such that $A = U_L \Sigma_{TL} V_L^H$.

Homework 2.3.4.1 Prove the above corollary.

Solution. Let $A = U \Sigma V^H = (U_L | U_R) \begin{pmatrix} \Sigma_{TL} & 0 \\ 0 & 0 \end{pmatrix} (V_L | V_R)^H$ be the SVD of A , where $U_L \in \mathbb{C}^{m \times r}$, $V_L \in \mathbb{C}^{n \times r}$ and $\Sigma_{TL} \in \mathbb{R}^{r \times r}$ with $\Sigma_{TL} = \text{diag}(\sigma_0, \sigma_1, \dots, \sigma_{r-1})$ and $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{r-1} > 0$. Then

$$\begin{aligned} A &= <\text{SVD of } A> \\ U \Sigma V^T &= <\text{Partitioning}> \\ (U_L | U_R) \begin{pmatrix} \Sigma_{TL} & 0 \\ 0 & 0 \end{pmatrix} (V_L | V_R)^H &= <\text{partitioned matrix-matrix multiplication}> \\ U_L \Sigma_{TL} V_L^H. \end{aligned}$$

Corollary 2.3.4.2 Let $A = U_L \Sigma_{TL} V_L^H$ be the Reduced SVD with $U_L = (u_0 | \dots | u_{r-1})$, $V_L = (v_0 | \dots | v_{r-1})$, and $\Sigma_{TL} = \begin{pmatrix} \sigma_0 & & & \\ & \ddots & & \\ & & \sigma_{r-1} & \end{pmatrix}$. Then

$$A = \sigma_0 u_0 v_0^H + \dots + \sigma_{r-1} u_{r-1} v_{r-1}^H.$$

Remark 2.3.4.3 This last result establishes that any matrix A with rank r can be written as a linear combination of r outer products:

$$A = \underbrace{\sigma_0 u_0 v_0^H}_{\sigma_0 | \dots} + \underbrace{\sigma_1 u_1 v_1^H}_{\sigma_1 | \dots} + \dots + \underbrace{\sigma_{r-1} u_{r-1} v_{r-1}^H}_{\sigma_{r-1} | \dots}.$$

2.3.5 SVD of nonsingular matrices



YouTube: <https://www.youtube.com/watch?v=5Gvmtll5T3k>

Homework 2.3.5.1 Let $A \in \mathbb{C}^{m \times m}$ and $A = U\Sigma V^H$ be its SVD.

TRUE/FALSE: A is nonsingular if and only if Σ is nonsingular.

Answer. TRUE

Solution. $\Sigma = U^H A V$. The product of square matrices is nonsingular if and only if each individual matrix is nonsingular. Since U and V are unitary, they are nonsingular.

Homework 2.3.5.2 Let $A \in \mathbb{C}^{m \times m}$ and $A = U\Sigma V^H$ be its SVD with

$$\Sigma = \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{m-1} \end{pmatrix}$$

TRUE/FALSE: A is nonsingular if and only if $\sigma_{m-1} \neq 0$.

Answer. TRUE

Solution. By the last homework, A is nonsingular if and only if Σ is nonsingular. A diagonal matrix is nonsingular if and only if its diagonal elements are all nonzero. $\sigma_0 \geq \cdots \geq \sigma_{m-1} > 0$. Hence the diagonal elements of Σ are nonzero if and only if $\sigma_{m-1} \neq 0$.

Homework 2.3.5.3 Let $A \in \mathbb{C}^{m \times m}$ be nonsingular and $A = U\Sigma V^H$ be its SVD.

ALWAYS/SOMETIMES/NEVER: The SVD of A^{-1} equals $V\Sigma^{-1}U^H$.

Answer. SOMETIMES

Explain it!

Solution. It would seem that the answer is ALWAYS: $A^{-1} = (U\Sigma V^H)^{-1} = (V^H)^{-1}\Sigma^{-1}U^{-1} =$

$V\Sigma^{-1}U^H$ with

$$\begin{aligned}\Sigma^{-1} &= \left\langle \begin{array}{c|c|c|c} \sigma_0 & 0 & \cdots & 0 \\ \hline 0 & \sigma_1 & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \sigma_{m-1} \end{array} \right\rangle^{-1} \\ &= \left\langle \begin{array}{c|c|c|c} 1/\sigma_0 & 0 & \cdots & 0 \\ \hline 0 & 1/\sigma_1 & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & 1/\sigma_{m-1} \end{array} \right\rangle.\end{aligned}$$

However, the SVD requires the diagonal elements to be positive and ordered from largest to smallest.

So, only if $\sigma_0 = \sigma_1 = \cdots = \sigma_{m-1}$ is it the case that $V\Sigma^{-1}U^H$ is the SVD of A^{-1} . In other words, when $\Sigma = \sigma_0 I$.

Homework 2.3.5.4 Let $A \in \mathbb{C}^{m \times m}$ be nonsingular and

$$\begin{aligned}A &= U\Sigma V^H \\ &= \left(\begin{array}{c|c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right) \left(\begin{array}{c|c|c|c} \sigma_0 & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & \sigma_{m-1} \end{array} \right) \left(\begin{array}{c|c|c|c} v_0 & \cdots & v_{m-1} \end{array} \right)^H\end{aligned}$$

be its SVD.

The SVD of A^{-1} is given by (indicate all correct answers):

1. $V\Sigma^{-1}U^H$.

2. $\left(\begin{array}{c|c|c|c} v_0 & \cdots & v_{m-1} \end{array} \right) \left(\begin{array}{c|c|c|c} 1/\sigma_0 & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & 1/\sigma_{m-1} \end{array} \right) \left(\begin{array}{c|c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right)^H$

3. $\left(\begin{array}{c|c|c|c} v_{m-1} & \cdots & v_0 \end{array} \right) \left(\begin{array}{c|c|c|c} 1/\sigma_{m-1} & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & 1/\sigma_0 \end{array} \right) \left(\begin{array}{c|c|c|c} u_{m-1} & \cdots & u_0 \end{array} \right)^H$.

4. $(VP^H)(P\Sigma^{-1}P^H)(UP^H)^H$ where $P = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{pmatrix}$

Answer. 3. and 4.

Explain it!

Solution. This question is a bit tricky.

1. It is the case that $A^{-1} = V\Sigma^{-1}U^H$. However, the diagonal elements of Σ^{-1} are ordered from smallest to largest, and hence this is not its SVD.
2. This is just Answer 1. but with the columns of U and V , and the elements of Σ , exposed.
3. This answer corrects the problems with the previous two answers: it reorders columns of U and V so that the diagonal elements of Σ end up ordered from largest to smallest.
4. This answer is just a reformulation of the last answer.

Homework 2.3.5.5 Let $A \in \mathbb{C}^{m \times m}$ be nonsingular. TRUE/FALSE: $\|A^{-1}\|_2 = 1 / \min_{\|x\|_2=1} \|Ax\|_2$.

Answer. TRUE

Solution.

$$\begin{aligned}
 & \|A^{-1}\|_2 \\
 &= <\text{definition}> \\
 & \max_{x \neq 0} \frac{\|A^{-1}x\|_2}{\|x\|_2} \\
 &= <\text{algebra}> \\
 & \max_{x \neq 0} \frac{\frac{1}{\|x\|_2}}{\frac{\|A^{-1}x\|_2}{\|x\|_2}} \\
 &= <\text{algebra}> \\
 & \frac{1}{\min_{x \neq 0} \frac{\|x\|_2}{\|A^{-1}x\|_2}} \\
 &= <\text{substitute } z = A^{-1}x> \\
 & \frac{1}{\min_{Az \neq 0} \frac{\|Az\|_2}{\|z\|_2}} \\
 &= <\text{A is nonsingular}> \\
 & \frac{1}{\min_{z \neq 0} \frac{\|Az\|_2}{\|z\|_2}} \\
 &= <x = z/\|z\|_2> \\
 & \frac{1}{\min_{\|x\|_2=1} \|Ax\|_2}
 \end{aligned}$$

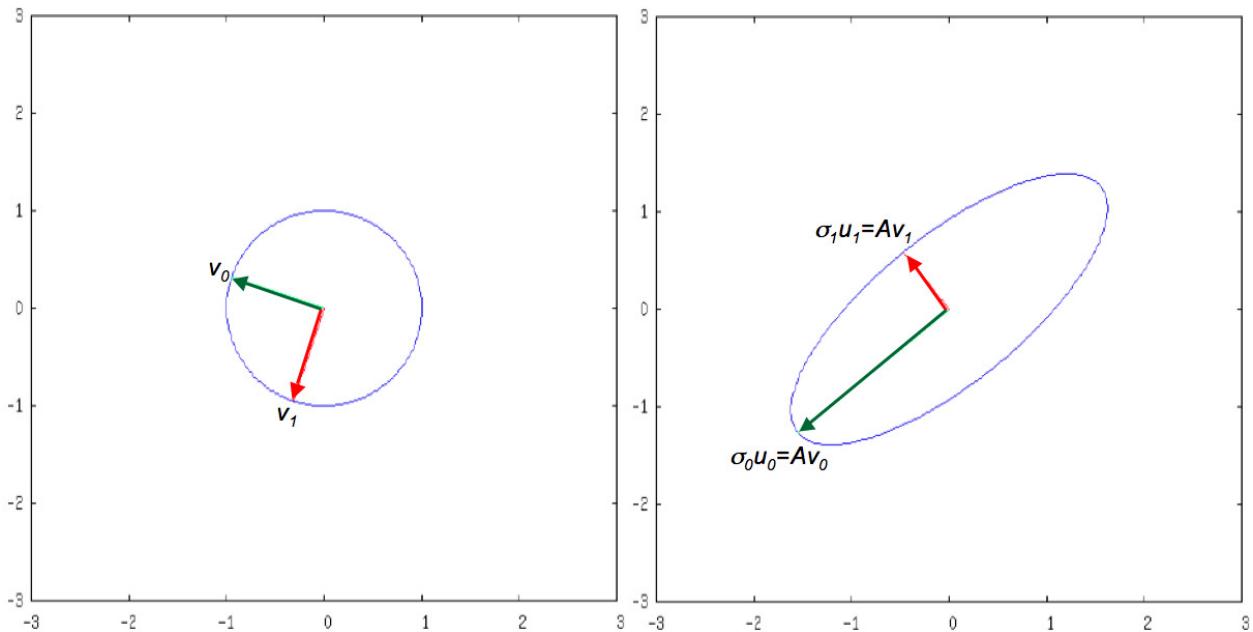
In Subsection 2.3.2, we discussed the case where $A \in \mathbb{R}^{2 \times 2}$. Letting $A = U\Sigma V^T$ and partitioning

$$A = \left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} \sigma_0 & 0 \\ 0 & \sigma_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T$$

yielded the pictures

\mathbb{R}^2 : Domain of A :

\mathbb{R}^2 : Range (codomain) of A :



This captures what the condition number $\kappa_2(A) = \sigma_0/\sigma_{n-1}$ captures: how elongated the oval that equals the image of the unit ball is. The more elongated, the greater the ratio σ_0/σ_{n-1} , and the worse the condition number of the matrix. In the limit, when $\sigma_{n-1} = 0$, the unit ball is mapped to a lower dimensional set, meaning that the transformation cannot be "undone."

Ponder This 2.3.5.6 For the 2D problem discussed in this unit, what would the image of the unit ball look like as $\kappa_2(A) \rightarrow \infty$? When is $\kappa_2(A) = \infty$?

2.3.6 Best rank-k approximation



YouTube: <https://www.youtube.com/watch?v=sN0DKG8vPhQ>

We are now ready to answer the question "How do we find the best rank-k approximation for a picture (or, more generally, a matrix)? " posed in [Subsection 2.1.1](#).

Theorem 2.3.6.1 Given $A \in \mathbb{C}^{m \times n}$, let $A = U\Sigma V^H$ be its SVD. Assume the entries on the main diagonal of Σ are $\sigma_0, \dots, \sigma_{\min(m,n)-1}$ with $\sigma_0 \geq \dots \geq \sigma_{\min(m,n)-1} \geq 0$. Given k such that $0 \leq k \leq \min(m, n)$, partition

$$U = \left(\begin{array}{c|c} U_L & U_R \end{array} \right), V = \left(\begin{array}{c|c} V_L & V_R \end{array} \right), \text{ and } \Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ 0 & \Sigma_{BR} \end{array} \right),$$

where $U_L \in \mathbb{C}^{m \times k}$, $V_L \in \mathbb{C}^{n \times k}$, and $\Sigma_{TL} \in \mathbb{R}^{k \times k}$. Then

$$B = U_L \Sigma_{TL} V_L^H$$

is the matrix in $\mathbb{C}^{m \times n}$ closest to A in the following sense:

$$\|A - B\|_2 = \min_{\substack{C \in \mathbb{C}^{m \times n} \\ \text{rank}(C) \leq k}} \|A - C\|_2.$$

In other words, B is the matrix with rank at most k that is closest to A as measured by the 2-norm. Also, for this B ,

$$\|A - B\|_2 = \begin{cases} \sigma_k & \text{if } k < \min(m, n) \\ 0 & \text{otherwise.} \end{cases}$$

The proof of this theorem builds on the following insight:

Homework 2.3.6.1 Given $A \in \mathbb{C}^{m \times n}$, let $A = U\Sigma V^H$ be its SVD. Show that

$$Av_j = \sigma_j u_j \text{ for } 0 \leq j < \min(m, n),$$

where u_j and v_j equal the columns of U and V indexed by j , and σ_j equals the diagonal element of Σ indexed with j .

Solution. W.l.o.g. assume $n \leq m$. Rewrite $A = U\Sigma V^H$ as $AV = U\Sigma$. Then

$$\begin{aligned} AV &= U\Sigma = <\text{partition}> \\ A \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right) &= \left(\begin{array}{c|c|c|c|c} u_0 & \cdots & u_{n-1} & u_n & \cdots & u_{m-1} \end{array} \right) \left(\begin{array}{c|c|c} \sigma_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{n-1} \\ 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & & 0 \end{array} \right) \\ &= <\text{multiply out}> \\ \left(\begin{array}{c|c|c} Av_0 & \cdots & Av_{n-1} \end{array} \right) &= \left(\begin{array}{c|c|c} \sigma_0 u_0 & \cdots & \sigma_{n-1} u_{n-1} \end{array} \right). \end{aligned}$$

Hence $Av_j = \sigma_j u_j$ for $0 \leq j < n$.

Proof of Theorem 2.3.6.1. First, if B is as defined, then $\|A - B\|_2 = \sigma_k$:

$$\begin{aligned}
 & \|A - B\|_2 \\
 &= \quad \text{multiplication with unitary matrices preserves 2-norm} \\
 &\|U^H(A - B)V\|_2 \\
 &= \quad \text{distribute} \\
 &\|U^H A V - U^H B V\|_2 \\
 &= \quad \text{use SVD of } A \text{ and partition} \\
 &\left\| \Sigma - \left(\begin{array}{c|c} U_L & U_R \end{array} \right)^H B \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \right\|_2 \\
 &= \quad \text{how } B \text{ was chosen} \\
 &\left\| \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ 0 & \Sigma_{BR} \end{array} \right) - \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ 0 & 0 \end{array} \right) \right\|_2 \\
 &= \quad \text{partitioned subtraction} \\
 &\left\| \left(\begin{array}{c|c} 0 & 0 \\ 0 & \Sigma_{BR} \end{array} \right) \right\|_2 \\
 &= \quad \text{=>} \\
 &\|\Sigma_{BR}\|_2 \\
 &= \quad \text{ } \Sigma_{TL} \text{ is } k \times k \\
 &\sigma_k
 \end{aligned}$$

(Obviously, this needs to be tidied up for the case where $k > \text{rank}(A)$.)

Next, assume that C has rank $r \leq k$ and $\|A - C\|_2 < \|A - B\|_2$. We will show that this leads to a contradiction.

- The null space of C has dimension at least $n - k$ since $\dim(\mathcal{N}(C)) = n - r$.
- If $x \in \mathcal{N}(C)$ then

$$\|Ax\|_2 = \|(A - C)x\|_2 \leq \|A - C\|_2 \|x\|_2 < \sigma_k \|x\|_2.$$

- Partition $U = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right)$ and $V = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right)$. Then $\|Av_j\|_2 = \|\sigma_j u_j\|_2 = \sigma_j \geq \sigma_k$ for $j = 0, \dots, k$.
- Now, let y be any linear combination of v_0, \dots, v_k : $y = \alpha_0 v_0 + \cdots + \alpha_k v_k$. Notice that

$$\|y\|_2^2 = \|\alpha_0 v_0 + \cdots + \alpha_k v_k\|_2^2 = |\alpha_0|^2 + \cdots + |\alpha_k|^2$$

since the vectors v_j are orthonormal. Then

$$\begin{aligned}
 & \|Ay\|_2^2 \\
 &= \langle y = \alpha_0 v_0 + \cdots + \alpha_k v_k \rangle \\
 &\|A(\alpha_0 v_0 + \cdots + \alpha_k v_k)\|_2^2 \\
 &= \langle \text{distributivity} \rangle \\
 &\|\alpha_0 A v_0 + \cdots + \alpha_k A v_k\|_2^2 \\
 &= \langle A v_j = \sigma_j u_j \rangle \\
 &\|\alpha_0 \sigma_0 u_0 + \cdots + \alpha_k \sigma_k u_k\|_2^2 \\
 &= \langle \text{this works because the } u_j \text{ are orthonormal} \rangle \\
 &\|\alpha_0 \sigma_0 u_0\|_2^2 + \cdots + \|\alpha_k \sigma_k u_k\|_2^2 \\
 &= \langle \text{norms are homogeneous and } \|u_j\|_2 = 1 \rangle \\
 &|\alpha_0|^2 \sigma_0^2 + \cdots + |\alpha_k|^2 \sigma_k^2 \\
 &\geq \langle \sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_k \geq 0 \rangle \\
 &(|\alpha_0|^2 + \cdots + |\alpha_k|^2) \sigma_k^2 \\
 &= \langle \|y\|_2^2 = |\alpha_0|^2 + \cdots + |\alpha_k|^2 \rangle \\
 &\sigma_k^2 \|y\|_2^2.
 \end{aligned}$$

so that $\|Ay\|_2 \geq \sigma_k \|y\|_2$. In other words, vectors in the subspace of all linear combinations of $\{v_0, \dots, v_k\}$ satisfy $\|Ax\|_2 \geq \sigma_k \|x\|_2$. The dimension of this subspace is $k+1$ (since $\{v_0, \dots, v_k\}$ form an orthonormal basis).

- Both these subspaces are subspaces of \mathbb{C}^n . One has dimension $k+1$ and the other $n-k$. This means that if you take a basis for one (which consists of $n-k$ linearly independent vectors) and add it to a basis for the other (which has $k+1$ linearly independent vectors), you end up with $n+1$ vectors. Since these cannot all be linearly independent in \mathbb{C}^n , there must be at least one nonzero vector z that satisfies both $\|Az\|_2 < \sigma_k \|z\|_2$ and $\|Az\|_2 \geq \sigma_k \|z\|_2$, which is a contradiction.

■

Theorem 2.3.6.1 tells us how to pick the best approximation to a given matrix of a given desired rank. In Section Subsection 2.1.1 we discussed how a low rank matrix can be used to compress data. The SVD thus gives the best such rank- k approximation. Let us revisit this.

Let $A \in \mathbb{R}^{m \times n}$ be a matrix that, for example, stores a picture. In this case, the i, j entry in A is, for example, a number that represents the grayscale value of pixel (i, j) .

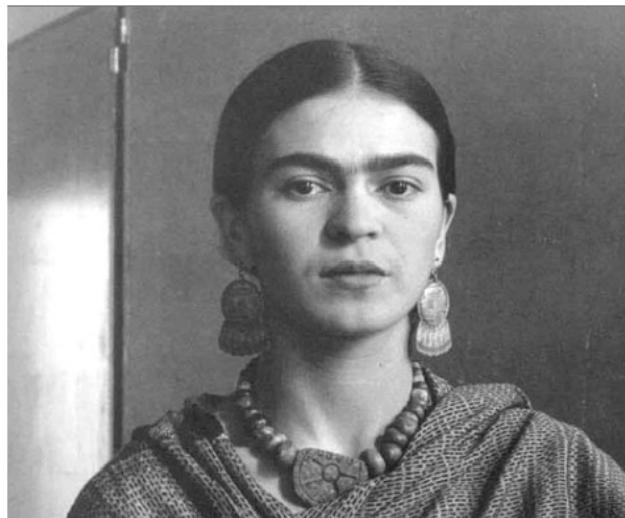
Homework 2.3.6.2 In Assignments/Week02/matlab execute

```

IMG = imread( 'Frida.jpg' );
A = double( IMG( :, :, 1 ) );
imshow( uint8( A ) )
size( A )

```

to generate the picture of Mexican artist Frida Kahlo



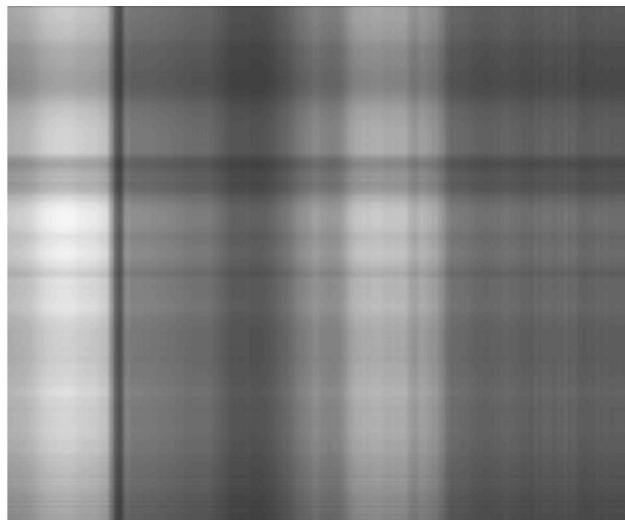
Although the picture is black and white, it was read as if it is a color image, which means a $m \times n \times 3$ array of pixel information is stored. Setting $A = \text{IMG}(:, :, 1)$ extracts a single matrix of pixel information. (If you start with a color picture, you will want to approximate $\text{IMG}(:, :, 1)$, $\text{IMG}(:, :, 2)$, and $\text{IMG}(:, :, 3)$ separately.)

Next, compute the SVD of matrix A

```
[ U, Sigma, V ] = svd( A );
```

and approximate the picture with a rank- k update, starting with $k = 1$:

```
k = 1
B = uint8( U( :, 1:k ) * Sigma( 1:k, 1:k ) * V( :, 1:k )' );
imshow( B );
```



Repeat this with increasing k .

```
r = min( size( A ) )
for k=1:r
```

```

imshow( uint8( U( :, 1:k ) * Sigma( 1:k,1:k ) * V( :, 1:k )' ) );
input( strcat( num2str( k ), "    press return" ) );
end

```

To determine a reasonable value for k , it helps to graph the singular values:

```

figure
r = min( size( A ) );
plot( [ 1:r ], diag( Sigma ), 'x' );

```

Since the singular values span a broad range, we may want to plot them with a log-log plot

```
loglog( [ 1:r ], diag( Sigma ), 'x' );
```

For this particular matrix (picture), there is no dramatic drop in the singular values that makes it obvious what k is a natural choice.

Solution.

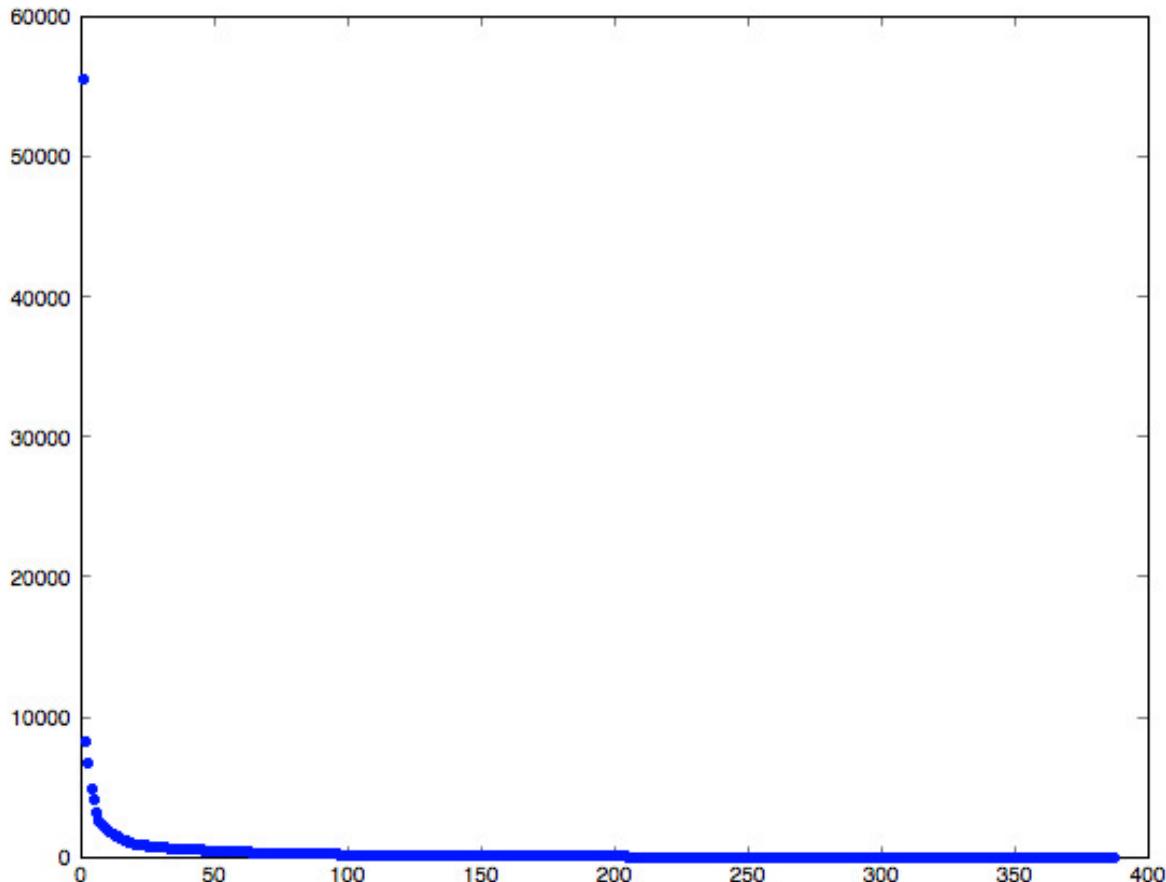
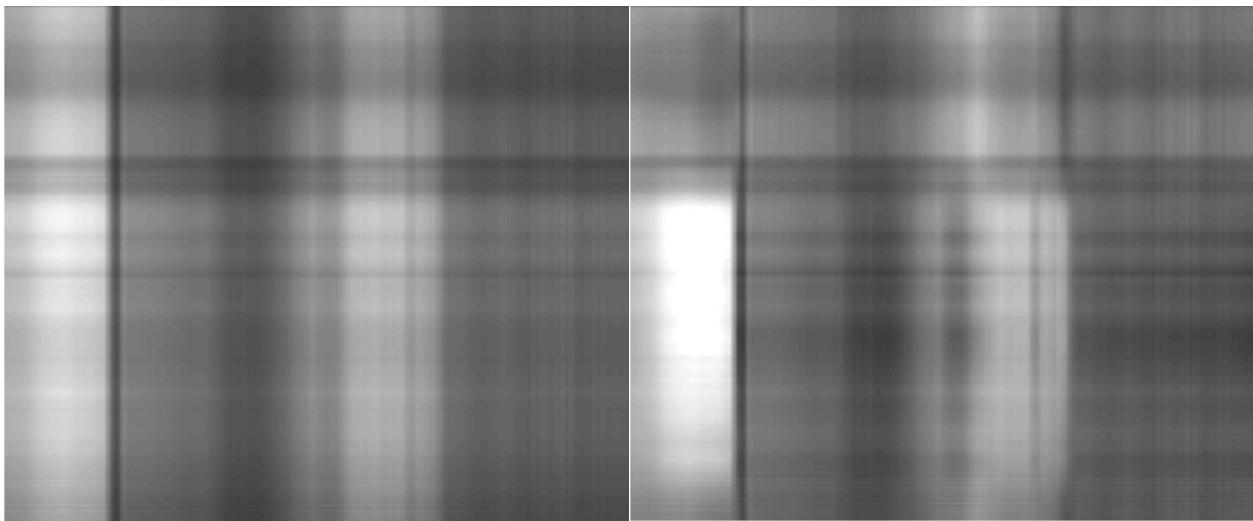


Figure 2.3.6.2 Distribution of singular values for the picture.

$k = 1$  $k = 2$ $k = 5$ $k = 10$  $k = 25$

Original picture

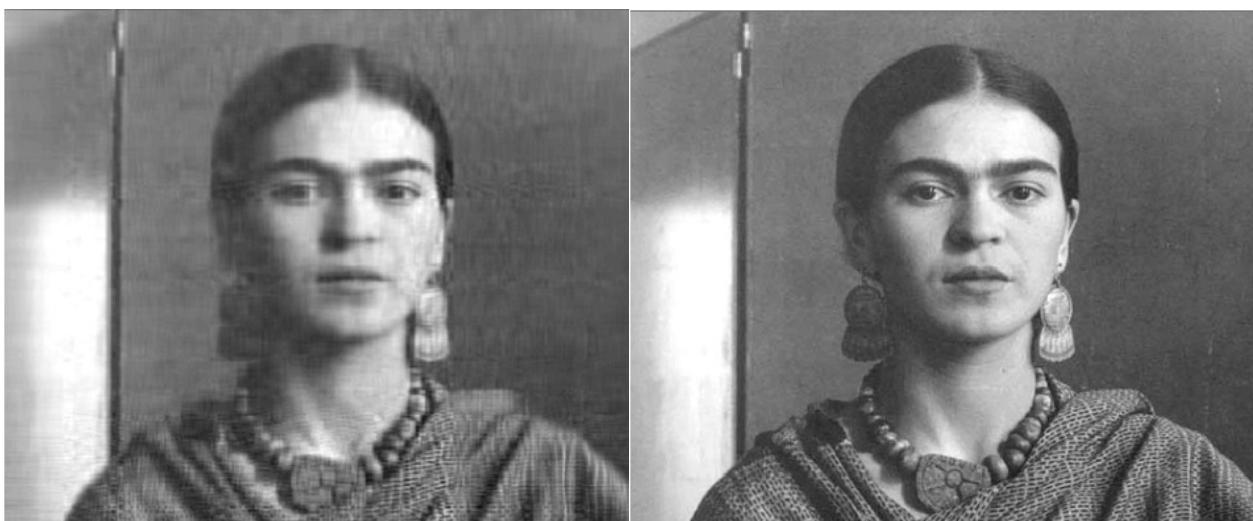


Figure 2.3.6.3 Multiple pictures as generated by the code.

2.4 Enrichments

2.4.1 Principle Component Analysis (PCA)

Principle Component Analysis (PCA) is a standard technique in data science related to the SVD. You may enjoy the article

- [30] J. Novembre, T. Johnson, K. Bryc, Z. Kutalik, A.R. Boyko, A. Auton, A. Indap, K.S. King, S. Bergmann, M.. Nelson, M. Stephens, C.D. Bustamante, , Nature, 2008.
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2735096/>.

In that article, PCA is cast as an eigenvalue problem rather than a singular value problem. Later in the course, in Week 11, we will link these.

2.5 Wrap Up

2.5.1 Additional homework

Homework 2.5.1.1 $U \in \mathbb{C}^{m \times m}$ is unitary if and only if $(Ux)^H(Uy) = x^H y$ for all $x, y \in \mathbb{C}^m$.

Hint. Revisit the proof of [Homework 2.2.4.6](#).

Homework 2.5.1.2 Let $A, B \in \mathbb{C}^{m \times n}$. Furthermore, let $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ be unitary.

TRUE/FALSE: $UAV^H = B$ iff $U^H BV = A$.

Answer. TRUE

Now prove it!

Homework 2.5.1.3 Prove that nonsingular $A \in \mathbb{C}^{n \times n}$ has condition number $\kappa_2(A) = 1$ if and only if $A = \sigma Q$ where Q is unitary and $\sigma \in \mathbb{R}$ is positive.

Hint. Use the SVD of A .

Homework 2.5.1.4 Let $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ be unitary.

ALWAYS/SOMETIMES/NEVER: The matrix $\begin{pmatrix} U & | & 0 \\ 0 & | & V \end{pmatrix}$ is unitary.

Answer. ALWAYS

Now prove it!

Homework 2.5.1.5 Matrix $A \in \mathbb{R}^{m \times m}$ is a stochastic matrix if and only if it is nonnegative (all its entries are nonnegative) and the entries in its columns sum to one: $\sum_{0 \leq i < m} \alpha_{i,j} = 1$. Such matrices are at the core of Markov processes. Show that a matrix A is both unitary matrix and a stochastic matrix if and only if it is a permutation matrix.

Homework 2.5.1.6 Show that if $\|\cdot\|$ is a norm and A is nonsingular, then $\|\cdot\|_{A^{-1}}$ defined by $\|x\|_{A^{-1}} = \|A^{-1}x\|$ is a norm.

Interpret this result in terms of the change of basis of a vector.

Homework 2.5.1.7 Let $A \in \mathbb{C}^{m \times m}$ be nonsingular and $A = U\Sigma V^H$ be its SVD with

$$\Sigma = \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{m-1} \end{pmatrix}$$

The condition number of A is given by (mark all correct answers):

1. $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2.$
2. $\kappa_2(A) = \sigma_0/\sigma_{m-1}.$
3. $\kappa_2(A) = u_0^H A v_0 / u_{m-1}^H A v_{m-1}.$
4. $\kappa_2(A) = \max_{\|x\|_2=1} \|Ax\|_2 / \min_{\|x\|_2=1} \|Ax\|_2.$

(Mark all correct answers.)

Homework 2.5.1.8 [Theorem 2.2.4.4](#) stated: If $A \in \mathbb{C}^{m \times m}$ preserves length ($\|Ax\|_2 = \|x\|_2$ for all $x \in \mathbb{C}^m$), then A is unitary. Give an alternative proof using the SVD.

Homework 2.5.1.9 In [Homework 1.3.7.2](#) you were asked to prove that $\|A\|_2 \leq \|A\|_F$ given $A \in \mathbb{C}^{m \times n}$. Give an alternative proof that leverages the SVD.

Homework 2.5.1.10 In [Homework 1.3.7.3](#), we skipped how the 2-norm bounds the Frobenius norm. We now have the tools to do so elegantly: Prove that, given $A \in \mathbb{C}^{m \times n}$,

$$\|A\|_F \leq \sqrt{r}\|A\|_2,$$

where r is the rank of matrix A .

2.5.2 Summary

Given $x, y \in \mathbb{C}^m$

- their dot product (inner product) is defined as

$$x^H y = \bar{x}^T y = \overline{\bar{x}^T} y = \bar{\chi}_0 \psi_0 + \bar{\chi}_1 \psi_1 + \cdots + \bar{\chi}_{m-1} \psi_{m-1} = \sum_{i=0}^{m-1} \bar{\chi}_i \psi_i.$$

- These vectors are said to be orthogonal (perpendicular) iff $x^H y = 0$.
- The component of y in the direction of x is given by

$$\frac{x^H y}{x^H x} x = \frac{x x^H}{x^H x} y.$$

The matrix that projects a vector onto the space spanned by x is given by

$$\frac{xx^H}{x^H x}.$$

- The component of y orthogonal to x is given by

$$y - \frac{x^H y}{x^H x} x = \left(I - \frac{xx^H}{x^H x} \right) y.$$

Thus, the matrix that projects a vector onto the space orthogonal to x is given by

$$I - \frac{xx^H}{x^H x}.$$

Given $u, v \in \mathbb{C}^m$ with u of unit length

- The component of v in the direction of u is given by

$$u^H vu = uu^H v.$$

- The matrix that projects a vector onto the space spanned by u is given by

$$uu^H$$

- The component of v orthogonal to u is given by

$$v - u^H vu = \left(I - uu^H \right) v.$$

- The matrix that projects a vector onto the space that is orthogonal to x is given by

$$I - uu^H$$

Let $u_0, u_1, \dots, u_{n-1} \in \mathbb{C}^m$. These vectors are said to be mutually orthonormal if for all $0 \leq i, j < n$

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

Let $Q \in \mathbb{C}^{m \times n}$ (with $n \leq m$). Then Q is said to be

- an orthonormal matrix iff $Q^H Q = I$.
- a unitary matrix iff $Q^H Q = I$ and $m = n$.
- an orthogonal matrix iff it is a unitary matrix and is real-valued.

Let $Q \in \mathbb{C}^{m \times n}$ (with $n \leq m$). Then $Q = \left(q_0 \mid \cdots \mid q_{n-1} \right)$ is orthonormal iff $\{q_0, \dots, q_{n-1}\}$ are mutually orthonormal.

Definition 2.5.2.1 Unitary matrix. Let $U \in \mathbb{C}^{m \times m}$. Then U is said to be a unitary matrix if and only if $U^H U = I$ (the identity). \diamond

If $U, V \in \mathbb{C}^{m \times m}$ are unitary, then

- $U^H U = I$.
- $UU^H = I$.
- $U^{-1} = U^H$.
- U^H is unitary.
- UV is unitary.

If $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ are unitary, $x \in \mathbb{C}^m$, and $A \in \mathbb{C}^{m \times n}$, then

- $\|Ux\|_2 = \|x\|_2$.
- $\|U^H A\|_2 = \|UA\|_2 = \|AV\|_2 = \|AV^H\|_2 = \|U^H AV\|_2 = \|UAV^H\|_2 = \|A\|_2$.
- $\|U^H A\|_F = \|UA\|_F = \|AV\|_F = \|AV^H\|_F = \|U^H AV\|_F = \|UAV^H\|_F = \|A\|_F$.
- $\|U\|_2 = 1$
- $\kappa_2(U) = 1$

Examples of unitary matrices:

- Rotation in 2D: $\begin{pmatrix} c & -s \\ s & c \end{pmatrix}$.
- Reflection: $I - 2uu^H$ where $u \in \mathbb{C}^m$ and $\|u\|_2 = 1$.

Change of orthonormal basis: If $x \in \mathbb{C}^m$ and $U = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right)$ is unitary, then

$$x = (u_0^H x)u_0 + \cdots + (u_{m-1}^H x)u_{m-1} = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right) \underbrace{\begin{pmatrix} u_0^H x \\ \vdots \\ u_{m-1}^H x \end{pmatrix}}_{U^H x} = UU^H x.$$

Let $A \in \mathbb{C}^{n \times n}$ be nonsingular and $x \in \mathbb{C}^n$ a nonzero vector. Consider

$$y = Ax \quad \text{and} \quad y + \delta y = A(x + \delta x).$$

Then

$$\frac{\|\delta y\|}{\|y\|} \leq \underbrace{\|A\| \|A^{-1}\|}_{\kappa(A)} \frac{\|\delta x\|}{\|x\|},$$

where $\|\cdot\|$ is an induced matrix norm.

Theorem 2.5.2.2 Singular Value Decomposition Theorem. Given $A \in \mathbb{C}^{m \times n}$ there exist unitary $U \in \mathbb{C}^{m \times m}$, unitary $V \in \mathbb{C}^{n \times n}$, and $\Sigma \in \mathbb{R}^{m \times n}$ such that $A = U\Sigma V^H$. Here $\Sigma = \begin{pmatrix} \Sigma_{TL} & 0 \\ 0 & 0 \end{pmatrix}$ with

$$\Sigma_{TL} = \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{r-1} \end{pmatrix} \quad \text{and} \quad \sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > 0.$$

The values $\sigma_0, \dots, \sigma_{r-1}$ are called the singular values of matrix A . The columns of U and V are called the left and right singular vectors, respectively.

Let $A \in \mathbb{C}^{m \times n}$ and $A = U\Sigma V^H$ its SVD with

$$U = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right),$$

$$V = \left(\begin{array}{c|c} V_L & V_R \end{array} \right) = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right),$$

and

$$\Sigma = \begin{pmatrix} \Sigma_{TL} & 0 \\ 0 & 0 \end{pmatrix}, \text{ where } \Sigma_{TL} = \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{r-1} \end{pmatrix} \quad \text{and} \quad \sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > 0.$$

Here $U_L \in \mathbb{C}^{m \times r}$, $V_L \in \mathbb{C}^{n \times r}$ and $\Sigma_{TL} \in \mathbb{R}^{r \times r}$. Then

- $\|A\|_2 = \sigma_0$. (The 2-norm of a matrix equals the largest singular value.)
- $\text{rank}(A) = r$.
- $\mathcal{C}(A) = \mathcal{C}(U_L)$.
- $\mathcal{N}(A) = \mathcal{C}(V_R)$.
- $\mathcal{R}(A) = \mathcal{C}(V_L)$.
- Left null-space of $A = \mathcal{C}(U_R)$.
- $A^H = V\Sigma^T U^H$.
- SVD: $A^H = V\Sigma U^H$.
- Reduced SVD: $A = U_L \Sigma_{TL} V_L^H$.
-

$$A = \underbrace{\sigma_0 u_0 v_0^H}_{\sigma_0 \mid \text{——}} + \underbrace{\sigma_1 u_1 v_1^H}_{\sigma_1 \mid \text{——}} + \cdots + \underbrace{\sigma_{r-1} u_{r-1} v_{r-1}^H}_{\sigma_{r-1} \mid \text{——}}.$$

- Reduced SVD: $A^H = V_L \Sigma U_L^H$.
- If $m \times m$ matrix A is nonsingular: $A^{-1} = V \Sigma^{-1} U^H$.
- If $A \in \mathbb{C}^{m \times m}$ then A is nonsingular if and only if $\sigma_{m-1} \neq 0$.
- If $A \in \mathbb{C}^{m \times m}$ is nonsingular then $\kappa_2(A) = \sigma_0/\sigma_{m-1}$.
- (Left) pseudo inverse: if A has linearly independent columns, then $A^\dagger = (A^H A)^{-1} A^H = V \Sigma_{TL}^{-1} U_L^H$.
- v_0 is the direction of maximal magnification.
- v_{n-1} is the direction of minimal magnification.
- If $n \leq m$, then $A v_j = \sigma_j u_j$, for $0 \leq j < n$.

Theorem 2.5.2.3 Given $A \in \mathbb{C}^{m \times n}$, let $A = U \Sigma V^H$ be its SVD. Assume the entries on the main diagonal of Σ are $\sigma_0, \dots, \sigma_{\min(m,n)-1}$ with $\sigma_0 \geq \dots \geq \sigma_{\min(m,n)-1} \geq 0$. Given k such that $0 \leq k \leq \min(m, n)$, partition

$$U = \left(\begin{array}{c|c} U_L & U_R \end{array} \right), V = \left(\begin{array}{c|c} V_L & V_R \end{array} \right), \text{ and } \Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right),$$

where $U_L \in \mathbb{C}^{m \times k}$, $V_L \in \mathbb{C}^{n \times k}$, and $\Sigma_{TL} \in \mathbb{R}^{k \times k}$. Then

$$B = U_L \Sigma_{TL} V_L^H$$

is the matrix in $\mathbb{C}^{m \times n}$ closest to A in the following sense:

$$\|A - B\|_2 = \min_{\substack{C \in \mathbb{C}^{m \times n} \\ \text{rank}(C) \leq k}} \|A - C\|_2.$$

In other words, B is the matrix with rank at most k that is closest to A as measured by the 2-norm. Also, for this B ,

$$\|A - B\|_2 = \begin{cases} \sigma_k & \text{if } k < \min(m, n) \\ 0 & \text{otherwise.} \end{cases}$$

Week 3

The QR Decomposition

3.1 Opening

3.1.1 Choosing the right basis



YouTube: <https://www.youtube.com/watch?v=5lEm5gZo27g>

A classic problem in numerical analysis is the approximation of a function, $f : \mathbb{R} \rightarrow \mathbb{R}$, with a polynomial of degree $n-1$. (The $n-1$ seems cumbersome. Think of it as a polynomial with n terms.)

$$f(\chi) \approx \gamma_0 + \gamma_1 \chi + \cdots + \gamma_{n-1} \chi^{n-1}.$$

* Now, often we know f only "sampled" at points $\chi_0, \dots, \chi_{m-1}$:

$$\begin{aligned} f(\chi_0) &= \phi_0 \\ &\vdots && \vdots \\ f(\chi_{m-1}) &= \phi_{m-1}. \end{aligned}$$

In other words, input to the process are the points

$$(\chi_0, \phi_0), \dots, (\chi_{m-1}, \phi_{m-1})$$

and we want to determine the polynomial that approximately fits these points. This means that

$$\begin{aligned} \gamma_0 + \gamma_1 \chi_0 + \cdots + \gamma_{n-1} \chi_0^{n-1} &\approx \phi_0 \\ \vdots &\vdots && \vdots \\ \gamma_0 + \gamma_1 \chi_{m-1} + \cdots + \gamma_{n-1} \chi_{m-1}^{n-1} &\approx \phi_{m-1}. \end{aligned}$$

This can be reformulated as the approximate linear system

$$\begin{pmatrix} 1 & \chi_0 & \cdots & \chi_0^{n-1} \\ 1 & \chi_1 & \cdots & \chi_1^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \chi_{m-1} & \cdots & \chi_{m-1}^{n-1} \end{pmatrix} \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_{n-1} \end{pmatrix} \approx \begin{pmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{m-1} \end{pmatrix}.$$

which can be solved using the techniques for linear least-squares in [Week 4](#). The matrix in the above equation is known as a **Vandermonde matrix**.

Homework 3.1.1.1 Choose $\chi_0, \chi_1, \dots, \chi_{m-1}$ to be equally spaced in the interval $[0, 1]$: for $i = 0, \dots, m - 1$, $\chi_i = ih$, where $h = 1/(m - 1)$. Write Matlab code to create the matrix

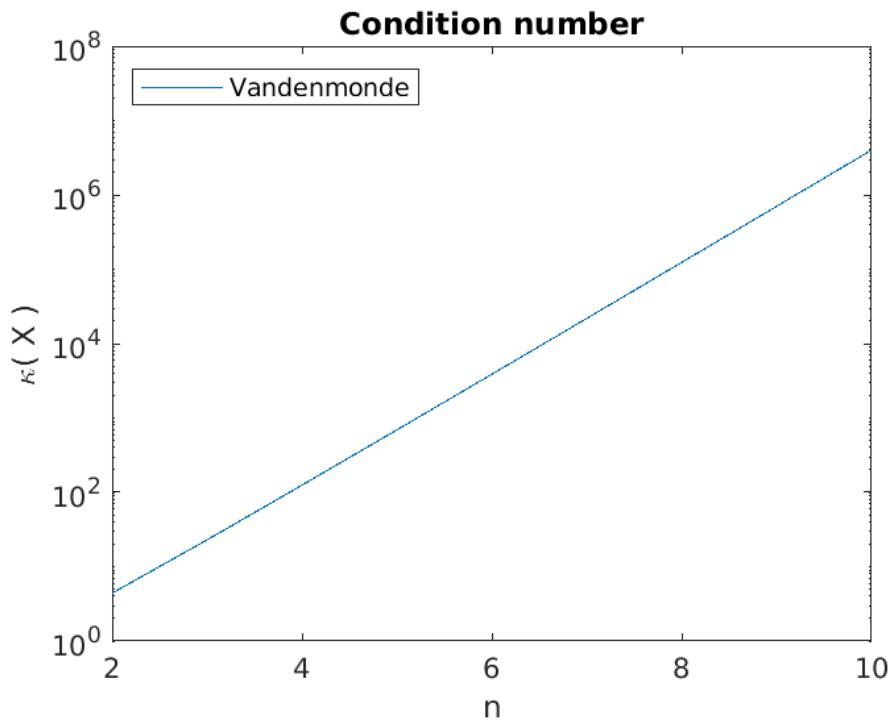
$$X = \begin{pmatrix} 1 & \chi_0 & \cdots & \chi_0^{n-1} \\ 1 & \chi_1 & \cdots & \chi_1^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \chi_{m-1} & \cdots & \chi_{m-1}^{n-1} \end{pmatrix}$$

as a function of n with $m = 5000$. Plot the condition number of X , $\kappa_2(X)$, as a function of n (Matlab's function for computing $\kappa_2(X)$ is `cond(X)`.)

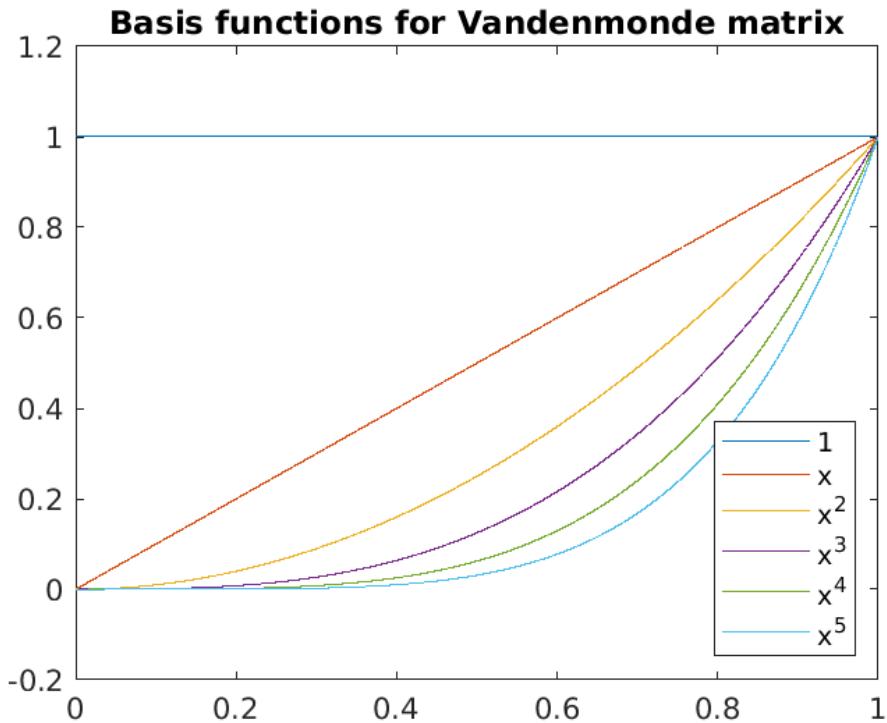
Hint. You may want to use the recurrence $x^{j+1} = xx^j$ and the fact that the `.*` operator in Matlab performs an element-wise multiplication.

Solution.

- Here is our implementation: [Assignments/Week03/answers/Vandermonde.m](#).
(Assignments/Week03/answers/Vandermonde.m)
- The graph of the condition number, $\kappa(X)$, as a function of n is given by

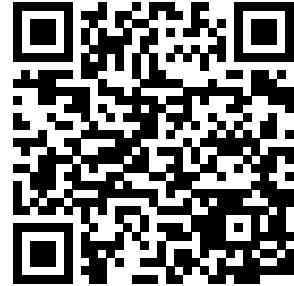


- The parent functions $1, x, x^2, \dots$ on the interval $[0, 1]$ are visualized as



Notice that the curves for x^j and x^{j+1} quickly start to look very similar, which explains why the columns of the Vandermonde matrix quickly become approximately linearly dependent.

Think about how this extends to even more columns of A .



YouTube: <https://www.youtube.com/watch?v=cBFt2dmXbu4>

An alternative set of polynomials that can be used are known as **Legendre polynomials**. A shifted version (appropriate for the interval $[0, 1]$) can be inductively defined by

$$\begin{aligned} P_0(\chi) &= 1 \\ P_1(\chi) &= 2\chi - 1 \\ \vdots &= \vdots \\ P_{n+1}(\chi) &= ((2n+1)(2\chi-1)P_n(\chi) - nP_{n-1}(\chi)) / (n+1). \end{aligned}$$

The polynomials have the property that

$$\int_0^1 P_s(\chi)P_t(\chi)d\chi = \begin{cases} C_s & \text{if } s = t \text{ for some nonzero constant } C_s \\ 0 & \text{otherwise} \end{cases}$$

which is an orthogonality condition on the polynomials.

The function $f : \mathbb{R} \rightarrow \mathbb{R}$ can now instead be approximated by

$$f(\chi) \approx \gamma_0 P_0(\chi) + \gamma_1 P_1(\chi) + \cdots + \gamma_{n-1} P_{n-1}(\chi).$$

and hence given points

$$(\chi_0, \phi_0), \dots, (\chi_{m-1}, \phi_{m-1})$$

we can determine the polynomial from

$$\begin{aligned} \gamma_0 P_0(\chi_0) + \gamma_1 P_1(\chi_0) + \cdots + \gamma_{n-1} P_{n-1}(\chi_0) &= \phi_0 \\ \vdots &= \vdots \\ \gamma_0 P_0(\chi_{m-1}) + \gamma_1 P_1(\chi_{m-1}) + \cdots + \gamma_{n-1} P_{n-1}(\chi_{m-1}) &= \phi_{m-1}. \end{aligned}$$

This can be reformulated as the approximate linear system

$$\begin{pmatrix} 1 & P_1(\chi_0) & \cdots & P_{n-1}(\chi_0) \\ 1 & P_1(\chi_1) & \cdots & P_{n-1}(\chi_1) \\ \vdots & \vdots & & \vdots \\ 1 & P_1(\chi_{m-1}) & \cdots & P_{n-1}(\chi_{m-1}) \end{pmatrix} \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_{n-1} \end{pmatrix} \approx \begin{pmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{m-1} \end{pmatrix}.$$

which can also be solved using the techniques for linear least-squares in Week 4. Notice that now the columns of the matrix are (approximately) orthogonal: Notice that if we "sample" x as $\chi_0, \dots, \chi_{n-1}$, then

$$\int_0^1 P_s(\chi)P_t(\chi)d\chi \approx \sum_{i=0}^{n-1} P_s(\chi_i)P_t(\chi_i),$$

which equals the dot product of the columns indexed with s and t .

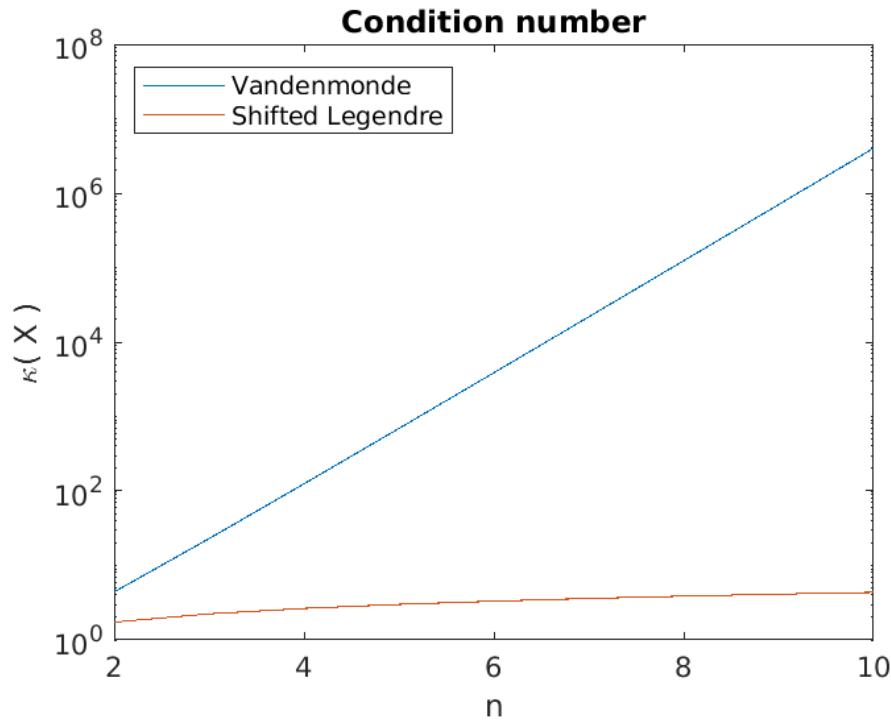
Homework 3.1.1.2 Choose $\chi_0, \chi_1, \dots, \chi_{m-1}$ to be equally spaced in the interval $[0, 1]$: for $i = 0, \dots, m - 1$, $\chi_i = ih$, where $h = 1/(m - 1)$. Write Matlab code to create the matrix

$$X = \begin{pmatrix} 1 & P_1(\chi_0) & \cdots & P_{n-1}(\chi_0) \\ 1 & P_1(\chi_1) & \cdots & P_{n-1}(\chi_1) \\ \vdots & \vdots & & \vdots \\ 1 & P(\chi_{m-1}) & \cdots & P_{n-1}(\chi_{m-1}) \end{pmatrix}$$

as a function of n with $m = 5000$. Plot $\kappa_2(X)$ as a function of n . To check whether the columns of X are mutually orthogonal, report $\|X^T X - D\|_2$ where D equals the diagonal of $X^T X$.

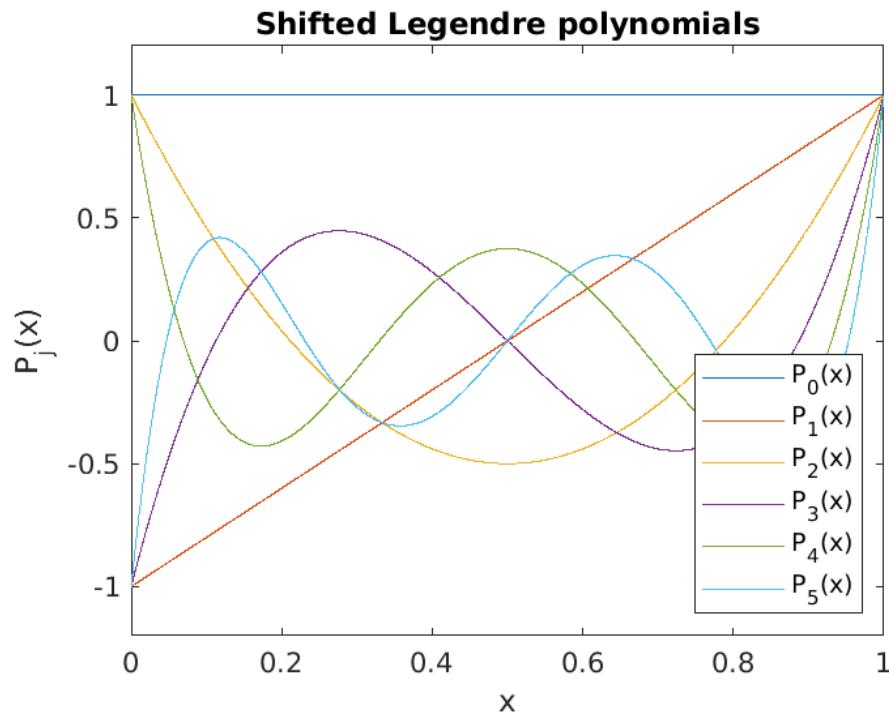
Solution.

- Here is our implementation: [ShiftedLegendre.m](#). (Assignments/Week03/answers/ShiftedLegendre.m)
- The graph of the condition number, as a function of n is given by



We notice that the matrices created from shifted Legendre polynomials have a very good condition numbers.

- The shifted Legendre polynomials are visualized as



- The columns of the matrix X are now reasonably orthogonal:

$X^T * X$ for $n=5$:

ans =

$$\begin{matrix} 5000 & 0 & 1 & 0 & 1 \\ 0 & 1667 & 0 & 1 & 0 \\ 1 & 0 & 1001 & 0 & 1 \\ 0 & 1 & 0 & 715 & 0 \\ 1 & 0 & 1 & 0 & 556 \end{matrix}$$



YouTube: <https://www.youtube.com/watch?v=syq-jOKWqTQ>

Remark 3.1.1.1 The point is that one ideally formulates a problem in a way that already captures orthogonality, so that when the problem is discretized ("sampled"), the matrices that arise will likely inherit that orthogonality, which we will see again and again is a good thing. In this chapter, we discuss how orthogonality can be exposed if it is not already part

of the underlying formulation of the problem.

3.1.2 Overview Week 3

- 3.1 Opening Remarks
 - 3.1.1 Choosing the right basis
 - 3.1.2 Overview Week 3
 - 3.1.3 What you will learn
- 3.2 3.2 Gram-Schmidt Orthogonalization
 - 3.2.1 Classical Gram-Schmidt (CGS)
 - 3.2.2 Gram-Schmidt and the QR factorization
 - 3.2.3 Classical Gram-Schmidt algorithm
 - 3.2.4 Modified Gram-Schmidt (MGS)
 - 3.2.5 In practice, MGS is more accurate
 - 3.2.6 Cost of Gram-Schmidt algorithms
- 3.3 Householder QR Factorization
 - 3.3.1 Using unitary matrices
 - 3.3.2 Householder transformation
 - 3.3.3 Practical computation of the Householder vector
 - 3.3.4 Householder QR factorization algorithm
 - 3.3.5 Forming Q
 - 3.3.6 Applying QH
 - 3.3.7 Orthogonality of resulting Q
- 3.4 Enrichments
 - 3.4.1 Blocked Householder QR factorization
- 3.5 Wrap Up
 - 3.5.1 Additional homework
 - 3.5.2 Summary

3.1.3 What you will learn

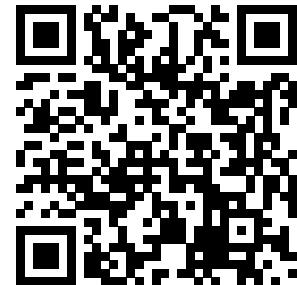
This chapter focuses on the QR factorization as a method for computing an orthonormal basis for the column space of a matrix.

Upon completion of this week, you should be able to

- Relate Gram-Schmidt orthogonalization of vectors to the QR factorization of a matrix.
- Show that Classical Gram-Schmidt and Modified Gram-Schmidt yield the same result (in exact arithmetic).
- Compare and contrast the Classical Gram-Schmidt and Modified Gram-Schmidt methods with regard to cost and robustness in the presence of roundoff error.
- Derive and explain the Householder transformations (reflections).
- Decompose a matrix to its QR factorization via the application of Householder transformations.
- Analyze the cost of the Householder QR factorization algorithm.
- Explain why Householder QR factorization yields a matrix Q with high quality orthonormal columns, even in the presence of roundoff error.

3.2 Gram-Schmidt Orthogonalization

3.2.1 Classical Gram-Schmidt (CGS)



YouTube: <https://www.youtube.com/watch?v=CWhBZB-3kg4>

Given a set of linearly independent vectors $\{a_0, \dots, a_{n-1}\} \subset \mathbb{C}^m$, the Gram-Schmidt process computes an orthonormal basis $\{q_0, \dots, q_{n-1}\}$ that spans the same subspace as the original vectors, i.e.

$$\text{Span}(\{a_0, \dots, a_{n-1}\}) = \text{Span}(\{q_0, \dots, q_{n-1}\}).$$

The process proceeds as follows:

- Compute vector q_0 of unit length so that $\text{Span}(\{a_0\}) = \text{Span}(\{q_0\})$:

- $\rho_{0,0} = \|a_0\|_2$
Computes the length of vector a_0 .
- $q_0 = a_0 / \rho_{0,0}$
Sets q_0 to a unit vector in the direction of a_0 .

Notice that $a_0 = q_0 \rho_{0,0}$

- Compute vector q_1 of unit length so that $\text{Span}(\{a_0, a_1\}) = \text{Span}(\{q_0, q_1\})$:

- $\rho_{0,1} = q_0^H a_1$
Computes $\rho_{0,1}$ so that $\rho_{0,1} q_0 = q_0^H a_1 q_0$ equals the component of a_1 in the direction of q_0 .
- $a_1^\perp = a_1 - \rho_{0,1} q_0$
Computes the component of a_1 that is orthogonal to q_0 .
- $\rho_{1,1} = \|a_1^\perp\|_2$
Computes the length of vector a_1^\perp .
- $q_1 = a_1^\perp / \rho_{1,1}$
Sets q_1 to a unit vector in the direction of a_1^\perp .

Notice that

$$\left(\begin{array}{c|c} a_0 & a_1 \end{array} \right) = \left(\begin{array}{c|c} q_0 & q_1 \end{array} \right) \left(\begin{array}{c|c} \rho_{0,0} & \rho_{0,1} \\ \hline 0 & \rho_{1,1} \end{array} \right).$$

- Compute vector q_2 of unit length so that $\text{Span}(\{a_0, a_1, a_2\}) = \text{Span}(\{q_0, q_1, q_2\})$:

- $\rho_{0,2} = q_0^H a_2$ or, equivalently, $\begin{pmatrix} \rho_{0,2} \\ \rho_{1,2} \end{pmatrix} = \begin{pmatrix} q_0 & q_1 \end{pmatrix}^H a_2$
Computes $\rho_{0,2}$ so that $\rho_{0,2} q_0 = q_0^H a_2 q_0$ and $\rho_{1,2} q_1 = q_1^H a_2 q_1$ equal the components of a_2 in the directions of q_0 and q_1 .
Or, equivalently, $\begin{pmatrix} q_0 & q_1 \end{pmatrix} \begin{pmatrix} \rho_{0,2} \\ \rho_{1,2} \end{pmatrix}$ is the component in $\text{Span}(\{q_0, q_1\})$.
- $a_2^\perp = a_2 - \rho_{0,2} q_0 - \rho_{1,2} q_1 = a_2 - \begin{pmatrix} q_0 & q_1 \end{pmatrix} \begin{pmatrix} \rho_{0,2} \\ \rho_{1,2} \end{pmatrix}$
Computes the component of a_2 that is orthogonal to q_0 and q_1 .
- $\rho_{2,2} = \|a_2^\perp\|_2$
Computes the length of vector a_2^\perp .
- $q_2 = a_2^\perp / \rho_{2,2}$
Sets q_2 to a unit vector in the direction of a_2^\perp .

Notice that

$$\left(\begin{array}{cc|c} a_0 & a_1 & a_2 \end{array} \right) = \left(\begin{array}{cc|c} q_0 & q_1 & q_2 \end{array} \right) \left(\begin{array}{cc|c} \rho_{0,0} & \rho_{0,1} & \rho_{0,2} \\ \hline 0 & \rho_{1,1} & \rho_{1,2} \\ \hline 0 & 0 & \rho_{2,2} \end{array} \right).$$

- And so forth.



YouTube: https://www.youtube.com/watch?v=AvXe0MfKl_0

Yet another way of looking at this problem is as follows.



YouTube: <https://www.youtube.com/watch?v=OZelM7YUwZo>

Consider the matrices

$$A = \left(\begin{array}{c|c|c|c|c|c|c} a_0 & \cdots & a_{k-1} & a_k & a_{k+1} & \cdots & a_{n-1} \end{array} \right)$$

and

$$Q = \left(\begin{array}{c|c|c|c|c|c} q_0 & \cdots & q_{k-1} & q_k & q_{k+1} & \cdots & q_{n-1} \end{array} \right)$$

We observe that

- $\text{Span}(\{a_0\}) = \text{Span}(\{q_0\})$
Hence $a_0 = \rho_{0,0}q_0$ for some scalar $\rho_{0,0}$.

- $\text{Span}(\{a_0, a_1\}) = \text{Span}(\{q_0, q_1\})$

Hence

$$a_1 = \rho_{0,1}q_0 + \rho_{1,1}q_1$$

for some scalars $\rho_{0,1}, \rho_{1,1}$.

- In general, $\text{Span}(\{a_0, \dots, a_{k-1}, a_k\}) = \text{Span}(\{q_0, \dots, q_{k-1}, q_k\})$

Hence

$$a_k = \rho_{0,k}q_0 + \cdots + \rho_{k-1,k}q_{k-1} + \rho_{k,k}q_k$$

for some scalars $\rho_{0,k}, \dots, \rho_{k,k}$.

Let's assume that q_0, \dots, q_{k-1} have already been computed and are mutually orthonormal.
Consider

$$a_k = \rho_{0,k}q_0 + \cdots + \rho_{k-1,k}q_{k-1} + \rho_{k,k}q_k.$$

Notice that

$$\begin{aligned} q_k^H a_k &= q_k^H (\rho_{0,k} q_0 + \cdots + \rho_{k-1,k} q_{k-1} + \rho_{k,k} q_k) \\ &= \underbrace{\rho_{0,k} q_k^H q_0}_0 + \cdots + \underbrace{\rho_{k-1,k} q_k^H q_{k-1}}_0 + \underbrace{\rho_{k,k} q_k^H q_k}_1 \end{aligned}$$

so that

$$\rho_{i,k} = q_i^H a_k,$$

for $i = 0, \dots, k - 1$. Next, we can compute

$$a_k^\perp = a_k - \rho_{0,k} q_0 - \cdots - \rho_{k-1,k} q_{k-1}$$

and, since $\rho_{k,k} q_k = a_k^\perp$, we can choose

$$\rho_{k,k} = \|a_k^\perp\|_2$$

and

$$q_k = a_k^\perp / \rho_{k,k}$$

Remark 3.2.1.1 For a review of Gram-Schmidt orthogonalization and exercises orthogonalizing real-valued vectors, you may want to look at Linear Algebra: Foundations to Frontiers (LAFF) [26] Week 11.

3.2.2 Gram-Schmidt and the QR factorization



YouTube: <https://www.youtube.com/watch?v=tHj20PSBCek>

The discussion in the last unit motivates the following theorem:

Theorem 3.2.2.1 QR Decomposition Theorem. Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns. Then there exists an orthonormal matrix Q and upper triangular matrix R such that $A = QR$, its QR decomposition. If the diagonal elements of R are taken to be real and positive, then the decomposition is unique.

In order to prove this theorem elegantly, we will first present the Gram-Schmidt orthogonalization algorithm using FLAME notation, in the next unit.

Ponder This 3.2.2.1 What happens in the Gram-Schmidt algorithm if the columns of A are NOT linearly independent? How might one fix this? How can the Gram-Schmidt algorithm be used to identify which columns of A are linearly independent?

Solution. If a_j is the first column such that $\{a_0, \dots, a_j\}$ are linearly dependent, then a_j^\perp will equal the zero vector and the process breaks down.

When a vector with a_j^\perp equal to the zero vector is encountered, the columns can be rearranged (permuted) so that that column (or those columns) come last.

Again, if $a_j^\perp = 0$ for some j , then the columns are linearly dependent since then a_j can be written as a linear combination of the previous columns.

3.2.3 Classical Gram-Schmidt algorithm



YouTube: <https://www.youtube.com/watch?v=YEEEJYp8snQ>

Remark 3.2.3.1 If the FLAME notation used in this unit is not intuitively obvious, you may to review some of the materials in Weeks 3-5 of Linear Algebra: Foundations to Frontiers (<http://www.ulaff.net>).

An alternative for motivating that algorithm is as follows:

- Consider $A = QR$.
- Partition A , Q , and R to yield

$$\left(\begin{array}{c|cc} A_0 & a_1 & a_2 \end{array} \right) = \left(\begin{array}{c|cc} Q_0 & q_1 & Q_2 \end{array} \right) \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array} \right).$$

- Assume that Q_0 and R_{00} have already been computed.
- Since corresponding columns of both sides must be equal, we find that

$$a_1 = Q_0 r_{01} + q_1 \rho_{11}. \quad (3.2.1)$$

Also, $Q_0^H Q_0 = I$ and $Q_0^H q_1 = 0$, since the columns of Q are mutually orthonormal.

- Hence

$$Q_0^H a_1 = Q_0^H Q_0 r_{01} + Q_0^H q_1 \rho_{11} = r_{01}.$$

- This shows how r_{01} can be computed from Q_0 and a_1 , which are already known:

$$r_{01} := Q_0^H a_1.$$

- Next,

$$a_1^\perp := a_1 - Q_0 r_{01}$$

is computed from (3.2.1). This is the component of a_1 that is perpendicular (orthogonal) to the columns of Q_0 . We know it is nonzero since the columns of A are linearly independent.

- Since $\rho_{11}q_1 = a_1^\perp$ and we know that q_1 has unit length, we now compute

$$\rho_{11} := \|a_1^\perp\|_2$$

and

$$q_1 := a_1^\perp / \rho_{11},$$

These insights are summarized in the algorithm in Figure 3.2.3.2.

$[Q, R] = \text{CGS-QR}(A)$
$A \rightarrow (A_L \mid A_R), Q \rightarrow (Q_L \mid Q_R), R \rightarrow \left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right)$
A_L and Q_L has 0 columns and R_{TL} is 0×0
while $n(A_L) < n(A)$
$(A_L \mid A_R) \rightarrow (A_0 \mid a_1 \ a_2), (Q_L \mid Q_R) \rightarrow (Q_0 \mid q_1 \ Q_2),$
$\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array} \right)$
$r_{01} := Q_0^H a_1$
$a_1^\perp := a_1 - Q_0 r_{01}$
$\rho_{11} := \ a_1^\perp\ _2$
$q_1 := a_1^\perp / \rho_{11}$
$(A_L \mid A_R) \leftarrow (A_0 \ a_1 \mid A_2), (Q_L \mid Q_R) \leftarrow (Q_0 \ q_1 \mid Q_2),$
$\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array} \right)$
endwhile

Figure 3.2.3.2 (Classical) Gram-Schmidt (CGS) algorithm for computing the QR factorization of a matrix A .

Having presented the algorithm in FLAME notation, we can provide a formal proof of Theorem 3.2.2.1.

Proof of Theorem 3.2.2.1. Informal proof: The process described earlier in this unit constructs the QR decomposition. The computation of $\rho_{j,j}$ is unique if it is restricted to be a real and positive number. This then prescribes all other results along the way.

Formal proof:

(By induction). Note that $n \leq m$ since A has linearly independent columns.

- Base case: $n = 1$. In this case $A = (A_0 \mid a_1)$, where A_0 has no columns. Since A has

linearly independent columns, $a_1 \neq 0$. Then

$$A = \begin{pmatrix} a_1 \end{pmatrix} = (q_1)(\rho_{11}),$$

where $\rho_{11} = \|a_1\|_2$ and $q_1 = a_1/\rho_{11}$, so that $Q = (q_1)$ and $R = (\rho_{11})$.

- Inductive step: Assume that the result is true for all A_0 with k linearly independent columns. We will show it is true for A with $k + 1$ linearly independent columns.

Let $A \in \mathbb{C}^{m \times (k+1)}$. Partition $A \rightarrow \begin{pmatrix} A_0 & | & a_1 \end{pmatrix}$.

By the induction hypothesis, there exist Q_0 and R_{00} such that $Q_0^H Q_0 = I$, R_{00} is upper triangular with nonzero diagonal entries and $A_0 = Q_0 R_{00}$. Also, by induction hypothesis, if the elements on the diagonal of R_{00} are chosen to be positive, then the factorization $A_0 = Q_0 R_{00}$ is unique.

We are looking for

$$\left(\begin{array}{c|c} \tilde{Q}_0 & q_1 \end{array} \right) \text{ and } \left(\begin{array}{c|c} \tilde{R}_{00} & r_{01} \\ \hline 0 & \rho_{11} \end{array} \right)$$

so that

$$\left(\begin{array}{c|c} A_0 & a_1 \end{array} \right) = \left(\begin{array}{c|c} \tilde{Q}_0 & q_1 \end{array} \right) \left(\begin{array}{c|c} \tilde{R}_{00} & r_{01} \\ \hline 0 & \rho_{11} \end{array} \right).$$

This means that

- $A_0 = \tilde{Q}_0 \tilde{R}_{00}$,

We choose $\tilde{Q}_0 = Q_0$ and $\tilde{R}_{00} = R_{00}$. If we insist that the elements on the diagonal be positive, this choice is unique. Otherwise, it is a choice that allows us to prove existence.

- $a_1 = Q_0 r_{01} + \rho_{11} q_1$ which is the unique choice if we insist on positive elements on the diagonal.

$a_1 = Q_0 r_{01} + \rho_{11} q_1$. Multiplying both sides by Q_0^H we find that r_{01} must equal $Q_0^H a_1$ (and is uniquely determined by this if we insist on positive elements on the diagonal).

- Letting $a_1^\perp = a_1 - Q_0 r_{01}$ (which equals the component of a_1 orthogonal to $\mathcal{C}(Q_0)$), we find that $\rho_{11} q_1 = a_1^\perp$. Since q_1 has unit length, we can choose $\rho_{11} = \|a_1^\perp\|_2$. If we insist on positive elements on the diagonal, then this choice is unique.

- Finally, we let $q_1 = a_1^\perp / \rho_{11}$.

- By the Principle of Mathematical Induction the result holds for all matrices $A \in \mathbb{C}^{m \times n}$ with $m \geq n$.

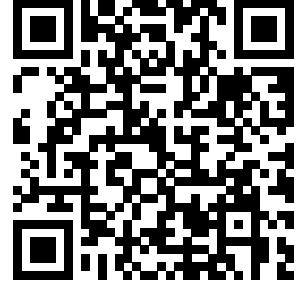
■

Homework 3.2.3.1 Implement the algorithm given in Figure 3.2.3.2 as
function [Q, R] = CGS_QR(A)

by completing the code in [Assignments/Week03/matlab/CGS_QR.m](#). Input is an $m \times n$ matrix A . Output is the matrix Q and the upper triangular matrix R . You may want to use [Assignments/Week03/matlab/test_CGS_QR.m](#) to check your implementation.

Solution. See [Assignments/Week03/answers/CGS_QR.m](#). ([Assignments/Week03/answers/CGS_QR.m](#))

3.2.4 Modified Gram-Schmidt (MGS)



YouTube: <https://www.youtube.com/watch?v=pOBJHhV3TKY>

In the video, we reasoned that the following two algorithms compute the same values, except that the columns of Q overwrite the corresponding columns of A :

```

for  $j = 0, \dots, n - 1$ 
   $a_j^\perp := a_j$ 
  for  $k = 0, \dots, j - 1$ 
     $\rho_{k,j} := q_k^H a_j^\perp$ 
     $a_j^\perp := a_j^\perp - \rho_{k,j} q_k$ 
  end
   $\rho_{j,j} := \|a_j^\perp\|_2$ 
   $q_j := a_j^\perp / \rho_{j,j}$ 
end

```

```

for  $j = 0, \dots, n - 1$ 
  for  $k = 0, \dots, j - 1$ 
     $\rho_{k,j} := a_k^H a_j$ 
     $a_j := a_j - \rho_{k,j} a_k$ 
  end
   $\rho_{j,j} := \|a_j\|_2$ 
   $a_j := a_j / \rho_{j,j}$ 
end

```

(a) MGS algorithm that computes Q and R from A .

(b) MGS algorithm that computes Q and R from A , overwriting A with Q .

Homework 3.2.4.1 Assume that q_0, \dots, q_{k-1} are mutually orthonormal. Let $\rho_{j,k} = q_j^H y$ for $j = 0, \dots, i - 1$. Show that

$$\underbrace{q_i^H y}_{\rho_{i,k}} = q_i^H (y - \rho_{0,k} q_0 - \dots - \rho_{i-1,k} q_{i-1})$$

for $i = 0, \dots, k - 1$.

Solution.

$$\begin{aligned}
 q_i^H (y - \rho_{0,k}q_0 - \cdots - \rho_{i-1,k}q_{i-1}) \\
 &= \quad <\text{distribute}> \\
 q_i^H y - q_i^H \rho_{0,k}q_0 - \cdots - q_i^H \rho_{i-1,k}q_{i-1} \\
 &= \quad <\rho_{0,k} \text{ is a scalar}> \\
 q_i^H y - \rho_{0,k} \underbrace{q_i^H q_0}_{0} - \cdots - \underbrace{\rho_{i-1,k} q_i^H q_{i-1}}_0
 \end{aligned}$$



YouTube: <https://www.youtube.com/watch?v=0ooNPondq5M>

This homework illustrates how, given a vector $y \in \mathbb{C}^m$ and a matrix $Q \in \mathbb{C}^{m \times k}$ the component orthogonal to the column space of Q , given by $(I - QQ^H)y$, can be computed by either of the two algorithms given in [Figure 3.2.4.1](#). The one on the left, $\text{Proj } \perp Q_{\text{CGS}}(Q, y)$ projects y onto the column space perpendicular to Q as did the Gram-Schmidt algorithm with which we started. The one on the right successfully subtracts out the component in the direction of q_i using a vector that has been updated in previous iterations (and hence is already orthogonal to q_0, \dots, q_{i-1}). The algorithm on the right is one variant of the Modified Gram-Schmidt (MGS) algorithm.

$[y^\perp, r] = \text{Proj } \perp Q_{\text{CGS}}(Q, y)$ (used by CGS)	$[y^\perp, r] = \text{Proj } \perp Q_{\text{MGS}}(Q, y)$ (used by MGS)
$y^\perp = y$ for $i = 0, \dots, k - 1$ $\rho_i := q_i^H y$ $y^\perp := y^\perp - \rho_i q_i$ endfor	$y^\perp = y$ for $i = 0, \dots, k - 1$ $\rho_i := q_i^H y^\perp$ $y^\perp := y^\perp - \rho_i q_i$ endfor

Figure 3.2.4.1 Two different ways of computing $y^\perp = (I - QQ^H)y = y - Qr$, where $r = Q^H y$. The computed y^\perp is the component of y orthogonal to $\mathcal{C}(Q)$, where Q has k orthonormal columns. (Notice the y on the left versus the y^\perp on the right in the computation of ρ_i .)

These insights allow us to present CGS and this variant of MGS in FLAME notation, in [Figure 3.2.4.2](#) (left and middle).

$[A, R] := \text{GS}(A)$ (overwrites A with Q)		
$A \rightarrow \left(\begin{array}{c c} A_L & A_R \end{array} \right), R \rightarrow \left(\begin{array}{c c} R_{TL} & R_{TR} \\ 0 & R_{BR} \end{array} \right)$		A_L has 0 columns and R_{TL} is 0×0
while $n(A_L) < n(A)$		
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c c} R_{TL} & R_{TR} \\ 0 & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array} \right)$		
CGS	MGS	MGS (alternative)
$r_{01} := A_0^H a_1$	$[a_1, r_{01}] = \text{Proj}_{\perp} \text{toQ}_{\text{MGS}}(A_0, a_1)$	
$a_1 := a_1 - A_0 r_{01}$	$\rho_{11} := \ a_1\ _2$	$\rho_{11} := \ a_1\ _2$
$\rho_{11} := \ a_1\ _2$	$a_1 := a_1 / \rho_{11}$	$a_1 := a_1 / \rho_{11}$
$a_1 := a_1 / \rho_{11}$		$r_{12}^T := a_1^H A_2$
		$A_2 := A_2 - a_1 r_{12}^T$
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c c} R_{TL} & R_{TR} \\ 0 & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array} \right)$		
endwhile		

Figure 3.2.4.2 Left: Classical Gram-Schmidt algorithm. Middle: Modified Gram-Schmidt algorithm. Right: Alternative Modified Gram-Schmidt algorithm. In this last algorithm, every time a new column, q_1 , of Q is computed, each column of A_2 is updated so that its component in the direction of q_1 is subtracted out. This means that at the start and finish of the current iteration, the columns of A_L are mutually orthonormal and the columns of A_R are orthogonal to the columns of A_L .

Next, we massage the MGS algorithm into the alternative MGS algorithmic variant given in Figure 3.2.4.2 (right).



YouTube: <https://www.youtube.com/watch?v=3XzHFWzV5iE>

The video discusses how MGS can be rearranged so that every time a new vector q_k is computed (overwriting a_k), the remaining vectors, $\{a_{k+1}, \dots, a_{n-1}\}$, can be updated by subtracting out the component in the direction of q_k . This is also illustrated through the next sequence of equivalent algorithms.

```

for  $j = 0, \dots, n - 1$ 
   $\rho_{j,j} := \|a_j\|_2$ 
   $a_j := a_j / \rho_{j,j}$ 
  for  $k = j + 1, \dots, n - 1$ 
     $\rho_{j,k} := a_j^H a_k$ 
     $a_k := a_k - \rho_{j,k} a_j$ 
  end
end

```

(c) MGS algorithm that normalizes the j th column to have unit length to compute q_j (overwriting a_j with the result) and then subtracts the component in the direction of q_j off the rest of the columns $(a_{j+1}, \dots, a_{n-1})$.

```

for  $j = 0, \dots, n - 1$ 
   $\rho_{j,j} := \|a_j\|_2$ 
   $a_j := a_j / \rho_{j,j}$ 
  
$$\left( \begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \\ a_j^H \left( \begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right) & := \\ \left( \begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right) & := \\ \left( \begin{array}{c|c|c} a_{j+1} - \rho_{j,j+1} a_j & \cdots & a_{n-1} - \rho_{j,n-1} a_j \end{array} \right) & \end{array} \right)$$

end

```

(e) Algorithm in (d) rewritten to expose only the outer loop.

```

for  $j = 0, \dots, n - 1$ 
   $\rho_{j,j} := \|a_j\|_2$ 
   $a_j := a_j / \rho_{j,j}$ 
  for  $k = j + 1, \dots, n - 1$ 
     $\rho_{j,k} := a_j^H a_k$ 
  end
  for  $k = j + 1, \dots, n - 1$ 
     $a_k := a_k - \rho_{j,k} a_j$ 
  end
end

```

(d) Slight modification of the algorithm in (c) that computes $\rho_{j,k}$ in a separate loop.

```

for  $j = 0, \dots, n - 1$ 
   $\rho_{j,j} := \|a_j\|_2$ 
   $a_j := a_j / \rho_{j,j}$ 
  
$$\left( \begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \\ a_j^H \left( \begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right) & := \\ \left( \begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right) & := \\ \left( \begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right) & - a_j \left( \begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right) \end{array} \right)$$

end

```

(f) Algorithm in (e) rewritten to expose the row-vector-times matrix multiplication $a_j^H \left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)$ and rank-1 update $\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right) - a_j \left(\begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right)$.

Figure 3.2.4.3 Various equivalent MGS algorithms.

This discussion shows that the updating of future columns by subtracting out the component in the direction of the latest column of Q to be computed can be cast in terms of a rank-1 update. This is also captured, using FLAME notation, in the algorithm in Figure 3.2.4.2, as is further illustrated in Figure 3.2.4.4:

Algorithm: $[A, R] := \text{MGS}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right),$
 $R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right)$
where A_L has 0 columns and R_{TL} is 0×0

while $n(A_L) < n(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right),$$

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array} \right)$$

$\rho_{11} := \|a_1\|_2$
 $a_1 := a_1 / \rho_{11}$
 $r_{12}^T := a_1^H A_2$
 $A_2 := A_2 - a_1 r_{12}^T$

Continue with

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right),$$

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array} \right)$$

endwhile

for $j = 0, \dots, n-1$
 $\rho_{j,j} := \|a_j\|_2$ ($\rho_{11} := \|a_1\|_2$)
 $a_j := a_j / \rho_{j,j}$ ($a_1 := a_1 / \rho_{11}$)

$$\overbrace{\left(\begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right)}^{r_{12}^T} := \underbrace{\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)}_{a_1^H A_2}$$

$$\overbrace{\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)}^{A_2} - \underbrace{a_j}_{a_1} \overbrace{\left(\begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right)}^{r_{12}^T} := \overbrace{\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)}^{A_2}$$

end

Figure 3.2.4.4 Alternative Modified Gram-Schmidt algorithm for computing the QR factorization of a matrix A .



YouTube: <https://www.youtube.com/watch?v=elwc14-1WF0>

Ponder This 3.2.4.2 Let A have linearly independent columns and let $A = QR$ be a QR factorization of A . Partition

$$A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right), \quad Q \rightarrow \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right), \quad \text{and} \quad R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right),$$

where A_L and Q_L have k columns and R_{TL} is $k \times k$.

As you prove the following insights, relate each to the algorithm in [Figure 3.2.4.4](#). In particular, at the top of the loop of a typical iteration, how have the different parts of A and R been updated?

1. $A_L = Q_L R_{TL}$.

($Q_L R_{TL}$ equals the QR factorization of A_L .)

2. $\mathcal{C}(A_L) = \mathcal{C}(Q_L)$.

(The first k columns of Q form an orthonormal basis for the space spanned by the first k columns of A .)

3. $R_{TR} = Q_L^H A_R$.

4. $(A_R - Q_L R_{TR})^H Q_L = 0$.

(Each column in $A_R - Q_L R_{TR}$ equals the component of the corresponding column of A_R that is orthogonal to $\text{Span}(Q_L)$.)

5. $\mathcal{C}(A_R - Q_L R_{TR}) = \mathcal{C}(Q_R)$.

6. $A_R - Q_L R_{TR} = Q_R R_{BR}$.

(The columns of Q_R form an orthonormal basis for the column space of $A_R - Q_L R_{TR}$.)

Solution. Consider the fact that $A = QR$. Then, multiplying the partitioned matrices,

$$\begin{aligned} \left(\begin{array}{c|c} A_L & A_R \end{array} \right) &= \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right) \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \\ &= \left(\begin{array}{c|c} Q_L R_{TL} & Q_L R_{TR} + Q_R R_{BR} \end{array} \right). \end{aligned}$$

Hence

$$A_L = Q_L R_{TL} \quad \text{and} \quad A_R = Q_L R_{TR} + Q_R R_{BR}. \quad (3.2.2)$$

1. The left equality in [\(3.2.2\)](#) answers 1.

2. $\mathcal{C}(A_L) = \mathcal{C}(Q_L)$ can be shown by noting that R is upper triangular and nonsingular and hence R_{TL} is upper triangular and nonsingular, and using this to show that $\mathcal{C}(A_L) \subset \mathcal{C}(Q_L)$ and $\mathcal{C}(Q_L) \subset \mathcal{C}(A_L)$:

- $\mathcal{C}(A_L) \subset \mathcal{C}(Q_L)$: Let $y \in \mathcal{C}(A_L)$. Then there exists x such that $A_L x = y$. But then $Q_L R_{TL} x = y$ and hence $Q_L(R_{TL} x) = y$ which means that $y \in \mathcal{C}(Q_L)$.
- $\mathcal{C}(Q_L) \subset \mathcal{C}(A_L)$: Let $y \in \mathcal{C}(Q_L)$. Then there exists x such that $Q_L x = y$. But then $A_L R_{TL}^{-1} x = y$ and hence $A_L(R_{TL}^{-1} x) = y$ which means that $y \in \mathcal{C}(A_L)$.

This answers 2.

3. Take $A_R - Q_L R_{TR} = Q_R R_{BR}$ and multiply both side by Q_L^H :

$$Q_L^H (A_R - Q_L R_{TR}) = Q_L^H Q_R R_{BR}$$

is equivalent to

$$Q_L^H A_R - \underbrace{Q_L^H Q_L}_I R_{TR} = \underbrace{Q_L^H Q_R}_0 R_{BR} = 0.$$

Rearranging yields 3.

4. Since $A_R - Q_L R_{TR} = Q_R R_{BR}$ we find that $(A_R - Q_L R_{TR})^H Q_L = (Q_R R_{BR})^H Q_L$ and

$$(A_R - Q_L R_{TR})^H Q_L = R_{BR}^H Q_R^H Q_L = 0.$$

5. Similar to the proof of 2.

6. Rearranging the right equality in (3.2.2) yields $A_R - Q_L R_{TR} = Q_R R_{BR}$, which answers 5.

7. Letting \hat{A} denote the original contents of A , at a typical point,

- A_L has been updated with Q_L .
- R_{TL} and R_{TR} have been computed.
- $A_R = \hat{A}_R - Q_L R_{TR}$.

Homework 3.2.4.3 Implement the algorithm in Figure 3.2.4.4 as
function [Aout, Rout] = MGS_QR(A, R)

Input is an $m \times n$ matrix A and a $n \times n$ matrix R . Output is the matrix Q , which has overwritten matrix A , and the upper triangular matrix R . (The values below the diagonal can be arbitrary.) You may want to use [Assignments/Week03/matlab/test_MGS_QR.m](#) to check your implementation.

Solution. See [Assignments/Week03/answers/MGS_QR.m](#).

3.2.5 In practice, MGS is more accurate



YouTube: <https://www.youtube.com/watch?v=7ArZnHE0PIw>

In theory, all Gram-Schmidt algorithms discussed in the previous sections are equivalent

in the sense that they compute the exact same QR factorizations when exact arithmetic is employed. In practice, in the presence of round-off error, the orthonormal columns of Q computed by MGS are often "more orthonormal" than those computed by CGS. We will analyze how round-off error affects linear algebra computations in the second part of the ALAFF. For now you will investigate it with a classic example.

When storing real (or complex) valued numbers in a computer, a limited accuracy can be maintained, leading to round-off error when a number is stored and/or when computation with numbers is performed. Informally, the machine epsilon (also called the unit roundoff error) is defined as the largest positive number, ϵ_{mach} , such that the stored value of $1 + \epsilon_{\text{mach}}$ is rounded back to 1.

Now, let us consider a computer where the only error that is ever incurred is when

$$1 + \epsilon_{\text{mach}}$$

is computed and rounded to 1.

Homework 3.2.5.1 Let $\epsilon = \sqrt{\epsilon_{\text{mach}}}$ and consider the matrix

$$A = \left(\begin{array}{c|c|c} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{array} \right) = \left(\begin{array}{c|c|c} a_0 & a_1 & a_2 \end{array} \right). \quad (3.2.3)$$

By hand, apply both the CGS and the MGS algorithms with this matrix, rounding $1 + \epsilon_{\text{mach}}$ to 1 whenever encountered in the calculation.

Upon completion, check whether the columns of Q that are computed are (approximately) orthonormal.

Solution. The complete calculation is given by

CGS	MGS
<u>First iteration</u> $\rho_{0,0} = \ a_0\ _2 = \sqrt{1+\epsilon^2} = \sqrt{1+\epsilon_{\text{mach}}}$ which is rounded to 1. $q_0 = a_0 / \rho_{0,0} = \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix} / 1 = \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix}$	<u>First iteration</u> $\rho_{0,0} = \ a_0\ _2 = \sqrt{1+\epsilon^2} = \sqrt{1+\epsilon_{\text{mach}}}$ which is rounded to 1. $q_0 = a_0 / \rho_{0,0} = \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix} / 1 = \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix}$
<u>Second iteration</u> $\rho_{0,1} = q_0^H a_1 = 1$ $a_1^\perp = a_1 - \rho_{0,1} q_0 = \begin{pmatrix} 0 \\ -\epsilon \\ \epsilon \\ 0 \end{pmatrix}$ $\rho_{1,1} = \ a_1^\perp\ _2 = \sqrt{2\epsilon^2} = \sqrt{2}\epsilon$ $q_1 = a_1^\perp / \rho_{1,1} = \begin{pmatrix} 0 \\ -\epsilon \\ \epsilon \\ 0 \end{pmatrix} / (\sqrt{2}\epsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}$	<u>Second iteration</u> $\rho_{0,1} = q_0^H a_1 = 1$ $a_1^\perp = a_1 - \rho_{0,1} q_0 = \begin{pmatrix} 0 \\ -\epsilon \\ \epsilon \\ 0 \end{pmatrix}$ $\rho_{1,1} = \ a_1^\perp\ _2 = \sqrt{2\epsilon^2} = \sqrt{2}\epsilon$ $q_1 = a_1^\perp / \rho_{1,1} = \begin{pmatrix} 0 \\ -\epsilon \\ \epsilon \\ 0 \end{pmatrix} / (\sqrt{2}\epsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}$
<u>Third iteration</u> $\rho_{0,2} = q_0^H a_2 = 1$ $\rho_{1,2} = q_1^H a_2 = 0$ $a_2^\perp = a_2 - \rho_{0,2} q_0 - \rho_{1,2} q_1 = \begin{pmatrix} 0 \\ -\epsilon \\ 0 \\ \epsilon \end{pmatrix}$ $\rho_{2,2} = \ a_2^\perp\ _2 = \sqrt{2\epsilon^2} = \sqrt{2}\epsilon$ $q_2 = a_2^\perp / \rho_{2,2} = \begin{pmatrix} 0 \\ -\epsilon \\ 0 \\ \epsilon \end{pmatrix} / (\sqrt{2}\epsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ 0 \\ \frac{\sqrt{2}}{2} \end{pmatrix}$	<u>Third iteration</u> $\rho_{0,2} = q_0^H a_2 = 1$ $\rho_{1,2} = q_1^H a_2^\perp = (\sqrt{2}/2)\epsilon$ $a_2^\perp = a_2^\perp - \rho_{1,2} q_1 = \begin{pmatrix} 0 \\ -\epsilon/2 \\ -\epsilon/2 \\ \epsilon \end{pmatrix}$ $\rho_{2,2} = \ a_2^\perp\ _2 = \sqrt{(6/4)\epsilon^2} = (\sqrt{6}/2)\epsilon$ $q_2 = a_2^\perp / \rho_{2,2} = \begin{pmatrix} 0 \\ -\frac{\epsilon}{2} \\ -\frac{\epsilon}{2} \\ \epsilon \end{pmatrix} / (\frac{\sqrt{6}}{2}\epsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{6}}{6} \\ -\frac{\sqrt{6}}{6} \\ \frac{\sqrt{6}}{3} \end{pmatrix}$

[Click here](#) to enlarge.

CGS yields the approximate matrix

$$Q \approx \left(\begin{array}{c|cc|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{array} \right)$$

while MGS yields

$$Q \approx \left(\begin{array}{c|cc|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & 0 & \frac{\sqrt{6}}{3} \end{array} \right)$$

Clearly, they don't compute the same answer.

If we now ask the question "Are the columns of Q orthonormal?" we can check this by computing $Q^H Q$, which should equal I , the identity.

- For CGS:

$$\begin{aligned} Q^H Q &= \\ &\left(\begin{array}{c|cc|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{array} \right)^H \left(\begin{array}{c|cc|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{array} \right) \\ &= \\ &\left(\begin{array}{ccc} 1 + \epsilon_{\text{mach}} & -\frac{\sqrt{2}}{2}\epsilon & -\frac{\sqrt{2}}{2}\epsilon \\ -\frac{\sqrt{2}}{2}\epsilon & 1 & \frac{1}{2} \\ -\frac{\sqrt{2}}{2}\epsilon & \frac{1}{2} & 1 \end{array} \right). \end{aligned}$$

Clearly, the computed second and third columns of Q are not mutually orthonormal.

What is going on? The answer lies with how a_2^\perp is computed in the last step $a_2^\perp := a_2 - (q_0^H a_2)q_0 - (q_1^H a_2)q_1$. Now, q_0 has a relatively small error in it and hence $q_0^H a_2 q_0$ has a relatively small error in it. It is likely that a part of that error is in the direction of q_1 . Relative to $q_0^H a_2 q_0$, that error in the direction of q_1 is small, but relative to $a_2 - q_0^H a_2 q_0$ it is not. The point is that then $a_2 - q_0^H a_2 q_0$ has a relatively large error in it in the direction of q_1 . Subtracting $q_1^H a_2 q_1$ does not fix this and since in the end a_2^\perp is small, it has a relatively large error in the direction of q_1 . This error is amplified when q_2 is computed by normalizing a_2^\perp .

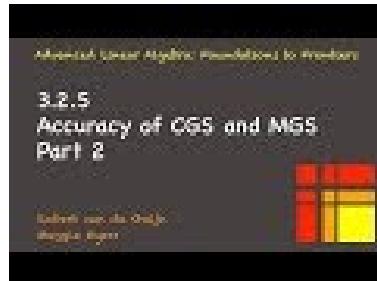
- For MGS:

$$\begin{aligned}
 Q^H Q &= \\
 &= \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & 0 & \frac{\sqrt{6}}{3} \end{array} \right)^H \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & 0 & \frac{\sqrt{6}}{3} \end{array} \right) \\
 &= \left(\begin{array}{ccc} 1 + \epsilon_{\text{mach}} & -\frac{\sqrt{2}}{2}\epsilon & -\frac{\sqrt{6}}{6}\epsilon \\ -\frac{\sqrt{2}}{2}\epsilon & 1 & 0 \\ -\frac{\sqrt{6}}{6}\epsilon & 0 & 1 \end{array} \right).
 \end{aligned}$$

Why is the orthogonality better? Consider the computation of $a_2^\perp := a_2 - (q_1^H a_2) q_1$:

$$a_2^\perp := a_2^\perp - q_1^H a_2^\perp q_1 = [a_2 - (q_0^H a_2) q_0] - (q_1^H [a_2 - (q_0^H a_2) q_0]) q_1.$$

This time, if $a_2 - q_0^H a_2^\perp q_0$ has an error in the direction of q_1 , this error is subtracted out when $(q_1^H a_2^\perp) q_1$ is subtracted from a_2^\perp . This explains the better orthogonality between the computed vectors q_1 and q_2 .



YouTube: <https://www.youtube.com/watch?v=OT4Yd-eVMS0>

We have argued via an example that MGS is more accurate than CGS. A more thorough analysis is needed to explain why this is generally so.

3.2.6 Cost of Gram-Schmidt algorithms

(No video for this unit.)

Homework 3.2.6.1 Analyze the cost of the CGS algorithm in [Figure 3.2.4.2](#) (left) assuming that $A \in \mathbb{C}^{m \times n}$.

Solution. During the k th iteration ($0 \leq k < n$), A_0 has k columns and A_2 has $n - k - 1$

columns. In each iteration

Operation	Approximate cost (in flops)
$r_{01} := A_0^H a_1$	$2mk$
$a_1 := a_1 - A_0 r_{01}$	$2mk$
$\rho_{11} := \ a_1\ _2$	$2m$
$a_1 := a_1 / \rho_{11}$	m

Thus, the total cost is (approximately)

$$\begin{aligned}
 & \sum_{k=0}^{n-1} [2mk + 2mk + 2m + m] \\
 & = \\
 & \sum_{k=0}^{n-1} [3m + 4mk] \\
 & = \\
 & 3mn + 4m \sum_{k=0}^{n-1} k \\
 & \approx \quad < \sum_{k=0}^{n-1} k = n(n-1)/2 \approx n^2/2 > \\
 & 3mn + 4m \frac{n^2}{2} \\
 & = \\
 & 3mn + 2mn^2 \\
 & \approx \quad < 3mn \text{ is of lower order} > \\
 & 2mn^2
 \end{aligned}$$

Homework 3.2.6.2 Analyze the cost of the MGS algorithm in [Figure 3.2.4.2](#) (right) assuming that $A \in \mathbb{C}^{m \times n}$.

Solution. During the k th iteration ($0 \leq k < n$), A_0 has k columns. and A_2 has $n - k - 1$ columns. In each iteration

Operation	Approximate cost (in flops)
$\rho_{11} := \ a_1\ _2$	$2m$
$a_1 := a_1 / \rho_{11}$	m
$r_{12}^T := a_1^H A_2$	$2m(n - k - 1)$
$A_2 := A_2 - a_1 r_{12}^T$	$2m(n - k - 1)$

Thus, the total cost is (approximately)

$$\begin{aligned}
 & \sum_{k=0}^{n-1} [2m(n - k - 1) + 2m(n - k - 1) + 2m + m] \\
 & = \\
 & \sum_{k=0}^{n-1} [3m + 4m(n - k - 1)] \\
 & = \\
 & 3mn + 4m \sum_{k=0}^{n-1} (n - k - 1) \\
 & = \quad <\text{Substitute } j = (n - k - 1) > \\
 & 3mn + 4m \sum_{j=0}^{n-1} j \\
 & \approx \quad <\sum_{j=0}^{n-1} j = n(n - 1)/2 \approx n^2/2 > \\
 & 3mn + 4m \frac{n^2}{2} \\
 & = \\
 & 3mn + 2mn^2 \\
 & \approx \quad <3mn \text{ is of lower order} > \\
 & 2mn^2
 \end{aligned}$$

Homework 3.2.6.3 Which algorithm requires more flops?

Solution. They require the approximately same number of flops.

A more careful analysis shows that, in exact arithmetic, they perform exactly the same computations, but in a different order. Hence the number of flops is exactly the same.

3.3 Householder QR Factorization

3.3.1 Using unitary matrices



YouTube: https://www.youtube.com/watch?v=NAdMU_1ZANK

A fundamental problem to avoid in numerical codes is the situation where one starts with large values and one ends up with small values with large relative errors in them. This is known as catastrophic cancellation. The Gram-Schmidt algorithms can inherently fall victim to this: column a_j is successively reduced in length as components in the directions of $\{q_0, \dots, q_{j-1}\}$ are subtracted, leaving a small vector if a_j was almost in the span of the first j columns of A . Application of a unitary transformation to a matrix or vector inherently preserves length. Thus, it would be beneficial if the QR factorization can be implemented as the successive application of unitary transformations. The Householder QR factorization accomplishes this.

The first fundamental insight is that the product of unitary matrices is itself unitary. If,

given $A \in \mathbb{C}^{m \times n}$ (with $m \geq n$), one could find a sequence of unitary matrices, $\{H_0, \dots, H_{n-1}\}$, such that

$$H_{n-1} \cdots H_0 A = \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where $R \in \mathbb{C}^{n \times n}$ is upper triangular, then

$$A = \underbrace{H_0^H \cdots H_{n-1}^H}_{Q} \begin{pmatrix} R \\ 0 \end{pmatrix}$$

which is closely related to the QR factorization of A .

Homework 3.3.1.1 Show that if $A \in \mathbb{C}^{m \times n}$ and $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$, where $Q \in \mathbb{C}^{m \times m}$ is unitary and R is upper triangular, then there exists $Q_L \in \mathbb{C}^{m \times n}$ such that $A = Q_L R$, is the QR factorization of A .

Solution.

$$Q \begin{pmatrix} R \\ 0 \end{pmatrix} = \begin{pmatrix} Q_L & Q_R \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_L R,$$

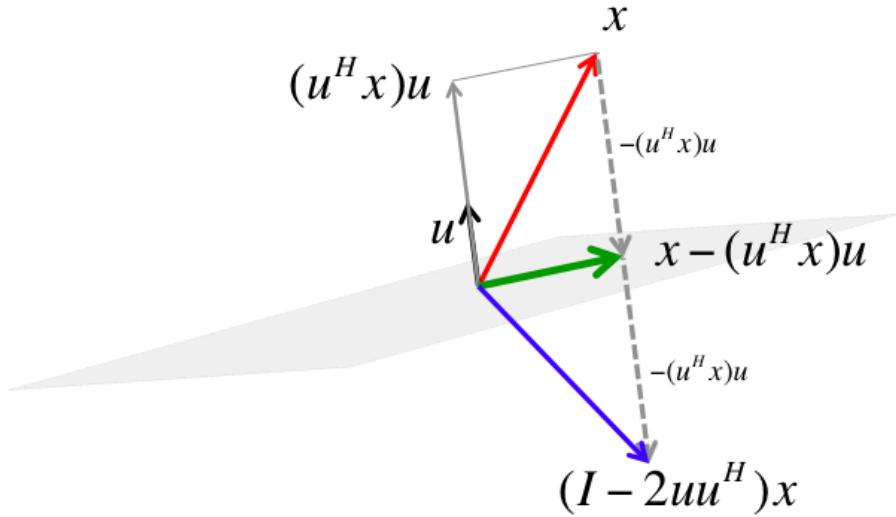
The second fundamental insight will be that the desired unitary transformations $\{H_0, \dots, H_{n-1}\}$ can be computed and applied cheaply, as we will discover in the remainder of this section.

3.3.2 Householder transformation



YouTube: <https://www.youtube.com/watch?v=6TIVIw4B5VA>

What we have discovered in this first video is how to construct a Householder transformation, also referred to as a reflector, since it acts like a mirroring with respect to the subspace orthogonal to the vector u , as illustrated in [Figure 3.3.2.1](#).



[PowerPoint](#)

[source](#)
HouseholderTransformation.pptx).

(Resources/Week03/

Figure 3.3.2.1 Given vector x and unit length vector u , the subspace orthogonal to u becomes a mirror for reflecting x represented by the transformation $(I - 2uu^H)$.

Definition 3.3.2.2 Let $u \in \mathbb{C}^n$ be a vector of unit length ($\|u\|_2 = 1$). Then $H = I - 2uu^H$ is said to be a Householder transformation or (Householder) reflector. \diamond

We observe:

- Any vector z that is perpendicular to u is left unchanged:

$$(I - 2uu^H)z = z - 2u(u^H z) = z.$$

- Any vector x can be written as $x = z + u^H xu$ where z is perpendicular to u and $u^H xu$ is the component of x in the direction of u . Then

$$\begin{aligned} (I - 2uu^H)x &= (I - 2uu^H)(z + u^H xu) = z + u^H xu - 2u \underbrace{u^H z}_0 - 2uu^H u^H xu \\ &= z + u^H xu - 2u^H x \underbrace{u^H u}_1 u = z - u^H xu. \end{aligned}$$

These observations can be interpreted as follows: The space perpendicular to u acts as a "mirror": a vector that is an element in that space (along the mirror) is not reflected. However, if a vector has a component that is orthogonal to the mirror, that component is reversed in direction, as illustrated in [Figure 3.3.2.1](#). Notice that a reflection preserves the length of a vector.

Homework 3.3.2.1 Show that if H is a reflector, then

- $HH = I$ (reflecting the reflection of a vector results in the original vector).
- $H = H^H$.
- $H^H H = HH^H = I$ (a reflector is unitary).

Solution. Show that if H is a reflector, then

- $HH = I$ (reflecting the reflection of a vector results in the original vector).

Solution:

$$\begin{aligned} & (I - 2uu^H)(I - 2uu^H) \\ &= \\ & I - 2uu^H - 2uu^H + 4u \underbrace{u^H u}_1 u^H \\ &= \\ & I - 4uu^H + 4uu^H = I \end{aligned}$$

- $H = H^H$.

Solution:

$$\begin{aligned} & (I - 2uu^H)^H \\ &= \\ & I - 2(u^H)^H u^H \\ &= \\ & I - 2uu^H \end{aligned}$$

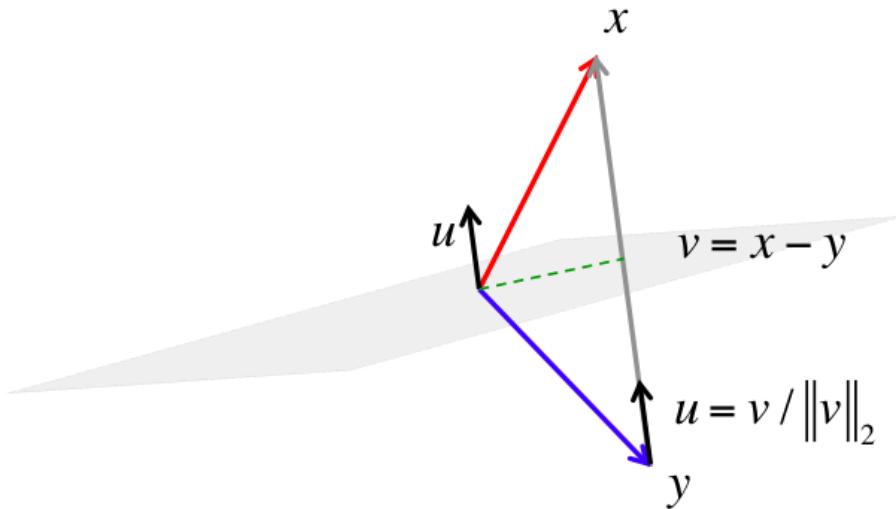
- $H^H H = I$ (a reflector is unitary).

Solution:

$$\begin{aligned} & H^H H \\ &= \\ & HH \\ &= \\ & I \end{aligned}$$



YouTube: <https://www.youtube.com/watch?v=wmjUHak9yHU>



[PowerPoint](#)

[source](#)

HouseholderTransformationAsUsed.pptx)

(Resources/Week03/

Figure 3.3.2.3 How to compute u given vectors x and y with $\|x\|_2 = \|y\|_2$.

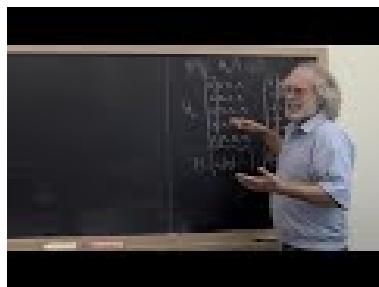
Next, let us ask the question of how to reflect a given $x \in \mathbb{C}^n$ into another vector $y \in \mathbb{C}^n$ with $\|x\|_2 = \|y\|_2$. In other words, how do we compute vector u so that

$$(I - 2uu^H)x = y.$$

From our discussion above, we need to find a vector u that is perpendicular to the space with respect to which we will reflect. From Figure 3.3.2.3 we notice that the vector from y to x , $v = x - y$, is perpendicular to the desired space. Thus, u must equal a unit vector in the direction v : $u = v / \|v\|_2$.

Remark 3.3.2.4 In subsequent discussion we will prefer to give Householder transformations as $I - uu^H/\tau$, where $\tau = u^H u / 2$ so that u needs no longer be a unit vector, just a direction. The reason for this will become obvious later.

When employing Householder transformations as part of a QR factorization algorithm, we need to introduce zeroes below the diagonal of our matrix. This requires a very special case of Householder transformation.



YouTube: https://www.youtube.com/watch?v=iMrgPGCWZ_o

As we compute the QR factorization via Householder transformations, we will need to find a Householder transformation H that maps a vector x to a multiple of the first unit basis vector (e_0). We discuss first how to find H in the case where $x \in \mathbb{R}^n$. We seek v so that $(I - \frac{2}{v^T v} v v^T)x = \pm \|x\|_2 e_0$. Since the resulting vector that we want is $y = \pm \|x\|_2 e_0$, we must choose $v = x - y = x \mp \|x\|_2 e_0$.

Example 3.3.2.5 Show that if $x \in \mathbb{R}^n$, $v = x \mp \|x\|_2 e_0$, and $\tau = v^T v / 2$ then $(I - \frac{1}{\tau} v v^T)x = \pm \|x\|_2 e_0$.

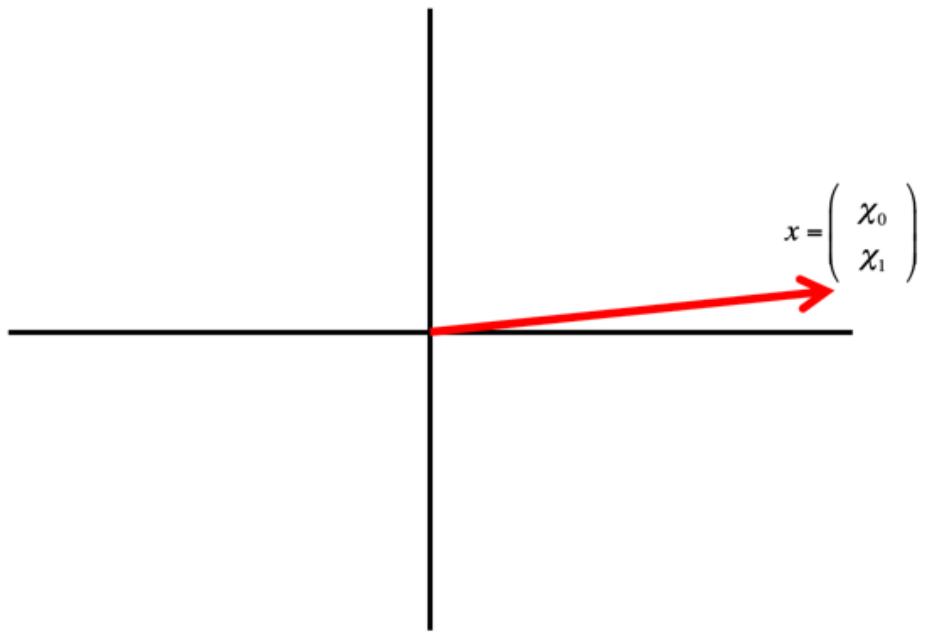
Solution. This is surprisingly messy... It is easier to derive the formula than it is to check it. So, we won't check it! \square

In practice, we choose $v = x + \text{sign}(\chi_1)\|x\|_2 e_0$ where χ_1 denotes the first element of x . The reason is as follows: the first element of v , v_1 , will be $v_1 = \chi_1 \mp \|x\|_2$. If χ_1 is positive and $\|x\|_2$ is almost equal to χ_1 , then $\chi_1 - \|x\|_2$ is a small number and if there is error in χ_1 and/or $\|x\|_2$, this error becomes large *relative* to the result $\chi_1 - \|x\|_2$, due to catastrophic cancellation. Regardless of whether χ_1 is positive or negative, we can avoid this by choosing $v = x + \text{sign}(\chi_1)\|x\|_2 e_0$:

$$v := x + \text{sign}(\chi_1)\|x\|_2 e_0 = \left(\begin{array}{c} \chi_1 \\ x_2 \end{array} \right) + \left(\begin{array}{c} \text{sign}(\chi_1)\|x\|_2 \\ 0 \end{array} \right) = \left(\begin{array}{c} \chi_1 + \text{sign}(\chi_1)\|x\|_2 \\ x_2 \end{array} \right).$$

Remark 3.3.2.6 This is a good place to clarify how we index in this course. Here we label the first element of the vector x as χ_1 , despite the fact that we have advocated in favor of indexing starting with zero. In our algorithms that leverage the FLAME notation (partitioning/repartitioning), you may have noticed that a vector or scalar indexed with 1 refers to the "current column/row" or "current element". In preparation of using the computation of the vectors v and u in the setting of such an algorithm, we use χ_1 here for the first element from which these vectors will be computed, which tends to be an element that is indexed with 1. So, there is reasoning behind the apparent insanity.

Ponder This 3.3.2.2 Consider $x \in \mathbb{R}^2$ as drawn below:

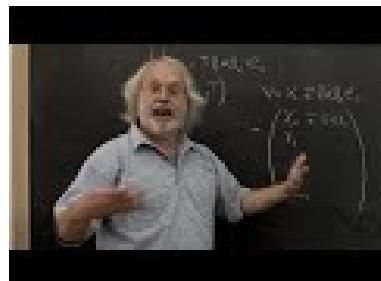


and let u be the vector such that $(I - uu^H/\tau)$ is a Householder transformation that maps x to a vector $\rho e_0 = \rho \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

- Draw a vector ρe_0 to which x is "mirrored."
- Draw the line that "mirrors."
- Draw the vector v from which u is computed.
- Repeat for the "other" vector ρe_0 .

Computationally, which choice of mirror is better than the other? Why?

3.3.3 Practical computation of the Householder vector



YouTube: https://www.youtube.com/watch?v=UX_QBt90jf8

3.3.3.1 The real case

Next, we discuss a slight variant on the above discussion that is used in practice. To do so, we view x as a vector that consists of its first element, χ_1 , and the rest of the vector, x_2 : More precisely, partition

$$x = \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix},$$

where χ_1 equals the first element of x and x_2 is the rest of x . Then we will wish to find a Householder vector $u = \begin{pmatrix} 1 \\ u_2 \end{pmatrix}$ so that

$$\left(I - \frac{1}{\tau} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^T \right) \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \pm \|x\|_2 \\ 0 \end{pmatrix}.$$

Notice that y in the previous discussion equals the vector $\begin{pmatrix} \pm \|x\|_2 \\ 0 \end{pmatrix}$, so the direction of u is given by

$$v = \begin{pmatrix} \chi_1 \mp \|x\|_2 \\ x_2 \end{pmatrix}.$$

We now wish to normalize this vector so its first entry equals "1":

$$u = \frac{v}{\nu_1} = \frac{1}{\chi_1 \mp \|x\|_2} \begin{pmatrix} \chi_1 \mp \|x\|_2 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ x_2/\nu_1 \end{pmatrix},$$

where $\nu_1 = \chi_1 \mp \|x\|_2$ equals the first element of v . (Note that if $\nu_1 = 0$ then u_2 can be set to 0.)

3.3.3.2 The complex case (optional)

Let us work out the complex case, dealing explicitly with x as a vector that consists of its first element, χ_1 , and the rest of the vector, x_2 : More precisely, partition

$$x = \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix},$$

where χ_1 equals the first element of x and x_2 is the rest of x . Then we will wish to find a Householder vector $u = \begin{pmatrix} 1 \\ u_2 \end{pmatrix}$ so that

$$\left(I - \frac{1}{\tau} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \right) \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \boxed{\pm} \|x\|_2 \\ 0 \end{pmatrix}.$$

Here $\boxed{\pm}$ denotes a complex scalar on the complex unit circle. By the same argument as before

$$v = \begin{pmatrix} \chi_1 - \boxed{\pm} \|x\|_2 \\ x_2 \end{pmatrix}.$$

We now wish to normalize this vector so its first entry equals "1":

$$u = \frac{v}{\nu_1} = \frac{1}{\chi_1 - \boxed{\pm} \|x\|_2} \begin{pmatrix} \chi_1 - \boxed{\pm} \|x\|_2 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ x_2/\nu_1 \end{pmatrix}.$$

where $\nu_1 = \chi_1 - \boxed{\pm} \|x\|_2$. (If $\nu_1 = 0$ then we set u_2 to 0.)

As was the case for the real-valued case, the choice $\boxed{\pm}$ is important. We choose $\boxed{\pm} = -\text{sign}(\chi_1) = -\frac{\chi_1}{|\chi_1|}$.

3.3.3.3 A routine for computing the Householder vector

The vector

$$\begin{pmatrix} 1 \\ u_2 \end{pmatrix}$$

is the Householder vector that reflects x into $\boxed{\pm} \|x\|_2 e_0$. The notation

$$\left[\begin{pmatrix} \rho \\ u_2 \end{pmatrix}, \tau \right] := \text{Housev} \left(\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \right)$$

represents the computation of the above mentioned vector u_2 , and scalars ρ and τ , from vector x . We will use the notation $H(x)$ for the transformation $I - \frac{1}{\tau} uu^H$ where u and τ are computed by $\text{Housev}(x)$.

Algorithm :	$\left[\begin{pmatrix} \rho \\ u_2 \end{pmatrix}, \tau \right] = \text{Housev} \left(\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \right)$
$\rho = -\text{sign}(\chi_1) \ x\ _2$ $\nu_1 = \chi_1 + \text{sign}(\chi_1) \ x\ _2$ $u_2 = x_2/\nu_1$ $\tau = (1 + u_2^H u_2)/2$	$\chi_2 := \ x_2\ _2$ $\alpha := \left\ \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} \right\ _2 (= \ x\ _2)$ $\rho := -\text{sign}(\chi_1) \alpha$ $\nu_1 := \chi_1 - \rho$ $u_2 := x_2/\nu_1$ $\chi_2 = \chi_2/ \nu_1 (= \ u_2\ _2)$ $\tau = (1 + \chi_2^2)/2$

Figure 3.3.3.1 Computing the Householder transformation. Left: simple formulation. Right: efficient computation. Note: I have not completely double-checked these formulas for the complex case. They work for the real case.

Remark 3.3.3.2 The function

```
function [ rho, ...
    u2, tau ] = Housev( chi1, ...
    x2 )
```

implements the function Housev. It can be found in [Assignments/Week03/matlab/Housev.m](#)

Homework 3.3.3.1 Function [Assignments/Week03/matlab/Housev.m](#) implements the steps in [Figure 3.3.3.1](#) (left). Update this implementation with the equivalent steps in [Figure 3.3.3.1](#) (right), which is $Sv^T v$ closer to how it is implemented in practice.

Solution. [Assignments/Week03/answers/Housev-alt.m](#)

3.3.4 Householder QR factorization algorithm



YouTube: <https://www.youtube.com/watch?v=5MeeuSoFBdY>

Let A be an $m \times n$ with $m \geq n$. We will now show how to compute $A \rightarrow QR$, the QR factorization, as a sequence of Householder transformations applied to A , which eventually zeroes out all elements of that matrix below the diagonal. The process is illustrated in [Figure 3.3.4.1](#).

Original matrix	$\left[\begin{pmatrix} \rho_{11} \\ u_{21} \end{pmatrix}, \tau_1 \right] =$ Housev $\left(\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix} \right)$	$\left(\begin{array}{cc} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{array} \right) :=$ $\left(\begin{array}{cc} \rho_{11} & a_{12}^T - w_{12}^T \\ 0 & A_{22} - u_{21}w_{12}^T \end{array} \right)$	“Move forward”
$\begin{matrix} \times & \times & \times & \times \\ \times & \times & \times & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ \times & \times & \times & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \end{matrix}$
$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & 0 & \times & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{matrix}$
$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & 0 & 0 & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 \end{matrix}$
$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 \end{matrix}$

Figure 3.3.4.1 Illustration of Householder QR factorization.

In the first iteration, we partition

$$A \rightarrow \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}.$$

Let

$$\left[\begin{pmatrix} \rho_{11} \\ u_{21} \end{pmatrix}, \tau_1 \right] = \text{Housev} \left(\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix} \right)$$

be the Householder transform computed from the first column of A . Then applying this Householder transform to A yields

$$\begin{aligned} \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} &:= \left(I - \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix}^H \right) \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} \\ &= \begin{pmatrix} \rho_{11} & a_{12}^T - w_{12}^T \\ 0 & A_{22} - u_{21}w_{12}^T \end{pmatrix}, \end{aligned}$$

where $w_{12}^T = (a_{12}^T + u_{21}^H A_{22})/\tau_1$. Computation of a full QR factorization of A will now proceed with the updated matrix A_{22} .



YouTube: <https://www.youtube.com/watch?v=WWe8yVccZy0>

Homework 3.3.4.1 Show that

$$\left(\begin{array}{c|c} I & 0 \\ \hline 0 & I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right) \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right)^H \end{array} \right) = \left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 0 \\ 1 \\ u_{21} \end{array} \right) \left(\begin{array}{c} 0 \\ 1 \\ u_{21} \end{array} \right)^H \right).$$

Solution.

$$\begin{aligned} \left(\begin{array}{c|c} I & 0 \\ \hline 0 & I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right) \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right)^H \end{array} \right) &= I - \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right) \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right)^H \end{array} \right) \\ &= I - \frac{1}{\tau_1} \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right) \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right)^H \end{array} \right) \\ &= I - \frac{1}{\tau_1} \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & 1 & u_2^H \\ 0 & u_2 & u_2 u_2^H \end{array} \right) \\ &= \left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 0 \\ 1 \\ u_{21} \end{array} \right) \left(\begin{array}{c} 0 \\ 1 \\ u_{21} \end{array} \right)^H \right). \end{aligned}$$

More generally, let us assume that after k iterations of the algorithm matrix A contains

$$A \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & A_{BR} \end{array} \right) = \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right),$$

where R_{TL} and R_{00} are $k \times k$ upper triangular matrices. Let

$$\left[\left(\begin{array}{c} \rho_{11} \\ u_{21} \end{array} \right), \tau_1 \right] = \text{Housev} \left(\left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right) \right).$$

and update

$$\begin{aligned}
 A &:= \left(\begin{array}{c|cc} I & 0 \\ 0 & \left(I - \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix}^H \right) \end{array} \right) \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) \\
 &= \left(I - \frac{1}{\tau_1} \begin{pmatrix} 0 \\ 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ u_{21} \end{pmatrix}^H \right) \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) \\
 &= \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & a_{12}^T - w_{12}^T \\ 0 & 0 & A_{22} - u_{21}w_{12}^T \end{array} \right),
 \end{aligned}$$

where, again, $w_{12}^T = (a_{12}^T + u_{21}^H A_{22})/\tau_1$.

Let

$$H_k = \left(I - \frac{1}{\tau_1} \begin{pmatrix} 0_k \\ 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 0_k \\ 1 \\ u_{21} \end{pmatrix}^H \right)$$

be the Householder transform so computed during the $(k+1)$ st iteration. Then upon completion matrix A contains

$$R = \begin{pmatrix} R_{TL} \\ 0 \end{pmatrix} = H_{n-1} \cdots H_1 H_0 \hat{A}$$

where \hat{A} denotes the original contents of A and R_{TL} is an upper triangular matrix. Rearranging this we find that

$$\hat{A} = H_0 H_1 \cdots H_{n-1} R$$

which shows that if $Q = H_0 H_1 \cdots H_{n-1}$ then $\hat{A} = QR$.

Typically, the algorithm overwrites the original matrix A with the upper triangular matrix, and at each step u_{21} is stored over the elements that become zero, thus overwriting a_{21} . (It is for this reason that the first element of u was normalized to equal "1".) In this case Q is usually not explicitly formed as it can be stored as the separate Householder vectors below the diagonal of the overwritten matrix. The algorithm that overwrites A in this manner is given in [Figure 3.3.4.2](#).

$[A, t] = \text{HQR_unb_var1}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \text{ and } t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right)$
A_{TL} is 0×0 and t_T has 0 elements
while $n(A_{BR}) > 0$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) \text{ and } \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right)$
$\left[\left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right), \tau_1 \right] := \left[\left(\begin{array}{c} \rho_{11} \\ u_{21} \end{array} \right), \tau_1 \right] = \text{Housev} \left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right)$
Update $\left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right) := \left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21}^H \end{array} \right) \left(\begin{array}{cc} 1 & u_{21}^H \end{array} \right) \right) \left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right)$
via the steps
$w_{12}^T := (a_{12}^T + u_{21}^H A_{22}) / \tau_1$
$\left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right) := \left(\begin{array}{c} a_{12}^T - w_{12}^T \\ A_{22} - a_{21} w_{12}^T \end{array} \right)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) \text{ and } \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right)$
endwhile

Figure 3.3.4.2 Unblocked Householder transformation based QR factorization.

In that figure,

$$[A, t] = \text{HQR_unb_var1}(A)$$

denotes the operation that computes the QR factorization of $m \times n$ matrix A , with $m \geq n$, via Householder transformations. It returns the Householder vectors and matrix R in the first argument and the vector of scalars " τ_i " that are computed as part of the Householder transformations in t .

Homework 3.3.4.2 Given $A \in \mathbb{R}^{m \times n}$ show that the cost of the algorithm in [Figure 3.3.4.2](#) is given by

$$C_{\text{HQR}}(m, n) \approx 2mn^2 - \frac{2}{3}n^3 \text{ flops.}$$

Solution. The bulk of the computation is in

$$w_{12}^T = (a_{12}^T + u_{21}^H A_{22}) / \tau_1$$

and

$$A_{22} - u_{21} w_{12}^T.$$

During the k th iteration (when R_{TL} is $k \times k$), this means a matrix-vector multiplication ($u_{21}^H A_{22}$) and rank-1 update with matrix A_{22} which is of size approximately $(m-k) \times (n-k)$

for a cost of $4(m - k)(n - k)$ flops. Thus the total cost is approximately

$$\begin{aligned}
 & \sum_{k=0}^{n-1} 4(m - k)(n - k) \\
 &= \\
 & 4 \sum_{j=0}^{n-1} (m - n + j)j \\
 &= \\
 & 4(m - n) \sum_{j=0}^{n-1} j + 4 \sum_{j=0}^{n-1} j^2 \\
 &= \\
 & 2(m - n)n(n - 1) + 4 \sum_{j=0}^{n-1} j^2 \\
 &\approx \\
 & 2(m - n)n^2 + 4 \int_0^n x^2 dx \\
 &= \\
 & 2mn^2 - 2n^3 + \frac{4}{3}n^3 \\
 &= \\
 & 2mn^2 - \frac{2}{3}n^3.
 \end{aligned}$$

Homework 3.3.4.3 Implement the algorithm given in [Figure 3.3.4.2](#) as
function [A_out, t] = HQR(A)

by completing the code in [Assignments/Week03/matlab/HQR.m](#). Input is an $m \times n$ matrix A . Output is the matrix A_{out} with the Householder vectors below its diagonal and R in its upper triangular part. You may want to use [Assignments/Week03/matlab/test_HQR.m](#) to check your implementation.

Solution. See [Assignments/Week03/answers/HQR.m](#). Warning: it only checks if R is computed correctly.

3.3.5 Forming Q



YouTube: <https://www.youtube.com/watch?v=cFWMsVNBzDY>

Given $A \in \mathbb{C}^{m \times n}$, let $[A, t] = \text{HQR_unb_var1}(A)$ yield the matrix A with the Householder vectors stored below the diagonal, R stored on and above the diagonal, and the scalars τ_i , $0 \leq i < n$, stored in vector t . We now discuss how to form the first n columns of $Q = H_0 H_1 \cdots H_{n-1}$. The computation is illustrated in [Figure 3.3.5.1](#).

Original matrix	$\left(\begin{array}{c c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \\ \hline \frac{1 - 1/\tau_1}{-\tau_1} & -(u_{21}^H A_{22})/\tau_1 \\ \hline -u_{21}/\tau_1 & A_{22} + u_{21} a_{12}^T \end{array} \right) :=$				“Move forward”
$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \end{array}$			$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \end{array}$	
	$\begin{array}{cc cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{array}$			$\begin{array}{cc cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{array}$	
	$\begin{array}{cc cc} 1 & 0 & 0 & 0 \\ 0 & \times & \times & \times \end{array}$			$\begin{array}{cc cc} 1 & 0 & 0 & 0 \\ 0 & \times & \times & \times \end{array}$	
	$\begin{array}{c ccc} \times & \times & \times & \times \\ \times & \times & \times & \times \end{array}$			$\begin{array}{c ccc} \times & \times & \times & \times \\ \times & \times & \times & \times \end{array}$	

Figure 3.3.5.1 Illustration of the computation of Q .

Notice that to pick out the first n columns we must form

$$Q \left(\begin{array}{c|c} I_{n \times n} & \\ \hline 0 & \end{array} \right) = H_0 \cdots H_{n-1} \left(\begin{array}{c|c} I_{n \times n} & \\ \hline 0 & \end{array} \right) = H_0 \cdots H_{k-1} \underbrace{H_k \cdots H_{n-1}}_{B_k} \left(\begin{array}{c|c} I_{n \times n} & \\ \hline 0 & \end{array} \right)$$

so that $Q = B_0$, where $B_k = H_k \cdots H_{n-1} \left(\begin{array}{c|c} I_{n \times n} & \\ \hline 0 & \end{array} \right)$.

Homework 3.3.5.1 ALWAYS/SOMETIMES/NEVER:

$$B_k = H_k \cdots H_{n-1} \left(\begin{array}{c|c} I_{n \times n} & \\ \hline 0 & \end{array} \right) = \left(\begin{array}{c|c} I_{k \times k} & 0 \\ \hline 0 & \tilde{B}_k \end{array} \right).$$

for some $(m-k) \times (n-k)$ matrix \tilde{B}_k .

Answer. ALWAYS

Solution. The proof of this is by induction on k :

- Base case: $k = n$. Then $B_n = \left(\begin{array}{c|c} I_{n \times n} & \\ \hline 0 & \end{array} \right)$, which has the desired form.

- Inductive step: Assume the result is true for B_k . We show it is true for B_{k-1} :

$$\begin{aligned}
 B_{k-1} &= \\
 &= H_{k-1} H_k \cdots H_{n-1} \left(\frac{I_{n \times n}}{0} \right) \\
 &= \\
 &= H_{k-1} B_k \\
 &= \\
 &= H_{k-1} \left(\frac{I_{k \times k}}{0} \mid \tilde{B}_k \right) \\
 &= \\
 &= \left(\begin{array}{cc|c} I_{(k-1) \times (k-1)} & 0 & 0 \\ 0 & I - \frac{1}{\tau_k} \left(\frac{1}{u_k} \right) (1 \mid u_k^H) & 0 \\ \hline 0 & 1 & \tilde{B}_k \end{array} \right) \\
 &= \\
 &= \left(\begin{array}{cc|c} I_{(k-1) \times (k-1)} & 0 & 0 \\ 0 & \left(I - \frac{1}{\tau_k} \left(\frac{1}{u_k} \right) (1 \mid u_k^H) \right) \left(\frac{1}{0} \mid \tilde{B}_k \right) & 0 \\ \hline 0 & 1 & \tilde{B}_k \end{array} \right) \\
 &= \text{choose } y_k^T = u_k^H \tilde{B}_k / \tau_k \\
 &= \left(\begin{array}{cc|c} I_{(k-1) \times (k-1)} & 0 & 0 \\ 0 & \left(\frac{1}{0} \mid \tilde{B}_k \right) - \left(\frac{1}{u_k} \right) (1/\tau_k \mid y_k^T) & 0 \\ \hline 0 & 1 & \tilde{B}_k \end{array} \right) \\
 &= \\
 &= \left(\begin{array}{cc|c} I_{(k-1) \times (k-1)} & 0 & 0 \\ 0 & \left(\frac{1 - 1/\tau_k}{-u_k/\tau_k} \mid \tilde{B}_k - u_k y_k^T \right) & 0 \\ \hline 0 & 1 & \tilde{B}_k - u_k y_k^T \end{array} \right) \\
 &= \\
 &= \left(\begin{array}{cc|c} I_{(k-1) \times (k-1)} & 0 & 0 \\ 0 & 1 - 1/\tau_k & -y_k^T \\ \hline 0 & -u_k/\tau_k & \tilde{B}_k - u_k y_k^T \end{array} \right) \\
 &= \\
 &= \left(\begin{array}{cc} I_{(k-1) \times (k-1)} & 0 \\ 0 & \tilde{B}_{k-1} \end{array} \right).
 \end{aligned}$$

- By the Principle of Mathematical Induction the result holds for B_0, \dots, B_n .



YouTube: <https://www.youtube.com/watch?v=pNEp5XlsZ4k>

The last exercise justifies the algorithm in [Figure 3.3.5.2](#),

$[A] = \text{FormQ}(A, t)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right)$
A_{TL} is $n(A) \times n(A)$ and t_T has $n(A)$ elements
while $n(A_{TL}) > 0$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{cc c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array} \right)$
Update $\left(\begin{array}{c c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) :=$ $\left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21}^H \end{array} \right) \left(\begin{array}{c c} 1 & u_{21}^H \end{array} \right) \right) \left(\begin{array}{c c} 1 & 0 \\ \hline 0 & A_{22} \end{array} \right)$
via the steps $\alpha_{11} := 1 - 1/\tau_1$ $a_{12}^T := -(a_{21}^H A_{22})/\tau_1$ $A_{22} := A_{22} + a_{21} a_{12}^T$ $a_{21} := -a_{21}/\tau_1$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array} \right)$
endwhile

Figure 3.3.5.2 Algorithm for overwriting A with Q from the Householder transformations stored as Householder vectors below the diagonal of A (as produced by $[A, t] = \text{HQR_unb_var1}(A, t)$).

which, given $[A, t] = \text{HQR_unb_var1}(A)$ from [Figure 3.3.4.2](#), overwrites A with the first $n = n(A)$ columns of Q .

Homework 3.3.5.2 Implement the algorithm in [Figure 3.3.5.2](#) as
function [A_out] = FormQ(A, t)

by completing the code in [Assignments/Week03/matlab/FormQ.m](#). You will want to use [Assignments/Week03/matlab/test_FormQ.m](#) to check your implementation. Input is the $m \times n$ matrix A and vector t that resulted from [A, t] = HQR(A). Output is the matrix Q for the QR factorization. You may want to use [Assignments/Week03/matlab/test_FormQ.m](#) to check your implementation.

Solution. See [Assignments/Week03/answers/FormQ.m](#)

Homework 3.3.5.3 Given $A \in \mathbb{C}^{m \times n}$, show that the cost of the algorithm in [Figure 3.3.5.2](#)

is given by

$$C_{\text{FormQ}}(m, n) \approx 2mn^2 - \frac{2}{3}n^3 \text{ flops.}$$

Hint. Modify the answer for [Homework 3.3.4.2](#).

Solution. When computing the Householder QR factorization, the bulk of the cost is in the computations

$$w_{12}^T := (a_{12}^T + u_{21}^H A_{22})/\tau_1$$

and

$$A_{22} - u_{21} w_{12}^T.$$

When forming Q , the cost is in computing

$$a_{12}^T := -(a_{21}^H A_{22}/\tau_1)$$

and

$$A_{22} := A_{22} + u_{21} w_{12}^T.$$

During the when A_{TL} is $k \times k$), these represent, essentially, identical costs: p the matrix-vector multiplication ($u_{21}^H A_{22}$) and rank-1 update with matrix A_{22} which is of size approximately $(m - k) \times (n - k)$ for a cost of $4(m - k)(n - k)$ flops. Thus the total cost is approximately

$$\begin{aligned} & \sum_{k=n-1}^0 4(m - k)(n - k) \\ &= \quad < \text{reverse the order of the summation} > \\ & \sum_{k=0}^{n-1} 4(m - k)(n - k) \\ &= \\ & 4 \sum_{j=1}^n (m - n + j)j \\ &= \\ & 4(m - n) \sum_{j=1}^n j + 4 \sum_{j=1}^n j^2 \\ &= \\ & 2(m - n)n(n + 1) + 4 \sum_{j=1}^n j^2 \\ &\approx \\ & 2(m - n)n^2 + 4 \int_0^n x^2 dx \\ &= \\ & 2mn^2 - 2n^3 + \frac{4}{3}n^3 \\ &= \\ & 2mn^2 - \frac{2}{3}n^3. \end{aligned}$$

Ponder This 3.3.5.4 If $m = n$ then Q could be accumulated by the sequence

$$Q = (\cdots ((IH_0)H_1)\cdots H_{n-1}).$$

Give a high-level reason why this would be (much) more expensive than the algorithm in [Figure 3.3.5.2](#)

3.3.6 Applying Q^H



YouTube: <https://www.youtube.com/watch?v=BfK3DVgfxIM>

In a future chapter, we will see that the QR factorization is used to solve the linear least-squares problem. To do so, we need to be able to compute $\hat{y} = Q^H y$ where $Q^H = H_{n-1} \cdots H_0$.

Let us start by computing $H_0 y$:

$$\begin{aligned} & \left(I - \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \right) \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} \\ &= \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} - \underbrace{\begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix}}_{\omega_1} / \tau_1 \\ &= \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} - \omega_1 \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \\ &= \begin{pmatrix} \psi_1 - \omega_1 \\ y_2 - \omega_1 u_2 \end{pmatrix}. \end{aligned}$$

More generally, let us compute $H_k y$:

$$\left(I - \frac{1}{\tau_1} \begin{pmatrix} 0 \\ 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ u_2 \end{pmatrix}^H \right) \begin{pmatrix} y_0 \\ \psi_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ \psi_1 - \omega_1 \\ y_2 - \omega_1 u_2 \end{pmatrix},$$

where $\omega_1 = (\psi_1 + u_2^H y_2) / \tau_1$. This motivates the algorithm in [Figure 3.3.6.1](#) for computing $y := H_{n-1} \cdots H_0 y$ given the output matrix A and vector t from routine `HQR_unb_var1`.

$[y] = \text{Apply_QH}(A, t, y)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ t_B \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$
A_{TL} is 0×0 and t_T, y_T have 0 elements
while $n(A_{BR}) < 0$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right),$
$\left(\begin{array}{c} t_T \\ t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$
Update $\left(\begin{array}{c} \psi_1 \\ y_2 \end{array} \right) := \left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right) \left(\begin{array}{cc} 1 & u_{21}^H \end{array} \right) \right) \left(\begin{array}{c} \psi_1 \\ y_2 \end{array} \right)$
via the steps
$\omega_1 := (\psi_1 + a_{21}^H y_2) / \tau_1$
$\left(\begin{array}{c} \psi_1 \\ y_2 \end{array} \right) := \left(\begin{array}{c} \psi_1 - \omega_1 \\ y_2 - \omega_1 u_{21} \end{array} \right)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right),$
$\left(\begin{array}{c} t_T \\ t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$
endwhile

Figure 3.3.6.1 Algorithm for computing $y := Q^H y (= H_{n-1} \cdots H_0 y)$ given the output from the algorithm HQR_unb_var1.

Homework 3.3.6.1 What is the approximate cost of algorithm in Figure 3.3.6.1 if Q (stored as Householder vectors in A) is $m \times n$.

Solution. The cost of this algorithm can be analyzed as follows: When y_T is of length k , the bulk of the computation is in a dot product with vectors of length $m - k - 1$ (to compute ω_1) and an axpy operation with vectors of length $m - k - 1$ to subsequently update ψ_1 and y_2 . Thus, the cost is approximately given by

$$\sum_{k=0}^{n-1} 4(m - k - 1) = 4 \sum_{k=0}^{n-1} m - 4 \sum_{k=0}^{n-1} (k - 1) \approx 4mn - 2n^2.$$

Notice that this is much cheaper than forming Q and then multiplying $Q^H y$.

3.3.7 Orthogonality of resulting Q

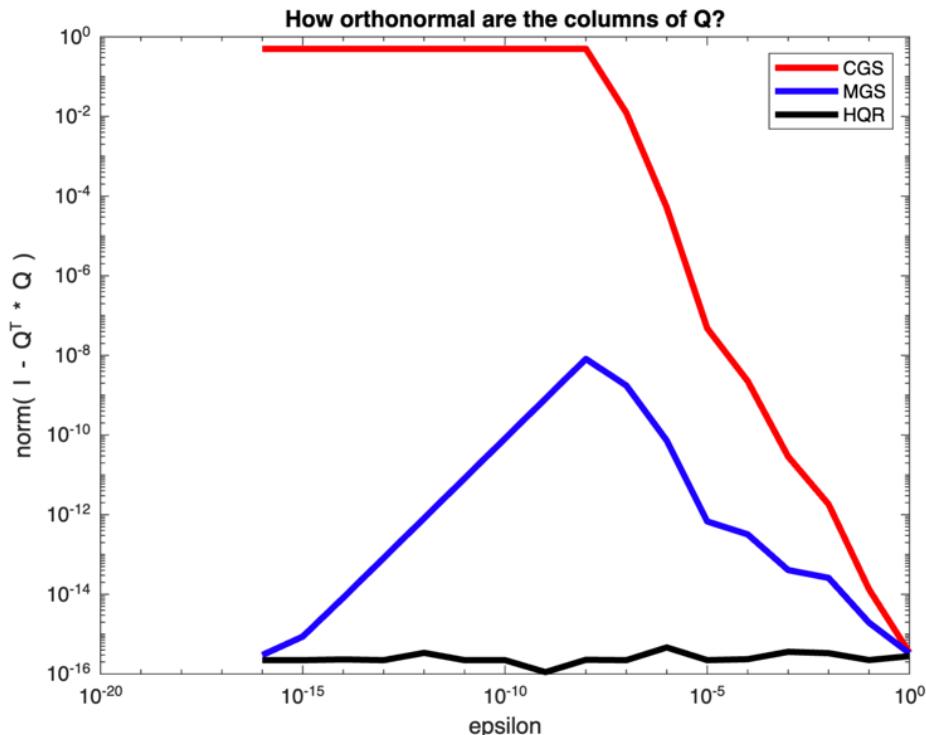
Homework 3.3.7.1 Previous programming assignments have the following routines for computing the QR factorization of a given matrix A :

- Classical Gram-Schmidt (CGS) [Homework 3.2.3.1](#):
 $[A_{\text{out}}, R_{\text{out}}] = \text{CGS_QR}(A).$
- Modified Gram-Schmidt (MGS) [Homework 3.2.4.3](#):
 $[A_{\text{out}}, R_{\text{out}}] = \text{MGS_QR}(A).$
- Householder QR factorization (HQR) [Homework 3.3.4.3](#):
 $[A_{\text{out}}, t_{\text{out}}] = \text{HQR}(A).$
- Form Q from Householder QR factorization [Homework 3.3.5.2](#):
 $Q = \text{FormQ}(A, t).$

Use these to examine the orthogonality of the computed Q by writing the Matlab script `Assignments/Week03/matlab/test_orthogonality.m` for the matrix

$$\left(\begin{array}{c|cc} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{array} \right).$$

Solution. Try [Assignments/Week03/answers/test_orthogonality.m](#).



Ponder This 3.3.7.2 In the last homework, we examined the orthogonality of the computed matrix Q for a very specific kind of matrix. The problem with that matrix is that the columns are nearly linearly dependent (the smaller ϵ is).

How can you quantify how close to being linearly dependent the columns of a matrix are?

How could you create a matrix of arbitrary size in such a way that you can control how close to being linearly dependent the columns are?

Homework 3.3.7.3 (Optional). Program up your solution to [Ponder This 3.3.7.2](#) and use it to compare how mutually orthonormal the columns of the computed matrices Q are.

3.4 Enrichments

3.4.1 Blocked Householder QR factorization

3.4.1.1 Casting computation in terms of matrix-matrix multiplication

Modern processors have very fast processors with very fast floating point units (which perform the multiply/adds that are the bread and butter of our computations), but very slow memory. Without getting into details, the reason is that modern memories are large and hence are physically far from the processor, with limited bandwidth between the two. To overcome this, smaller "cache" memories are closer to the CPU of the processor. In order to achieve high performance (efficient use of the fast processor), the strategy is to bring data into such a cache and perform a lot of computations with this data before writing a result out to memory.

Operations like a dot product of vectors or an "axpy" ($y := \alpha x + y$) perform $O(m)$ computation with $O(m)$ data and hence don't present much opportunity for reuse of data. Similarly, matrix-vector multiplication and rank-1 update operations perform $O(m^2)$ computation with $O(m^2)$ data, again limiting the opportunity for reuse. In contrast, matrix-matrix multiplication performs $O(m^3)$ computation with $O(m^2)$ data, and hence there is an opportunity to reuse data.

The goal becomes to rearrange computation so that most computation is cast in terms of matrix-matrix multiplication-like operations. Algorithms that achieve this are called *blocked algorithms*.

It is probably best to return to this enrichment after you have encountered simpler algorithms and their blocked variants later in the course, since Householder QR factorization is one of the more difficult operations to cast in terms of matrix-matrix multiplication.

3.4.1.2 Accumulating Householder transformations

Given a sequence of Householder transformations, computed as part of Householder QR factorization, these Householder transformations can be accumulated into a new transformation: If H_0, \dots, H_{k-1} are Householder transformations, then

$$H_0 H_1 \cdots H_{k-1} = I - UT^{-1}U^H,$$

where T is an upper triangular matrix. If U stores the Householder vectors that define H_0, \dots, H_{k-1} (with "1"s explicitly on its diagonal) and t holds the scalars $\tau_0, \dots, \tau_{k-1}$, then

```
T := FormT( U, t )
```

computes the desired matrix T . Now, applying this UT transformation to a matrix B yields

$$(I - UT^{-1}U^H)B = B - U(T^{-1}(U^H B)),$$

which demonstrates that this operations requires the matrix-matrix multiplication $W := U^H B$, the triangular matrix-matrix multiplication $W := T^{-1}W$ and the matrix-matrix multiplication $B - UW$, each of which can attain high performance.

In [23] we call the transformation $I - UT^{-1}U^H$ that equals the accumulated Householder transformations the **UT transform** and prove that T can instead be computed as

$$T = \text{triu}(U^H U)$$

(the upper triangular part of $U^H U$) followed by either dividing the diagonal elements by two or setting them to $\tau_0, \dots, \tau_{k-1}$ (in order). In that paper, we point out similar published results [8] [35] [45] [32].

3.4.1.3 A blocked algorithm

A QR factorization that exploits the insights that yielded the UT transform can now be described:

- Partition

$$A \rightarrow \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

where A_{11} is $b \times b$.

- We can use the unblocked algorithm in Subsection 3.3.4 to factor the panel $\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$

$$[\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, t_1] := \text{HouseQR_unb_var1}(\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}),$$

overwriting the entries below the diagonal with the Householder vectors $\begin{pmatrix} U_{11} \\ U_{21} \end{pmatrix}$ (with the ones on the diagonal implicitly stored) and the upper triangular part with R_{11} .

- Form T_{11} from the Householder vectors using the procedure described earlier in this unit:

$$T_{11} := \text{FormT}(\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix})$$

- Now we need to also apply the Householder transformations to the rest of the columns:

$$\begin{aligned}
 & \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} \\
 &= \\
 & \left(I - \begin{pmatrix} U_{11} \\ U_{21} \end{pmatrix} T_{11}^{-1} \begin{pmatrix} U_{11} \\ U_{21} \end{pmatrix}^H \right)^H \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} \\
 &= \\
 & \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} - \begin{pmatrix} U_{11} \\ U_{21} \end{pmatrix} W_{12} \\
 &= \\
 & \begin{pmatrix} A_{12} - U_{11}W_{12} \\ A_{22} - U_{21}W_{12} \end{pmatrix},
 \end{aligned}$$

where

$$W_{12} = T_{11}^{-H} (U_{11}^H A_{12} + U_{21}^H A_{22}).$$

This motivates the blocked algorithm in [Figure 3.4.1.1](#).

$[A, t] := \text{HouseQR_blk_var1}(A, t)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right)$
A_{TL} is 0×0 , t_T has 0 rows
while $m(A_{TL}) < m(A)$
choose block size b
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline t_1 \\ t_2 \end{array} \right)$
A_{11} is $b \times b$, t_1 has b rows
$[\left(\begin{array}{c} A_{11} \\ A_{21} \end{array} \right), t_1] := \text{HQR_unb_var1}(\left(\begin{array}{c} A_{11} \\ A_{21} \end{array} \right))$
$T_{11} := \text{FormT}(\left(\begin{array}{c} A_{11} \\ A_{21} \end{array} \right), t_1)$
$W_{12} := T_{11}^{-H} (U_{11}^H A_{12} + U_{21}^H A_{22})$
$\left(\begin{array}{c} A_{12} \\ A_{22} \end{array} \right) := \left(\begin{array}{c} A_{12} - U_{11}W_{12} \\ A_{22} - U_{21}W_{12} \end{array} \right)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & A_{01} & A_{02} & \\ \hline A_{10} & A_{11} & A_{12} & \\ A_{20} & A_{21} & A_{22} & \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline t_1 \\ t_2 \end{array} \right)$
endwhile

Figure 3.4.1.1 Blocked Householder transformation based QR factorization.

Details can be found in [\[23\]](#).

3.4.1.4 The WY transform

An alternative (and more usual) way of expressing a Householder transform is

$$I - \beta vv^H,$$

where $\beta = 2/v^H v$ ($= 1/\tau$, where τ is as discussed before). This leads to an alternative accumulation of Householder transforms known as the compact WY transform [35]:

$$I - USU^H$$

where upper triangular matrix S relates to the matrix T in the UT transform via $S = T^{-1}$. Obviously, T can be computed first and then inverted via the insights in the next exercise. Alternatively, inversion of matrix T can be incorporated into the algorithm that computes T (which is what is done in the implementation in LAPACK [1]).

3.4.2 Systematic derivation of algorithms

We have described two algorithms for Gram-Schmidt orthogonalization: the Classical Gram-Schmidt (CGS) and the Modified Gram-Schmidt (MGS) algorithms. In this section we use this operation to introduce our FLAME methodology for systematically deriving algorithms hand-in-hand with their proof of correctness. Those who want to see the finer points of this methodologies may want to consider taking our Massive Open Online Course titled "LAFF-On: Programming for Correctness," offered on edX.

The idea is as follows: We first specify the input (the **precondition**) and output (the **postcondition**) for the algorithm. factorization

- The precondition for the QR factorization is

$$A = \hat{A}.$$

A contains the original matrix, which we specify by \hat{A} since A will be overwritten as the algorithm proceeds.

- The postcondition for the QR factorization is

$$A = Q \wedge \hat{A} = QR \wedge Q^H Q = I. \quad (3.4.1)$$

This specifies that A is to be overwritten by an orthonormal matrix Q and that QR equals the original matrix \hat{A} . We will not explicitly specify that R is upper triangular, but keep that in mind as well.

Now, we know that we march through the matrices in a consistent way. At some point in the algorithm we will have divided them as follows:

$$A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right), Q \rightarrow \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right), R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right),$$

where these partitionings are "conformal" (they have to fit in context). To come up with algorithms, we now ask the question "What are the contents of A and R at a typical stage of the loop?" To answer this, we instead first ask the question "In terms of the parts of the matrices are that naturally exposed by the loop, what is the final goal?" To answer that question, we take the partitioned matrices, and enter them in the postcondition (3.4.1):

$$\begin{aligned} \underbrace{\left(\begin{array}{c|c} A_L & A_R \end{array} \right)}_A &= \underbrace{\left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)}_Q \\ \wedge \underbrace{\left(\begin{array}{c|c} \hat{A}_L & \hat{A}_R \end{array} \right)}_{\hat{A}} &= \underbrace{\left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)}_Q \underbrace{\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ 0 & R_{BR} \end{array} \right)}_R \\ \wedge \underbrace{\left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)^H}_{Q^H} \underbrace{\left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)}_Q &= \underbrace{\left(\begin{array}{c|c} I & 0 \\ 0 & I \end{array} \right)}_I. \end{aligned}$$

(Notice that R_{BL} becomes a zero matrix since R is upper triangular.) Applying the rules of linear algebra (multiplying out the various expressions) yields

$$\begin{aligned} \left(\begin{array}{c|c} A_L & A_R \end{array} \right) &= \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right) \\ \wedge \left(\begin{array}{c|c} \hat{A}_L & \hat{A}_R \end{array} \right) &= \left(\begin{array}{c|c} Q_L R_{TL} & Q_L R_{TR} + Q_R R_{BR} \end{array} \right) \\ \wedge \left(\begin{array}{c|c} Q_L^H Q_L & Q_L^T Q_R \\ Q_R^H Q_L & Q_R^H Q_R \end{array} \right) . &= \left(\begin{array}{c|c} I & 0 \\ 0 & I \end{array} \right). \end{aligned} \tag{3.4.2}$$

We call this the **Partitioned Matrix Expression** (PME). It is a recursive definition of the operation to be performed.

The different algorithms differ in what is in the matrices A and R as the loop iterates. Can we systematically come up with an expression for their contents at a typical point in the iteration? The observation is that when the loop has not finished, only part of the final result has been computed. So, we should be able to take the PME in (3.4.2) and remove terms to come up with partial results towards the final result. There are some dependencies (some parts of matrices must be computed before others). Taking this into account gives us two **loop invariants**:

- Loop invariant 1:

$$\begin{aligned} \left(\begin{array}{c|c} A_L & A_R \end{array} \right) &= \left(\begin{array}{c|c} Q_L & \hat{A}_R \end{array} \right) \\ \wedge \hat{A}_L &= Q_L R_{TL} \\ \wedge Q_L^H Q_L &= I \end{aligned} \tag{3.4.3}$$

- Loop invariant 2:

$$\begin{aligned} \left(\begin{array}{c|c} A_L & A_R \end{array} \right) &= \left(\begin{array}{c|c} Q_L & \hat{A}_R - Q_L R_{TR} \end{array} \right) \\ \wedge \left(\begin{array}{c|c} \hat{A}_L & \hat{A}_R \end{array} \right) &= \left(\begin{array}{c|c} Q_L R_{TL} & Q_L R_{TR} + Q_R R_{BR} \end{array} \right) \\ \wedge Q_L^H Q_L &= I \end{aligned}$$

We note that our knowledge of linear algebra allows us to manipulate this into

$$\begin{aligned} \left(\begin{array}{c|c} A_L & A_R \end{array} \right) &= \left(\begin{array}{c|c} Q_L & \hat{A}_R - Q_L R_{TR} \end{array} \right) \\ \wedge \hat{A}_L &= Q_L R_{TL} \wedge Q_L^H \hat{A}_L = R_{TR} \wedge Q_L^H Q_L = I. \end{aligned} \quad (3.4.4)$$

The idea now is that we *derive* the loop that computes the QR factorization by systematically *deriving* the algorithm that maintains the state of the variables described by a chosen loop invariant. If you use (3.4.3), then you end up with CGS. If you use (3.4.4), then you end up with MGS.

Interested in details? We have a MOOC for that: [LAFF-On Programming for Correctness](#).

3.5 Wrap Up

3.5.1 Additional homework

Homework 3.5.1.1 Consider the matrix $\begin{pmatrix} A \\ B \end{pmatrix}$ where A has linearly independent columns.

Let

- $A = Q_A R_A$ be the QR factorization of A .
- $\begin{pmatrix} R_A \\ B \end{pmatrix} = Q_B R_B$ be the QR factorization of $\begin{pmatrix} R_A \\ B \end{pmatrix}$.
- $\begin{pmatrix} A \\ B \end{pmatrix} = QR$ be the QR factorization of $\begin{pmatrix} A \\ B \end{pmatrix}$.

Assume that the diagonal entries of R_A , R_B , and R are all positive. Show that $R = R_B$.

Solution.

$$\begin{pmatrix} A \\ B \end{pmatrix} = \left(\begin{array}{c|c} Q_A & 0 \\ \hline 0 & I \end{array} \right) \begin{pmatrix} R_A \\ B \end{pmatrix} = \left(\begin{array}{c|c} Q_A & 0 \\ \hline 0 & I \end{array} \right) Q_B R_B$$

Also, $\begin{pmatrix} A \\ B \end{pmatrix} = QR$. By the uniqueness of the QR factorization (when the diagonal elements of the triangular matrix are restricted to be positive), $Q = \left(\begin{array}{c|c} Q_A & 0 \\ \hline 0 & I \end{array} \right) Q_B$ and $R = R_B$.

Remark 3.5.1.1 This last exercise gives a key insight that is explored in the paper

- [20] Brian C. Gunter, Robert A. van de Geijn, Parallel out-of-core computation and updating of the QR factorization, ACM Transactions on Mathematical Software (TOMS), 2005.

3.5.2 Summary

Classical Gram-Schmidt orthogonalization: Given a set of linearly independent vectors $\{a_0, \dots, a_{n-1}\} \subset \mathbb{C}^m$, the Gram-Schmidt process computes an orthonormal basis $\{q_0, \dots, q_{n-1}\}$ that spans the same subspace as the original vectors, i.e.

$$\text{Span}(\{a_0, \dots, a_{n-1}\}) = \text{Span}(\{q_0, \dots, q_{n-1}\}).$$

The process proceeds as follows:

- Compute vector q_0 of unit length so that $\text{Span}(\{a_0\}) = \text{Span}(\{q_0\})$:
 - $\rho_{0,0} = \|a_0\|_2$
Computes the length of vector a_0 .
 - $q_0 = a_0 / \rho_{0,0}$
Sets q_0 to a unit vector in the direction of a_0 .

Notice that $a_0 = q_0 \rho_{0,0}$

- Compute vector q_1 of unit length so that $\text{Span}(\{a_0, a_1\}) = \text{Span}(\{q_0, q_1\})$:
 - $\rho_{0,1} = q_0^H a_1$
Computes $\rho_{0,1}$ so that $\rho_{0,1} q_0 = q_0^H a_1 q_0$ equals the component of a_1 in the direction of q_0 .
 - $a_1^\perp = a_1 - \rho_{0,1} q_0$
Computes the component of a_1 that is orthogonal to q_0 .
 - $\rho_{1,1} = \|a_1^\perp\|_2$
Computes the length of vector a_1^\perp .
 - $q_1 = a_1^\perp / \rho_{1,1}$
Sets q_1 to a unit vector in the direction of a_1^\perp .

Notice that

$$\left(\begin{array}{c|c} a_0 & a_1 \end{array} \right) = \left(\begin{array}{c|c} q_0 & q_1 \end{array} \right) \left(\begin{array}{c|c} \rho_{0,0} & \rho_{0,1} \\ \hline 0 & \rho_{1,1} \end{array} \right).$$

- Compute vector q_2 of unit length so that $\text{Span}(\{a_0, a_1, a_2\}) = \text{Span}(\{q_0, q_1, q_2\})$:
 - $\rho_{0,2} = q_0^H a_2$ or, equivalently, $\begin{pmatrix} \rho_{0,2} \\ \rho_{1,2} \end{pmatrix} = \begin{pmatrix} q_0 & q_1 \end{pmatrix}^H a_2$
Computes $\rho_{0,2}$ so that $\rho_{0,2} q_0 = q_0^H a_2 q_0$ and $\rho_{1,2} q_1 = q_1^H a_2 q_1$ equal the components of a_2 in the directions of q_0 and q_1 .
Or, equivalently, $\begin{pmatrix} q_0 & q_1 \end{pmatrix} \begin{pmatrix} \rho_{0,2} \\ \rho_{1,2} \end{pmatrix}$ is the component in $\text{Span}(\{q_0, q_1\})$.

- $a_2^\perp = a_2 - \rho_{0,2}q_0 - \rho_{1,2}q_1 = a_2 - \begin{pmatrix} q_0 & q_1 \end{pmatrix} \begin{pmatrix} \rho_{0,2} \\ \rho_{1,2} \end{pmatrix}$

Computes the component of a_2 that is orthogonal to q_0 and q_1 .

- $\rho_{2,2} = \|a_2^\perp\|_2$

Computes the length of vector a_2^\perp .

- $q_2 = a_2^\perp / \rho_{2,2}$

Sets q_2 to a unit vector in the direction of a_2^\perp .

Notice that

$$\left(\begin{array}{cc|c} a_0 & a_1 & a_2 \end{array} \right) = \left(\begin{array}{cc|c} q_0 & q_1 & q_2 \end{array} \right) \left(\begin{array}{cc|c} \rho_{0,0} & \rho_{0,1} & \rho_{0,2} \\ 0 & \rho_{1,1} & \rho_{1,2} \\ 0 & 0 & \rho_{2,2} \end{array} \right).$$

- And so forth.

Theorem 3.5.2.1 QR Decomposition Theorem. Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns. Then there exists an orthonormal matrix Q and upper triangular matrix R such that $A = QR$, its QR decomposition. If the diagonal elements of R are taken to be real and positive, then the decomposition is unique.

Projection a vector y onto the orthonormal columns of $Q \in \mathbb{C}^{m \times n}$:

$[y^\perp, r] = \text{Proj}_{\perp Q_{\text{CGS}}}(Q, y)$ (used by CGS)	$[y^\perp, r] = \text{Proj}_{\perp Q_{\text{MGS}}}(Q, y)$ (used by MGS)
$y^\perp = y$ for $i = 0, \dots, k - 1$ $\rho_i := q_i^H y$ $y^\perp := y^\perp - \rho_i q_i$ endfor	$y^\perp = y$ for $i = 0, \dots, k - 1$ $\rho_i := q_i^H y^\perp$ $y^\perp := y^\perp - \rho_i q_i$ endfor

Gram-Schmidt orthogonalization algorithms:

$[A, R] := \text{GS}(A)$ (overwrites A with Q) $A \rightarrow \left(\begin{array}{c c} A_L & A_R \end{array} \right), R \rightarrow \left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right)$ A_L has 0 columns and R_{TL} is 0×0 while $n(A_L) < n(A)$		
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array} \right)$		
CGS $r_{01} := A_0^H a_1$ $a_1 := a_1 - A_0 r_{01}$ $\rho_{11} := \ a_1\ _2$ $a_1 := a_1 / \rho_{11}$	MGS $[a_1, r_{01}] = \text{Proj}_{\perp} \text{toQ}_{\text{MGS}}(A_0, a_1)$ $\rho_{11} := \ a_1\ _2$ $q_1 := a_1 / \rho_{11}$	MGS (alternative) $\rho_{11} := \ a_1\ _2$ $a_1 := a_1 / \rho_{11}$ $r_{12}^T := a_1^H A_2$ $A_2 := A_2 - a_1 r_{12}^T$
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array} \right)$		
endwhile		

Classic example that shows that the columns of Q , computed by MGS, are "more orthogonal" than those computed by CGS:

$$A = \left(\begin{array}{c|c|c} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{array} \right) = \left(\begin{array}{c|c|c} a_0 & a_1 & a_2 \end{array} \right).$$

Cost of Gram-Schmidt algorithms: approximately $2mn^2$ flops.

Definition 3.5.2.2 Let $u \in \mathbb{C}^n$ be a vector of unit length ($\|u\|_2 = 1$). Then $H = I - 2uu^H$ is said to be a Householder transformation or (Householder) reflector. \diamond

If H is a Householder transformation (reflector), then

- $HH = I$.
- $H = H^H$.
- $H^H H = HH^H = I$.
- $H^{-1} = H^H = H$.

Computing a Householder transformation $I - 2uu^H$:

- Real case:

- $v = x \mp \|x\|_2 e_0$.
 $v = x + \text{sign}(\chi_1) \|x\|_2 e_0$ avoids catastrophic cancellation.
- $u = v/\|v\|_2$
- Complex case:
 - $v = x \mp \boxed{\pm} \|x\|_2 e_0$.
(Picking $\boxed{\pm}$ carefully avoids catastrophic cancellation.)
 - $u = v/\|v\|_2$

Practical computation of u and τ so that $I - uu^H/\tau$ is a Householder transformation (reflector):

Algorithm : $\left[\begin{pmatrix} \rho \\ u_2 \end{pmatrix}, \tau \right] = \text{Housev} \left(\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \right)$	
	$\chi_2 := \ x_2\ _2$
	$\alpha := \left\ \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} \right\ _2 (= \ x\ _2)$
$\rho = -\text{sign}(\chi_1) \ x\ _2$ $\nu_1 = \chi_1 + \text{sign}(\chi_1) \ x\ _2$ $u_2 = x_2/\nu_1$ $\tau = (1 + u_2^H u_2)/2$	$\rho := -\text{sign}(\chi_1) \alpha$ $\nu_1 := \chi_1 - \rho$ $u_2 := x_2/\nu_1$ $\chi_2 = \chi_2/ \nu_1 (= \ u_2\ _2)$ $\tau = (1 + \chi_2^2)/2$

Householder QR factorization algorithm:

$[A, t] = \text{HQR_unb_var1}(A)$	
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$ and $t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right)$ A_{TL} is 0×0 and t_T has 0 elements	
while $n(A_{BR}) > 0$	
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$ and $\left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right)$	
$\left[\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}, \tau_1 \right] := \left[\begin{pmatrix} \rho_{11} \\ u_{21} \end{pmatrix}, \tau_1 \right] = \text{Housev} \left(\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix} \right)$ Update $\left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right) := \left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21}^H \end{array} \right) \left(\begin{array}{c} 1 & u_{21}^H \end{array} \right) \right) \left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right)$ via the steps $w_{12}^T := (a_{12}^T + a_{21}^H A_{22})/\tau_1$ $\left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right) := \left(\begin{array}{c} a_{12}^T - w_{12}^T \\ A_{22} - a_{21} w_{12}^T \end{array} \right)$	
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$ and $\left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right)$	
endwhile	

Cost: approximately $2mn^2 - \frac{2}{3}n^3$ flops.

Algorithm for forming Q given output of Householder QR factorization algorithm:

$[A] = \text{FormQ}(A, t)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right)$
A_{TL} is $n(A) \times n(A)$ and t_T has $n(A)$ elements
while $n(A_{TL}) > 0$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{cc c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array} \right)$
Update $\left(\begin{array}{c c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) :=$
$\left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21}^H \end{array} \right) \left(\begin{array}{c c} 1 & u_{21}^H \end{array} \right) \right) \left(\begin{array}{c c} 1 & 0 \\ \hline 0 & A_{22} \end{array} \right)$
via the steps
$\alpha_{11} := 1 - 1/\tau_1$
$a_{12}^T := -(a_{21}^H A_{22})/\tau_1$
$A_{22} := A_{22} + a_{21} a_{12}^T$
$a_{21} := -a_{21}/\tau_1$
endwhile

Cost: approximately $2mn^2 - \frac{2}{3}n^3$ flops.

Algorithm for applying Q^H given output of Householder QR factorization algorithm:

$[y] = \text{Apply_QH}(A, t, y)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right)$
A_{TL} is 0×0 and t_T, y_T have 0 elements
while $n(A_{BR}) < 0$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right),$ $\left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array} \right)$
<hr/> Update $\left(\begin{array}{c} \psi_1 \\ \hline y_2 \end{array} \right) := \left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right) \left(\begin{array}{cc} 1 & u_{21}^H \end{array} \right) \right) \left(\begin{array}{c} \psi_1 \\ \hline y_2 \end{array} \right)$ via the steps $\omega_1 := (\psi_1 + a_{21}^H y_2) / \tau_1$ $\left(\begin{array}{c} \psi_1 \\ \hline y_2 \end{array} \right) := \left(\begin{array}{c} \psi_1 - \omega_1 \\ y_2 - \omega_1 u_2 \end{array} \right)$
<hr/> $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right),$ $\left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array} \right)$
endwhile

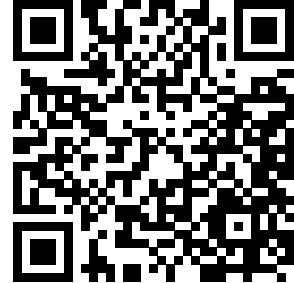
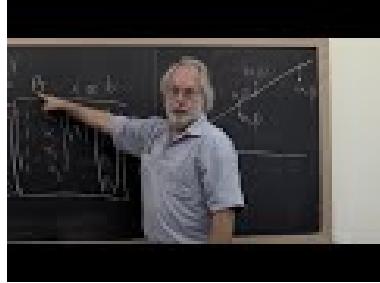
Cost: approximately $4mn - n^2$ flops.

Week 4

Linear Least Squares

4.1 Opening

4.1.1 Fitting the best line



YouTube: <https://www.youtube.com/watch?v=LPfd0YoQQU0>

A classic problem is to fit the "best" line through a given set of points: Given

$$\{(\chi_i, \psi_i)\}_{i=0}^{m-1},$$

we wish to fit the line $f(\chi) = \gamma_0 + \gamma_1\chi$ to these points, meaning that the coefficients γ_0 and γ_1 are to be determined. Now, in the end we want to formulate this as approximately solving $Ax = b$ and for that reason, we change the labels we use: Starting with points

$$\{(\alpha_i, \beta_i)\}_{i=0}^{m-1},$$

we wish to fit the line $f(\alpha) = \chi_0 + \chi_1\alpha$ through these points so that

$$\begin{aligned}\chi_0 + \chi_1\alpha_0 &\approx \beta_0 \\ \chi_0 + \chi_1\alpha_1 &\approx \beta_1 \\ \vdots &\quad \vdots \quad \vdots \\ \chi_0 + \chi_1\alpha_{m-1} &\approx \beta_{m-1},\end{aligned}$$

which we can instead write as

$$Ax \approx b,$$

where

$$A = \begin{pmatrix} 1 & \alpha_0 \\ 1 & \alpha_1 \\ \vdots & \vdots \\ 1 & \alpha_{m-1} \end{pmatrix}, x = \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}, \text{ and } b = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{m-1} \end{pmatrix}.$$

Homework 4.1.1.1 Use the script in [Assignments/Week04/matlab/LineFittingExercise.m](#) to fit a line to the given data by guessing the coefficients χ_0 and χ_1 .

Ponder This 4.1.1.2 Rewrite the script for [Homework 4.1.1.1](#) to be a bit more engaging...)

4.1.2 Overview

- 4.1 Opening
 - 4.1.1 Fitting the best line
 - 4.1.2 Overview
 - 4.1.3 What you will learn
- 4.2 Solution via the Method of Normal Equations
 - 4.2.1 The four fundamental spaces of a matrix
 - 4.2.2 The Method of Normal Equations
 - 4.2.3 Solving the normal equations
 - 4.2.4 Conditioning of the linear least squares problem
 - 4.2.5 Why using the Method of Normal Equations could be bad
- 4.3 Solution via the SVD
 - 4.3.1 The SVD and the four fundamental spaces
 - 4.3.2 Case 1: A has linearly independent columns
 - 4.3.3 Case 2: General case
- 4.4 Solution via the QR factorization
 - 4.4.1 A has linearly independent columns
 - 4.4.2 Via Gram-Schmidt QR factorization
 - 4.4.3 Via the Householder QR factorization
 - 4.4.4 A has linearly dependent columns
- 4.5 Enrichments
 - 4.5.1 Rank Revealing QR (RRQR) via MGS

- 4.5.2 Rank Revealing Householder QR factorization
- 4.6 Wrap Up
 - 4.6.1 Additional homework
 - 4.6.2 Summary

4.1.3 What you will learn

This week is all about solving linear least squares, a fundamental problem encountered when fitting data or approximating matrices.

Upon completion of this week, you should be able to

- Formulate a linear least squares problem.
- Transform the least squares problem into normal equations.
- Relate the solution of the linear least squares problem to the four fundamental spaces.
- Describe the four fundamental spaces of a matrix using its singular value decomposition.
- Solve the solution of the linear least squares problem via Normal Equations, the Singular Value Decomposition, and the QR decomposition.
- Compare and contrast the accuracy and cost of the different approaches for solving the linear least squares problem.

4.2 Solution via the Method of Normal Equations

4.2.1 The four fundamental spaces of a matrix



YouTube: <https://www.youtube.com/watch?v=9mdDqC1SChg>

We assume that the reader remembers theory related to (vector) subspaces. If a review is in order, we suggest Weeks 9 and 10 of Linear Algebra: Foundations to Frontiers (LAFF) [26].

At some point in your linear algebra education, you should also have learned about the four fundamental spaces of a matrix $A \in \mathbb{C}^{m \times n}$ (although perhaps only for the real-valued case):

- The column space, $\mathcal{C}(A)$, which is equal to the set of all vectors that are linear combinations of the columns of A

$$\{y \mid y = Ax\}.$$

- The null space, $\mathcal{N}(A)$, which is equal to the set of all vectors that are mapped to the zero vector by A

$$\{x \mid Ax = 0\}.$$

- The row space, $\mathcal{R}(A)$, which is equal to the set

$$\{y \mid y^H = x^H A\}.$$

Notice that $\mathcal{R}(A) = \mathcal{C}(A^H)$.

- The left null space, which is equal to the set of all vectors

$$\{x \mid x^H A = 0\}.$$

Notice that this set is equal to $\mathcal{N}(A^H)$.

Definition 4.2.1.1 Orthogonal subspaces. Two subspaces $S, T \subset \mathbb{C}^n$ are orthogonal if any two arbitrary vectors (and hence all vectors) $x \in S$ and $y \in T$ are orthogonal: $x^H y = 0$. \diamond

The following exercises help you refresh your skills regarding these subspaces.

Homework 4.2.1.1 Let $A \in \mathbb{C}^{m \times n}$. Show that its row space, $\mathcal{R}(A)$, and null space, $\mathcal{N}(A)$, are orthogonal.

Solution. Pick arbitrary $x \in \mathcal{R}(A)$ and $y \in \mathcal{N}(A)$. We need to show that these two vectors are orthogonal. Then

$$\begin{aligned} & x^H y \\ &= \langle x \in \mathcal{R}(A) \text{ iff there exists } z \text{ s.t. } x = A^H z \rangle \\ & (A^H z)^H y \\ &= \langle \text{transposition of product} \rangle \\ & z^H A y \\ &= \langle y \in \mathcal{N}(A) \rangle \\ & z^H 0 = 0. \end{aligned}$$

Homework 4.2.1.2 Let $A \in \mathbb{C}^{m \times n}$. Show that its column space, $\mathcal{C}(A)$, and left null space, $\mathcal{N}(A^H)$, are orthogonal.

Solution. Pick arbitrary $x \in \mathcal{C}(A)$ and $y \in \mathcal{N}(A^H)$. Then

$$\begin{aligned} x^H y &= \langle x \in \mathcal{C}(A) \text{ iff there exists } z \text{ s.t. } x = Az \rangle \\ (Az)^H y &= \langle \text{transposition of product} \rangle \\ z^H A^H y &= \langle y \in \mathcal{N}(A^H) \rangle \\ z^H 0 &= 0. \end{aligned}$$

Homework 4.2.1.3 Let $\{s_0, \dots, s_{r-1}\}$ be a basis for subspace $\mathcal{S} \subset \mathbb{C}^n$ and $\{t_0, \dots, t_{k-1}\}$ be a basis for subspace $\mathcal{T} \subset \mathbb{C}^n$. Show that the following are equivalent statements:

1. Subspaces \mathcal{S}, \mathcal{T} are orthogonal.
2. The vectors in $\{s_0, \dots, s_{r-1}\}$ are orthogonal to the vectors in $\{t_0, \dots, t_{k-1}\}$.
3. $s_i^H t_j = 0$ for all $0 \leq i < r$ and $0 \leq j < k$.
4. $(s_0 | \dots | s_{r-1})^H (t_0 | \dots | t_{k-1}) = 0$, the zero matrix of appropriate size.

Solution. We are going to prove the equivalence of all the statements by showing that 1. implies 2., 2. implies 3., 3. implies 4., and 4. implies 1.

- 1. implies 2.

Subspaces \mathcal{S} and \mathcal{T} are orthogonal if any vectors $x \in \mathcal{S}$ and $y \in \mathcal{T}$ are orthogonal. Obviously, this means that s_i is orthogonal to t_j for $0 \leq i < r$ and $0 \leq j < k$.

- 2. implies 3.

This is true by definition of what it means for two sets of vectors to be orthogonal.

- 3. implies 4.

$$(s_0 | \dots | s_{r-1})^H (t_0 | \dots | t_{k-1}) = \begin{pmatrix} s_0^H t_0 & s_0^H t_1 & \dots \\ s_1^H t_0 & s_1^H t_1 & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

- 4. implies 1.

We need to show that if $x \in \mathcal{S}$ and $y \in \mathcal{T}$ then $x^H y = 0$.

Notice that

$$x = (s_0 | \dots | s_{r-1}) \begin{pmatrix} \hat{\chi}_0 \\ \vdots \\ \hat{\chi}_{r-1} \end{pmatrix} \text{ and } y = (t_0 | \dots | t_{k-1}) \begin{pmatrix} \hat{\psi}_0 \\ \vdots \\ \hat{\psi}_{k-1} \end{pmatrix}$$

for appropriate choices of \hat{x} and \hat{y} . But then

$$\begin{aligned} x^H y &= \left(\begin{pmatrix} s_0 & \cdots & s_{r-1} \end{pmatrix} \begin{pmatrix} \hat{\chi}_0 \\ \vdots \\ \hat{\chi}_{r-1} \end{pmatrix} \right)^H \begin{pmatrix} t_0 & \cdots & t_{k-1} \end{pmatrix} \begin{pmatrix} \hat{\psi}_0 \\ \vdots \\ \hat{\psi}_{k-1} \end{pmatrix} \\ &= \begin{pmatrix} \hat{\chi}_0 \\ \vdots \\ \hat{\chi}_{r-1} \end{pmatrix}^H \underbrace{\begin{pmatrix} s_0 & \cdots & s_{r-1} \end{pmatrix}^H \begin{pmatrix} t_0 & \cdots & t_{k-1} \end{pmatrix}}_{0_{r \times k}} \begin{pmatrix} \hat{\psi}_0 \\ \vdots \\ \hat{\psi}_{k-1} \end{pmatrix} \\ &= 0 \end{aligned}$$

Homework 4.2.1.4 Let $A \in \mathbb{C}^{m \times n}$. Show that any vector $x \in \mathbb{C}^n$ can be written as $x = x_r + x_n$, where $x_r \in \mathcal{R}(A)$ and $x_n \in \mathcal{N}(A)$, and $x_r^H x_n = 0$.

Hint. Let r be the rank of matrix A . In a basic linear algebra course you learned that then the dimension of the row space, $\mathcal{R}(A)$, is r and the dimension of the null space, $\mathcal{N}(A)$, is $n - r$.

Let $\{w_0, \dots, w_{r-1}\}$ be a basis for $\mathcal{R}(A)$ and $\{w_r, \dots, w_{n-1}\}$ be a basis for $\mathcal{N}(A)$.

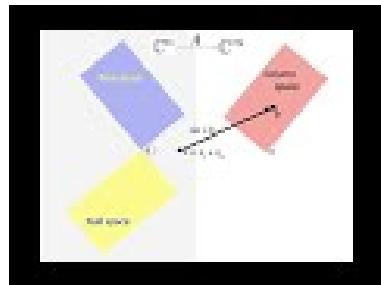
Answer. TRUE

Now prove it!

Solution. Let r be the rank of matrix A . In a basic linear algebra course you learned that then the dimension of the row space, $\mathcal{R}(A)$, is r and the dimension of the null space, $\mathcal{N}(A)$, is $n - r$.

Let $\{w_0, \dots, w_{r-1}\}$ be a basis for $\mathcal{R}(A)$ and $\{w_r, \dots, w_{n-1}\}$ be a basis for $\mathcal{N}(A)$. Since we know that these two spaces are orthogonal, we know that $\{w_0, \dots, w_{r-1}\}$ are orthogonal to $\{w_r, \dots, w_{n-1}\}$. Hence $\{w_0, \dots, w_{n-1}\}$ are linearly independent and form a basis for \mathbb{C}^n . Thus, there exist coefficients $\{\alpha_0, \dots, \alpha_{n-1}\}$ such that

$$\begin{aligned} x &= \alpha_0 w_0 + \cdots + \alpha_{n-1} w_{n-1} \\ &= \underbrace{\alpha_0 w_0 + \cdots + \alpha_{r-1} w_{r-1}}_{x_r} + \underbrace{\alpha_r w_r + \cdots + \alpha_{n-1} w_{n-1}}_{x_n}. \end{aligned}$$



YouTube: https://www.youtube.com/watch?v=ZdlraR_7cMA

Figure 4.2.1.2 captures the insights so far.

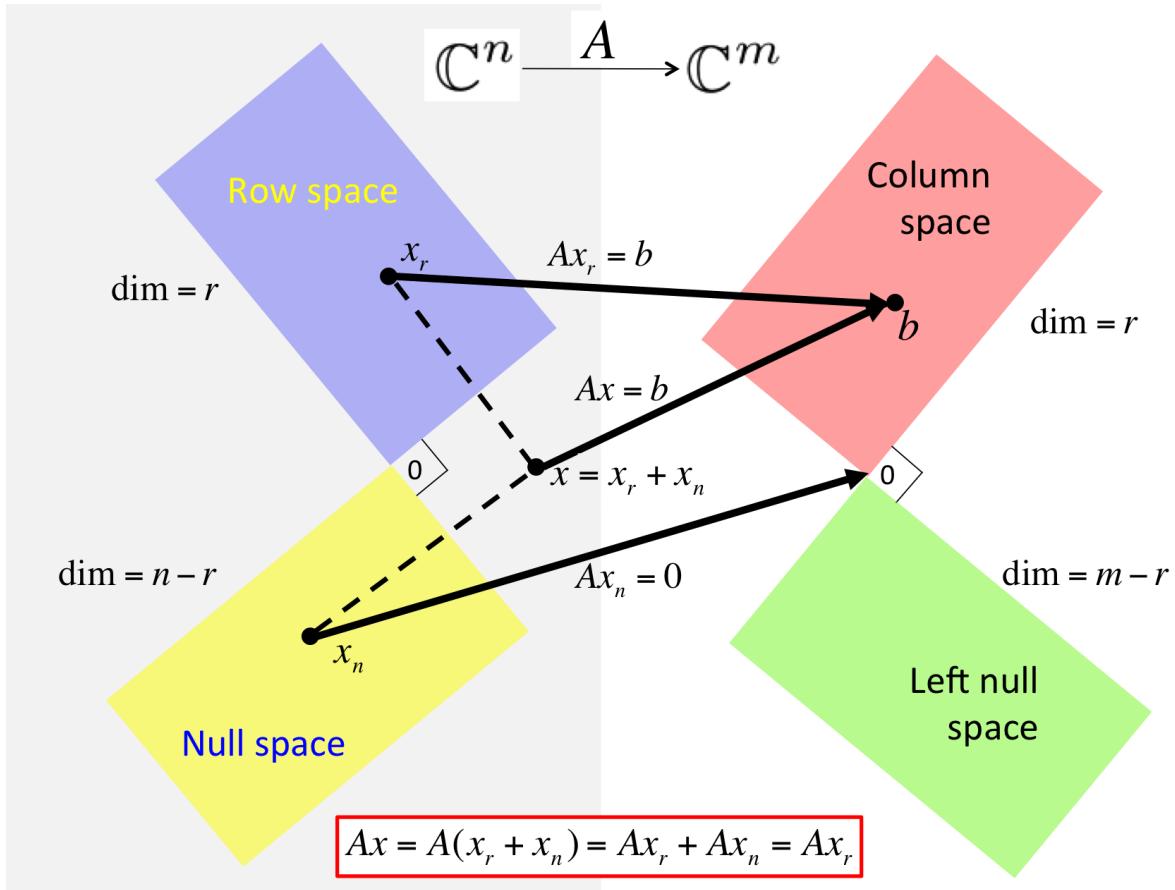


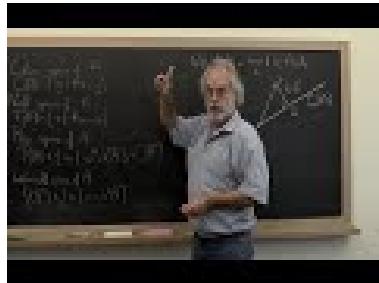
Figure 4.2.1.2 Illustration of the four fundamental spaces and the mapping of a vector $x \in \mathbb{C}^n$ by matrix $A \in \mathbb{C}^{m \times n}$.

That figure also captures that if r is the rank of matrix, then

- $\dim(\mathcal{R}(A)) = \dim(\mathcal{C}(A)) = r$;
- $\dim(\mathcal{N}(A)) = n - r$;
- $\dim(\mathcal{N}(A^H)) = m - r$.

Proving this is a bit cumbersome given the knowledge we have so far, but becomes very easy once we relate the various spaces to the SVD, in [Subsection 4.3.1](#). So, we just state it for now.

4.2.2 The Method of Normal Equations



YouTube: <https://www.youtube.com/watch?v=oT4KI0xx-f4>

Consider again the LLS problem: Given $A \in \mathbb{C}^{m \times n}$ and $b \in \mathbb{C}^m$ find $\hat{x} \in \mathbb{C}^n$ such that

$$\|b - A\hat{x}\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2.$$

We list a sequence of observations that you should have been exposed to in previous study of linear algebra:

- $\hat{b} = A\hat{x}$ is in the column space of A .
- \hat{b} equals the member of the column space of A that is closest to b , making it the orthogonal projection of b onto the column space of A .
- Hence the residual, $b - \hat{b}$, is orthogonal to the column space of A .
- From Figure 4.2.1.2 we deduce that $b - \hat{b} = b - A\hat{x}$ is in $\mathcal{N}(A^H)$, the left null space of A .
- Hence $A^H(b - A\hat{x}) = 0$ or, equivalently,

$$A^H A \hat{x} = A^H b.$$

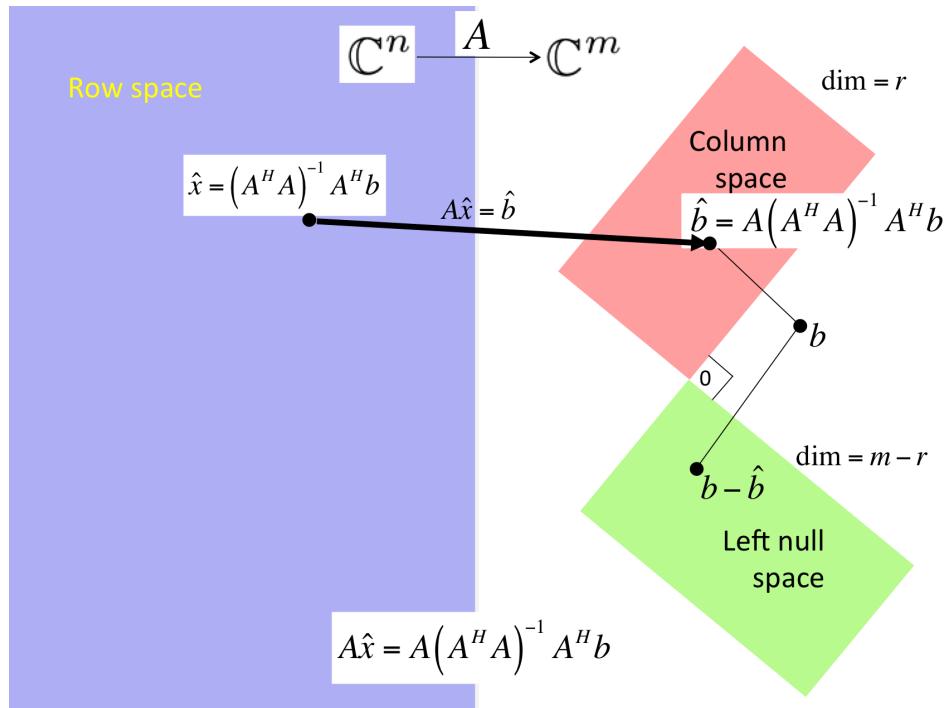
This linear system of equations is known as the normal equations.

- If A has linearly independent columns, then $\text{rank}(A) = n$, $\mathcal{N}(A) = \emptyset$, and $A^H A$ is nonsingular. In this case,

$$\hat{x} = (A^H A)^{-1} A^H b.$$

Obviously, this solution is in the row space, since $\mathcal{R}(A) = \mathbb{C}^n$.

With this, we have discovered what is known as the Method of Normal Equations. These steps are summarized in Figure 4.2.2.1



[PowerPoint Source](#)

Figure 4.2.2.1 Solving LLS via the Method of Normal Equations when A has linearly independent columns (and hence the row space of A equals \mathbb{C}^n).

Definition 4.2.2.2 (Left) pseudo inverse. Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns. Then

$$A^\dagger = (A^H A)^{-1} A^H$$

is its (left) pseudo inverse. \diamond

Homework 4.2.2.1 Let $A \in \mathbb{C}^{m \times m}$ be nonsingular. Then $A^{-1} = A^\dagger$.

Solution.

$$AA^\dagger = A(A^H A)^{-1} A^H = AA^{-1} A^{-H} A^H = II = I.$$

Homework 4.2.2.2 Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns. ALWAYS/SOMETIMES/NEVER: $AA^\dagger = I$.

Hint. Consider $A = (e_0)$.

Answer. SOMETIMES

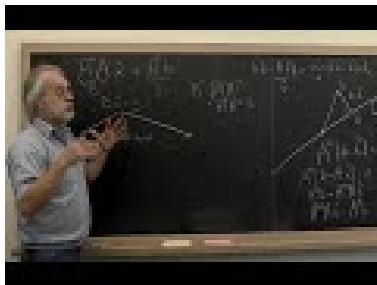
Solution. An example where $AA^\dagger = I$ is the case where $m = n$ and hence A is nonsingular.

An example where $AA^\dagger \neq I$ is $A = e_0$ for $m > 1$. Then

$$\begin{aligned}
 AA^\dagger &= <\text{ instantiate}> \\
 \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix} &\left(\underbrace{\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 1 \end{pmatrix}^H \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix}}_{1} \right)^{-1} \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix}^H \\
 &= <\text{simplify}> \\
 \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix} &\left(\begin{matrix} 1 & 0 & \cdots \end{matrix} \right) \\
 &= <\text{multiply out}> \\
 \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ \vdots & \vdots \end{pmatrix} & \\
 &= <\text{m}>1> \\
 &\neq I.
 \end{aligned}$$

Ponder This 4.2.2.3 The last exercise suggests there is also a right pseudo inverse. How would you define it?

4.2.3 Solving the normal equations



YouTube: <https://www.youtube.com/watch?v=ln4XogsWcOE>

Let us review a method you have likely seen before for solving the LLS problem when matrix A has linearly independent columns. We already used these results in [Subsection 2.1.1](#)

We wish to solve $A^H A \hat{x} = A^H b$, where A has linearly independent columns. If we form $B = A^H A$ and $y = A^H b$, we can instead solve $B \hat{x} = y$. Some observations:

- Since A has linearly independent columns, B is nonsingular. Hence, \hat{x} is unique.
- B is Hermitian since $B^H = (A^H A)^H = A^H (A^H)^H = A^H A = B$.

- B is Hermitian Positive Definite (HPD): $x \neq 0$ implies that $x^H B x > 0$. This follows from the fact that

$$x^H B x = x^H A^H A x = (Ax)^H (Ax) = \|Ax\|_2^2.$$

Since A has linearly independent columns, $x \neq 0$ implies that $Ax \neq 0$ and hence $\|Ax\|_2^2 > 0$.

In [Section 5.4](#), you will find out that since B is HPD, there exists a lower triangular matrix L such that $B = LL^H$. This is known as the Cholesky factorization of B . The steps for solving the normal equations then become

- Compute $B = A^H A$.

Notice that since B is Hermitian symmetric, only the lower or upper triangular part needs to be computed. This is known as a Hermitian rank-k update (where in this case $k = n$). The cost is, approximately, mn^2 flops. (See [Subsection C.0.1](#).)

- Compute $y = A^H b$.

The cost of this matrix-vector multiplication is, approximately, $2mn$ flops. (See [Subsection C.0.1](#).)

- Compute the Cholesky factorization $B \rightarrow LL^H$.

Later we will see that this costs, approximately, $\frac{1}{3}n^3$ flops. (See [Subsection 5.4.3](#).)

- Solve

$$Lz = y$$

(solve with a lower triangular matrix) followed by

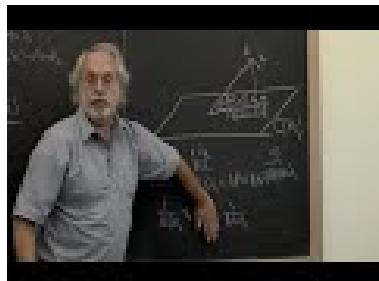
$$L^H \hat{x} = z$$

(solve with an upper triangular matrix).

Together, these triangular solves cost, approximately, $2n^2$ flops. (See [Subsection C.0.1](#).)

We will revisit this in [Section 5.4](#).

4.2.4 Conditioning of the linear least squares problem



YouTube: https://www.youtube.com/watch?v=etx_1VZ4VFk

Given $A \in \mathbb{C}^{m \times n}$ with linearly independent columns and $b \in \mathbb{C}^m$, consider the linear least squares (LLS) problem

$$\|b - A\hat{x}\|_2 = \min_x \|b - Ax\|_2 \quad (4.2.1)$$

and the perturbed problem

$$\|(b + \delta b) - A(\hat{x} + \delta \hat{x})\|_2 = \min_x \|(b + \delta b) - Ax\|_2 \quad (4.2.2)$$

The question we want to examine is by how much the relative error in b is amplified into a relative error in \hat{x} . We will restrict our discussion to the case where A has linearly independent columns.

Now, we discovered that \hat{b} , the projection of b onto the column space of A , satisfies

$$\hat{b} = A\hat{x} \quad (4.2.3)$$

and the projection of $b + \delta b$ satisfies

$$\hat{b} + \hat{\delta b} = A(\hat{x} + \delta \hat{x}) \quad (4.2.4)$$

where $\hat{\delta b}$ equals the projection of δb onto the column space of A .

Let θ equal the angle between vectors b and its projection \hat{b} (which equals the angle between b and the column space of A). Then

$$\cos(\theta) = \|\hat{b}\|_2 / \|b\|_2$$

and hence

$$\cos(\theta)\|b\|_2 = \|\hat{b}\|_2 = \|A\hat{x}\|_2 \leq \|A\|_2 \|\hat{x}\|_2 = \sigma_0 \|\hat{x}\|_2$$

which (as long as $\hat{x} \neq 0$) can be rewritten as

$$\frac{1}{\|\hat{x}\|_2} \leq \frac{\sigma_0}{\cos(\theta)} \frac{1}{\|b\|_2}. \quad (4.2.5)$$

Subtracting (4.2.3) from (4.2.4) yields

$$\hat{\delta b} = A\delta\hat{x}$$

or, equivalently,

$$A\delta\hat{x} = \hat{\delta b}$$

which is solved by

$$\begin{aligned} \delta\hat{x} &= A^\dagger \hat{\delta b} \\ &= A^\dagger A (A^H A)^{-1} A^H \delta b \\ &= (A^H A)^{-1} A^H A (A^H A)^{-1} A^H \delta b \\ &= A^\dagger \delta b, \end{aligned}$$

where $A^\dagger = (A^H A)^{-1} A^H$ is the pseudo inverse of A and we recall that $\hat{\delta b} = A(A^H A)^{-1} A^H \delta b$. Hence

$$\|\delta\hat{x}\|_2 \leq \|A^\dagger\|_2 \|\delta b\|_2. \quad (4.2.6)$$

Homework 4.2.4.1 Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns. Show that

$$\|(A^H A)^{-1} A^H\|_2 = 1/\sigma_{n-1},$$

where σ_{n-1} equals the smallest singular value of A .

Hint. Use the reduced SVD of A .

Solution. Let $A = U_L \Sigma_{TL} V^H$ be the reduced SVD of A , where V is square because A has linearly independent columns. Then

$$\begin{aligned} & \|(A^H A)^{-1} A^H\|_2 \\ &= \|((U_L \Sigma_{TL} V^H)^H U_L \Sigma_{TL} V^H)^{-1} (U_L \Sigma_{TL} V^H)^H\|_2 \\ &= \|(V \Sigma_{TL} U_L^H U_L \Sigma_{TL} V^H)^{-1} V \Sigma_{TL} U_L^H\|_2 \\ &= \|(V \Sigma_{TL}^{-1} \Sigma_{TL}^{-1} V^H) V \Sigma_{TL} U_L^H\|_2 \\ &= \|V \Sigma_{TL}^{-1} U_L^H\|_2 \\ &= \|\Sigma_{TL}^{-1} U_L^H\|_2 \\ &= 1/\sigma_{n-1}. \end{aligned}$$

This last step needs some more explanation: Clearly $\|\Sigma_{TL} U_L^H\|_2 \leq \|\Sigma_{TL}\|_2 \|U_L^H\|_2 = \sigma_0 \|U_L^H\|_2 \leq \sigma_0$. We need to show that there exists a vector x with $\|x\|_2 = 1$ such that $\|\Sigma_{TL} U_L^H x\|_2 = \|\Sigma_{TL} U_L^H\|_2$. If we pick $x = u_0$ (the first column of U_L), then $\|\Sigma_{TL} U_L^H x\|_2 = \|\Sigma_{TL} U_L^H u_0\|_2 = \|\Sigma_{TL} e_0\|_2 = \|\sigma_0 e_0\|_2 = \sigma_0$.

Combining (4.2.5), (4.2.6), and the result in this last homework yields

$$\frac{\|\hat{x}\|_2}{\|\hat{x}\|_2} \leq \frac{1}{\cos(\theta)} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta b\|_2}{\|b\|_2}. \quad (4.2.7)$$

Notice the effect of the $\cos(\theta)b$. If b is almost perpendicular to $\mathcal{C}(A)$, then its projection \hat{b} is small and $\cos \theta$ is small. Hence a small relative change in b can be greatly amplified. This makes sense: if b is almost perpendicular to $\mathcal{C}(A)$, then $\hat{x} \approx 0$, and any small $\delta b \in \mathcal{C}(A)$ can yield a relatively large change δx .

Definition 4.2.4.1 Condition number of matrix with linearly independent columns. Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns (and hence $n \leq m$). Then its condition number (with respect to the 2-norm) is defined by

$$\kappa_2(A) = \|A\|_2 \|A^\dagger\|_2 = \frac{\sigma_0}{\sigma_{n-1}}.$$

◊

It is informative to explicitly expose $\cos(\theta) = \|\hat{b}\|_2/\|b\|_2$ in (4.2.7):

$$\frac{\|\hat{x}\|_2}{\|\hat{b}\|_2} \leq \frac{\|b\|_2}{\|\hat{b}\|_2} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta b\|_2}{\|b\|_2}.$$

Notice that the ratio

$$\frac{\|\delta b\|_2}{\|b\|_2}$$

can be made smaller by adding a component, b_r , to b that is orthogonal to $\mathcal{C}(A)$ (and hence does not change the projection onto the column space, \hat{b}):

$$\frac{\|\delta b\|_2}{\|b + b_r\|_2}.$$

The factor $1/\cos(\theta)$ ensures that this does not magically reduce the relative error in \hat{x} :

$$\frac{\|\hat{x}\|_2}{\|\hat{b}\|_2} \leq \frac{\|b + b_r\|_2}{\|\hat{b}\|_2} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta b\|_2}{\|b + b_r\|_2}.$$

4.2.5 Why using the Method of Normal Equations could be bad



YouTube: <https://www.youtube.com/watch?v=W-HnQDsZs0w>

Homework 4.2.5.1 Show that $\kappa_2(A^H A) = (\kappa_2(A))^2$.

Hint. Use the SVD of A .

Solution. Let $A = U\Sigma V^H$ be the reduced SVD of A . Then

$$\begin{aligned} \kappa_2(A^H A) &= \|A^H A\|_2 \|(A^H A)^{-1}\|_2 \\ &= \|(U\Sigma V^H)^H U\Sigma V^H\|_2 \|((U\Sigma V^H)^H U\Sigma V^H)^{-1}\|_2 \\ &= \|V\Sigma^2 V^H\|_2 \|V(\Sigma^{-1})^2 V^H\|_2 \\ &= \|\Sigma^2\|_2 \|(\Sigma^{-1})^2\|_2 \\ &= \frac{\sigma_0^2}{\sigma_{n-1}^2} = \left(\frac{\sigma_0}{\sigma_{n-1}}\right)^2 = \kappa_2(A)^2. \end{aligned}$$

Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns. If one uses the Method of Normal Equations to solve the linear least squares problem $\min_x \|b - Ax\|_2$ via the steps

- Compute $B = A^H A$.

- Compute $y = A^H b$.
- Solve $B\hat{x} = y$.

the condition number of B equals the square of the condition number of A . So, while the sensitivity of the LLS problem is captured by

$$\frac{\|\delta\hat{x}\|_2}{\|\hat{x}\|_2} \leq \frac{1}{\cos(\theta)} \kappa_2(A) \frac{\|\delta b\|_2}{\|b\|_2}.$$

the sensitivity of computing \hat{x} from $B\hat{x} = y$ is captured by

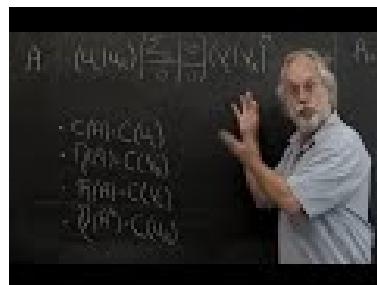
$$\frac{\|\delta\hat{x}\|_2}{\|\hat{x}\|_2} \leq \kappa_2(A)^2 \frac{\|\delta y\|_2}{\|y\|_2}.$$

If $\kappa_2(A)$ is relatively small (meaning that A is not close to a matrix with linearly dependent columns), then this may not be a problem. But if the columns of A are nearly linearly dependent, or high accuracy is desired, alternatives to the Method of Normal Equations should be employed.

Remark 4.2.5.1 It is important to realize that this squaring of the condition number is an artifact of the chosen algorithm rather than an inherent sensitivity to change of the problem.

4.3 Solution via the SVD

4.3.1 The SVD and the four fundamental spaces



YouTube: <https://www.youtube.com/watch?v=Zj72oRSSsH8>

Theorem 4.3.1.1 Given $A \in \mathbb{C}^{m \times n}$, let $A = U_L \Sigma_{TL} V_L^H$ equal its Reduced SVD and $A = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H$ its SVD. Then

- $\mathcal{C}(A) = \mathcal{C}(U_L)$,
- $\mathcal{N}(A) = \mathcal{C}(V_R)$,
- $\mathcal{R}(A) = \mathcal{C}(A^H) = \mathcal{C}(V_L)$, and
- $\mathcal{N}(A^H) = \mathcal{C}(U_R)$.

Proof. We prove that $\mathcal{C}(A) = \mathcal{C}(U_L)$, leaving the other parts as exercises.

Let $A = U_L \Sigma_{TL} V_L^H$ be the Reduced SVD of A . Then

- $U_L^H U_L = I$ (U_L is orthonormal),
- $V_L^H V_L = I$ (V_L is orthonormal), and
- Σ_{TL} is nonsingular because it is diagonal and the diagonal elements are all nonzero.

We will show that $\mathcal{C}(A) = \mathcal{C}(U_L)$ by showing that $\mathcal{C}(A) \subset \mathcal{C}(U_L)$ and $\mathcal{C}(U_L) \subset \mathcal{C}(A)$

- $\mathcal{C}(A) \subset \mathcal{C}(U_L)$:

Let $z \in \mathcal{C}(A)$. Then there exists a vector $x \in \mathbb{C}^n$ such that $z = Ax$. But then $z = Ax = U_L \Sigma_{TL} V_L^H x = U_L \underbrace{\Sigma_{TL} V_L^H x}_{\hat{x}} = U_L \hat{x}$. Hence $z \in \mathcal{C}(U_L)$.

- $\mathcal{C}(U_L) \subset \mathcal{C}(A)$:

Let $z \in \mathcal{C}(U_L)$. Then there exists a vector $x \in \mathbb{C}^r$ such that $z = U_L x$. But then $z = U_L x = U_L \underbrace{\Sigma_{TL} V_L^H V_L \Sigma_{TL}^{-1} x}_{I \quad \hat{x}} = A \underbrace{V_L \Sigma_{TL}^{-1} x}_{\hat{x}} = A \hat{x}$. Hence $z \in \mathcal{C}(A)$.

We leave the other parts as exercises for the learner. ■

Homework 4.3.1.1 For the last theorem, prove that $\mathcal{R}(A) = \mathcal{C}(A^H) = \mathcal{C}(V_L)$.

Solution. $\mathcal{R}(A) = \mathcal{C}(V_L)$:

The slickest way to do this is to recognize that if $A = U_L \Sigma_{TL} V_L^H$ is the Reduced SVD of A then $A^H = V_L \Sigma_{TL} U_L^H$ is the Reduced SVD of A^H . One can then invoke the fact that $\mathcal{C}(A) = \mathcal{C}(U_L)$ where in this case A is replaced by A^H and U_L by V_L .

Ponder This 4.3.1.2 For the last theorem, prove that $\mathcal{N}(A^H) = \mathcal{C}(U_R)$.

Homework 4.3.1.3 Given $A \in \mathbb{C}^{m \times n}$, let $A = U_L \Sigma_{TL} V_L^H$ equal its Reduced SVD and $A = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H$ its SVD, and $r = \text{rank}(A)$.

- ALWAYS/SOMETIMES/NEVER: $r = \text{rank}(A) = \dim(\mathcal{C}(A)) = \dim(\mathcal{C}(U_L))$,
- ALWAYS/SOMETIMES/NEVER: $r = \dim(\mathcal{R}(A)) = \dim(\mathcal{C}(V_L))$,
- ALWAYS/SOMETIMES/NEVER: $n - r = \dim(\mathcal{N}(A)) = \dim(\mathcal{C}(V_R))$, and
- ALWAYS/SOMETIMES/NEVER: $m - r = \dim(\mathcal{N}(A^H)) = \dim(\mathcal{C}(U_R))$.

Answer.

- ALWAYS: $r = \text{rank}(A) = \dim(\mathcal{C}(A)) = \dim(\mathcal{C}(U_L))$,
- ALWAYS: $r = \dim(\mathcal{R}(A)) = \dim(\mathcal{C}(V_L))$,

- ALWAYS: $n - r = \dim(\mathcal{N}(A)) = \dim(\mathcal{C}(V_R))$, and
- ALWAYS: $m - r = \dim(\mathcal{N}(A^H)) = \dim(\mathcal{C}(U_R))$.

Now prove it.

Solution.

- ALWAYS: $r = \text{rank}(A) = \dim(\mathcal{C}(A)) = \dim(\mathcal{C}(U_L))$,

The dimension of a space equals the number of vectors in a basis. A basis is any set of linearly independent vectors such that the entire set can be created by taking linear combinations of those vectors. The rank of a matrix is equal to the dimension of its column space which is equal to the dimension of its row space.

Now, clearly the columns of U_L are linearly independent (since they are orthonormal) and form a basis for $\mathcal{C}(U_L)$. This, together with [Theorem 4.3.1.1](#), yields the fact that $r = \text{rank}(A) = \dim(\mathcal{C}(A)) = \dim(\mathcal{C}(U_L))$.

- ALWAYS: $r = \dim(\mathcal{R}(A)) = \dim(\mathcal{C}(V_L))$,

There are a number of ways of reasoning this. One is a small modification of the proof that $r = \text{rank}(A) = \dim(\mathcal{C}(A)) = \dim(\mathcal{C}(U_L))$. Another is to look at A^H and to apply the last subproblem.

- ALWAYS: $n - r = \dim(\mathcal{N}(A)) = \dim(\mathcal{C}(V_R))$.

We know that $\dim(\mathcal{N}(A)) + \dim(\mathcal{R}(A)) = n$. The answer follows directly from this and the last subproblem.

- ALWAYS: $m - r = \dim(\mathcal{N}(A^H)) = \dim(\mathcal{C}(U_R))$.

We know that $\dim(\mathcal{N}(A^H)) + \dim(\mathcal{C}(A)) = m$. The answer follows directly from this and the first subproblem.

Homework 4.3.1.4 Given $A \in \mathbb{C}^{m \times n}$, let $A = U_L \Sigma_{TL} V_L^H$ equal its Reduced SVD and $A = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H$ its SVD.

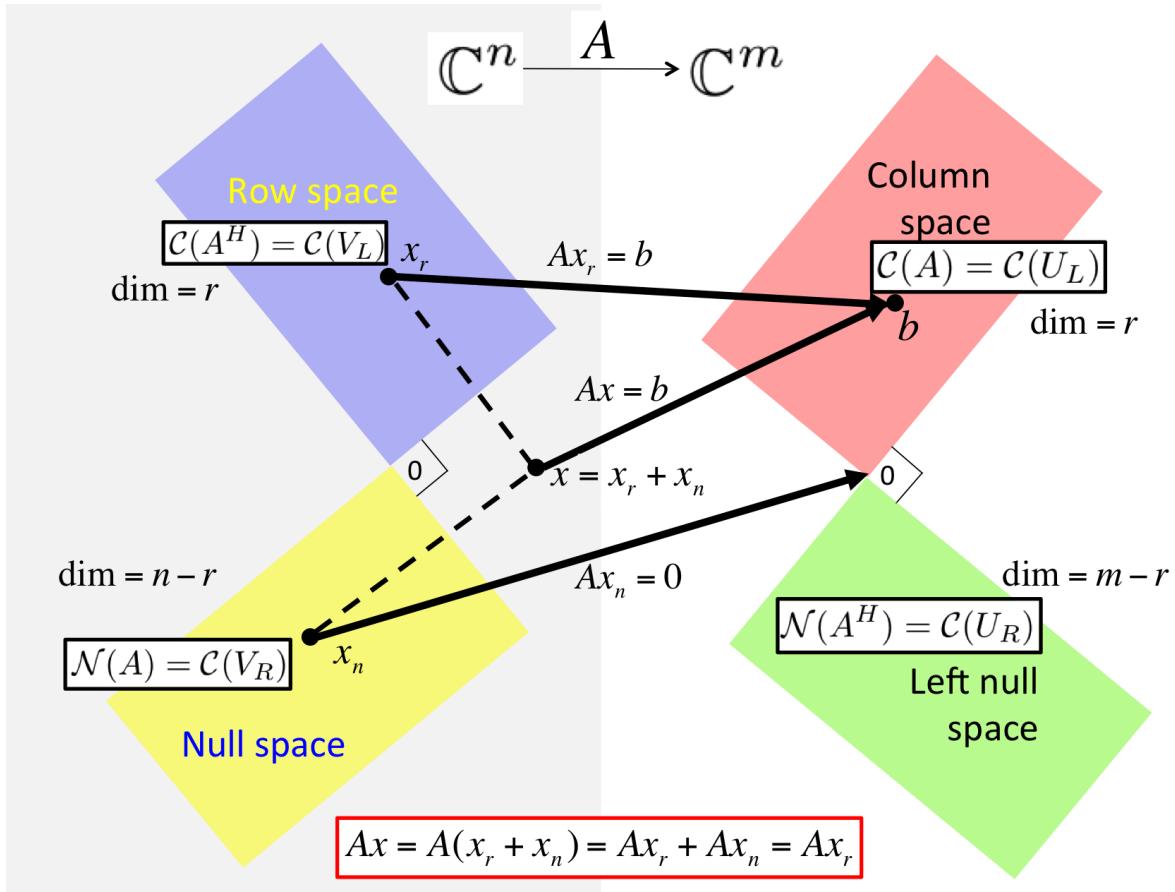
Any vector $x \in \mathbb{C}^n$ can be written as $x = x_r + x_n$ where $x_r \in \mathcal{C}(V_L)$ and $x_n \in \mathcal{C}(V_R)$.
TRUE/FALSE

Answer. TRUE

Now prove it!

Solution.

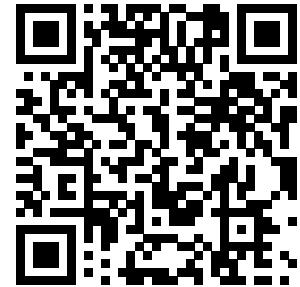
$$\begin{aligned}
 x &= Ix = VV^H x \\
 &= \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H x \\
 &= \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c} V_L^H \\ V_R^H \end{array} \right) x \\
 &= \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c} V_L^H x \\ V_R^H x \end{array} \right) \\
 &= \underbrace{V_L V_L^H x}_{x_r} + \underbrace{V_R V_R^H x}_{x_n}.
 \end{aligned}$$



[PowerPoint Source](#)

Figure 4.3.1.2 Illustration of relationship between the SVD of matrix A and the four fundamental spaces.

4.3.2 Case 1: A has linearly independent columns



YouTube: <https://www.youtube.com/watch?v=wLCN0yOLFkM>

Let us start by discussing how to use the SVD to find \hat{x} that satisfies

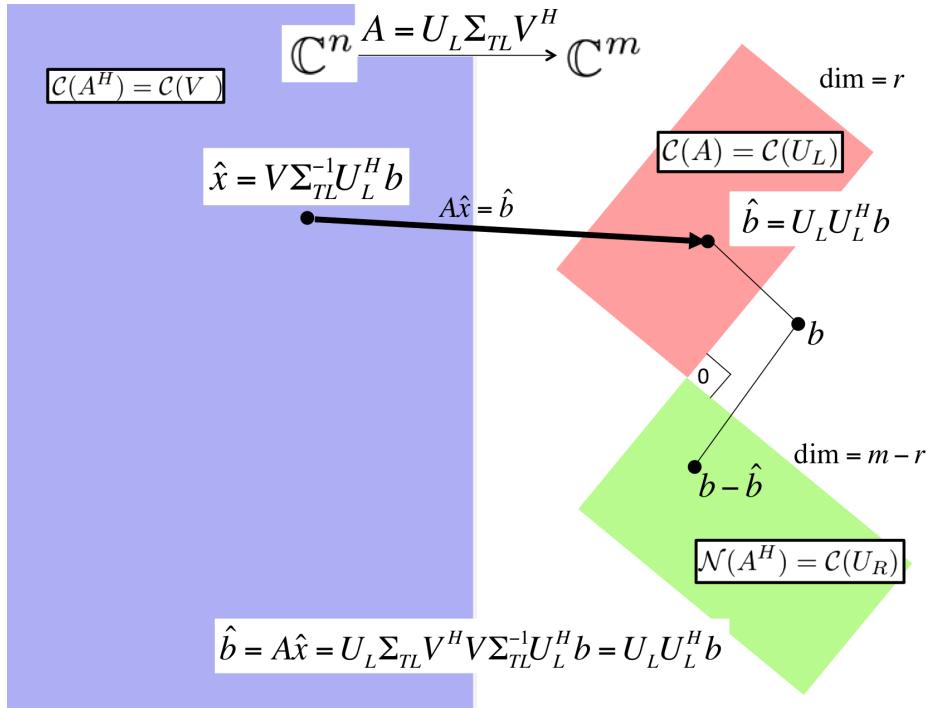
$$\|b - A\hat{x}\|_2 = \min_x \|b - Ax\|_2,$$

for the case where $A \in \mathbb{C}^{m \times n}$ has linearly independent columns (in other words, $\text{rank}(A) = n$).

Let $A = U_L \Sigma_{TL} V^H$ be its reduced SVD decomposition. (Notice that $V_L = V$ since A has linearly independent columns and hence V_L is $n \times n$ and equals V .)

Here is a way to find the solution based on what we encountered before: Since A has linearly independent columns, the solution is given by $\hat{x} = (A^H A)^{-1} A^H b$ (the solution to the normal equations). Now,

$$\begin{aligned} \hat{x} &= \langle \text{solution to the normal equations} \rangle \\ (A^H A)^{-1} A^H b &= \langle A = U_L \Sigma_{TL} V^H \rangle \\ [(U_L \Sigma_{TL} V^H)^H (U_L \Sigma_{TL} V^H)]^{-1} (U_L \Sigma_{TL} V^H)^H b &= \langle (BCD)^H = (D^H C^H B^H) \text{ and } \Sigma_{TL}^H = \Sigma_{TL} \rangle \\ [(V \Sigma_{TL} U_L^H) (U_L \Sigma_{TL} V^H)]^{-1} (V \Sigma_{TL} U_L^H) b &= \langle U_L^H U_L = I \rangle \\ [V \Sigma_{TL} \Sigma_{TL} V^H]^{-1} V \Sigma_{TL} U_L^H b &= \langle V^{-1} = V^H \text{ and } (BCD)^{-1} = D^{-1} C^{-1} B^{-1} \rangle \\ V \Sigma_{TL}^{-1} \Sigma_{TL}^{-1} V^H V \Sigma_{TL} U_L^H b &= \langle V^H V = I \text{ and } \Sigma_{TL}^{-1} \Sigma_{TL} = I \rangle \\ V \Sigma_{TL}^{-1} U_L^H b & \end{aligned}$$



[PowerPoint Source](#)

Figure 4.3.2.1 Solving LLS via the SVD when A had linearly independent columns (and hence the row space of A equals \mathbb{C}^n).

Alternatively, we can come to the same conclusion without depending on the Method of Normal Equations, in preparation for the more general case discussed in the next subsection. The derivation is captured in [Figure 4.3.2.1](#).

$$\begin{aligned}
 & \min_{x \in \mathbb{C}^n} \|b - Ax\|_2^2 \\
 &= <\text{substitute the SVDA} = U\Sigma V^H> \\
 & \min_{x \in \mathbb{C}^n} \|b - U\Sigma V^H x\|_2^2 \\
 &= <\text{substitute } I = UU^H \text{ and factor out } U> \\
 & \min_{x \in \mathbb{C}^n} \|U(U^H b - \Sigma V^H x)\|_2^2 \\
 &= <\text{multiplication by a unitary matrix preserves two-norm}> \\
 & \min_{x \in \mathbb{C}^n} \|U^H b - \Sigma V^H x\|_2^2 \\
 &= <\text{partition, partitioned matrix-matrix multiplication}> \\
 & \min_{x \in \mathbb{C}^n} \left\| \left(\frac{U_L^H b}{U_R^H b} \right) - \left(\frac{\Sigma_{TL}}{0} \right) V^H x \right\|_2^2 \\
 &= <\text{partitioned matrix-matrix multiplication and addition}> \\
 & \min_{x \in \mathbb{C}^n} \left\| \left(\frac{U_L^H b - \Sigma_{TL} V^H x}{U_R^H b} \right) \right\|_2^2 \\
 &= <\left\| \left(\frac{v_T}{v_B} \right) \right\|_2^2 = \|v_T\|_2^2 + \|v_B\|_2^2> \\
 & \min_{x \in \mathbb{C}^n} \|U_L^H b - \Sigma_{TL} V^H x\|_2^2 + \|U_R^H b\|_2^2
 \end{aligned}$$

The x that solves $\Sigma_{TL}V^Hx = U_L^Hb$ minimizes the expression. That x is given by

$$\hat{x} = V\Sigma_{TL}^{-1}U_L^Hb.$$

since Σ_{TL} is a diagonal matrix with only nonzeros on its diagonal and V is unitary.

Here is yet another way of looking at this: we wish to compute \hat{x} that satisfies

$$\|b - A\hat{x}\|_2 = \min_x \|b - Ax\|_2,$$

for the case where $A \in \mathbb{C}^{m \times n}$ has linearly independent columns. We know that $A = U_L\Sigma_{TL}V^H$, its Reduced SVD. To find the x that minimizes, we first project b onto the column space of A . Since the column space of A is identical to the column space of U_L , we can project onto the column space of U_L instead:

$$\hat{b} = U_LU_L^Hb.$$

(Notice that this is *not* because U_L is unitary, since it isn't. It is because the matrix $U_LU_L^H$ projects onto the columns space of U_L since U_L is orthonormal.) Now, we wish to find \hat{x} that exactly solves $A\hat{x} = \hat{b}$. Substituting in the Reduced SVD, this means that

$$U_L\Sigma_{TL}V^H\hat{x} = U_LU_L^Hb.$$

Multiplying both sides by U_L^H yields

$$\Sigma_{TL}V^H\hat{x} = U_L^Hb.$$

and hence

$$\hat{x} = V\Sigma_{TL}^{-1}U_L^Hb.$$

We believe this last explanation probably leverages the Reduced SVD in a way that provides the most insight, and it nicely motivates how to find solutions to the LLS problem when $\text{rank}(A) < r$.

The steps for solving the linear least squares problem via the SVD, when $A \in \mathbb{C}^{m \times n}$ has linearly independent columns, and the costs of those steps are given by

- Compute the Reduced SVD $A = U_L\Sigma_{TL}V^H$.

We will not discuss practical algorithms for computing the SVD until much later. We will see that the cost is $O(mn^2)$ with a large constant.

- Compute $\hat{x} = V\Sigma_{TL}^{-1}U_L^Hb$.

The cost of this is approximately,

- Form $y_T = U_L^Hb$: $2mn$ flops.
- Scale the individual entries in y_T by dividing by the corresponding singular values: n divides, overwriting $y_T = \Sigma_{TL}^{-1}y_T$. The cost of this is negligible.
- Compute $\hat{x} = Vy_T$: $2n^2$ flops.

The devil is in the details of how the SVD is computed and whether the matrices U_L and/or V are explicitly formed.

4.3.3 Case 2: General case



YouTube: <https://www.youtube.com/watch?v=qhsPHQk1id8>

Now we show how to use the SVD to find \hat{x} that satisfies

$$\|b - A\hat{x}\|_2 = \min_x \|b - Ax\|_2,$$

where $\text{rank}(A) = r$, with no assumptions about the relative size of m and n . In our discussion, we let $A = U_L \Sigma_{TL} V_L^H$ equal its Reduced SVD and

$$A = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ 0 & 0 \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H$$

its SVD.

The first observation is, once more, that an \hat{x} that minimizes $\|b - Ax\|_2$ satisfies

$$A\hat{x} = \hat{b},$$

where $\hat{b} = U_L U_L^H b$, the orthogonal projection of b onto the column space of A . Notice our use of "an \hat{x} " since the solution won't be unique if $r < m$ and hence the null space of A is not trivial. Substituting in the SVD this means that

$$\left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ 0 & 0 \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H \hat{x} = U_L U_L^H b.$$

Multiplying both sides by U_L^H yields

$$\left(\begin{array}{c|c} I & 0 \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ 0 & 0 \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H \hat{x} = U_L^H b$$

or, equivalently,

$$\Sigma_{TL} V_L^H \hat{x} = U_L^H b. \quad (4.3.1)$$

Any solution to this can be written as the sum of a vector in the row space of A with a vector in the null space of A :

$$\hat{x} = Vz = \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c} z_T \\ z_B \end{array} \right) = \underbrace{V_L z_T}_{x_r} + \underbrace{V_R z_B}_{x_n}.$$

Substituting this into (4.3.1) we get

$$\Sigma_{TL} V_L^H (V_L z_T + V_R z_B) = U_L^H b,$$

which leaves us with

$$\Sigma_{TL} z_T = U_L^H b.$$

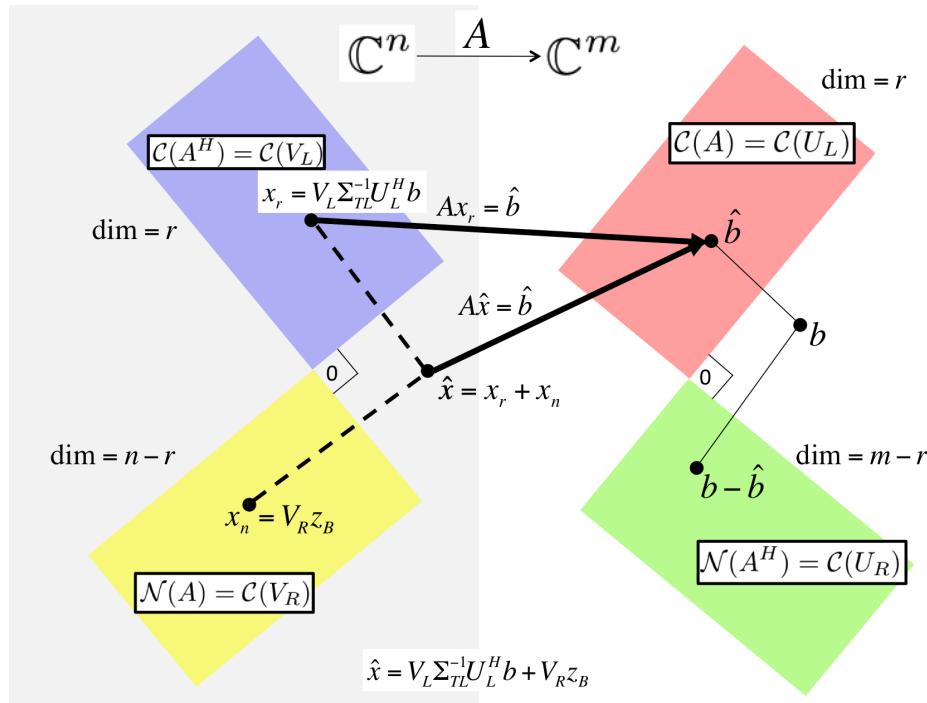
Thus, the solution in the row space is given by

$$x_r = V_L z_T = V_L \Sigma_{TL}^{-1} U_L^H b$$

and the general solution is given by

$$\hat{x} = V_L \Sigma_{TL}^{-1} U_L^H b + V_R z_B,$$

where z_B is any vector in \mathbb{C}^{n-r} . This reasoning is captured in Figure 4.3.3.1.



[PowerPoint Source](#)

Figure 4.3.3.1 Solving LLS via the SVD of A .

Homework 4.3.3.1 Reason that

$$\hat{x} = V_L \Sigma_{TL}^{-1} U_L^H b$$

is the solution to the LLS problem with minimal length (2-norm). In other words, if x^* satisfies

$$\|b - Ax^*\|_2 = \min_x \|b - Ax\|_2$$

then $\|\hat{x}\|_2 \leq \|x^*\|_2$.

Solution. The important insight is that

$$x^* = \underbrace{V_L \Sigma_{TL}^{-1} U_L^H b}_{\hat{x}} + V_R z_B$$

and that

$$V_L \Sigma_{TL}^{-1} U_L^H b \quad \text{and} \quad V_R z_B$$

are orthogonal to each other (since $V_L^H V_R = 0$). If $u^H v = 0$ then $\|u + v\|_2^2 = \|u\|_2^2 + \|v\|_2^2$. Hence

$$\|x^*\|_2^2 = \|\hat{x} + V_R z_B\|_2^2 = \|\hat{x}\|_2^2 + \|V_R z_B\|_2^2 \geq \|\hat{x}\|_2^2$$

and hence $\|\hat{x}\|_2 \leq \|x^*\|_2$.

4.4 Solution via the QR factorization

4.4.1 A has linearly independent columns



YouTube: <https://www.youtube.com/watch?v=mKAZjYX656Y>

Theorem 4.4.1.1 Assume $A \in \mathbb{C}^{m \times n}$ has linearly independent columns and let $A = QR$ be its QR factorization with orthonormal matrix $Q \in \mathbb{C}^{m \times n}$ and upper triangular matrix $R \in \mathbb{C}^{n \times n}$. Then the LLS problem

$$\text{Find } \hat{x} \in \mathbb{C}^n \text{ such that } \|b - A\hat{x}\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2$$

is solved by the unique solution of

$$R\hat{x} = Q^H b.$$

Proof 1. Since $A = QR$, minimizing $\|b - Ax\|_2$ means minimizing

$$\|b - Q \underbrace{Rx}_z\|_2.$$

Since R is nonsingular, we can first find z that minimizes

$$\|b - Qz\|_2$$

after which we can solve $Rx = z$ for x . But from the Method of Normal Equations we know

that the minimizing z solves

$$Q^H Q z = Q^H b.$$

Since Q has orthonormal columns, we thus deduce that

$$z = Q^H b.$$

Hence, the desired \hat{x} must satisfy

$$R\hat{x} = Q^H b.$$

■

Proof 2. Let $A = Q_L R_{TL}$ be the QR factorization of A . We know that then there exists a matrix Q_R such that $Q = \begin{pmatrix} Q_L & Q_R \end{pmatrix}$ is unitary: Q_R is an orthonormal basis for the space orthogonal to the space spanned by Q_L . Now,

$$\begin{aligned} & \min_{x \in \mathbb{C}^n} \|b - Ax\|_2^2 \\ &= <\text{ substitute } A = Q_L R_{TL} > \\ & \min_{x \in \mathbb{C}^n} \|b - Q_L R_{TL} x\|_2^2 \\ &= <\text{ two norm is preserved since } Q^H \text{ is unitary} > \\ & \min_{x \in \mathbb{C}^n} \|Q^H(b - Q_L R_{TL} x)\|_2^2 \\ &= <\text{ partitioning; distributing} > \\ & \min_{x \in \mathbb{C}^n} \left\| \left(\frac{Q_L^H}{Q_R^H} \right) b - \left(\frac{Q_L^H}{Q_R^H} \right) Q_L R_{TL} x \right\|_2^2 \\ &= <\text{ partitioned matrix-matrix multiplication} > \\ & \min_{x \in \mathbb{C}^n} \left\| \left(\frac{Q_L^H b}{Q_R^H b} \right) - \left(\frac{R_{TL} x}{0} \right) \right\|_2^2 \\ &= <\text{ partitioned matrix addition} > \\ & \min_{x \in \mathbb{C}^n} \left\| \left(\frac{Q_L^H b - R_{TL} x}{Q_R^H b} \right) \right\|_2^2 \\ &= <\text{ property of the 2-norm: } \left\| \left(\frac{u}{v} \right) \right\|_2^2 = \|u\|_2^2 + \|v\|_2^2 > \\ & \min_{x \in \mathbb{C}^n} \left(\|Q_L^H b - R_{TL} x\|_2^2 + \|Q_R^H b\|_2^2 \right) \\ &= <\text{ } Q_R^H b \text{ is independent of } x > \\ & \left(\min_{x \in \mathbb{C}^n} \|Q_L^H b - R_{TL} x\|_2^2 \right) + \|Q_R^H b\|_2^2 \\ &= <\text{ minimized by } \hat{x} \text{ that satisfies } R_{TL} \hat{x} = Q_L^H b > \\ & \|Q_R^H b\|_2^2. \end{aligned}$$

Thus, the desired \hat{x} that minimizes the linear least squares problem solves $R_{TL} \hat{x} = Q_L^H b$. The solution is unique because R_{TL} is nonsingular (because A has linearly independent columns). ■

Homework 4.4.1.1 Yet another alternative proof for [Theorem 4.4.1.1](#) starts with the observation that the solution is given by $\hat{x} = (A^H A)^{-1} A^H b$ and then substitutes in $A = QR$. Give a proof that builds on this insight.

Solution. Recall that we saw in [Subsection 4.2.2](#) that, if A has linearly independent columns, the LLS solution is given by $\hat{x} = (A^H A)^{-1} A^H b$ (the solution to the normal equations). Also, if A has linearly independent columns and $A = QR$ is its QR factorization, then the upper triangular matrix R is nonsingular (and hence has no zeroes on its diagonal).

Now,

$$\begin{aligned}\hat{x} &= \text{< Solution to the Normal Equations >} \\ (A^H A)^{-1} A^H b &= \text{< } A = QR \text{ >} \\ [(QR)^H (QR)]^{-1} (QR)^H b &= \text{< } (BC)^H = (C^H B^H) \text{ >} \\ [R^H Q^H QR]^{-1} R^H Q^H b &= \text{< } Q^H Q = I \text{ >} \\ [R^H R]^{-1} R^H Q^H b &= \text{< } (BC)^{-1} = C^{-1} B^{-1} \text{ >} \\ R^{-1} R^{-H} R^H Q^H b &= \text{< } R^{-H} R^H = I \text{ >} \\ R^{-1} Q^H b.\end{aligned}$$

Thus, the \hat{x} that solves $R\hat{x} = Q^H b$ solves the LLS problem.

Ponder This 4.4.1.2 Create a picture similar to [Figure 4.3.2.1](#) that uses the QR factorization rather than the SVD.

4.4.2 Via Gram-Schmidt QR factorization

In [Section 3.2](#), you were introduced to the (Classical and Modified) Gram-Schmidt process and how it was equivalent to computing a QR factorization of the matrix, A , that has as columns the linearly independent vectors being orthonormalized. The resulting Q and R can be used to solve the linear least squares problem by first computing $y = Q^H b$ and next solving $R\hat{x} = y$.

Starting with $A \in \mathbb{C}^{m \times n}$ let's explicitly state the steps required to solve the LLS problem via either CGS or MGS and analyze the cost.:

- From [Homework 3.2.6.1](#) or [Homework 3.2.6.2](#), factoring $A = QR$ via CGS or MGS costs, approximately, $2mn^2$ flops.
- Compute $y = Q^H b$: $2mn$ flops.
- Solve $R\hat{x} = y$: n^2 flops.

Total: $2mn^2 + 2mn + n^2$ flops.

4.4.3 Via the Householder QR factorization



YouTube: https://www.youtube.com/watch?v=Mk-Y_15aGGc

Given $A \in \mathbb{C}^{m \times n}$ with linearly independent columns, the Householder QR factorization yields n Householder transformations, H_0, \dots, H_{n-1} , so that

$$\underbrace{H_{n-1} \cdots H_0}_{Q^H} A = \begin{pmatrix} R_{TL} \\ 0 \end{pmatrix}.$$

$[A, t] = \text{HouseQR_unb_var1}(A)$ overwrites A with the Householder vectors that define H_0, \dots, H_{n-1} below the diagonal and R_{TL} in the upper triangular part.

Rather than explicitly computing Q and then computing $\tilde{y} := Q^H y$, we can instead apply the Householder transformations:

$$\tilde{y} := H_{n-1} \cdots H_0 y,$$

overwriting y with \tilde{y} . After this, the vector y is partitioned as $y = \begin{pmatrix} y_T \\ y_B \end{pmatrix}$ and the triangular system $R_{TL}\hat{x} = y_T$ yields the desired solution.

The steps and theirs costs of this approach are

- From Subsection 3.3.4, factoring $A = QR$ via the Householder QR factorization costs, approximately, $2mn^2 - \frac{2}{3}n^3$ flops.
- From Homework 3.3.6.1, applying Q as a sequence of Householder transformations costs, approximately, $4mn - 2n^2$ flops.
- Solve $R_{TL}\hat{x} = y_T$: n^2 flops.

Total: $2mn^2 - \frac{2}{3}n^3 + 4mn - n^2 \approx 2mn^2 - \frac{2}{3}n^3$ flops.

4.4.4 A has linearly dependent columns

Let us now consider the case where $A \in \mathbb{C}^{m \times n}$ has rank $r \leq n$. In other words, it has r linearly independent columns. Let $p \in \mathbb{R}^n$ be a permutation vector, by which we mean a permutation of the vector

$$\begin{pmatrix} 0 \\ 1 \\ \vdots \\ n-1 \end{pmatrix}$$

And $P(p)$ be the matrix that, when applied to a vector $x \in \mathbb{C}^n$ permutes the entries of x according to the vector p :

$$P(p)x = \underbrace{\begin{pmatrix} e_{\pi_0}^T \\ e_{\pi_1}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}}_{P(p)} x = \begin{pmatrix} e_{\pi_0}^T x \\ e_{\pi_1}^T x \\ \vdots \\ e_{\pi_{n-1}}^T x \end{pmatrix} = \begin{pmatrix} \chi_{\pi_0} \\ \chi_{\pi_1} \\ \vdots \\ \chi_{\pi_{n-1}} \end{pmatrix}.$$

where e_j equals the columns of $I \in \mathbb{R}^{n \times n}$ indexed with j (and hence the standard basis vector indexed with j).

If we apply $P(p)^T$ to $A \in \mathbb{C}^{m \times n}$ from the right, we get

$$\begin{aligned} AP(p)^T &= <\text{definition of } P(p)> \\ A \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}^T &= <\text{transpose}> \\ A(e_{\pi_0} | \cdots | e_{\pi_{n-1}}) &= <\text{matrix multiplication by columns}> \\ (Ae_{\pi_0} | \cdots | Ae_{\pi_{n-1}}) &= <Be_j = b_j> \\ (a_{\pi_0} | \cdots | a_{\pi_{n-1}}). \end{aligned}$$

In other words, applying the transpose of the permutation matrix to A from the right permutes its columns as indicated by the permutation vector p .

The discussion about permutation matrices gives us the ability to rearrange the columns of A so that the first $r = \text{rank}(A)$ columns are linearly independent.

Theorem 4.4.4.1 Assume $A \in \mathbb{C}^{m \times n}$ and that $r = \text{rank}(A)$. Then there exists a permutation vector $p \in \mathbb{R}^n$, orthonormal matrix $Q_L \in \mathbb{C}^{m \times r}$, upper triangular matrix $R_{TL} \in \mathbb{C}^{r \times r}$, and $R_{TR} \in \mathbb{C}^{r \times (n-r)}$ such that

$$AP(p)^T = Q_L (R_{TL} | R_{TR}).$$

Proof. Let p be the permutation vector such that the first r columns of $A^P = AP(p)^T$ are linearly independent. Partition

$$A^P = AP(p)^T = (A_L^P | A_R^P)$$

where $A_L^P \in \mathbb{C}^{m \times r}$. Since A_L^P has linearly independent columns, its QR factorization, $A^P = Q_L R_{TL}$, exists. Since all the linearly independent columns of matrix A were permuted to the left, the remaining columns, now part of A_R^P , are in the column space of A_L^P and hence in the column space of Q_L . Hence $A_R^P = Q_L R_{TR}$ for some matrix R_{TR} , which then must satisfy

$Q_L^H A_R^P = R_{TR}$ giving us a means by which to compute it. We conclude that

$$A^P = AP(p)^T = \left(\begin{array}{c|c} A_L^P & A_R^P \end{array} \right) = Q_L \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right).$$

■

Let us examine how this last theorem can help us solve the LLS

$$\text{Find } \hat{x} \in \mathbb{C}^n \text{ such that } \|b - Ax\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2$$

when $\text{rank}(A) \leq n$:

$$\begin{aligned} & \min_{x \in \mathbb{C}^n} \|b - Ax\|_2 \\ &= \langle P(p)^T P(p) = I \rangle \\ & \min_{x \in \mathbb{C}^n} \|b - AP(p)^T P(p)x\|_2 \\ &= \langle AP(p)^T = Q_L \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) \rangle \\ & \min_{x \in \mathbb{C}^n} \underbrace{\|b - Q_L \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) P(p)x\|_2}_w \\ &= \langle \text{substitute } w = \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) P(p)x \rangle \\ & \min_{w \in \mathbb{C}^r} \|b - Q_L w\|_2 \end{aligned}$$

which is minimized when $w = Q_L^H b$. Thus, we are looking for vector \hat{x} such that

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) P(p)\hat{x} = Q_L^H b.$$

Substituting

$$z = \left(\begin{array}{c} z_T \\ z_B \end{array} \right)$$

for $P(p)\hat{x}$ we find that

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) \left(\begin{array}{c} z_T \\ z_B \end{array} \right) = Q_L^H b.$$

Now, we can pick $z_B \in \mathbb{C}^{n-r}$ to be an arbitrary vector, and determine a corresponding z_T by solving

$$R_{TL}z_T = Q_L^H b - R_{TR}z_B.$$

A convenient choice is $z_B = 0$ so that z_T solves

$$R_{TL}z_T = Q_L^H b.$$

Regardless of choice of z_B , the solution \hat{x} is given by

$$\hat{x} = P(p)^T \left(\frac{R_{TL}^{-1}(Q_L^H b - R_{TR}z_B)}{z_B} \right).$$

(a permutation of vector z .) This defines an infinite number of solutions if $\text{rank}(A) < n$.

The problem is that we don't know which columns are linearly independent in advance. In enrichments in [Subsection 4.5.1](#) and [Subsection 4.5.2](#), rank-revealing QR factorization algorithms are discussed that overcome this problem.

4.5 Enrichments

4.5.1 Ran -Revealing QR (RRQR) via MGS

The discussion in [Subsection 4.4.4](#) falls short of being a practical algorithm for at least two reasons:

- One needs to be able to determine in advance what columns of A are linearly independent; and
- Due to roundoff error or error in the data from which the matrix was created, a column may be linearly independent of other columns when for practical purposes it should be considered dependent.

We now discuss how the MGS algorithm can be modified so that appropriate linearly independent columns can be determined "on the fly" as well as the defacto rank of the matrix. The result is known as the **Rank Revealing QR factorization (RRQR)**. It is also known as **QR factorization with column pivoting**. We are going to give a modification of the MGS algorithm for computing the RRQR.

For our discussion, we introduce an elementary pivot matrix, $\tilde{P}(j) \in \mathbb{C}^{n \times n}$, that swaps the first element of the vector to which it is applied with the element indexed with j :

$$\tilde{P}(j)x = \begin{pmatrix} e_j^T \\ e_1^T \\ \vdots \\ e_{j-1}^T \\ e_0^T \\ e_{j+1}^T \\ \vdots \\ e_{n-1}^T \end{pmatrix} x = \begin{pmatrix} e_j^T x \\ e_1^T x \\ \vdots \\ e_{j-1}^T x \\ e_0^T x \\ e_{j+1}^T x \\ \vdots \\ e_{n-1}^T x \end{pmatrix} = \begin{pmatrix} \chi_j \\ \chi_1 \\ \vdots \\ \chi_{j-1} \\ \chi_0 \\ \chi_{j+1} \\ \vdots \\ \chi_{n-1} \end{pmatrix}.$$

Another way of stating this is that

$$\tilde{P}(j) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & I_{(j-1) \times (j-1)} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{(n-j-1) \times (n-j-1)} \end{pmatrix},$$

where $I_{k \times k}$ equals the $k \times k$ identity matrix. When applying $\tilde{P}(j)$ from the right to a matrix, it swaps the first column and the column indexed with j . Notice that $\tilde{P}(j)^T = \tilde{P}(j)$ and $\tilde{P}(j) = \tilde{P}(j)^{-1}$.

Remark 4.5.1.1 For a more detailed discussion of permutation matrices, you may want to consult Week 7 of "Linear Algebra: Foundations to Frontiers" (LAFF) [\[26\]](#). We also revisit this in [Section 5.3](#) when discussing LU factorization with partial pivoting.

Here is an outline of the algorithm:

- Determine the index π_1 such that the column of A indexed with π_1 has the largest 2-norm (is the longest).
- Permute $A := A\tilde{P}(\pi_1)$, swapping the first column with the column that is longest.
- Partition

$$A \rightarrow \begin{pmatrix} a_1 & A_2 \end{pmatrix}, Q \rightarrow \begin{pmatrix} q_1 & Q_2 \end{pmatrix}, R \rightarrow \begin{pmatrix} \rho_{11} & r_{12}^T \\ 0 & R_{22} \end{pmatrix}, p \rightarrow \begin{pmatrix} \pi_1 \\ p_2 \end{pmatrix}$$

- Compute $\rho_{11} := \|a_1\|_2$.
- $q_1 := a_1 / \rho_{11}$.
- Compute $r_{12}^T := q_1^T A_2$.
- Update $A_2 := A_2 - q_1 r_{12}^T$.

This subtracts the component of each column that is in the direction of q_1 .

- Continue the process with the updated matrix A_2 .

The complete algorithm, which overwrites A with Q , is given in [Figure 4.5.1.2](#). Observe that the elements on the diagonal of R will be positive and in non-increasing order because updating $A_2 := A_2 - q_1 r_{12}^T$ inherently does not increase the length of the columns of A_2 . After all, the component in the direction of q_1 is being subtracted from each column of A_2 , leaving the component orthogonal to q_1 .

$[A, R, p] := \text{RRQR_MGS_simple}(A, R, p)$
$A \rightarrow \left(\begin{array}{c c} A_L & A_R \end{array} \right), R \rightarrow \left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right)$
A_L has 0 columns, R_{TL} is 0×0 , p_T has 0 rows
while $n(A_L) < n(A)$
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_0 & a_1 & A_2 \end{array} \right),$ $\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ R_{20} & r_{21} & R_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ p_2 \end{array} \right)$
$\pi_1 = \text{DetermineColumnIndex}(\left(\begin{array}{cc} a_1 & A_2 \end{array} \right))$
$\left(\begin{array}{cc} a_1 & A_2 \end{array} \right) := \left(\begin{array}{cc} a_1 & A_2 \end{array} \right) \tilde{P}(\pi_1)$
$\rho_{11} := \ a_1\ _2$
$a_1 := a_1 / \rho_{11}$
$r_{12}^T := a_1^T A_2$
$A_2 := A_2 - a_1 r_{12}^T$
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{cc c} A_0 & a_1 & A_2 \end{array} \right),$
$\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ R_{20} & r_{21} & R_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ p_2 \end{array} \right)$
endwhile

Figure 4.5.1.2 Simple implementation of RRQR via MGS. Incorporating a stopping criteria that checks whether ρ_{11} is small would allow the algorithm to determine the effective rank of the input matrix.

The problem with the algorithm in Figure 4.5.1.2 is that determining the index π_1 requires the 2-norm of all columns in A_R to be computed, which costs $O(m(n-j))$ flops when A_L has j columns (and hence A_R has $n-j$ columns). The following insight reduces this cost:

Let $A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$, $v = \begin{pmatrix} \nu_0 \\ \nu_1 \\ \vdots \\ \nu_{n-1} \end{pmatrix} = \begin{pmatrix} \|a_0\|_2^2 \\ \|a_1\|_2^2 \\ \vdots \\ \|a_{n-1}\|_2^2 \end{pmatrix}$, $q^T q = 1$ (here q is of the same size as the columns of A), and $r = A^T q = \begin{pmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_{n-1} \end{pmatrix}$. Compute $B := A - qr^T$ with

$B = \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right)$. Then

$$\begin{pmatrix} \|b_0\|_2^2 \\ \|b_1\|_2^2 \\ \vdots \\ \|b_{n-1}\|_2^2 \end{pmatrix} = \begin{pmatrix} \nu_0 - \rho_0^2 \\ \nu_1 - \rho_1^2 \\ \vdots \\ \nu_{n-1} - \rho_{n-1}^2 \end{pmatrix}.$$

To verify this, notice that

$$a_i = (a_i - a_i^T qq) + a_i^T qq$$

and

$$(a_i - a_i^T qq)^T q = a_i^T q - a_i^T qq^T q = a_i^T q - a_i^T q = 0.$$

This means that

$$\|a_i\|_2^2 = \|(a_i - a_i^T qq) + a_i^T qq\|_2^2 = \|a_i - a_i^T qq\|_2^2 + \|a_i^T qq\|_2^2 = \|a_i - \rho_i q\|_2^2 + \|\rho_i q\|_2^2 = \|b_i\|_2^2 + \rho_i^2$$

so that

$$\|b_i\|_2^2 = \|a_i\|_2^2 - \rho_i^2 = \nu_i - \rho_i^2.$$

Building on this insight, we make an important observation that greatly reduces the cost of determining the column that is longest. Let us start by computing v as the vector such that the i th entry in v equals the square of the length of the i th column of A . In other words, the i th entry of v equals the dot product of the i column of A with itself. In the above outline for the MGS with column pivoting, we can then also partition

$$v \rightarrow \begin{pmatrix} \nu_1 \\ v_2 \end{pmatrix}.$$

The question becomes how v_2 before the update $A_2 := A_2 - q_1 r_{12}^T$ compares to v_2 after that update. The answer is that the i th entry of v_2 must be updated by subtracting off the square of the i th entry of r_{12}^T .

Let us introduce the functions $v = \text{ComputeWeights}(A)$ and $v = \text{UpdateWeights}(v, r)$ to compute the described weight vector v and to update a weight vector v by subtracting from its elements the squares of the corresponding entries of r . Also, the function `DeterminePivot` returns the index of the largest in the vector, and swaps that entry with the first entry. An optimized RRQR via MGS algorithm, RRQR-MGS, is now given in [Figure 4.5.1.3](#). In that algorithm, A is overwritten with Q .

$[A, R, p] := \text{RRQR_MSG}(A, R, p)$
$v := \text{ComputeWeights}(A)$
$A \rightarrow (A_L A_R), R \rightarrow \left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right), v \rightarrow \left(\begin{array}{c} v_T \\ \hline v_B \end{array} \right)$
A_L has 0 columns, R_{TL} is 0×0 , p_T has 0 rows, v_T has 0 rows
while $n(A_L) < n(A)$
$(A_L A_R) \rightarrow (A_0 a_1 \ A_2),$
$\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ R_{20} & r_{21} & R_{22} \end{array} \right),$
$\left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ \hline v_B \end{array} \right) \rightarrow \left(\begin{array}{c} v_0 \\ \hline \nu_1 \\ v_2 \end{array} \right)$
$\left[\left(\begin{array}{c} \nu_1 \\ \hline v_2 \end{array} \right), \pi_1 \right] = \text{DeterminePivot}\left(\left(\begin{array}{c} \nu_1 \\ \hline v_2 \end{array} \right) \right)$
$(A_0 a_1 \ A_2) := (A_0 a_1 \ A_2) \tilde{P}(\pi_1)^T$
$\rho_{11} := \ a_1\ _2$
$a_1 := a_1 / \rho_{11}$
$r_{12}^T := q_1^T A_2$
$A_2 := A_2 - q_1 r_{12}^T$
$v_2 := \text{UpdateWeights}(v_2, r_{12})$
$(A_L A_R) \leftarrow (A_0 \ a_1 A_2),$
$\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ R_{20} & r_{21} & R_{22} \end{array} \right),$
$\left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ \hline v_B \end{array} \right) \leftarrow \left(\begin{array}{c} v_0 \\ \hline \nu_1 \\ v_2 \end{array} \right)$
endwhile

Figure 4.5.1.3 RRQR via MGS, with optimization. Incorporating a stopping criteria that checks whether ρ_{11} is small would allow the algorithm to determine the effective rank of the input matrix.

Let us revisit the fact that the diagonal elements of R are positive and in nonincreasing order. This upper triangular matrix is singular if a diagonal element equals zero (and hence all subsequent diagonal elements equal zero). Hence, if ρ_{11} becomes small relative to prior diagonal elements, the remaining columns of the (updated) A_R are essentially zero vectors, and the original matrix can be approximated with

$$A \approx Q_L \left(\begin{array}{cc} R_{TL} & R_{TR} \end{array} \right) = \boxed{\quad} \cdot \boxed{\quad}.$$

If Q_L has k columns, then this becomes a rank-k approximation.

Remark 4.5.1.4 Notice that in updating the weight vector v , the accuracy of the entries may progressively deteriorate due to catastrophic cancellation. Since these values are only used to determine the order of the columns and, importantly, when they become very small the rank of the matrix has revealed itself, this is in practice not a problem.

4.5.2 Rank Revealing Householder QR factorization

The unblocked QR factorization discussed in [Section 3.3](#) can be supplemented with column pivoting, yielding HQRP_unb_var1 in [Figure 4.5.2.1](#). In that algorithm, we incorporate the idea that the weights that are used to determine how to pivot can be updated at each step by using information in the partial row r_{12}^T , which overwrites a_{12}^T , just like it was in [Subsection 4.5.1](#).

$[A, t, p] = \text{HQRP_unb_var1}(A)$
$v := \text{ComputeWeights}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ t_B \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ p_B \end{array} \right), v \rightarrow \left(\begin{array}{c} v_T \\ v_B \end{array} \right)$
A_{TL} is 0×0 and t_T has 0 elements
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right),$
$\left(\begin{array}{c} p_T \\ p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \pi_1 \\ p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ v_B \end{array} \right) \rightarrow \left(\begin{array}{c} v_0 \\ \nu_1 \\ v_2 \end{array} \right)$
$\left[\begin{array}{c} \nu_1 \\ v_2 \end{array} \right], \pi_1] = \text{DeterminePivot}\left(\begin{array}{c} \nu_1 \\ v_2 \end{array} \right)$
$\left(\begin{array}{cc} a_{01} & A_{02} \\ \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{cc} a_{01} & A_{02} \\ \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{array} \right) P(\pi_1)^T$
$\left[\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right], \tau_1] := \left[\begin{array}{c} \rho_{11} \\ u_{21} \end{array} \right], \tau_1] = \text{Housev} \left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right)$
$w_{12}^T := (a_{12}^T + a_{21}^H A_{22})/\tau_1$
$\left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right) := \left(\begin{array}{c} a_{12}^T - w_{12}^T \\ A_{22} - a_{21} w_{12}^T \end{array} \right)$
$v_2 = \text{UpdateWeight}(v_2, a_{12})$
...
endwhile

Figure 4.5.2.1 Rank Revealing Householder QR factorization algorithm.

Combining a blocked Householder QR factorization algorithm, as discussed in [Subsubsection 3.4.1.3](#), with column pivoting is tricky, since half the computational cost is inherently

in computing the parts of R that are needed to update the weights and that stands in the way of a true blocked algorithm (that casts most computation in terms of matrix-matrix multiplication). The following papers are related to this:

- [33] Gregorio Quintana-Orti, Xiaobai Sun, and Christof H. Bischof, A BLAS-3 version of the QR factorization with column pivoting, SIAM Journal on Scientific Computing, 19, 1998.

discusses how to cast approximately half the computation in terms of matrix-matrix multiplication.

- [25] Per-Gunnar Martinsson, Gregorio Quintana-Orti, Nathan Heavner, Robert van de Geijn, Householder QR Factorization With Randomization for Column Pivoting (HQRRP), SIAM Journal on Scientific Computing, Vol. 39, Issue 2, 2017.

shows how a randomized algorithm can be used to cast most computation in terms of matrix-matrix multiplication.

4.6 Wrap Up

4.6.1 Additional homework

We start with some concrete problems from our undergraduate course titled "Linear Algebra: Foundations to Frontiers" [26]. If you have trouble with these, we suggest you look at Chapter 11 of that course.

Homework 4.6.1.1 Consider $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$ and $b = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$.

- Compute an orthonormal basis for $\mathcal{C}(A)$.
- Use the method of normal equations to compute the vector \hat{x} that minimizes $\min_x \|b - Ax\|_2$
- Compute the orthogonal projection of b onto $\mathcal{C}(A)$.
- Compute the QR factorization of matrix A .
- Use the QR factorization of matrix A to compute the vector \hat{x} that minimizes $\min_x \|b - Ax\|_2$

Homework 4.6.1.2 The vectors

$$q_0 = \frac{\sqrt{2}}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}, \quad q_1 = \frac{\sqrt{2}}{2} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}.$$

- TRUE/FALSE: These vectors are mutually orthonormal.

- Write the vector $\begin{pmatrix} 4 \\ 2 \end{pmatrix}$ as a linear combination of vectors q_0 and q_1 .

4.6.2 Summary

The LLS problem can be stated as: Given $A \in \mathbb{C}^{m \times n}$ and $b \in \mathbb{C}^m$ find $\hat{x} \in \mathbb{C}^n$ such that

$$\|b - A\hat{x}\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2.$$

Given $A \in \mathbb{C}^{m \times n}$,

- The column space, $\mathcal{C}(A)$, which is equal to the set of all vectors that are linear combinations of the columns of A

$$\{y \mid y = Ax\}.$$

- The null space, $\mathcal{N}(A)$, which is equal to the set of all vectors that are mapped to the zero vector by A

$$\{x \mid Ax = 0\}.$$

- The row space, $\mathcal{R}(A)$, which is equal to the set

$$\{y \mid y^H = x^H A\}.$$

Notice that $\mathcal{R}(A) = \mathcal{C}(A^H)$.

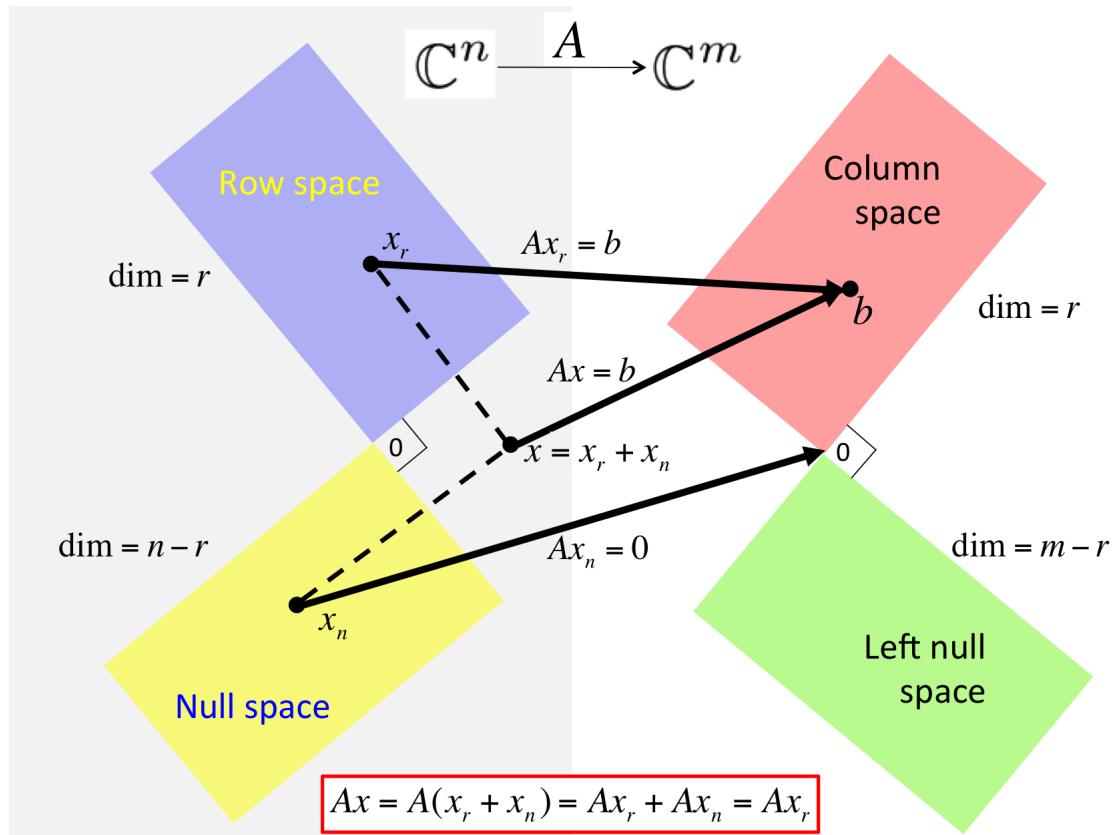
- The left null space, which is equal to the set of all vectors

$$\{x \mid x^H A = 0\}.$$

Notice that this set is equal to $\mathcal{N}(A^H)$.

- If $Ax = b$ then there exist $x_r \in \mathcal{R}(A)$ and $x = x_r + x_n$ where $x_r \in \mathcal{R}(A)$ and $x_n \in \mathcal{N}(A)$.

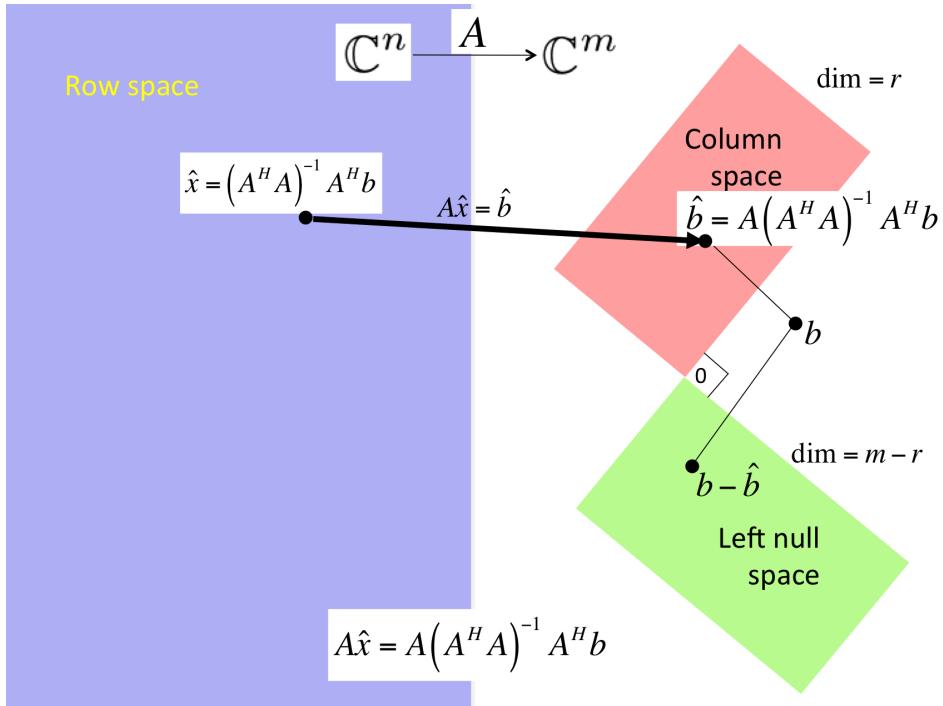
These insights are summarized in the following picture, which also captures the orthogonality of the spaces.



If A has linearly independent columns, then the solution of LLS, \hat{x} , equals the solution of the normal equations

$$(A^H A)\hat{x} = A^H b.$$

as summarized in



The (left) pseudo inverse of A is given by $A^\dagger = (A^H A)^{-1} A^H$ so that the solution of LLS is given by $\hat{x} = A^\dagger b$.

Definition 4.6.2.1 Condition number of matrix with linearly independent columns. Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns (and hence $n \leq m$). Then its condition number (with respect to the 2-norm) is defined by

$$\kappa_2(A) = \|A\|_2 \|A^\dagger\|_2 = \frac{\sigma_0}{\sigma_{n-1}}.$$

◊

Assuming A has linearly independent columns, let $\hat{b} = A\hat{x}$ where \hat{b} is the projection of b onto the column space of A (in other words, \hat{x} solves the LLS problem), $\cos(\theta) = \|\hat{b}\|_2/\|b\|_2$, and $\hat{b} + \delta\hat{b} = A(\hat{x} + \delta\hat{x})$, where $\delta\hat{b}$ equals the projection of δb onto the column space of A . Then

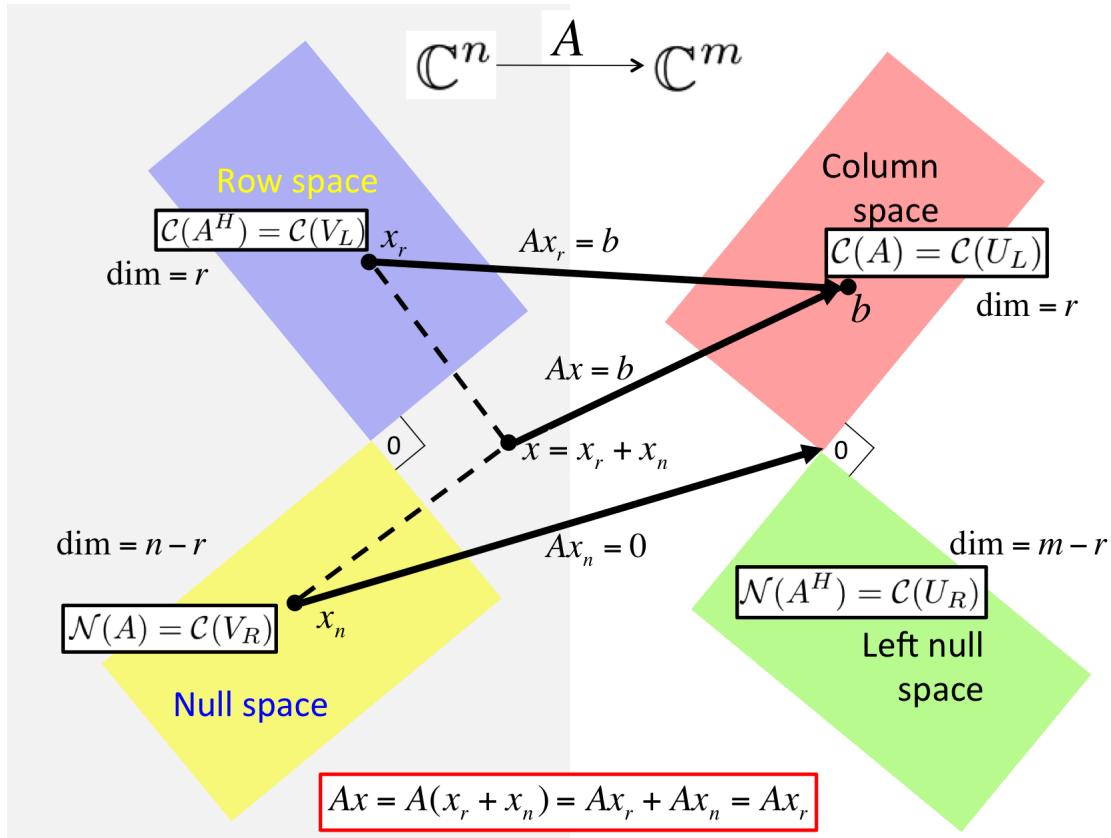
$$\frac{\|\delta\hat{x}\|_2}{\|\hat{x}\|_2} \leq \frac{1}{\cos(\theta)} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta b\|_2}{\|b\|_2}$$

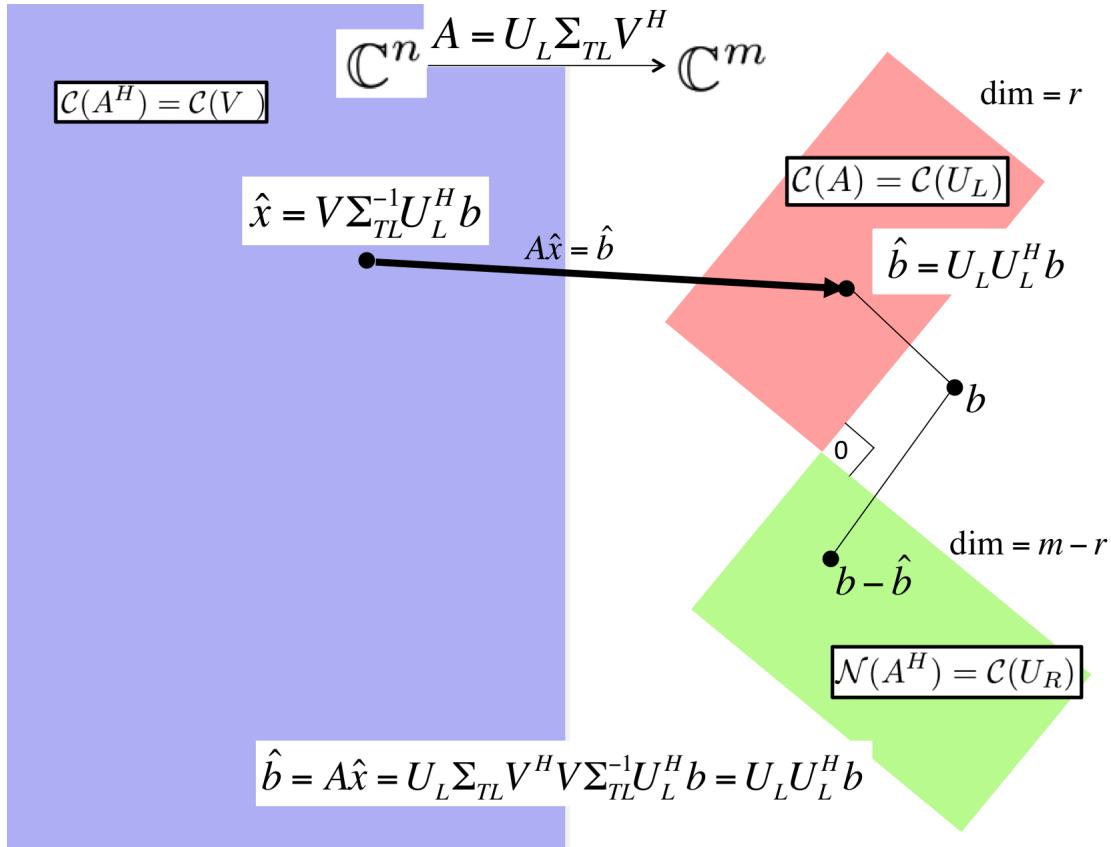
captures the sensitivity of the LLS problem to changes in the right-hand side.

Theorem 4.6.2.2 Given $A \in \mathbb{C}^{m \times n}$, let $A = U_L \Sigma_{TL} V_L^H$ equal its Reduced SVD and $A = (U_L | U_R) \begin{pmatrix} \Sigma_{TL} & 0 \\ 0 & 0 \end{pmatrix} (V_L | V_R)^H$ its SVD. Then

- $\mathcal{C}(A) = \mathcal{C}(U_L)$,
- $\mathcal{N}(A) = \mathcal{C}(V_R)$,
- $\mathcal{R}(A) = \mathcal{C}(A^H) = \mathcal{C}(V_L)$, and

- $\mathcal{N}(A^H) = \mathcal{C}(U_R)$.





If A has linearly independent columns and $A = U_L \Sigma_{TL} V_L^H$ is its Reduced SVD, then

$$\hat{x} = V_L \Sigma_{TL}^{-1} U_L^H b$$

solves LLS.

Given $A \in \mathbb{C}^{m \times n}$, let $A = U_L \Sigma_{TL} V_L^H$ equal its Reduced SVD and $A = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ 0 & 0 \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H$ its SVD. Then

$$\hat{x} = V_L \Sigma_{TL} U_L^H b + V_R z_b,$$

is the general solution to LLS, where z_b is any vector in \mathbb{C}^{n-r} .

Theorem 4.6.2.3 Assume $A \in \mathbb{C}^{m \times n}$ has linearly independent columns and let $A = QR$ be its QR factorization with orthonormal matrix $Q \in \mathbb{C}^{m \times n}$ and upper triangular matrix $R \in \mathbb{C}^{n \times n}$. Then the LLS problem

$$\text{Find } \hat{x} \in \mathbb{C}^n \text{ such that } \|b - A\hat{x}\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2$$

is solved by the unique solution of

$$R\hat{x} = Q^H b.$$

Solving LLS via Gram-Schmidt QR factorization for $A \in \mathbb{C}^{m \times n}$:

- Compute QR factorization via (Classical or Modified) Gram-Schmidt: approximately $2mn^2$ flops.

- Compute $y = Q^H b$: approximately $2mn^2$ flops.
- Solve $R\hat{x} = y$: approximately n^2 flops.

Solving LLS via Householder QR factorization for $A \in \mathbb{C}^{m \times n}$:

- Householder QR factorization: approximately $2mn^2 - \frac{2}{3}n^3$ flops.
- Compute $y_T = Q^H bnn$ by applying Householder transformations: approximately $4mn - 2n^2$ flops.
- Solve $R_{TL}\hat{x} = y_T$: approximately n^2 flops.

Part II

Solving Linear Systems

Week 5

The LU and Cholesky Factorizations

5.1 Opening

5.1.1 Of Gaussian elimination and LU factorization



YouTube: <https://www.youtube.com/watch?v=fszE2KNxTmo>

Homework 5.1.1.1 Reduce the appended system

$$\begin{array}{ccc|c} 2 & -1 & 1 & 1 \\ -2 & 2 & 1 & -1 \\ 4 & -4 & 1 & 5 \end{array}$$

to upper triangular form, overwriting the zeroes that are introduced with the multipliers.

Solution.

$$\begin{array}{ccc|c} 2 & -1 & 1 & 1 \\ -1 & 1 & 2 & 0 \\ 2 & -2 & 3 & 3 \end{array}$$



YouTube: <https://www.youtube.com/watch?v=Tt00Qikd-nI>

$A = LU(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & a_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$a_{21} := a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{12}^T$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & a_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$
endwhile

Figure 5.1.1.1 Algorithm that overwrites A with its LU factorization.

Homework 5.1.1.2 The execution of the LU factorization algorithm with

$$A = \begin{pmatrix} 2 & -1 & 1 \\ -2 & 2 & 1 \\ 4 & -4 & 1 \end{pmatrix}$$

in the video overwrites A with

$$\begin{pmatrix} 2 & -1 & 1 \\ -1 & 1 & 2 \\ 2 & -2 & 3 \end{pmatrix}.$$

Multiply the L and U stored in that matrix and compare the result with the original matrix, let's call it \hat{A} .

Solution.

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & -2 & 1 \end{pmatrix} \text{ and } U = \begin{pmatrix} 2 & -1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 3 \end{pmatrix}.$$

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & -2 & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 2 & -1 & 1 \\ -2 & 2 & 1 \\ 4 & -4 & 1 \end{pmatrix} = \hat{A}.$$

5.1.2 Overview

- 5.1 Opening
 - 5.1.1 Of Gaussian elimination and LU factorization
 - 5.1.2 Overview
 - 5.1.3 What you will learn
- 5.2 From Gaussian elimination to LU factorization
 - 5.2.1 Gaussian elimination
 - 5.2.2 LU factorization: The right-looking algorithm
 - 5.2.3 Existence of the LU factorization
 - 5.2.4 Gaussian elimination via Gauss transforms
- 5.3 LU factorization with (row) pivoting
 - 5.3.1 Gaussian elimination with row exchanges
 - 5.3.2 Permutation matrices
 - 5.3.3 LU factorization with partial pivoting
 - 5.3.4 Solving $A x = y$ via LU factorization with pivoting
 - 5.3.5 Solving with a triangular matrix
 - 5.3.6 LU factorization with complete pivoting
 - 5.3.7 Improving accuracy via iterative refinement
- 5.4 Cholesky factorization
 - 5.4.1 Hermitian Positive Definite matrices
 - 5.4.2 The Cholesky Factorization Theorem
 - 5.4.3 Cholesky factorization algorithm (right-looking variant)
 - 5.4.4 Proof of the Cholesky Factorizaton Theorem
 - 5.4.5 Cholesky factorization and solving LLS
 - 5.4.6 Implementation with the classical BLAS
- 5.5 Enrichments
 - 5.5.1 Other LU factorization algorithms
- 5.6 Wrap Up
 - 5.6.1 Additional homework
 - 5.6.2 Summary

5.1.3 What you will learn

This week is all about solving nonsingular linear systems via LU (with or without pivoting) and Cholesky factorization. In practice, solving $Ax = b$ is *not* accomplished by forming the inverse explicitly and then computing $x = A^{-1}b$. Instead, the matrix A is factored into the product of triangular matrices and it is these triangular matrices that are employed to solve the system. This requires fewer computations.

Upon completion of this week, you should be able to

- Link Gaussian elimination to LU factorization.
- View LU factorization in different ways: as Gaussian elimination, as the application of a sequence of Gauss transforms, and the operation that computes L and U such that $A = LU$.
- State and prove necessary conditions for the existence of the LU factorization.
- Extend the ideas behind Gaussian elimination and LU factorization to include pivoting.
- Derive different algorithms for LU factorization and for solving the resulting triangular systems.
- Employ the LU factorization, with or without pivoting, to solve $Ax = b$.
- Identify, prove, and apply properties of Hermitian Positive Definite matrices.
- State and prove conditions related to the existence of the Cholesky factorization.
- Derive Cholesky factorization algorithms.
- Analyze the cost of the different factorization algorithms and related algorithms for solving triangular systems.

5.2 From Gaussian elimination to LU factorization

5.2.1 Gaussian elimination



YouTube: <https://www.youtube.com/watch?v=UdN0W8Czj8c>

Homework 5.2.1.1 Solve

$$\begin{pmatrix} 2 & -1 & 1 \\ -4 & 0 & 1 \\ 4 & 0 & -2 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} -6 \\ 2 \\ 0 \end{pmatrix}.$$

Answer.

$$\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ -2 \end{pmatrix}.$$

Solution. We employ Gaussian elimination applied to an appended system:

-

$$\left(\begin{array}{ccc|c} 2 & -1 & 1 & -6 \\ -4 & 0 & 1 & 2 \\ 4 & 0 & -2 & 0 \end{array} \right)$$

- Compute the multiplier $\lambda_{10} = (-4)/(2) = -2$
- Subtract $\lambda_{10} = -2$ times the first row from the second row, yielding

$$\left(\begin{array}{ccc|c} 2 & -1 & 1 & -6 \\ 0 & -2 & 3 & -10 \\ 4 & 0 & -2 & 0 \end{array} \right)$$

- Compute the multiplier $\lambda_{20} = (4)/(2) = 2$
- Subtract $\lambda_{20} = 2$ times the first row from the third row, yielding

$$\left(\begin{array}{ccc|c} 2 & -1 & 1 & -6 \\ 0 & -2 & 3 & -10 \\ 0 & 2 & -4 & 12 \end{array} \right)$$

- Compute the multiplier $\lambda_{21} = (2)/(-2) = -1$
- Subtract $\lambda_{21} = -1$ times the second row from the third row, yielding

$$\left(\begin{array}{ccc|c} 2 & -1 & 1 & -6 \\ 0 & -2 & 3 & -10 \\ 0 & 0 & -1 & 2 \end{array} \right)$$

- Solve the triangular system

$$\begin{pmatrix} 2 & -1 & 1 \\ 0 & -2 & 3 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} -6 \\ -10 \\ 2 \end{pmatrix}$$

to yield

$$\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ -2 \end{pmatrix}.$$

The exercise in [Homework 5.2.1.1](#) motivates the following algorithm, which reduces the linear system $Ax = b$ stored in $n \times n$ matrix A and right-hand side vector b of size n to an upper triangular system.

```

for  $j := 0, \dots, n - 1$ 
  for  $i := j + 1, \dots, n - 1$ 
     $\lambda_{i,j} := \alpha_{i,j}/\alpha_{j,j}$ 
     $\alpha_{i,j} := 0$ 
    for  $k = j + 1, \dots, n - 1$ 
       $\alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j}\alpha_{j,k}$ 
    endfor
     $\beta_i := \beta_i - \lambda_{i,j}\beta_j$ 
  endfor
endfor

```

$\left. \begin{array}{l} \alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j}\alpha_{j,k} \\ \beta_i := \beta_i - \lambda_{i,j}\beta_j \end{array} \right\}$ subtract $\lambda_{i,j}$ times row j from row k

This algorithm completes as long as no divide by zero is encountered.

Let us manipulate this a bit. First, we notice that we can first reduce the matrix to an upper triangular matrix, and then update the right-hand side using the multipliers that were computed along the way (if these are stored):

```

reduce  $A$  to upper triangular form
for  $j := 0, \dots, n - 1$ 
  for  $i := j + 1, \dots, n - 1$ 
     $\lambda_{i,j} := \alpha_{i,j}/\alpha_{j,j}$ 
     $\alpha_{i,j} := 0$ 
    for  $k = j + 1, \dots, n - 1$ 
       $\alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j}\alpha_{j,k}$ 
    endfor
  endfor
endfor

```

update b using multipliers (forward substitution)

```

for  $j := 0, \dots, n - 1$ 
  for  $i := j + 1, \dots, n - 1$ 
     $\beta_i := \beta_i - \lambda_{i,j}\beta_j$ 
  endfor
endfor

```

Ignoring the updating of the right-hand side (a process known as forward substitution), for

each iteration we can first compute the multipliers and then update the matrix:

```

for  $j := 0, \dots, n - 1$ 
  for  $i := j + 1, \dots, n - 1$ 
     $\lambda_{i,j} := \alpha_{i,j}/\alpha_{j,j}$ 
     $\alpha_{i,j} := 0$ 
  endfor
  } compute multipliers
  for  $i := j + 1, \dots, n - 1$ 
    for  $k = j + 1, \dots, n - 1$ 
       $\alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j}\alpha_{j,k}$ 
    endfor
  } subtract  $\lambda_{i,j}$  times row  $j$  from row  $k$ 
  endfor
endfor

```

Since we know that $\alpha_{i,j}$ is set to zero, we can use its location to store the multiplier:

```

for  $j := 0, \dots, n - 1$ 
  for  $i := j + 1, \dots, n - 1$ 
     $\alpha_{i,j} := \lambda_{i,j} = \alpha_{i,j}/\alpha_{j,j}$ 
  endfor
  } compute all multipliers
  for  $i := j + 1, \dots, n - 1$ 
    for  $k = j + 1, \dots, n - 1$ 
       $\alpha_{i,k} := \alpha_{i,k} - \alpha_{i,j}\alpha_{j,k}$ 
    endfor
  } subtract  $\lambda_{i,j}$  times row  $j$  from row  $k$ 
  endfor
endfor

```

Finally, we can cast the computation in terms of operations with vectors and submatrices:

```

for  $j := 0, \dots, n - 1$ 
  
$$\begin{pmatrix} \alpha_{j+1,j} \\ \vdots \\ \alpha_{n-1,j} \end{pmatrix} := \begin{pmatrix} \lambda_{j+1,j} \\ \vdots \\ \lambda_{n-1,j} \end{pmatrix} / \alpha_{j,j}$$

  
$$\begin{pmatrix} \alpha_{j+1,j+1} & \cdots & \alpha_{j+1,n-1} \\ \vdots & & \vdots \\ \alpha_{n-1,j+1} & \cdots & \alpha_{n-1,n-1} \end{pmatrix} :=$$

  
$$\begin{pmatrix} \alpha_{j+1,j+1} & \cdots & \alpha_{j+1,n-1} \\ \vdots & & \vdots \\ \alpha_{n-1,j+1} & \cdots & \alpha_{n-1,n-1} \end{pmatrix} - \begin{pmatrix} \alpha_{j+1,j} \\ \vdots \\ \alpha_{n-1,j} \end{pmatrix} \begin{pmatrix} \alpha_{j,j+1} & \cdots & \alpha_{j,n-1} \end{pmatrix}$$

endfor

```

In Figure 5.2.1.1 this algorithm is presented with our FLAME notation.

$A = \text{GE}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$a_{21} := l_{21} = a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{12}^T$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$
endwhile

Figure 5.2.1.1 Gaussian elimination algorithm that reduced a matrix A to upper triangular form, storing the multipliers below the diagonal.

Homework 5.2.1.2 Apply the algorithm [Figure 5.2.1.1](#) to the matrix

$$\begin{pmatrix} 2 & -1 & 1 \\ -4 & 0 & 1 \\ 4 & 0 & -2 \end{pmatrix}$$

and report the resulting matrix. Compare the contents of that matrix to the upper triangular matrix computed in the solution of [Homework 5.2.1.1](#).

Answer.

$$\begin{pmatrix} 2 & -1 & 1 \\ -2 & -2 & 3 \\ 2 & -1 & -1 \end{pmatrix}$$

Solution. Partition:

$$\left(\begin{array}{ccc} 2 & -1 & 1 \\ -4 & 0 & 1 \\ 4 & 0 & -2 \end{array} \right)$$

- First iteration:

◦ $\alpha_{21} := \lambda_{21} = a_{21}/\alpha_{11}$:

$$\left(\begin{array}{ccc} 2 & -1 & 1 \\ -2 & 0 & 1 \\ 2 & 0 & -2 \end{array} \right)$$

◦ $A_{22} := A_{22} - a_{21}a_{12}^T$:

$$\left(\begin{array}{ccc} 2 & -1 & 1 \\ -2 & -2 & 3 \\ 2 & 2 & -4 \end{array} \right)$$

- State at bottom of iteration:

$$\left(\begin{array}{c|cc} 2 & -1 & 1 \\ \hline -2 & -2 & 3 \\ 2 & 2 & -4 \end{array} \right)$$

- Second iteration:

- $\alpha_{21} := \lambda_{21} = \alpha_{21}/\alpha_{11}$:

$$\left(\begin{array}{c|cc} 2 & -1 & 1 \\ \hline -2 & -2 & 3 \\ 2 & -1 & -4 \end{array} \right)$$

- $A_{22} := A_{22} - a_{21}a_{12}^T$:

$$\left(\begin{array}{c|cc} 2 & -1 & 1 \\ \hline -2 & -2 & 3 \\ 2 & -1 & -1 \end{array} \right)$$

- State at bottom of iteration:

$$\left(\begin{array}{c|cc} 2 & -1 & 1 \\ \hline -2 & -2 & 3 \\ 2 & -1 & -1 \end{array} \right)$$

- Third iteration:

- $\alpha_{21} := \lambda_{21} = \alpha_{21}/\alpha_{11}$:

$$\left(\begin{array}{c|cc} 2 & -1 & 1 \\ \hline -2 & -2 & 3 \\ 2 & -1 & -1 \end{array} \right)$$

(computation with empty vector).

- $A_{22} := A_{22} - a_{21}a_{12}^T$:

$$\left(\begin{array}{c|cc} 2 & -1 & 1 \\ \hline -2 & -2 & 3 \\ 2 & -1 & -1 \end{array} \right)$$

(update of empty matrix)

- State at bottom of iteration:

$$\left(\begin{array}{ccc} 2 & -1 & 1 \\ -2 & -2 & 3 \\ 2 & -1 & -1 \end{array} \right)$$

The upper triangular matrix computed in [Homework 5.2.1.1](#) was

$$\left(\begin{array}{ccc} 2 & -1 & 1 \\ 0 & -2 & 3 \\ 0 & 0 & -1 \end{array} \right)$$

which can be found in the upper triangular part of the updated matrix A .

Homework 5.2.1.3 Applying Figure 5.2.1.1 to the matrix

$$A = \begin{pmatrix} 2 & -1 & 1 \\ -4 & 0 & 1 \\ 4 & 0 & -2 \end{pmatrix}$$

yielded

$$\begin{pmatrix} 2 & -1 & 1 \\ -2 & -2 & 3 \\ 2 & -1 & -1 \end{pmatrix}.$$

This can be thought of as an array that stores the unit lower triangular matrix L below the diagonal (with implicit ones on its diagonal) and upper triangular matrix U on and above its diagonal:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 2 & -1 & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 2 & -1 & 1 \\ 0 & -2 & 3 \\ 0 & 0 & -1 \end{pmatrix}$$

Compute $B = LU$ and compare it to A .

Answer. Magic! $B = A$!

Solution.

$$B = LU = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 2 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 & 1 \\ 0 & -2 & 3 \\ 0 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 2 & -1 & 1 \\ -4 & 0 & 1 \\ 4 & 0 & -2 \end{pmatrix} = A.$$

5.2.2 LU factorization: The right-looking algorithm



YouTube: https://www.youtube.com/watch?v=GfpB_RU8pIo

In the launch of this week, we mentioned an algorithm that computes the LU factorization of a given matrix A so that

$$A = LU,$$

where L is a unit lower triangular matrix and U is an upper triangular matrix. We now derive that algorithm, which is often called the right-looking algorithm for computing the LU factorization.

Partition A , L , and U as follows:

$$A \rightarrow \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}, \quad L \rightarrow \begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix}, \quad \text{and} \quad U \rightarrow \begin{pmatrix} v_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix}.$$

Then $A = LU$ means that

$$\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} v_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix} = \begin{pmatrix} v_{11} & u_{12}^T \\ l_{21}v_{11} & l_{21}u_{12}^T + L_{22}U_{22} \end{pmatrix}.$$

Hence

$$\begin{aligned} \alpha_{11} &= v_{11} & a_{12}^T &= u_{12}^T \\ a_{21} &= v_{11}l_{21} & A_{22} &= l_{21}u_{12}^T + L_{22}U_{22} \end{aligned}$$

or, equivalently,

$$\begin{aligned} \alpha_{11} &= v_{11} & a_{12}^T &= u_{12}^T \\ a_{21} &= v_{11}l_{21} & A_{22} - l_{21}u_{12}^T &= L_{22}U_{22}. \end{aligned}$$

If we overwrite the upper triangular part of A with U and the strictly lower triangular part of A with the strictly lower triangular part of L (since we know that its diagonal consists of ones), we deduce that we must perform the computations

- $a_{21} := l_{21} = a_{21}/\alpha_{11}$.
- $A_{22} := A_{22} - l_{21}a_{12}^T = A_{22} - a_{21}a_{12}^T$.
- Continue by computing the LU factorization of the updated A_{22} .

The resulting algorithm is given in [Figure 5.2.2.1](#).

$A = \text{LU-right-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$\underline{a_{21} := a_{21}/\alpha_{11}}$
$\underline{A_{22} := A_{22} - a_{21}a_{12}^T}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.2.2.1 Right-looking LU factorization algorithm.

Before we discuss the cost of this algorithm, let us discuss a trick that is often used in the analysis of the cost of algorithms in linear algebra. We can approximate sums with integrals:

$$\sum_{k=0}^{n-1} k^p \approx \int_0^n x^p dx = \frac{1}{p+1} x^{p+1} \Big|_0^n = \frac{1}{p+1} n^{p+1}.$$

Homework 5.2.2.1 Give the approximate cost incurred by the algorithm in Figure 5.2.2.1 when applied to an $n \times n$ matrix.

Answer. Approximately $\frac{2}{3}n^3$ flops.

Solution. Consider the iteration where A_{TL} is (initially) $k \times k$. Then

- a_{21} is of size $n - k - 1$. Thus $a_{21} := a_{21}/\alpha_{11}$ is typically computed by first computing $1/\alpha_{11}$ and then $a_{21} := (1/\alpha_{11})a_{21}$, which requires $(n - k - 1)$ flops. (The cost of computing $1/\alpha_{11}$ is inconsequential when n is large, so it is usually ignored.)
- A_{22} is of size $(n - k - 1) \times (n - k - 1)$ and hence the rank-1 update $A_{22} := A_{22} - a_{21}a_{12}^T$ requires $2(n - k - 1)(n - k - 1)$ flops.

Now, the cost of updating a_{21} is small relative to that of the update of A_{22} and hence will be ignored. Thus, the total cost is given by, approximately,

$$\sum_{k=0}^{n-1} 2(n - k - 1)^2 \text{ flops.}$$

Let us now simplify this:

$$\begin{aligned} & \sum_{k=0}^{n-1} 2(n - k - 1)^2 \\ &= < \text{change of variable: } j = n - k - 1 > \\ & \sum_{j=0}^{n-1} 2j^2 \\ &= < \text{algebra} > \\ & 2 \sum_{j=0}^{n-1} j^2 \\ &\approx < \sum_{j=0}^{n-1} j^2 \approx \int_0^n x^2 dx = n^3/3 > \\ & \frac{2}{3}n^3 \end{aligned}$$

Homework 5.2.2.2 Give the approximate cost incurred by the algorithm in Figure 5.2.2.1 when applied to an $m \times n$ matrix.

Answer. Approximately $mn^2 - \frac{1}{3}n^3$ flops.

Solution. Consider the iteration where A_{TL} is (initially) $k \times k$. Then

- a_{21} is of size $m - k - 1$. Thus $a_{21} := a_{21}/\alpha_{11}$ is typically computed by first computing $1/\alpha_{11}$ and then $a_{21} := (1/\alpha_{11})a_{21}$, which requires $(m - k - 1)$ flops. (The cost of computing $1/\alpha_{11}$ is inconsequential when m is large.)
- A_{22} is of size $(m - k - 1) \times (n - k - 1)$ and hence the rank-1 update $A_{22} := A_{22} - a_{21}a_{12}^T$ requires $2(m - k - 1)(n - k - 1)$ flops.

Now, the cost of updating a_{21} is small relative to that of the update of A_{22} and hence will be ignored. Thus, the total cost is given by, approximately,

$$\sum_{k=0}^{n-1} 2(m - k - 1)(n - k - 1) \text{ flops.}$$

Let us now simplify this:

$$\begin{aligned}
 & \sum_{k=0}^{n-1} 2(m-k-1)(n-k-1) \\
 & = \quad < \text{change of variable: } j = n - k - 1 > \\
 & \sum_{j=0}^{n-1} 2(m-(n-j-1)-1)j \\
 & = \quad < \text{simplify} > \\
 & \sum_{j=0}^{n-1} 2(m-n+j)j \\
 & = \quad < \text{algebra} > \\
 & 2(m-n) \sum_{j=0}^{n-1} j + 2 \sum_{j=0}^{n-1} j^2 \\
 & \approx \quad < \sum_{j=0}^{n-1} j \approx n^2/2 \text{ and } \sum_{j=0}^{n-1} j^2 \approx n^3/3 > \\
 & (m-n)n^2 + \frac{2}{3}n^3 \\
 & = \quad < \text{simplify} > \\
 & mn^2 - \frac{1}{3}n^3
 \end{aligned}$$

Remark 5.2.2.2 In a practical application of LU factorization, it is uncommon to factor a non-square matrix. However, high-performance implementations of the LU factorization that use "blocked" algorithms perform a factorization of a rectangular submatrix of A , which is why we generalize beyond the square case.

Homework 5.2.2.3 It is a good idea to perform a "git pull" in the Assignments directory to update with the latest files before you start new programming assignments.

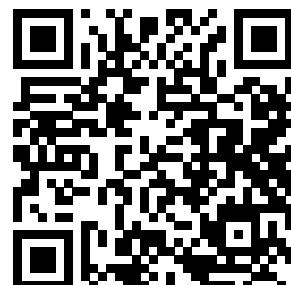
Implement the algorithm given in [Figure 5.2.2.1](#) as

```
function [ A_out ] = LU_right_looking( A )
```

by completing the code in [Assignments/Week05/matlab/LU_right_looking.m](#). Input is an $m \times n$ matrix A . Output is the matrix A that has been overwritten by the LU factorization. You may want to use [Assignments/Week05/matlab/test_LU_right_looking.m](#) to check your implementation.

Solution. See [Assignments/Week05/answers/LU_right_looking.m](#). ([Assignments/Week05/answers/LU_right_looking.m](#))

5.2.3 Existence of the LU factorization



YouTube: <https://www.youtube.com/watch?v=Aaa9n97N1qc>

Now that we have an algorithm for computing the LU factorization, it is time to talk about when this LU factorization exists (in other words: when we can guarantee that the algorithm completes).

We would like to talk about the existence of the LU factorization for the more general case where A is an $m \times n$ matrix, with $m \geq n$. What does this mean?

Definition 5.2.3.1 Given a matrix $A \in \mathbb{C}^{m \times n}$ with $m \geq n$, its LU factorization is given by $A = LU$ where $L \in \mathbb{C}^{m \times n}$ is unit lower trapezoidal and $U \in \mathbb{C}^{n \times n}$ is upper triangular with nonzeros on its diagonal. \diamond

The first question we will ask is when the LU factorization exists. For this, we need another definition.

Definition 5.2.3.2 Principal leading submatrix. For $k \leq n$, the $k \times k$ principal leading submatrix of a matrix A is defined to be the square matrix $A_{TL} \in \mathbb{C}^{k \times k}$ such that $A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$. \diamond

This definition allows us to state necessary and sufficient conditions for when a matrix with n linearly independent columns has an LU factorization:

Lemma 5.2.3.3 Let $L \in \mathbb{C}^{n \times n}$ be a unit lower triangular matrix and $U \in \mathbb{C}^{n \times n}$ be an upper triangular matrix. Then $A = LU$ is nonsingular if and only if U has no zeroes on its diagonal.

Homework 5.2.3.1 Prove Lemma 5.2.3.3.

Hint. You may use the fact that a triangular matrix has an inverse if and only if it has no zeroes on its diagonal.

Solution. The proof hinges on the fact that a triangular matrix is nonsingular if and only if it doesn't have any zeroes on its diagonal. Hence we can instead prove that $A = LU$ is nonsingular if and only if U is nonsingular (since L is unit lower triangular and hence has no zeroes on its diagonal).

- (\Rightarrow): Assume $A = LU$ is nonsingular. Since L is nonsingular, $U = L^{-1}A$. We can show that U is nonsingular in a number of ways:
 - We can explicitly give its inverse:

$$U(A^{-1}L) = L^{-1}AA^{-1}L = I.$$

Hence U has an inverse and is thus nonsingular.

- Alternatively, we can reason that the product of two nonsingular matrices, namely L^{-1} and A , is nonsingular.

- (\Leftarrow): Assume $A = LU$ and U has no zeroes on its diagonal. We then know that both L^{-1} and U^{-1} exist. Again, we can either explicitly verify a known inverse of A :

$$A(U^{-1}L^{-1}) = LUU^{-1}L^{-1} = I$$

or we can recall that the product of two nonsingular matrices, namely U^{-1} and L^{-1} , is nonsingular.

Theorem 5.2.3.4 Existence of the LU factorization. Let $A \in \mathbb{C}^{m \times n}$ and $m \geq n$ have linearly independent columns. Then A has a (unique) LU factorization if and only if all its principal leading submatrices are nonsingular.



YouTube: <https://www.youtube.com/watch?v=SPlE5xJF9hY>

Proof.

- (\Rightarrow): Let nonsingular A have a (unique) LU factorization. We will show that its principal leading submatrices are nonsingular.

Let

$$\underbrace{\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)}_A = \underbrace{\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right)}_L \underbrace{\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right)}_U$$

be the LU factorization of A , where $A_{TL}, L_{TL}, U_{TL} \in \mathbb{C}^{k \times k}$. By the assumption that LU is the LU factorization of A , we know that U cannot have a zero on the diagonal and hence is nonsingular. Now, since

$$\begin{aligned} \underbrace{\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)}_A &= \underbrace{\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right)}_L \underbrace{\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right)}_U \\ &= \left(\begin{array}{c|c} L_{TL}U_{TL} & L_{TL}U_{TR} \\ \hline L_{BL}U_{TL} & L_{BL}U_{TR} + L_{BL}U_{BR} \end{array} \right), \end{aligned}$$

the $k \times k$ principal leading submatrix A_{TL} equals $A_{TL} = L_{TL}U_{TL}$, which is nonsingular since L_{TL} has a unit diagonal and U_{TL} has no zeroes on the diagonal. Since k was chosen arbitrarily, this means that all principal leading submatrices are nonsingular.

- (\Leftarrow): We will do a proof by induction on n .

- Base Case: $n = 1$. Then A has the form $A = \begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}$ where α_{11} is a scalar.

Since the principal leading submatrices are nonsingular $\alpha_{11} \neq 0$. Hence $A = \underbrace{\begin{pmatrix} 1 \\ a_{21}/\alpha_{11} \end{pmatrix}}_L \underbrace{\alpha_{11}}_U$ is the LU factorization of A . This LU factorization is unique

because the first element of L must be 1.

- Inductive Step: Assume the result is true for all matrices with $n = k$. Show it is true for matrices with $n = k + 1$.

Let A of size $n = k + 1$ have nonsingular principal leading submatrices. Now, if an LU factorization of A exists, $A = LU$, then it would have to form

$$\underbrace{\left(\begin{array}{c|c} A_{00} & a_{01} \\ \hline a_{10}^T & \alpha_{11} \\ A_{20} & a_{21} \end{array} \right)}_A = \underbrace{\left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \\ L_{20} & l_{21} \end{array} \right)}_L \underbrace{\left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & v_{11} \end{array} \right)}_U. \quad (5.2.1)$$

If we can show that the different parts of L and U exist, are unique, and $v_{11} \neq 0$, we are done (since then U is nonsingular). (5.2.1) can be rewritten as

$$\left(\begin{array}{c} A_{00} \\ \hline a_{10}^T \\ A_{20} \end{array} \right) = \left(\begin{array}{c} L_{00} \\ \hline l_{10}^T \\ L_{20} \end{array} \right) U_{00} \text{ and } \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ a_{21} \end{array} \right) = \left(\begin{array}{c} L_{00} u_{01} \\ \hline l_{10}^T u_{01} + v_{11} \\ L_{20} u_{01} + l_{21} v_{11} \end{array} \right),$$

or, equivalently,

$$\begin{cases} L_{00} u_{01} = a_{01} \\ v_{11} = \alpha_{11} - l_{10}^T u_{01} \\ l_{21} = (a_{21} - L_{20} u_{01})/v_{11} \end{cases}$$

Now, by the Inductive Hypothesis L_{00} , l_{10}^T , and L_{20} exist and are unique. So the question is whether u_{01} , v_{11} , and l_{21} exist and are unique:

- u_{01} exists and is unique. Since L_{00} is nonsingular (it has ones on its diagonal) $L_{00}u_{01} = a_{01}$ has a solution that is unique.
- v_{11} exists, is unique, and is nonzero. Since l_{10}^T and u_{01} exist and are unique, $v_{11} = \alpha_{11} - l_{10}^T u_{01}$ exists and is unique. It is also nonzero since the principal leading submatrix of A given by

$$\left(\begin{array}{c|c} A_{00} & a_{01} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & v_{11} \end{array} \right),$$

is nonsingular by assumption and therefore v_{11} must be nonzero.

- l_{21} exists and is unique. Since v_{11} exists, is unique, and is nonzero,

$$l_{21} = (a_{21} - L_{20}a_{01})/v_{11}$$

exists and is uniquely determined.

Thus the $m \times (k + 1)$ matrix A has a unique LU factorization.

- By the Principal of Mathematical Induction the result holds.

■

The formulas in the inductive step of the proof of Theorem 5.2.3.4 suggest an alternative

algorithm for computing the LU factorization of a $m \times n$ matrix A with $m \geq n$, given in Figure 5.2.3.5. This algorithm is often referred to as the (unblocked) left-looking algorithm.

$A = \text{LU-left-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
Solve $L_{00}u_{01} = a_{01}$ overwriting a_{01} with u_{01}
$\alpha_{11} := v_{11} = \alpha_{11} - a_{10}^T a_{01}$
$a_{21} := a_{21} - A_{20}a_{01}$
$a_{21} := l_{21} = a_{21}/\alpha_{11}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.2.3.5 Left-looking LU factorization algorithm. L_{00} is the unit lower triangular matrix stored in the strictly lower triangular part of A_{00} (with the diagonal implicitly stored).

Homework 5.2.3.2 Show that if the left-looking algorithm in Figure 5.2.3.5 is applied to an $m \times n$ matrix, with $m \geq n$, the cost is approximately $mn^2 - \frac{1}{3}n^3$ flops (just like the right-looking algorithm).

Solution. Consider the iteration where A_{TL} is (initially) $k \times k$. Then

- Solving $L_{00}u_{01} = a_{21}$ requires approximately k^2 flops.
- Updating $\alpha_{11} := \alpha_{11} - a_{10}^T a_{01}$ requires approximately $2k$ flops, which we will ignore.
- Updating $a_{21} := a_{21} - A_{20}a_{01}$ requires approximately $2(m - k - 1)k$ flops.
- Updating $a_{21} := a_{21}/\alpha_{11}$ requires approximately $(m - k - 1)$ flops, which we will ignore.

Thus, the total cost is given by, approximately,

$$\sum_{k=0}^{n-1} (k^2 + 2(m - k - 1)k) \text{ flops.}$$

Let us now simplify this:

$$\begin{aligned}
 & \sum_{k=0}^{n-1} (k^2 + 2(m-k-1)k) \\
 & = \quad < \text{algebra} > \\
 & \sum_{k=0}^{n-1} k^2 + 2 \sum_{k=0}^{n-1} (m-k-1)k \\
 & = \quad < \text{algebra} > \\
 & \sum_{k=0}^{n-1} 2(m-1)k - \sum_{k=0}^{n-1} k^2 \\
 & \approx \quad < \sum_{j=0}^{n-1} j \approx n^2/2 \text{ and } \sum_{j=0}^{n-1} j^2 \approx n^3/3 > \\
 & (m-1)n^2 - \frac{1}{3}n^3
 \end{aligned}$$

Had we not ignored the cost of $\alpha_{11} := \alpha_{11} - a_{10}^T a_{01}$, which approximately $2k$, then the result would have been approximately

$$mn^2 - \frac{1}{3}n^3$$

instead of $(m-1)n^2 - \frac{1}{3}n^3$, which is identical to that of the right-looking algorithm in [Figure 5.2.2.1](#). This makes sense, since the two algorithms perform the same operations in a different order.

Of course, regardless,

$$(m-1)n^2 - \frac{1}{3}n^3 \approx mn^2 - \frac{1}{3}n^3$$

if m is large.

Remark 5.2.3.6 A careful analysis would show that the left- and right-looking algorithms perform the exact same operations with the same elements of A , except in a different order. Thus, it is no surprise that the costs of these algorithms are the same.

Ponder This 5.2.3.3 If A is $m \times m$ (square!), then yet another algorithm can be derived by partitioning A , L , and U so that

$$A = \begin{pmatrix} A_{00} & a_{01} \\ a_{10}^T & \alpha_{11} \end{pmatrix}, L = \begin{pmatrix} L_{00} & 0 \\ l_{10}^T & 1 \end{pmatrix}, U = \begin{pmatrix} U_{00} & u_{01} \\ 0 & v_{11} \end{pmatrix}.$$

Assume that L_{00} and U_{00} have already been computed in previous iterations, and determine

how to compute u_{01} , l_{10}^T , and v_{11} in the current iteration. Then fill in the algorithm:

$A = \text{LU-bordered}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & a_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & a_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$

This algorithm is often called the *bordered* LU factorization algorithm.

Next, modify the proof of [Theorem 5.2.3.4](#) to show the existence of the LU factorization *when A is square* and has nonsingular leading principal submatrices.

Finally, show that this bordered algorithm also requires approximately $2m^3/3$ flops.

Homework 5.2.3.4 Implement the algorithm given in [Figure 5.2.3.5](#) as
function [A_{out}] = LU_left_looking(A)

by completing the code in [Assignments/Week05/matlab/LU_left_looking.m](#). Input is an $m \times n$ matrix A . Output is the matrix A that has been overwritten by the LU factorization. You may want to use [Assignments/Week05/matlab/test_LU_left_looking.m](#) to check your implementation.

Solution. See [Assignments/Week05/answers/LU_left_looking.m](#). ([Assignments/Week05/answers/LU_left_looking.m](#))

5.2.4 Gaussian elimination via Gauss transforms



YouTube: <https://www.youtube.com/watch?v=YDtynD4iAVM>

Definition 5.2.4.1 A matrix L_k of the form

$$L_k = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array} \right),$$

where I_k is the $k \times k$ identity matrix and I is an identity matrix "of appropriate size" is called a Gauss transform. \diamond

Gauss transforms, when applied to a matrix, take multiples of the row indexed with k and add these multiples to other rows. In our use of Gauss transforms to explain the LU factorization, we subtract instead:

Example 5.2.4.2 Evaluate

$$\left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & -\lambda_{21} & 1 & 0 \\ 0 & -\lambda_{31} & 0 & 1 \end{array} \right) \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \tilde{a}_2^T \\ \tilde{a}_3^T \end{pmatrix} =$$

Solution.

$$\left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & -\lambda_{21} & 1 & 0 \\ 0 & -\lambda_{31} & 0 & 1 \end{array} \right) \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \tilde{a}_2^T \\ \tilde{a}_3^T \end{pmatrix} = \left(\frac{\tilde{a}_0^T}{\tilde{a}_1^T} \right) = \left(\frac{\tilde{a}_0^T}{\tilde{a}_1^T - \left(\begin{pmatrix} \tilde{a}_2^T \\ \tilde{a}_3^T \end{pmatrix} - \begin{pmatrix} \lambda_{21} \\ \lambda_{31} \end{pmatrix} \tilde{a}_1^T \right)} \right) = \left(\frac{\tilde{a}_0^T}{\tilde{a}_1^T - \lambda_{21}\tilde{a}_1^T} \right).$$

\square

Notice the similarity with what one does in Gaussian elimination: take multiples of one row and subtracting these from other rows.

Homework 5.2.4.1 Evaluate

$$\left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right)$$

where I_k is the $k \times k$ identity matrix and A_0 has k rows. If we compute

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & \hat{a}_{21} & \hat{A}_{22} \end{array} \right) := \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right)$$

how should l_{21} be chosen if we want \hat{a}_{21} to be a zero vector?

Solution.

$$\begin{aligned}
 & \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) \\
 &= \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & -l_{21}\alpha_{11} + a_{21} & -l_{21}a_{12}^T + A_{22} \end{array} \right) \\
 &= \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} - \alpha_{11}l_{21} & A_{22} - l_{21}a_{12}^T \end{array} \right)
 \end{aligned}$$

If $l_{21} = a_{21}/\alpha_{11}$ then $\tilde{a}_{21} = a_{21} - \alpha_{11}a_{21}/\alpha_{11} = 0$.

Hopefully you notice the parallels between the computation in the last homework, and the algorithm in [Figure 5.2.1.1](#).

Now, assume that the right-looking LU factorization has proceeded to where A contains

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right),$$

where A_{00} is upper triangular (recall: it is being overwritten by U !). What we would like to do is eliminate the elements in a_{21} by taking multiples of the "current row" $(\alpha_{11} | a_{12}^T)$ and subtract these from the rest of the rows: $(a_{21} | A_{22})$ in order to introduce zeroes below α_{11} . The vehicle is an appropriately chosen Gauss transform, inspired by [Homework 5.2.4.1](#). We must determine l_{21} so that

$$\left(\begin{array}{c|cc} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right).$$

As we saw in [Homework 5.2.4.1](#), this means we must pick $l_{21} = a_{21}/\alpha_{11}$. The resulting algorithm is summarized in [Figure 5.2.4.3](#). Notice that this algorithm is, once again, identical to the algorithm in [Figure 5.2.1.1](#) (except that it does not overwrite the lower triangular matrix).

$A = \text{GE-via-Gauss-transforms}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$l_{21} := a_{21}/\alpha_{11}$
$\left. \begin{aligned} & \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) \\ & := \left(\begin{array}{c ccc} I & 0 & 0 & & \\ \hline 0 & 1 & 0 & & \\ 0 & -l_{21} & 0 & & \end{array} \right) \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) \\ & = \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right) \end{aligned} \right\} \begin{aligned} a_{21} &:= 0 \\ A_{22} &:= A_{22} - l_{21}a_{12}^T \end{aligned}$
endwhile
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$

Figure 5.2.4.3 Gaussian elimination, formulated as a sequence of applications of Gauss transforms.

Homework 5.2.4.2 Show that

$$\left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right)^{-1} = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array} \right)$$

where I_k denotes the $k \times k$ identity matrix.

Hint. To show that $B = A^{-1}$, it suffices to show that $BA = I$ (if A and B are square).

Solution.

$$\begin{aligned} & \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I_{(n-k-1) \times (n-k-1)} \end{array} \right) \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array} \right) \\ &= \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} + l_{21} & I \end{array} \right) \\ &= \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & I \end{array} \right) \end{aligned}$$

Starting with an $m \times m$ matrix A , the algorithm computes a sequence of m Gauss transforms L_0, \dots, L_{m-1} , each of the form

$$L_k = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right), \quad (5.2.2)$$

such that $L_{m-1}L_{m-2} \cdots L_1L_0A = U$. Equivalently, $A = L_0^{-1}L_1^{-1} \cdots L_{m-2}^{-1}L_{m-1}^{-1}U$, where

$$L_k^{-1} = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array} \right).$$

It is easy to show that the product of unit lower triangular matrices is itself unit lower triangular. Hence

$$L = L_0^{-1}L_1^{-1} \cdots L_{n-2}^{-1}L_{n-1}^{-1}$$

is unit lower triangular. However, it turns out that this L is particularly easy to compute, as the following homework suggests.

Homework 5.2.4.3 Let

$$\tilde{L}_{k-1} = L_0^{-1}L_1^{-1} \cdots L_{k-1}^{-1} = \left(\begin{array}{c|cc} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ L_{20} & 0 & I \end{array} \right) \quad \text{and} \quad L_k^{-1} = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array} \right),$$

where L_{00} is a $k \times k$ unit lower triangular matrix. Show that

$$\tilde{L}_k = \tilde{L}_{k-1}^{-1}L_k^{-1} = \left(\begin{array}{c|cc} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ L_{20} & l_{21} & I \end{array} \right).$$

Solution.

$$\begin{aligned} \tilde{L}_k &= L_0^{-1}L_1^{-1} \cdots L_{k-1}^{-1}L_k^{-1} = \tilde{L}_{k-1}^{-1}L_k^{-1} \\ &= \left(\begin{array}{c|cc} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ L_{20} & 0 & I \end{array} \right) \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array} \right) \\ &= \left(\begin{array}{c|cc} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ L_{20} & l_{21} & I \end{array} \right). \end{aligned}$$

What this exercise shows is that $L = L_0^{-1}L_1^{-1} \cdots L_{n-2}^{-1}L_{n-1}^{-1}$ is the triangular matrix that is created by simply placing the computed vectors l_{21} below the diagonal of a unit lower triangular matrix. This insight explains the "magic" observed in [Homework 5.2.1.3](#). We conclude that the algorithm in [Figure 5.2.1.1](#) overwrites $n \times n$ matrix A with unit lower triangular matrix L and upper triangular matrix U such that $A = LU$. This is known as the LU factorization or LU decomposition of A .

Ponder This 5.2.4.4 Let

$$L_k = \left(\begin{array}{c|cc} I_{k \times k} & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right).$$

Show that

$$\kappa_2(L_k) \geq \|l_{21}\|_2^2.$$

What does this mean about how error in A may be amplified if the pivot (the α_{11} by which entries in a_{21} are divided to compute l_{21}) encountered in the right-looking LU factorization algorithm is small in magnitude relative to the elements below it? How can we choose which row to swap so as to minimize $\|l_{21}\|_2$?

Hint. Revisit [Homework 1.3.5.5](#).

5.3 LU factorization with (row) pivoting

5.3.1 Gaussian elimination with row exchanges

!



YouTube: <https://www.youtube.com/watch?v=t6cK75IE6d8>

Homework 5.3.1.1 Perform Gaussian elimination as explained in [Subsection 5.2.1](#) to solve

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Solution. The appended system is given by

$$\left(\begin{array}{cc|c} 0 & 1 & 2 \\ 1 & 0 & 1 \end{array} \right).$$

In the first step, the multiplier is computed as $\lambda_{1,0} = 1/0$ and the algorithm fails. Yet, it is clear that the (unique) solution is

$$\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

The point of the exercise: Gaussian elimination and, equivalently, LU factorization as we have discussed so far can fail if a "divide by zero" is encountered. The element on the

diagonal used to compute the multipliers in a current iteration of the outer-most loop is called the pivot (element). Thus, if a zero pivot is encountered, the algorithms fail. Even if the pivot is merely small (in magnitude), as we will discuss in a future week, roundoff error encountered when performing floating point operations will likely make the computation "numerically unstable," which is the topic of next week's material.

The simple observation is that the rows of the matrix (and corresponding right-hand side element) correspond to linear equations that must be simultaneously solved. Reordering these does not change the solution. Reordering in advance so that no zero pivot is encountered is problematic, since pivots are generally updated by prior computation. However, when a zero pivot is encountered, the row in which it appears can simply be swapped with another row so that the pivot is replaced with a nonzero element (which then becomes the pivot). In exact arithmetic, it suffices to ensure that the pivot is nonzero after swapping. As mentioned, in the presence of roundoff error, any element that is small in magnitude can create problems. For this reason, we will swap rows so that the element with the largest magnitude (among the elements in the "current" column below the diagonal) becomes the pivot. This is known as *partial pivoting* or *row pivoting*.

Homework 5.3.1.2 When performing Gaussian elimination as explained in [Subsection 5.2.1](#) to solve

$$\begin{pmatrix} 10^{-k} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

set

$$1 - 10^k$$

to

$$-10^k$$

(since we will assume k to be large and hence 1 is very small relative to 10^k). With this modification (which simulates roundoff error that may be encountered when performing floating point computation), what is the answer?

Next, solve

$$\begin{pmatrix} 1 & 0 \\ 10^{-k} & 1 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

What do you observe?

Solution. The appended system is given by

$$\left(\begin{array}{cc|c} 10^{-k} & 1 & 1 \\ 1 & 0 & 1 \end{array} \right).$$

In the first step, the multiplier is computed as $\lambda_{1,0} = 10^k$ and the updated appended system becomes

$$\left(\begin{array}{cc|c} 10^{-k} & 1 & 1 \\ 0 & -10^k & 1 - 10^k \end{array} \right)$$

which is rounded to

$$\left(\begin{array}{cc|c} 10^{-k} & 1 & 1 \\ 0 & -10^k & -10^k \end{array} \right).$$

We then compute

$$\chi_1 = (-10^k)/(-10^k) = 1$$

and

$$\chi_0 = (1 - \chi_1)/10^{-k} = (1 - 1)/10^{-k} = 0.$$

If we instead start with the equivalent system

$$\left(\begin{array}{cc|c} 1 & 0 & 1 \\ 10^{-k} & 1 & 1 \end{array} \right).$$

the appended system after one step becomes

$$\left(\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 1 - 10^{-k} \end{array} \right)$$

which yields the solution

$$\left(\begin{array}{c} \chi_0 \\ \chi_1 \end{array} \right) = \left(\begin{array}{c} 1 \\ 1 - 10^{-k} \end{array} \right).$$

which becomes

$$\left(\begin{array}{c} \chi_0 \\ \chi_1 \end{array} \right) = \left(\begin{array}{c} 1 \\ 1 \end{array} \right).$$

as k gets large.

What this illustrates is how a large multiple of a row being added to another row can wipe out information in that second row. After one step of Gaussian elimination, the system becomes equivalent to one that started with

$$\left(\begin{array}{cc|c} 10^{-k} & 1 & 1 \\ 1 & 0 & 0 \end{array} \right).$$

5.3.2 Permutation matrices



YouTube: <https://www.youtube.com/watch?v=4lRnLbvrdtg>

Recall that we already discussed permutation in [Subsection 4.4.4](#) in the setting of column pivoting when computing the QR factorization.

Definition 5.3.2.1 Given

$$p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{n-1} \end{pmatrix},$$

where $\{\pi_0, \pi_1, \dots, \pi_{n-1}\}$ is a permutation (rearrangement) of the integers $\{0, 1, \dots, n-1\}$, we define the permutation matrix $P(p)$ by

$$P(p) = \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}.$$

◊

Homework 5.3.2.1 Let

$$p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{n-1} \end{pmatrix} \text{ and } x = \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{n-1} \end{pmatrix}.$$

Evaluate $P(p)x$.

Solution.

$$\begin{aligned} P(p)x &= \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix} x &< \text{definition} > \\ &= \begin{pmatrix} e_{\pi_0}^T x \\ \vdots \\ e_{\pi_{n-1}}^T x \end{pmatrix} &< \text{matrix-vector multiplication by rows} > \\ &= \begin{pmatrix} \chi_{\pi_0} \\ \vdots \\ \chi_{\pi_{n-1}} \end{pmatrix} &< e_j^T x = x_j > \end{aligned}$$

The last homework shows that applying $P(p)$ to a vector x rearranges the elements of that vector according to the permutation indicated by the vector p .

Homework 5.3.2.2 Let

$$p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{n-1} \end{pmatrix} \text{ and } A = \begin{pmatrix} \tilde{a}_0^T \\ \vdots \\ \tilde{a}_{n-1}^T \end{pmatrix}.$$

Evaluate $P(p)A$.

Solution.

$$\begin{aligned}
 P(p)A &= \left(\begin{array}{c} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{array} \right) A && \text{definition} \\
 &= \left(\begin{array}{c} e_{\pi_0}^T A \\ \vdots \\ e_{\pi_{n-1}}^T A \end{array} \right) && \text{matrix-matrix multiplication by rows} \\
 &= \left(\begin{array}{c} e_j^T A = \tilde{a}_j^T \\ \vdots \\ \tilde{a}_{\pi_0}^T \\ \vdots \\ \tilde{a}_{\pi_{n-1}}^T \end{array} \right) && e_j^T A = \tilde{a}_j^T
 \end{aligned}$$

The last homework shows that applying $P(p)$ to a matrix A rearranges the rows of that matrix according to the permutation indicated by the vector p .

Homework 5.3.2.3 Let

$$p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{n-1} \end{pmatrix} \text{ and } A = \begin{pmatrix} a_0 & \cdots & a_{n-1} \end{pmatrix}.$$

Evaluate $AP(p)^T$.

Solution.

$$\begin{aligned}
 AP(p)^T &= \left(\begin{array}{c} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{array} \right)^T && \text{definition} \\
 A \left(\begin{array}{c} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{array} \right)^T &= \left(\begin{array}{c} e_{\pi_0} & \cdots & e_{\pi_{n-1}} \end{array} \right) && \text{transpose } P(p) \\
 &= \left(\begin{array}{c} Ae_{\pi_0} & \cdots & Ae_{\pi_{n-1}} \end{array} \right) && \text{matrix-matrix multiplication by columns} \\
 &= \left(\begin{array}{c} Ae_j = a_j \\ \vdots \\ a_{\pi_0} \\ \vdots \\ a_{\pi_{n-1}} \end{array} \right) && Ae_j = a_j
 \end{aligned}$$

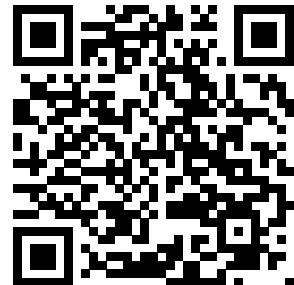
The last homework shows that applying $P(p)^T$ from the right to a matrix A rearranges the columns of that matrix according to the permutation indicated by the vector p .

Homework 5.3.2.4 Evaluate $P(p)P(p)^T$.

Answer. $P(p)P(p)^T = I$

Solution.

$$\begin{aligned}
 & PP(p)^T \\
 &= \left(\begin{array}{c} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{array} \right) \left(\begin{array}{c} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{array} \right)^T && \text{definition} \\
 &= \left(\begin{array}{c} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{array} \right) \left(\begin{array}{ccc} e_{\pi_0} & \cdots & e_{\pi_{n-1}} \end{array} \right) && \text{transpose } P(p) \\
 &= \left(\begin{array}{cccc} e_{\pi_0}^T e_{\pi_0} & e_{\pi_0}^T e_{\pi_1} & \cdots & e_{\pi_0}^T e_{\pi_{n-1}} \\ e_{\pi_1}^T e_{\pi_0} & e_{\pi_1}^T e_{\pi_1} & \cdots & e_{\pi_1}^T e_{\pi_{n-1}} \\ \vdots & \vdots & & \vdots \\ e_{\pi_{n-1}}^T e_{\pi_0} & e_{\pi_{n-1}}^T e_{\pi_1} & \cdots & e_{\pi_{n-1}}^T e_{\pi_{n-1}} \end{array} \right) && \text{evaluate} \\
 &= \left(\begin{array}{cccc} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{array} \right) && \langle e_i^T e_j = \dots \rangle
 \end{aligned}$$



YouTube: <https://www.youtube.com/watch?v=1qvSlLn65Ws>

We will see that when discussing the LU factorization with partial pivoting, a permutation matrix that swaps the first element of a vector with the π -th element of that vector is a fundamental tool.

Definition 5.3.2.2 Elementary pivot matrix. Given $\pi \in \{0, \dots, n-1\}$ define the

elementary pivot matrix

$$\tilde{P}(\pi) = \begin{pmatrix} \frac{e_\pi^T}{e_1^T} \\ \vdots \\ \frac{e_{\pi-1}^T}{e_0^T} \\ \frac{e_\pi^T}{e_{\pi+1}^T} \\ \vdots \\ e_{n-1}^T \end{pmatrix}$$

or, equivalently,

$$\tilde{P}(\pi) = \begin{cases} I_n & \text{if } \pi = 0 \\ \left(\begin{array}{c|c|c|c} 0 & 0 & 1 & 0 \\ \hline 0 & I_{\pi-1} & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & I_{n-\pi-1} \end{array} \right) & \text{otherwise,} \end{cases}$$

where n is the size of the permutation matrix. \diamond

When $\tilde{P}(\pi)$ is applied to a vector, it swaps the top element with the element indexed with π . When it is applied to a matrix, it swaps the top row with the row indexed with π . The size of matrix $\tilde{P}(\pi)$ is determined by the size of the vector or the row size of the matrix to which it is applied.

In discussing LU factorization with pivoting, we will use elementary pivot matrices in a very specific way, which necessitates the definition of how a sequence of such pivots are applied. Let p be a vector of integers satisfying the conditions

$$p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{k-1} \end{pmatrix}, \quad \text{where } 1 \leq k \leq n \text{ and } 0 \leq \pi_i < n - i, \quad (5.3.1)$$

then $\tilde{P}(p)$ will denote the sequence of pivots

$$\tilde{P}(p) = \begin{pmatrix} I_{k-1} & 0 \\ 0 & \tilde{P}(\pi_{k-1}) \end{pmatrix} \begin{pmatrix} I_{k-2} & 0 \\ 0 & \tilde{P}(\pi_{k-2}) \end{pmatrix} \cdots \begin{pmatrix} 1 & 0 \\ 0 & \tilde{P}(\pi_1) \end{pmatrix} \tilde{P}(\pi_0).$$

(Here $\tilde{P}(\cdot)$ is always an elementary pivot matrix "of appropriate size.") What this exactly does is best illustrated through an example:

Example 5.3.2.3 Let

$$p = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} \quad \text{and} \quad A = \begin{pmatrix} 0.0 & 0.1 & 0.2 \\ 1.0 & 1.1 & 1.2 \\ 2.0 & 2.1 & 2.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix}.$$

Evaluate $\tilde{P}(p)A$.

Solution.

$$\begin{aligned}
 & \tilde{P}(p)A \\
 &= \quad <\text{ instantiate}> \\
 & \tilde{P}\left(\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}\right) \begin{pmatrix} 0.0 & 0.1 & 0.2 \\ 1.0 & 1.1 & 1.2 \\ 2.0 & 2.1 & 2.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \\
 &= \quad <\text{ definition of } \tilde{P}(\cdot)> \\
 & \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \tilde{P}\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) \end{array} \right) \tilde{P}(2) \begin{pmatrix} 0.0 & 0.1 & 0.2 \\ 1.0 & 1.1 & 1.2 \\ 2.0 & 2.1 & 2.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \\
 &= \quad <\text{ swap first row with row indexed with 2}> \\
 & \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \tilde{P}\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) \end{array} \right) \begin{pmatrix} 2.0 & 2.1 & 2.2 \\ 1.0 & 1.1 & 1.2 \\ 0.0 & 0.1 & 0.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \\
 &= \quad <\text{ partitioned matrix-matrix multiplication}> \\
 & \left(\begin{array}{c} \left(\begin{array}{ccc} 2.0 & 2.1 & 2.2 \end{array} \right) \\ \hline \tilde{P}\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) \begin{pmatrix} 1.0 & 1.1 & 1.2 \\ 0.0 & 0.1 & 0.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \end{array} \right) = \quad <\text{ swap current first row with row indexed with 1 relative to}> \\
 & \left(\begin{array}{c} \left(\begin{array}{ccc} 2.0 & 2.1 & 2.2 \\ 0.0 & 0.1 & 0.2 \end{array} \right) \\ \hline \tilde{P}\left(\begin{pmatrix} 1 \end{pmatrix}\right) \begin{pmatrix} 1.0 & 1.1 & 1.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \end{array} \right) \\
 &= \quad <\text{ swap current first row with row indexed with 1 relative to that row}> \\
 & \left(\begin{array}{c} \left(\begin{array}{ccc} 2.0 & 2.1 & 2.2 \\ 0.0 & 0.1 & 0.2 \\ 3.0 & 3.1 & 3.2 \end{array} \right) \\ \hline \left(\begin{array}{ccc} 1.0 & 1.1 & 1.2 \end{array} \right) \end{array} \right) \\
 &= \\
 & \left(\begin{array}{ccc} 2.0 & 2.1 & 2.2 \\ 0.0 & 0.1 & 0.2 \\ 3.0 & 3.1 & 3.2 \\ 1.0 & 1.1 & 1.2 \end{array} \right)
 \end{aligned}$$

□

The relation between $\tilde{P}(\cdot)$ and $P(\cdot)$ is tricky to specify:

$$\tilde{P}\left(\begin{array}{c} \pi_0 \\ \pi_1 \\ \vdots \\ \pi_{k-1} \end{array}\right) = P\left(\begin{array}{cc} I_{k-1} & 0 \\ 0 & \tilde{P}(\pi_{k-1}) \end{array}\right) \cdots \left(\begin{array}{cc} 1 & 0 \\ 0 & \tilde{P}(\pi_1) \end{array}\right) \tilde{P}(\pi_0) \left(\begin{array}{c} 0 \\ 1 \\ \vdots \\ k-1 \end{array}\right).$$

5.3.3 LU factorization with partial pivoting



YouTube: <https://www.youtube.com/watch?v=QSnoqrsQNag>

Having introduced our notation for permutation matrices, we can now define the LU factorization with partial pivoting: Given an $m \times n$ matrix A , we wish to compute

- vector p of n integers that indicates how rows are pivoting as the algorithm proceeds,
- a unit lower trapezoidal matrix L , and
- an upper triangular matrix U

so that $\tilde{P}(p)A = LU$. We represent this operation by

$$[A, p] := \text{LUpiv}A,$$

where upon completion A has been overwritten by $\{L \setminus U\}$, which indicates that U overwrites the upper triangular part of A and L is stored in the strictly lower triangular part of A .

Let us start with revisiting the derivation of the right-looking LU factorization in [Subsection 5.2.2](#). The first step is to find a first permutation matrix $\tilde{P}(\pi_1)$ such that the element on the diagonal in the first column is maximal in value. (Mathematically, any nonzero value works. We will see that ensuring that the multiplier is less than one in magnitude reduces the potential for accumulation of error.) For this, we will introduce the function

$$\text{maxi}(x)$$

which, given a vector x , returns the index of the element in x with maximal magnitude (absolute value). The algorithm then proceeds as follows:

- Partition A , L as follows:

$$A \rightarrow \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}, \quad \text{and} \quad L \rightarrow \begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix}$$

- Compute $\pi_1 = \max_i \left(\frac{\alpha_{11}}{a_{21}} \right)$.
- Permute the rows: $\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} := \tilde{P}(\pi_1) \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}$.
- Compute $l_{21} := a_{21}/\alpha_{11}$.
- Update $A_{22} := A_{22} - l_{21}a_{12}^T$.

This completes the introduction of zeroes below the diagonal of the first column.

Now, more generally, assume that the computation has proceeded to the point where matrix A has been overwritten by

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right)$$

where A_{00} is upper triangular. If no pivoting was added one would compute $l_{21} := a_{21}/\alpha_{11}$ followed by the update

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|cc} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right).$$

Now, instead one performs the steps

- Compute

$$\pi_1 := \max_i \left(\frac{\alpha_{11}}{a_{21}} \right).$$

- Permute the rows:

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|cc} I & 0 & 0 \\ \hline 0 & \tilde{P}(\pi_1) & \end{array} \right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right)$$

- Update

$$l_{21} := a_{21}/\alpha_{11}.$$

- Update

$$\begin{aligned} \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) &:= \left(\begin{array}{c|cc} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) \\ &= \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right). \end{aligned}$$

This algorithm is summarized in [Figure 5.3.3.1](#). In that algorithm, the lower triangular matrix L is accumulated below the diagonal.

```
[A, p] = LUpiv-right-looking(A)
A →  $\left( \begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$ , p →  $\left( \begin{array}{c} p_T \\ \hline p_B \end{array} \right)$ 
    ATL is 0 × 0, pT has 0 elements
while n(ATL) < n(A)
     $\left( \begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left( \begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left( \begin{array}{c} p_T \\ \hline p_B \end{array} \right) \rightarrow \left( \begin{array}{c} p_0 \\ \hline \pi_1 \\ p_2 \end{array} \right)$ 
     $\pi_1 := \max_i \left( \frac{\alpha_{11}}{a_{21}} \right)$ 
     $\left( \begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) := \left( \begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right) \left( \begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$ 
     $a_{21} := a_{21}/\alpha_{11}$ 
     $A_{22} := A_{22} - a_{21}a_{12}^T$ 
     $\left( \begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left( \begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left( \begin{array}{c} p_T \\ \hline p_B \end{array} \right) \leftarrow \left( \begin{array}{c} p_0 \\ \hline \pi_1 \\ p_2 \end{array} \right)$ 
endwhile
```

Figure 5.3.3.1 Right-looking LU factorization algorithm with partial pivoting.



YouTube: <https://www.youtube.com/watch?v=n-Kl62HrYhM>

What this algorithm computes is a sequence of Gauss transforms L_0, \dots, L_{n-1} and permutations P_0, \dots, P_{n-1} such that

$$L_{n-1}P_{n-1} \cdots L_0P_0A = U$$

or, equivalently,

$$A = P_0^T L_0^{-1} \cdots P_{n-1}^T L_{n-1}^{-1} U.$$

Actually, since $P_k = \left(\begin{array}{c|c} I_{k \times k} & 0 \\ \hline 0 & \tilde{P}(\pi) \end{array} \right)$ for some π , we know that $P_k^T = P_k$ and hence

$$A = P_0 L_0^{-1} \cdots P_{n-1} L_{n-1}^{-1} U.$$

What we will finally show is that there are Gauss transforms L_0^*, \dots, L_{n-1}^* such that

$$A = P_0 \cdots P_{n-1} \underbrace{L_0^* \cdots L_{n-1}^*}_L U$$

or, equivalently,

$$\tilde{P}(p)A = P_{n-1} \cdots P_0 A = \underbrace{L_0^* \cdots L_{n-1}^*}_L U,$$

which is what we set out to compute.

Here is the insight. If only we know how to order the rows of A and right-hand side b correctly, then we would not have to pivot. But we only know how to pivot as the computation unfolds. Recall that the multipliers can overwrite the elements they zero in Gaussian elimination and do so when we formulate it as an LU factorization. By not only pivoting the elements of

$$\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}$$

but also all of

$$\left(\begin{array}{c|cc} a_{12}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right),$$

we are moving the computed multipliers with the rows that are being swapped. It is for this reason that we end up computing the LU factorization of the permuted matrix: $\tilde{P}(p)A$.

Homework 5.3.3.1 Implement the algorithm given in [Figure 5.3.3.1](#) as
function [A_out] = LUpiv_right_looking(A)

by completing the code in [Assignments/Week05/matlab/LUpiv_right_looking.m](#). Input is an $m \times n$ matrix A . Output is the matrix A that has been overwritten by the LU factorization and pivot vector p . You may want to use [Assignments/Week05/matlab/test_LUpiv_right_looking.m](#) to check your implementation.

The following utility functions may come in handy:

- [Assignments/Week05/matlab/maxi.m](#)
- [Assignments/Week05/matlab/Swap.m](#)

which we hope are self explanatory.

Solution. See [Assignments/Week05/answers/LUpiv_right_looking.m](#). ([Assignments/Week05/answers/LUpiv_right_looking.m](#))

5.3.4 Solving $A x = y$ via LU factorization with pivoting



YouTube: <https://www.youtube.com/watch?v=kqj3n1EUCkw>

Given nonsingular matrix $A \in \mathbb{C}^{m \times n}$, the above discussions have yielded an algorithm for computing permutation matrix P , unit lower triangular matrix L and upper triangular matrix U such that $PA = LU$. We now discuss how these can be used to solve the system of linear equations $Ax = y$.

Starting with

$$Ax = b$$

where nonsingular matrix A is $n \times n$ (and hence square),

- Overwrite A with its LU factorization, accumulating the pivot information in vector p :

$$[A, p] := \text{LU piv}(A).$$

A now contains L and U and $\tilde{P}(p)A = LU$.

- We notice that $Ax = b$ is equivalent to $\tilde{P}(p)Ax = \tilde{P}(p)b$. Thus, we compute $y := \tilde{P}(p)b$. Usually, y overwrites b .
- Next, we recognize that $\tilde{P}(p)Ax = y$ is equivalent to $L \underbrace{(Ux)}_z = y$. Hence, we can compute z by solving the unit lower triangular system

$$Lz = y$$

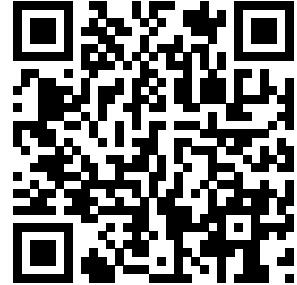
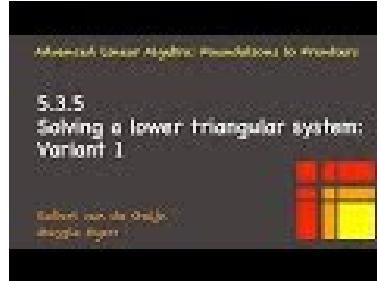
and next compute x by solving the upper triangular system

$$Ux = z.$$

5.3.5 Solving with a triangular matrix

We are left to discuss how to solve $Lz = y$ and $Ux = z$.

5.3.5.1 Algorithmic Variant 1



YouTube: https://www.youtube.com/watch?v=qc_4NsNp3q0

Consider $Lz = y$ where L is unit lower triangular. Partition

$$L \rightarrow \begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix}, \quad z \rightarrow \begin{pmatrix} \zeta_1 \\ z_2 \end{pmatrix} \quad \text{and} \quad y \rightarrow \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix}.$$

Then

$$\underbrace{\begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix}}_L \underbrace{\begin{pmatrix} \zeta_1 \\ z_2 \end{pmatrix}}_z = \underbrace{\begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix}}_y.$$

Multiplying out the left-hand side yields

$$\begin{pmatrix} \zeta_1 \\ \zeta_1 l_{21} + L_{22} z_2 \end{pmatrix} = \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix}$$

and the equalities

$$\begin{aligned} \zeta_1 &= \psi_1 \\ \zeta_1 l_{21} + L_{22} z_2 &= y_2, \end{aligned}$$

which can be rearranged as

$$\begin{aligned} \zeta_1 &= \psi_1 \\ L_{22} z_2 &= y_2 - \zeta_1 l_{21}. \end{aligned}$$

We conclude that in the current iteration

- ψ_1 needs not be updated.
- $y_2 := y_2 - \psi_1 l_{21}$

So that in future iterations $L_{22} z_2 = y_2$ (updated!) will be solved, updating z_2 .

These insights justify the algorithm in Figure 5.3.5.1, which overwrites y with the solution to $Lz = y$.

Solve $Lz = y$, overwriting y with z (Variant 1)
$L \rightarrow \left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$
L_{TL} is 0×0 and y_T has 0 elements
while $n(L_{TL}) < n(L)$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$
$y_2 := y_2 - \psi_1 l_{21}$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$
endwhile

Figure 5.3.5.1 Lower triangular solve (with unit lower triangular matrix), Variant 1

Homework 5.3.5.1 Derive a similar algorithm for solving $Ux = z$. Update the below skeleton algorithm with the result. (Don't forget to put in the lines that indicate how you "partition and repartition" through the matrix.)

Solve $Ux = z$, overwriting z with x (Variant 1)
$U \rightarrow \left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right), z \rightarrow \left(\begin{array}{c} z_T \\ z_B \end{array} \right)$
U_{BR} is 0×0 and z_B has 0 elements
while $n(U_{BR}) < n(U)$
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \rightarrow \left(\begin{array}{ccc} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \rightarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \leftarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$
endwhile

Hint: Partition

$$\left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right) \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right) = \left(\begin{array}{c} z_0 \\ \zeta_1 \end{array} \right).$$

Solution. Multiplying this out yields

$$\left(\begin{array}{c} U_{00}x_0 + u_{01}\chi_1 \\ v_{11}\chi_1 \end{array} \right) = \left(\begin{array}{c} z_0 \\ \zeta_1 \end{array} \right).$$

So, $\chi_1 = \zeta_1/v_{11}$ after which x_0 can be computed by solving $U_{00}x_0 = z_0 - \chi_1 u_{01}$. The resulting algorithm is then given by

Solve $Ux = z$, overwriting z with x (Variant 1)	
$U \rightarrow \left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right), z \rightarrow \left(\begin{array}{c} z_T \\ z_B \end{array} \right)$	
U_{BR} is 0×0 and z_B has 0 elements	
while $n(U_{BR}) < n(U)$	
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \rightarrow \left(\begin{array}{cc c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \rightarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$	
$\zeta_1 := \zeta_1/v_{11}$	
$z_0 := z_0 - \zeta_1 u_{01}$	
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \leftarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$	
endwhile	

5.3.5.2 Algorithmic Variant 2



YouTube: <https://www.youtube.com/watch?v=2tvfYnD9NrQ>

An alternative algorithm can be derived as follows: Partition

$$L \rightarrow \left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right), \quad z \rightarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \end{array} \right) \quad \text{and} \quad y \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right).$$

Then

$$\underbrace{\left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right)}_L \underbrace{\left(\begin{array}{c} z_0 \\ \zeta_1 \end{array} \right)}_z = \underbrace{\left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right)}_y.$$

Multiplying out the left-hand side yields

$$\left(\begin{array}{c} L_{00}z_0 \\ l_{10}^T z_0 + \zeta_1 \end{array} \right) = \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right)$$

and the equalities

$$\begin{aligned} L_{00}z_0 &= y_0 \\ l_{10}^T z_0 + \zeta_1 &= \psi_1. \end{aligned}$$

The idea now is as follows: Assume that the elements of z_0 were computed in previous iterations in the algorithm in [Figure 5.3.5.2](#), overwriting y_0 . Then in the current iteration we must compute $\zeta_1 := \psi_1 - l_{10}^T z_0$, overwriting ψ_1 .

Solve $Lz = y$, overwriting y with z (Variant 2)	
$L \rightarrow \left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$	
L_{TL} is 0×0 and y_T has 0 elements	
while $n(L_{TL}) < n(L)$	
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$	
$\psi_1 := \psi_1 - l_{10}^T y_0$	
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$	
endwhile	

Figure 5.3.5.2 Lower triangular solve (with unit lower triangular matrix), Variant 2

Homework 5.3.5.2 Derive a similar algorithm for solving $Ux = z$. Update the below skeleton algorithm with the result. (Don't forget to put in the lines that indicate how you "partition and repartition" through the matrix.)

Solve $Ux = z$, overwriting z with x (Variant 2)	
$U \rightarrow \left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right), z \rightarrow \left(\begin{array}{c} z_T \\ z_B \end{array} \right)$	
U_{BR} is 0×0 and z_B has 0 elements	
while $n(U_{BR}) < n(U)$	
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c ccc} U_{00} & u_{01} & U_{02} & \\ \hline u_{10}^T & v_{11} & u_{12}^T & \\ U_{20} & u_{21} & U_{22} & \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \rightarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$	
$\zeta_1 := z_0 - u_{10}^T z_1$	
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c ccc} U_{00} & u_{01} & U_{02} & \\ \hline u_{10}^T & v_{11} & u_{12}^T & \\ U_{20} & u_{21} & U_{22} & \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \leftarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$	
endwhile	

Hint: Partition

$$U \rightarrow \left(\begin{array}{cc} v_{11} & u_{12}^T \\ 0 & U_{22} \end{array} \right).$$

Solution. Partition

$$\begin{pmatrix} v_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix} \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \zeta_1 \\ z_2 \end{pmatrix}.$$

Multiplying this out yields

$$\begin{pmatrix} v_{11}\chi_1 + u_{12}^T x_2 \\ U_{22}x_2 \end{pmatrix} = \begin{pmatrix} \zeta_1 \\ z_2 \end{pmatrix}.$$

So, if we assume that x_2 has already been computed and has overwritten z_2 , then χ_1 can be computed as

$$\chi_1 = (\zeta_1 - u_{12}^T x_2)/v_{11}$$

which can then overwrite ζ_1 . The resulting algorithm is given by

Solve $Ux = z$, overwriting z with x (Variant 2)	
$U \rightarrow \left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right), z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array} \right)$	U_{BR} is 0×0 and z_B has 0 elements
while $n(U_{BR}) < n(U)$	
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \rightarrow \left(\begin{array}{cc c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array} \right) \rightarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ z_2 \end{array} \right)$	
$\zeta_1 := \zeta_1 - u_{12}^T z_2$	
$\zeta_1 := \zeta_1/v_{11}$	
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array} \right) \leftarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ z_2 \end{array} \right)$	
endwhile	

Homework 5.3.5.3 Let L be an $m \times m$ unit lower triangular matrix. If a multiply and add each require one flop, what is the approximate cost of solving $Lx = y$?

Solution. Let us analyze Variant 1.

Let L_{00} be $k \times k$ in a typical iteration. Then y_2 is of size $m - k - 1$ and $y_2 := y_2 - \psi_1 l_{21}$ requires $2(m - k - 1)$ flops. Summing this over all iterations requires

$$\sum_{k=0}^{m-1} [2(m - k - 1)] \text{ flops.}$$

The change of variables $j = m - k - 1$ yields

$$\sum_{k=0}^{m-1} [2(m - k - 1)] = 2 \sum_{j=0}^{m-1} j \approx m^2.$$

Thus, the cost is approximately m^2 flops.

5.3.5.3 Discussion

Computation tends to be more efficient when matrices are accessed by column, since in scientific computing applications tend to store matrices by columns (in column-major order). This dates back to the days when Fortran ruled supreme. Accessing memory consecutively improves performance, so computing with columns tends to be more efficient than computing with rows.

Variant 1 for each of the algorithms casts computation in terms of columns of the matrix that is involved;

$$y_2 := y_2 - \psi_1 l_{21}$$

and

$$z_0 := z_0 - \zeta_1 u_{01}.$$

These are called **axpy** operations:

$$y := \alpha x + y.$$

"alpha times x plus y ." In contrast, Variant 2 casts computation in terms of rows of the matrix that is involved:

$$\psi_1 := \psi_1 - l_{10}^T y_0$$

and

$$\zeta_1 := \zeta_1 - u_{12}^T z_2$$

perform dot products.

5.3.6 LU factorization with complete pivoting

LU factorization with partial pivoting builds on the insight that pivoting (rearranging) rows in a linear system does not change the solution: if $Ax = b$ then $P(p)Ax = P(p)b$, where p is a pivot vector. Now, if r is another pivot vector, then notice that $P(r)^T P(r) = I$ (a simple property of pivot matrices) and $AP(r)^T$ permutes the columns of A in exactly the same order as $P(r)A$ permutes the rows of A .

What this means is that if $Ax = b$ then $P(p)AP(r)^T(P(r)x) = P(p)b$. This supports the idea that one might want to not only permute rows of A , as in partial pivoting, but also columns of A . This is done in a variation on LU factorization that is known as LU factorization with *complete pivoting*.

The idea is as follows: Given matrix A , partition

$$A = \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}.$$

Now, instead of finding the largest element in magnitude in the first column, find the largest element in magnitude in the entire matrix. Let's say it is element (π_1, ρ_1) . Then, one permutes

$$\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} := P(\pi_1) \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} P(\rho_1)^T,$$

making α_{11} the largest element in magnitude. We will later see that the magnitude of α_{11} impacts *element growth* in the remaining matrix (A_{22}) and that in turn impacts the numerical stability (accuracy) of the algorithm. By choosing α_{11} to be as large as possible in magnitude, the magnitude of multipliers is reduced as is element growth.

The problem is that complete pivoting requires $O(n^2)$ comparisons per iteration. Thus, the number of comparisons is of the same order as the number of floating point operations. Worse, it completely destroys the ability to cast most computation in terms of matrix-matrix multiplication, thus impacting the ability to attain much greater performance.

In practice LU with complete pivoting is not used.

5.3.7 Improving accuracy via iterative refinement

When solving $Ax = b$ on a computer, error is inherently incurred. Instead of the exact solution x , an approximate solution \hat{x} is computed, which instead solves $A\hat{x} = \hat{b}$. The difference between x and \hat{x} satisfies

$$A(x - \hat{x}) = b - \hat{b}.$$

We can compute $\hat{b} = A\hat{x}$ and hence we can compute $\delta b = b - \hat{b}$. We can then solve $A\delta x = \delta b$. If this computation is completed without error, then $x = \hat{x} + \delta x$ and we are left with the exact solution. Obviously, there is error in δx as well, and hence we have merely computed an improved approximate solution to $Ax = b$. This process can be repeated. As long as solving with A yields at least one digit of accuracy, this process can be used to improve the computed result, limited by the accuracy in the right-hand side b and the condition number of A .

This process is known as iterative refinement.

5.4 Cholesky factorization

5.4.1 Hermitian Positive Definite matrices



YouTube: <https://www.youtube.com/watch?v=nxGR8NgXYxg>

Hermitian Positive Definite (HPD) are a special class of matrices that are frequently encountered in practice.

Definition 5.4.1.1 Hermitian positive definite matrix. A matrix $A \in \mathbb{C}^{n \times n}$ is Hermitian positive definite (HPD) if and only if it is Hermitian ($A^H = A$) and for all nonzero

vectors $x \in \mathbb{C}^n$ it is the case that $x^H Ax > 0$. If in addition $A \in \mathbb{R}^{n \times n}$ then A is said to be symmetric positive definite (SPD). \diamond

If you feel uncomfortable with complex arithmetic, just replace the word "Hermitian" with "symmetric" in this document and the Hermitian transpose operation, H , with the transpose operation, T .

Example 5.4.1.2 Consider the case where $n = 1$ so that A is a real scalar, α . Notice that then A is SPD if and only if $\alpha > 0$. This is because then for all nonzero $\chi \in \mathbb{R}$ it is the case that $\alpha\chi^2 > 0$. \square

Let's get some practice with reasoning about Hermitian positive definite matrices.

Homework 5.4.1.1 Let $B \in \mathbb{C}^{m \times n}$ have linearly independent columns.

ALWAYS/SOMETIMES/NEVER: $A = B^H B$ is HPD.

Answer. ALWAYS

Now prove it!

Solution. Let $x \in \mathbb{C}^m$ be a nonzero vector. Then $x^H B^H B x = (Bx)^H (Bx)$. Since B has linearly independent columns we know that $Bx \neq 0$. Hence $(Bx)^H (Bx) > 0$.

Homework 5.4.1.2 Let $A \in \mathbb{C}^{m \times m}$ be HPD.

ALWAYS/SOMETIMES/NEVER: The diagonal elements of A are real and positive.

Hint. Consider the standard basis vector e_j .

Answer. ALWAYS

Now prove it!

Solution. Let e_j be the j th unit basis vectors. Then $0 < e_j^H A e_j = \alpha_{j,j}$.

Homework 5.4.1.3 Let $A \in \mathbb{C}^{m \times m}$ be HPD. Partition

$$A = \left(\begin{array}{c|c} \alpha_{11} & a_{21}^H \\ \hline a_{21} & A_{22} \end{array} \right).$$

ALWAYS/SOMETIMES/NEVER: A_{22} is HPD.

Answer. ALWAYS

Now prove it!

Solution. We need to show that $x_2^H A_{22} x_2 > 0$ for any nonzero $x_2 \in \mathbb{C}^{m-1}$.

Let $x_2 \in \mathbb{C}^{m-1}$ be a nonzero vector and choose $x = \begin{pmatrix} 0 \\ x_2 \end{pmatrix}$. Then

$$\begin{aligned} & 0 \\ & < \quad < A \text{ is HPD} > \\ & x^H A x \\ & = < \text{partition} > \\ & \left(\begin{array}{c} 0 \\ x_2 \end{array} \right)^H \left(\begin{array}{c|c} \alpha_{11} & a_{21}^H \\ \hline a_{21} & A_{22} \end{array} \right) \left(\begin{array}{c} 0 \\ x_2 \end{array} \right) \\ & = < \text{multiply out} > \\ & x_2^H A_{22} x_2. \end{aligned}$$

We conclude that A_{22} is HPD.

5.4.2 The Cholesky Factorization Theorem



YouTube: <https://www.youtube.com/watch?v=w8a9xVHVmAI>

We will prove the following theorem in [Subsection 5.4.4](#).

Theorem 5.4.2.1 Cholesky Factorization Theorem. *Given an HPD matrix A there exists a lower triangular matrix L such that $A = LL^H$. If the diagonal elements of L are restricted to be positive, L is unique.*

Obviously, there similarly exists an upper triangular matrix U such that $A = U^HU$ since we can choose $U^H = L$.

The lower triangular matrix L is known as the Cholesky factor and LL^H is known as the Cholesky factorization of A . It is unique if the diagonal elements of L are restricted to be positive. Typically, only the lower (or upper) triangular part of A is stored, and it is that part that is then overwritten with the result. In our discussions, we will assume that the lower triangular part of A is stored and overwritten.

5.4.3 Cholesky factorization algorithm (right-looking variant)



YouTube: <https://www.youtube.com/watch?v=x4grvf-MfTk>

The most common algorithm for computing the Cholesky factorization of a given HPD matrix A is derived as follows:

- Consider $A = LL^H$, where L is lower triangular.

Partition

$$A = \begin{pmatrix} \alpha_{11} & * \\ a_{21} & A_{22} \end{pmatrix} \quad \text{and} \quad L = \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix}. \quad (5.4.1)$$

Since A is HPD, we know that

- α_{11} is a positive number ([Homework 5.4.1.2](#)).
- A_{22} is HPD ([Homework 5.4.1.3](#)).
- By substituting these partitioned matrices into $A = LL^H$ we find that

$$\begin{aligned} \begin{pmatrix} \alpha_{11} & \star \\ a_{21} & A_{22} \end{pmatrix} &= \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix}^H = \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} \bar{\lambda}_{11} & l_{21}^H \\ 0 & L_{22}^H \end{pmatrix} \\ &= \begin{pmatrix} |\lambda_{11}|^2 & \star \\ \bar{\lambda}_{11}l_{21} & l_{21}l_{21}^H + L_{22}L_{22}^H \end{pmatrix}, \end{aligned}$$

from which we conclude that

$$\begin{aligned} \alpha_{11} &= |\lambda_{11}|^2 \\ a_{21} &= \lambda_{11}l_{21} \quad A_{22} = l_{21}l_{21}^H + L_{22}L_{22}^H \end{aligned}$$

or, equivalently,

$$\begin{aligned} \lambda_{11} &= \pm\sqrt{\alpha_{11}} \\ l_{21} &= a_{21}/\bar{\lambda}_{11} \quad L_{22} = \text{Chol}(A_{22} - l_{21}l_{21}^H) \end{aligned}$$

- These equalities motivate the following algorithm for overwriting the lower triangular part of A with the Cholesky factor of A :

 - Partition $A \rightarrow \begin{pmatrix} \alpha_{11} & \star \\ a_{21} & A_{22} \end{pmatrix}$.
 - Overwrite $\alpha_{11} := \lambda_{11} = \sqrt{\alpha_{11}}$. (Picking $\lambda_{11} = \sqrt{\alpha_{11}}$ makes it positive and real, and ensures uniqueness.)
 - Overwrite $a_{21} := l_{21} = a_{21}/\lambda_{11}$.
 - Overwrite $A_{22} := A_{22} - l_{21}l_{21}^H$ (updating only the lower triangular part of A_{22}). This operation is called a *symmetric rank-1 update*.
 - Continue by computing the Cholesky factor of A_{22} .

The resulting algorithm is often called the "right-looking" variant and is summarized in [Figure 5.4.3.1](#).

$A = \text{Chol-right-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$\alpha_{11} := \lambda_{11} = \sqrt{\alpha_{11}}$
$a_{21} := l_{21} = a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{21}^H$ (syr: update only lower triangular part)
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.4.3.1 Cholesky factorization algorithm (right-looking variant). The operation "syr" refers to "symmetric rank-1 update", which performs a rank-1 update, updating only the lower triangular part of the matrix in this algorithm.

Homework 5.4.3.1 Give the approximate cost incurred by the algorithm in Figure 5.4.3.1 when applied to an $n \times n$ matrix.

Answer. $\frac{1}{3}n^3$ flops.

Solution.



YouTube: <https://www.youtube.com/watch?v=6twDI6QhqCY>

The cost of the Cholesky factorization of $A \in \mathbb{C}^{n \times n}$ can be analyzed as follows: In Figure 5.4.3.1 during the k th iteration (starting k at zero) A_{00} is $k \times k$. Thus, the operations in that iteration cost

- $\alpha_{11} := \sqrt{\alpha_{11}}$: this cost is negligible when k is large.
- $a_{21} := a_{21}/\alpha_{11}$: approximately $(n-k-1)$ flops. This operation is typically implemented as $(1/\alpha_{11})a_{21}$.
- $A_{22} := A_{22} - a_{21}a_{21}^H$ (updating only the lower triangular part of A_{22}): approximately $(n-k-1)^2$ flops.

Thus, the total cost in flops is given by

$$\begin{aligned}
 C_{\text{Chol}}(n) &\approx < \text{sum over all iterations} > \\
 &\underbrace{\sum_{k=0}^{n-1} (n-k-1)^2}_{(\text{Due to update of } A_{22})} + \underbrace{\sum_{k=0}^{n-1} (n-k-1)}_{(\text{Due to update of } a_{21})} \\
 &= < \text{change of variables } j = n - k - 1 > \\
 &\sum_{j=0}^{n-1} j^2 + \sum_{j=0}^{n-1} j \\
 &\approx < \sum_{j=0}^{n-1} j^2 \approx n^3/3; \sum_{j=0}^{n-1} j \approx n^2/2 > \\
 &\frac{1}{3}n^3 + \frac{1}{2}n^2 \\
 &\approx < \text{remove lower order term} > \\
 &\frac{1}{3}n^3.
 \end{aligned}$$

Remark 5.4.3.2 Comparing the cost of the Cholesky factorization to that of the LU factorization in [Homework 5.2.2.1](#), we see that taking advantage of symmetry cuts the cost approximately in half.

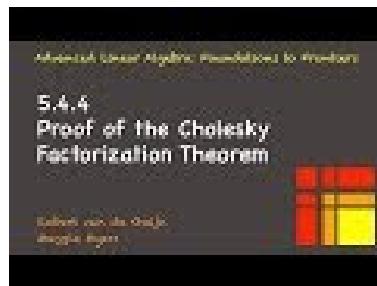
Homework 5.4.3.2 Implement the algorithm given in [Figure 5.4.3.1](#) as

```
function [ A_out ] = Chol_right_looking( A )
```

by completing the code in [Assignments/Week05/matlab/Chol_right_looking.m](#). Input is a HPD $m \times n$ matrix A with only the lower triangular part stored. Output is the matrix A that has its lower triangular part overwritten with the Cholesky factor. You may want to use [Assignments/Week05/matlab/test_Chol_right_looking.m](#) to check your implementation.

Solution. See [Assignments/Week05/answers/Chol_right_looking.m](#). ([Assignments/Week05/answers/Chol_right_looking.m](#))

5.4.4 Proof of the Cholesky Factorization Theorem



YouTube: <https://www.youtube.com/watch?v=unpQfRgIH0g>

Partition, once again,

$$A \rightarrow \begin{pmatrix} \alpha_{11} & a_{21}^H \\ a_{21} & A_{22} \end{pmatrix}.$$

The following lemmas are key to the proof of the Cholesky Factorization Theorem:

Lemma 5.4.4.1 Let $A \in \mathbb{C}^{n \times n}$ be HPD. Then α_{11} is real and positive.

Proof. This is special case of [Homework 5.4.1.1](#). ■

Lemma 5.4.4.2 Let $A \in \mathbb{C}^{n \times n}$ be HPD and $l_{21} = a_{21}/\sqrt{\alpha_{11}}$. Then $A_{22} - l_{21}l_{21}^H$ is HPD.

Proof. Since A is Hermitian so are A_{22} and $A_{22} - l_{21}l_{21}^H$.

Let $x_2 \in \mathbb{C}^{n-1}$ be an arbitrary nonzero vector. Define $x = \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix}$ where $\chi_1 = -a_{21}^H x_2 / \alpha_{11}$.

Then, since clearly $x \neq 0$,

$$\begin{aligned}
 0 & < A \text{ is HPD} > \\
 x^H Ax & = < \text{partition} > \\
 \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix}^H \begin{pmatrix} \alpha_{11} & a_{21}^H \\ a_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} & = < \text{partitioned multiplication} > \\
 \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix}^H \begin{pmatrix} \alpha_{11}\chi_1 + a_{21}^H x_2 \\ a_{21}\chi_1 + A_{22}x_2 \end{pmatrix} & = < \text{partitioned multiplication} > \\
 \alpha_{11}\bar{\chi}_1\chi_1 + \bar{\chi}_1 a_{21}^H x_2 + x_2^H a_{21}\chi_1 + x_2^H A_{22}x_2 & = < \text{linear algebra} > \\
 \alpha_{11} \frac{a_{21}^H x_2}{\alpha_{11}} \frac{x_2^H a_{21}}{\alpha_{11}} - \frac{x_2^H a_{21}}{\alpha_{11}} a_{21}^H x_2 - x_2^H a_{21} \frac{a_{21}^H x_2}{\alpha_{11}} + x_2^H A_{22}x_2 & = < \text{since } x_2^H a_{21}, a_{21}^H x_2 \text{ are scalars and hence can move around; } \alpha_{11}/\alpha_{11} = 1 > \\
 x_2^H a_{21} \frac{a_{21}^H x_2}{\alpha_{11}} - x_2^H a_{21} \frac{a_{21}^H x_2}{\alpha_{11}} - x_2^H a_{21} \frac{a_{21}^H x_2}{\alpha_{11}} + x_2^H A_{22}x_2 & = < \text{cancel terms; factor out } x_2^H \text{ and } x_2 > \\
 x_2^H (A_{22} - \frac{a_{21}a_{21}^H}{\alpha_{11}}) x_2 & = < \text{simplify} > \\
 x_2^H (A_{22} - l_{21}l_{21}^H) x_2. &
 \end{aligned}$$

We conclude that $A_{22} - l_{21}l_{21}^H$ is HPD. ■

Proof of the Cholesky Factorization Theorem. Proof by induction.

1. Base case: $n = 1$:

Clearly the result is true for a 1×1 matrix $A = \alpha_{11}$: In this case, the fact that A is HPD means that α_{11} is real and positive and a Cholesky factor is then given by $\lambda_{11} = \sqrt{\alpha_{11}}$, with uniqueness if we insist that λ_{11} is positive.

2. Inductive step: Assume the result is true for $n = k$. We will show that it holds for $n = k + 1$.

Let $A \in \mathbb{C}^{(k+1) \times (k+1)}$ be HPD. Partition

$$A = \begin{pmatrix} \alpha_{11} & a_{21}^H \\ a_{21} & A_{22} \end{pmatrix} \text{ and } L = \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix}.$$

Let

- $\lambda_{11} = \sqrt{\alpha_{11}}$ (which is well-defined by Lemma 5.4.4.1,
- $l_{21} = a_{21}/\lambda_{11}$,
- $A_{22} - l_{21}l_{21}^H = L_{22}L_{22}^H$ (which exists as a consequence of the Inductive Hypothesis and Lemma 5.4.4.2.)

Then L is the desired Cholesky factor of A .

3. By the Principal of Mathematical Induction, the theorem holds.

■

5.4.5 Cholesky factorization and solving LLS



YouTube: <https://www.youtube.com/watch?v=C7LEuhS4H94>

Recall from Section 4.2 that the solution $\hat{x} \in \mathbb{C}^n$ to the linear least-squares (LLS) problem

$$\|b - A\hat{x}\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2 \quad (5.4.2)$$

equals the solution to the normal equations

$$\underbrace{A^H A}_B \hat{x} = \underbrace{A^H b}_y .$$

Since $A^H A$ is Hermitian, it would be good to take advantage of that special structure to factor it more cheaply. If $A^H A$ were HPD, then the Cholesky factorization can be employed. Fortunately, from Homework 5.4.1.1 we know that if A has linearly independent columns, then $A^H A$ is HPD. Thus, the steps required to solve the LLS problem (5.4.2) when $A \in \mathbb{C}^{m \times n}$ Are

- Form $B = A^H A$. Cost: approximately mn^2 flops.
- Factor $B = LL^H$ (Cholesky factorization). Cost: approximately $n^3/3$ flops.
- Compute $y = A^H b$. Cost: approximately $2mn$ flops.
- Solve $Lz = y$. Cost: approximately n^2 flops.
- Solve $L^H \hat{x} = z$. Cost: approximately n^2 flops.

for a total of, approximately, $mn^2 + n^3/3$ flops.

Ponder This 5.4.5.1 Consider $A \in \mathbb{C}^{m \times n}$ with linearly independent columns. Recall that A has a QR factorization, $A = QR$ where Q has orthonormal columns and R is an upper triangular matrix with positive diagonal elements. How are the Cholesky factorization of $A^H A$ and the QR factorization of A related?

5.4.6 Implementation with the classical BLAS

The Basic Linear Algebra Subprograms (BLAS) are an interface to commonly used fundamental linear algebra operations. In this section, we illustrate how the unblocked and blocked Cholesky factorization algorithms can be implemented in terms of the BLAS. The explanation draws from the entry we wrote for the BLAS in the Encyclopedia of Parallel Computing [38].

5.4.6.1 What are the BLAS?

The BLAS interface [24] [15] [14] was proposed to support portable high-performance implementation of applications that are matrix and/or vector computation intensive. The idea is that one casts computation in terms of the BLAS interface, leaving the architecture-specific optimization of that interface to an expert.

5.4.6.2 Implementation in Fortran

We start with a simple implementation in Fortran. A simple algorithm that exposes three loops and the corresponding code in Fortran are given by

```

for  $j := 0, \dots, n - 1$            do  $j=1, n$ 
     $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$        $A(j,j) = \text{sqrt}(A(j,j))$ 
    for  $i := j + 1, \dots, n - 1$ 
         $\alpha_{i,j} := \alpha_{i,j}/\alpha_{j,j}$     do  $i=j+1,n$ 
    end                                 $A(i,j) = A(i,j) / A(j,j)$ 
    for  $k := j + 1, \dots, n - 1$ 
        for  $i := k, \dots, n - 1$ 
             $\alpha_{i,k} := \alpha_{i,k} - \alpha_{i,j}\alpha_{k,j}$     enddo
        endfor                            do  $k=j+1,n$ 
    endfor                            do  $i=k,n$ 
    endfor                             $A(i,k) = A(i,k) - A(i,j) * A(k,j)$ 
    enddo                             enddo
    enddo                           enddo

```

Notice that Fortran starts indexing at one when addressing an array.

Next, exploit the fact that the BLAS interface supports a number of "vector-vector" operations known as the Level-1 BLAS. Of these, we will use

```
dscal( n, alpha, x, incx )
```

which updates the vector x stored in memory starting at address x and increment between entries of $incx$ and of size n with αx where α is stored in $alpha$, and

```
daxpy( n, alpha, x, incx, y, incy )
```

which updates the vector y stored in memory starting at address y and increment between entries of incy and of size n with $\alpha x + y$ where x is stored at address x with increment incx and α is stored in alpha . With these, the implementation becomes

```

do j=1, n
  A(j,j) = sqrt(A(j,j))
  call dscal( n-j, 1.0d00 / a(j,j), a(j+1,j), 1 )
for j := 0, ..., n - 1
   $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$ 
   $\alpha_{j+1:n-1,j} := \alpha_{j+1:n-1,j} / \alpha_{j,j}$ 
  for k := j + 1, ..., n - 1
     $\alpha_{k:n-1,k} := \alpha_{k:n-1,k} - \alpha_{k,j} \alpha_{k:n-1,j}$ 
    call daxpy( n-k+1, -A(k,j), A(k,j), 1,
    endfor
    A(k,k), 1 )
  endfor
enddo
enddo
```

$$\text{Here } \alpha_{j+1:n-1,j} = \begin{pmatrix} \alpha_{j+1,j} \\ \vdots \\ \alpha_{n-1,j} \end{pmatrix}.$$

The entire update $A_{22} := A_{22} - a_{21}a_{21}^T$ can be cast in terms of a matrix-vector operation (level-2 BLAS call) to

```
dsyr( uplo, n, alpha, x, A, lda )
```

which updates the matrix A stored in memory starting at address A with leading dimension lda of size n by n with $\alpha xx^T + A$ where x is stored at address x with increment incx and α is stored in alpha . Since both A and $\alpha xx^T + A$ are symmetric, only the triangular part indicated by uplo is updated. This is captured by the below algorithm and implementation.

```

do j=1, n
  A(j,j) = sqrt(A(j,j))
  call dscal( n-j, 1.0d00 / a(j,j), a(j+1,j), 1 )
for j := 0, ..., n - 1
   $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$ 
   $\alpha_{j+1:n-1,j} := \alpha_{j+1:n-1,j} / \alpha_{j,j}$ 
   $\alpha_{j+1:n-1,j+1:n-1} :=$ 
    call dsyr( "Lower triangular", n-j+1, -1.0d00,
     $- \text{tril}(\alpha_{j+1:n-1,j} \alpha_{j+1:n-1,j}^T)$ 
    A(j+1,j), 1, A(j+1,j+1), lda )
  enddo
enddo
```

Notice how the code that cast computation in terms of the BLAS uses a higher level of abstraction, through routines that implement the linear algebra operations that are encountered in the algorithms.

$A = \text{Chol-blocked-right-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12}^T \\ A_{20} & A_{21} & A_{22} \end{array} \right)$
$A_{11} := L_{11} = \text{Chol}(\cdot)A_{11}$
Solve $L_{21}L_{11}^{-H} = A_{21}$ overwriting A_{21} with L_{21}
$A_{22} := A_{22} - A_{21}A_{21}^H$ syrk: update only lower triangular part
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12}^T \\ A_{20} & A_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.4.6.1 Blocked Cholesky factorization Variant 3 (right-looking) algorithm. The operation "syrk" refers to "symmetric rank-k update", which performs a rank-k update (matrix-matrix multiplication with a small "k" size), updating only the lower triangular part of the matrix in this algorithm.

Finally, a blocked right-looking Cholesky factorization algorithm, which casts most computation in terms of a matrix-matrix multiplication operation referred to as a "symmetric rank-k update" is given in [Figure 5.4.6.1](#). There, we use FLAME notation to present the algorithm. It translates into Fortran code that exploits the BLAS given below.

```

do j=1, nb ,n
    jb = min( nb, n-j )
    Chol( jb, A( j, j ) );

    dtrsm( "Right", "Lower triangular", "Transpose",
           "Unit diag", jb, n-j-jb+1, 1.0d00, A( j,j ), LDA,
           A( j+jb, j ), LDA )

    dsyrk( "Lower triangular", n-j-jb+1, jb, 1.0d00,
           A( j+jb,j ), LDA, 1.0d00, A( j+jb, j+jb ), LDA )
enddo

```

The routines dtrsm and dsyrk are level-3 BLAS routines:

- The call to dtrsm implements $A_{21} := L_{21}$ where $L_{21}L_{11}^T = A_{21}$.
- The call to dsyrk implements $A_{22} := -A_{21}A_{21}^T + A_{22}$, updating only the lower triangular part of the matrix.

The bulk of the computation is now cast in terms of matrix-matrix operations which can achieve high performance.

5.5 Enrichments

5.5.1 Other LU factorization algorithms

There are actually five different (unblocked) algorithms for computing the LU factorization that were discovered over the course of the centuries. Here we show how to systematically derive all five. For details, we suggest Week 6 of our Massive Open Online Course titled "LAFF-On Programming for Correctness" [28].

Remark 5.5.1.1 To put yourself in the right frame of mind, we highly recommend you spend about an hour reading the paper

- [31] Devangi N. Parikh, Margaret E. Myers, Richard Vuduc, Robert A. van de Geijn, A Simple Methodology for Computing Families of Algorithms, FLAME Working Note #87, The University of Texas at Austin, Department of Computer Science, Technical Report TR-18-06. [arXiv:1808.07832](https://arxiv.org/abs/1808.07832).

$A = \text{LU-var1}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$

⋮
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.5.1.2 LU factorization algorithm skeleton.

Finding the algorithms starts with the following observations.

- Our algorithms will overwrite the matrix A , and hence we introduce \hat{A} to denote the original contents of A . We will say that the precondition for the algorithm is that

$$A = \hat{A}$$

(A starts by containing the original contents of A .)

- We wish to overwrite A with L and U . Thus, the postcondition for the algorithm (the state in which we wish to exit the algorithm) is that

$$A = L \setminus U \wedge LU = \hat{A}$$

(A is overwritten by L below the diagonal and U on and above the diagonal, where multiplying L and U yields the original matrix A .)

- All the algorithms will march through the matrices from top-left to bottom-right, giving us the code skeleton in [Figure 5.5.1.2](#). Since the computed L and U overwrite A , throughout they are partitioned conformal to (in the same way) as is A .
- Thus, before and after each iteration of the loop the matrices are viewed as quadrants:

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right), \text{ and } U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right).$$

where A_{TL} , L_{TL} , and U_{TL} are all square and equally sized.

- In terms of these exposed quadrants, in the end we wish for matrix A to contain

$$\begin{aligned} \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) &= \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & L \setminus U_{BR} \end{array} \right) \\ \wedge \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right) \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right) &= \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right) \end{aligned}$$

- Manipulating this yields what we call the Partitioned Matrix Expression (PME), which can be viewed as a recursive definition of the LU factorization:

$$\begin{aligned} \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) &= \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & L \setminus U_{BR} \end{array} \right) \\ \wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \Bigg| \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BR}U_{BR} = \hat{A}_{BR} - L_{BL}U_{TR}} \end{aligned}$$

- Now, consider the code skeleton for the LU factorization in [Figure 5.5.1.2](#). At the top of the loop (right after the **while**), we want to maintain certain contents in matrix A . Since we are in a loop, we haven't yet overwritten A with the final result. Instead, some progress toward this final result has been made. The way we can find what the state of A is that we would like to maintain is to take the PME and delete subexpressions. For example, consider the following condition on the contents of A :

$$\begin{aligned} \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) &= \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline \hat{A}_{BR} - L_{BL}U_{TR} & \end{array} \right) \\ \wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \Bigg| \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{\quad\quad\quad} \end{aligned} .$$

What we are saying is that A_{TL} , A_{TR} , and A_{BL} have been completely updated with the corresponding parts of L and U , and A_{BR} has been partially updated. This is exactly the state that the right-looking algorithm that we discussed in [Subsection 5.2.2](#) maintains! What is left is to factor A_{BR} , since it contains $\hat{A}_{BR} - L_{BL}U_{TR}$, and $\hat{A}_{BR} - L_{BL}U_{TR} = L_{BR}U_{BR}$.

- By carefully analyzing the order in which computation must occur (in compiler lingo: by performing a dependence analysis), we can identify five states that can be maintained at the top of the loop, by deleting subexpressions from the PME. These are called loop invariants. There are five for LU factorization:

$$\begin{array}{c|c}
 \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) & \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) \\
 \text{Invariant 1} & \text{Invariant 2} \\
 \hline
 \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & \widehat{A}_{TR} \\ \hline \widehat{L}_{BL} & \widehat{A}_{BR} \end{array} \right) & \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} \end{array} \right) \\
 \text{Invariant 3} & \text{Invariant 4} \\
 \hline
 \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} - L_{BL}U_{TR} \end{array} \right) & \\
 \text{Invariant 5} &
 \end{array}$$

- Key to figuring out what updates must occur in the loop for each of the variants is to look at how the matrices are repartitioned at the top and bottom of the loop body.

For each of the five algorithms for LU factorization, we will derive the loop invariant, and then derive the algorithm from the loop invariant.

5.5.1.1 Variant 1: Bordered algorithm

Consider the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) \wedge L_{TL}U_{TL} = \widehat{A}_{TL},$$

meaning that the leading principal submatrix A_{TL} has been overwritten with its LU factorization, and the remainder of the matrix has not yet been touched.

At the top of the loop, after repartitioning, A then contains

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} L \setminus U_{00} & \widehat{a}_{01} & \widehat{A}_{02} \\ \hline \widehat{a}_{10}^T & \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array} \right) \wedge L_{00}U_{00} = \widehat{A}_{00}$$

while after updating A it must contain

$$\underbrace{\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} L \setminus U_{00} & u_{01} & \widehat{A}_{02} \\ \hline l_{10}^T & v_{11} & \widehat{a}_{12}^T \\ \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array} \right)}_{\begin{array}{l} L_{00}U_{00} = \widehat{A}_{00} \\ l_{10}^T U_{00} = \widehat{a}_{10}^T \end{array}} \wedge \underbrace{\left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right) = \left(\begin{array}{cc} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{array} \right)}_{\begin{array}{l} L_{00}u_{01} = \widehat{a}_{01} \\ l_{10}^T u_{01} + v_{11} = \widehat{\alpha}_{11} \end{array}}$$

for the loop invariant to again hold after the iteration. Here the entries in red are known (in addition to the ones marked with a "hat") and the entries in blue are to be computed. With this, we can compute the desired parts of L and U :

- Solve $L_{00}u_{01} = a_{01}$, overwriting a_{01} with the result. (Notice that $a_{01} = \hat{a}_{01}$ before this update.)
- Solve $l_{10}^T U_{00} = a_{10}^T$ (or, equivalently, $U_{00}^T(l_{10}^T)^T = (a_{10}^T)^T$ for l_{10}^T), overwriting a_{10}^T with the result. (Notice that $a_{10}^T = \hat{a}_{10}^T$ before this update.)
- Update $\alpha_{11} := v_{11} = \alpha_{11} - l_{10}^T u_{01}$. (Notice that by this computation, $a_{10}^T = l_{10}^T$ and $a_{01} = u_{01}$.)

The resulting algorithm is captured in [Figure 5.5.1.3](#).

$A = \text{LU-var1}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$ A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
Solve $L_{00}u_{01} = a_{01}$ overwriting a_{01} with the result Solve $l_{10}^T U_{00} = a_{10}^T$ overwriting a_{10}^T with the result $\alpha_{11} := v_{11} = \alpha_{11} - a_{10}^T a_{01}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$
endwhile

Figure 5.5.1.3 Variant 1 (bordered) LU factorization algorithm. Here A_{00} stores $L \setminus U_{00}$.

Homework 5.5.1.1 If A is $n \times n$, show the cost of Variant 1 is approximately $\frac{2}{3}n^3$.

Solution. During the k th iteration, A_{00} is $k \times k$, for $k = 0, \dots, n-1$. Then the (approximate) cost of each of the steps is given by

- Solve $L_{00}u_{01} = a_{01}$, overwriting a_{01} with the result. Cost: approximately k^2 flops.
- Solve $l_{10}^T U_{00} = a_{10}^T$ (or, equivalently, $U_{00}^T(l_{10}^T)^T = (a_{10}^T)^T$ for l_{10}^T), overwriting a_{10}^T with the result. Cost: approximately k^2 flops.
- Compute $v_{11} = \alpha_{11} - l_{10}^T u_{01}$, overwriting α_{11} with the result. Cost: $2k$ flops.

Thus, the total cost is given by

$$\sum_{k=0}^{n-1} (k^2 + k^2 + 2k) \approx 2 \sum_{k=0}^{n-1} k^2 \approx 2 \frac{1}{3} n^3 = \frac{2}{3} n^3.$$

5.5.1.2 Variant 2: Up-looking algorithm

Consider next the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) \wedge L_{TL}U_{TL} = \widehat{A}_{TL} \mid L_{TL}U_{TR} = \widehat{A}_{TR}$$

meaning that the leading principal submatrix A_{TL} has been overwritten with its LU factorization and U_{TR} has overwritten A_{TR} .

At the top of the loop, after repartitioning, A then contains

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} L \setminus U_{00} & u_{01} & U_{02} \\ \hline \widehat{a}_{10}^T & \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array} \right)$$

$$\wedge \underbrace{L_{00}U_{00} = \widehat{A}_{00} \mid L_{00} \left(\begin{array}{cc} u_{01} & U_{02} \end{array} \right) = \left(\begin{array}{cc} \widehat{a}_{01} & \widehat{A}_{02} \end{array} \right)}_{L_{00}U_{00} = \widehat{A}_{00} \mid L_{00}u_{01} = \widehat{a}_{01} \quad L_{00}U_{02} = \widehat{A}_{02}}$$

while after updating A it must contain

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} L \setminus U_{00} & u_{01} & U_{02} \\ \hline l_{10}^T & v_{11} & u_{12}^T \\ \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array} \right)$$

$$\wedge \underbrace{\left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right) = \left(\begin{array}{cc} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{array} \right) \mid \left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c} U_{02} \\ u_{12}^T \end{array} \right) = \left(\begin{array}{c} \widehat{A}_{00} \\ \widehat{a}_{12}^T \end{array} \right)}_{L_{00}U_{00} = \widehat{A}_{00} \quad L_{00}u_{01} = \widehat{a}_{01} \quad L_{00}U_{02} = \widehat{A}_{02} \quad l_{10}^T U_{00} = \widehat{a}_{10}^T \quad l_{10}^T u_{01} + v_{11} = \widehat{\alpha}_{11} \quad l_{10}^T U_{02} + u_{12}^T = \widehat{a}_{12}^T}$$

for the loop invariant to again hold after the iteration. Here, again, the entries in red are known (in addition to the ones marked with a "hat") and the entries in blue are to be computed. With this, we can compute the desired parts of L and U :

- Solve $l_{10}^T U_{00} = a_{10}^T$, overwriting a_{10}^T with the result.
- Update $\alpha_{11} := v_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$.
- Update $a_{12}^T := u_{12}^T = a_{12}^T - l_{10}^T U_{02} = a_{12}^T - a_{10}^T A_{02}$.

The resulting algorithm is captured in [Figure 5.5.1.4](#).

$A = \text{LU-var2}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
Solve $l_{10}^T U_{00} = a_{10}^T$ overwriting a_{10}^T with the result
$\alpha_{11} := v_{11} = \alpha_{11} - a_{10}^T a_{01}$
$a_{12}^T := u_{12}^T = a_{12}^T - l_{10}^T U_{02}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.5.1.4 Variant 2 (up-looking) LU factorization algorithm. Here A_{00} stores $L \setminus U_{00}$.

Homework 5.5.1.2 If A is $n \times n$, show the cost of Variant 2 is approximately $\frac{2}{3}n^3$.

Solution. During the k th iteration, A_{00} is $k \times k$, for $k = 0, \dots, n-1$. Then the (approximate) cost of each of the steps is given by

- Solve $l_{10}^T U_{00} = a_{10}^T$, overwriting a_{10}^T with the result. Approximate cost: k^2 flops.
- Update $\alpha_{11} := v_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$. Approximate cost: $2k$ flops.
- Update $a_{12}^T := u_{12}^T = a_{12}^T - l_{10}^T U_{02} = a_{12}^T - a_{10}^T A_{02}$. Approximate cost: $2k(n-k-1)$ flops.

Thus, the total cost is approximately given by

$$\begin{aligned}
 & \sum_{k=0}^{n-1} (k^2 + 2k + 2k(n-k-1)) \\
 & \quad = \quad < \text{simplify} > \\
 & \sum_{k=0}^{n-1} (2kn - k^2) \\
 & \quad = \quad < \text{algebra} > \\
 & 2n \sum_{k=0}^{n-1} k - \sum_{k=0}^{n-1} k^2 \\
 & \quad \approx \quad < \sum_{k=0}^{n-1} k \approx n^2/2; \sum_{k=0}^{n-1} k^2 \approx k^3/3 > \\
 & n^3 - \frac{n^3}{3} \\
 & \quad = \quad < \text{simplify} > \\
 & \frac{2}{3}n^3.
 \end{aligned}$$

5.5.1.3 Variant 3: Left-looking algorithm

Consider the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & \hat{A}_{TR} \\ \hline L_{BL} & \hat{A}_{BR} \end{array} \right) \wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BxL}} \quad \boxed{\quad}$$

At the top of the loop, after repartitioning, A then contains

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} L \setminus U_{00} & \hat{a}_{01} & \hat{A}_{02} \\ \hline l_{10}^T & \hat{\alpha}_{11} & \hat{a}_{12}^T \\ L_{20} & \hat{a}_{21} & \hat{A}_{22} \end{array} \right) \wedge \frac{L_{00}U_{00} = \hat{A}_{00}}{l_{10}^T U_{00} = \hat{a}_{10}^T} \quad \frac{L_{20}U_{00} = \hat{A}_{20}}{L_{20}U_{00} = \hat{A}_{20}}$$

while after updating A it must contain

$$\begin{aligned} \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) &= \left(\begin{array}{c|cc} L \setminus U_{00} & u_{01} & \hat{A}_{02} \\ \hline l_{10}^T & v_{11} & \hat{a}_{12}^T \\ L_{20} & l_{21} & \hat{A}_{22} \end{array} \right) \\ \wedge \underbrace{\left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right)}_{\left(\begin{array}{cc} L_{20} & l_{21} \end{array} \right) \left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right)} &= \left(\begin{array}{cc} \hat{A}_{00} & \hat{a}_{01} \\ \hat{a}_{10}^T & \hat{\alpha}_{11} \end{array} \right) \\ &\quad \boxed{\begin{array}{c} L_{00}U_{00} = \hat{A}_{00} & L_{00}u_{01} = \hat{a}_{01} \\ l_{10}^T U_{00} = \hat{a}_{10}^T & l_{10}^T u_{01} + v_{11} = \hat{\alpha}_{11} \\ L_{20}U_{00} = \hat{A}_{20} & L_{20}u_{01} + l_{21}v_{11} = \hat{a}_{21} \end{array}} \end{aligned}$$

for the loop invariant to again hold after the iteration. With this, we can compute the desired parts of L and U :

- Solve $L_{00}u_{01} = a_{01}$, overwriting a_{01} with the result.
- Update $\alpha_{11} := v_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$.
- Update $a_{21} := l_{21} = (a_{21} - L_{20}u_{01})/v_{11} = (a_{21} - A_{20}a_{10})/\alpha_{11}$.

The resulting algorithm is captured in [Figure 5.5.1.5](#).

$A = \text{LU-var3}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
Solve $L_{00}u_{01} = a_{01}$ overwriting a_{01} with the result
$\alpha_{11} := v_{11} = \alpha_{11} - a_{10}^T a_{01}$
$a_{21} := l_{21} = (a_{21} - A_{20}a_{10})/\alpha_{11}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$
endwhile

Figure 5.5.1.5 Variant 3 (left-looking) LU factorization algorithm. Here A_{00} stores $L \setminus U_{00}$.

Homework 5.5.1.3 If A is $n \times n$, show the cost of Variant 3 is approximately $\frac{2}{3}n^3$.

Solution. During the k th iteration, A_{00} is $k \times k$, for $k = 0, \dots, n-1$. Then the (approximate) cost of each of the steps is given by

- Solve $L_{00}u_{01} = a_{01}$, overwriting a_{01} with the result. Approximate cost: k^2 flops.
- Update $\alpha_{11} := v_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$. Approximate cost: $2k$ flops.
- Update $a_{21} := l_{21} = (a_{21} - L_{20}u_{01})/v_{11} = (a_{21} - A_{20}a_{10})/\alpha_{11}$. Approximate cost: $2(n-k-1)$ flops.

Thus, the total cost is approximately given by

$$\begin{aligned}
 & \sum_{k=0}^{n-1} (k^2 + 2k + 2k(n-k-1)) \\
 & \quad = \quad < \text{simplify} > \\
 & \sum_{k=0}^{n-1} (2kn - k^2) \\
 & \quad = \quad < \text{algebra} > \\
 & 2n \sum_{k=0}^{n-1} k - \sum_{k=0}^{n-1} k^2 \\
 & \quad \approx \quad < \sum_{k=0}^{n-1} k \approx n^2/2; \sum_{k=0}^{n-1} k^2 \approx k^3/3 > \\
 & n^3 - \frac{n^3}{3} \\
 & \quad = \quad < \text{simplify} > \\
 & \frac{2}{3}n^3.
 \end{aligned}$$

5.5.1.4 Variant 4: Crout variant

Consider next the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} \end{array} \right) \wedge \frac{L_{TL}U_{TL} = \widehat{A}_{TL}}{L_{BL}U_{TL} = \widehat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \widehat{A}_{TR}}{L_{BL}U_{TR} = \widehat{A}_{BL}}$$

At the top of the loop, after repartitioning, A then contains

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} L \setminus U_{00} & u_{01} & U_{02} \\ \hline l_{10}^T & \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ L_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array} \right)$$

$$\wedge \frac{L_{00}U_{00} = \widehat{A}_{00}}{l_{10}^T U_{00} = \widehat{a}_{10}^T} \quad \frac{L_{00}u_{01} = \widehat{a}_{01}}{L_{20}u_{01} + l_{21}v_{11} = \widehat{a}_{21}} \quad \frac{L_{00}U_{02} = \widehat{A}_{02}}{l_{10}^T U_{02} + u_{12}^T = \widehat{a}_{12}^T}$$

while after updating A it must contain

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc} L \setminus U_{00} & u_{01} & U_{02} \\ \hline l_{10}^T & v_{11} & u_{12}^T \\ L_{20} & l_{21} & \widehat{A}_{22} \end{array} \right)$$

$$\wedge \underbrace{\left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right)}_{\left(\begin{array}{cc} L_{20} & l_{21} \end{array} \right) \left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right)} = \left(\begin{array}{cc} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{array} \right) \left| \begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c} U_{02} \\ u_{12}^T \end{array} \right) = \left(\begin{array}{c} \widehat{A}_{02} \\ \widehat{a}_{12}^T \end{array} \right)$$

$$\underbrace{\left(\begin{array}{cc} L_{00}U_{00} = \widehat{A}_{00} \\ l_{10}^T U_{00} = \widehat{a}_{10}^T \\ L_{20}U_{00} = \widehat{A}_{20} \end{array} \right) \left(\begin{array}{cc} L_{00}u_{01} = \widehat{a}_{01} \\ l_{10}^T u_{01} + v_{11} = \widehat{a}_{11} \\ L_{20}u_{01} + l_{21}v_{11} = \widehat{a}_{21} \end{array} \right)}_{\left(\begin{array}{cc} L_{00}U_{02} = \widehat{A}_{02} \\ l_{10}^T U_{02} + u_{12}^T = \widehat{a}_{12}^T \end{array} \right)}$$

for the loop invariant to again hold after the iteration. With this, we can compute the desired parts of L and U :

- Update $\alpha_{11} := v_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$.
- Update $a_{12}^T := u_{12}^T = a_{12}^T - l_{10}^T U_{02} = a_{12}^T - a_{10}^T A_{02}$.
- Update $a_{21} := l_{21} = (a_{21} - L_{20}u_{01})/v_{11} = (a_{21} - A_{20}a_{10})/\alpha_{11}$.

The resulting algorithm is captured in [Figure 5.5.1.6](#).

$A = \text{LU-var4}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$\alpha_{11} := v_{11} = \alpha_{11} - a_{10}^T a_{01}$
$a_{12}^T := u_{12}^T = a_{12}^T - a_{10}^T A_{02}$
$a_{21} := l_{21} = (a_{21} - A_{20} a_{10}) / \alpha_{11}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.5.1.6 Variant 4 (Crout) LU factorization algorithm.

Homework 5.5.1.4 If A is $n \times n$, show the cost of Variant 4 is approximately $\frac{2}{3}n^3$.

Solution. During the k th iteration, A_{00} is $k \times k$, for $k = 0, \dots, n-1$. Then the (approximate) cost of each of the steps is given by

- Update $\alpha_{11} := v_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$. Approximate cost: $2k$ flops.
- Update $a_{12}^T := u_{12}^T = a_{12}^T - l_{10}^T U_{02} = a_{12}^T - a_{10}^T A_{02}$. Approximate cost: $2k(n-k-1)$ flops.
- Update $a_{21} := l_{21} = (a_{21} - L_{20} u_{01}) / v_{11} = (a_{21} - A_{20} a_{10}) / \alpha_{11}$. Approximate cost: $2k(n-k-1) + (n-k-1)$ flops.

Thus, ignoring the $2k$ flops for the dot product and the $n-k-1$ flops for multiplying with $1/\alpha_{11}$ in each iteration, the total cost is approximately given by

$$\begin{aligned}
& \sum_{k=0}^{n-1} 4k(n-k-1) \\
& \approx \quad < \text{remove lower order term} > \sum_{k=0}^{n-1} 4k(n-k) \\
& = \quad < \text{algebra} > \\
& 4n \sum_{k=0}^{n-1} k - 4 \sum_{k=0}^{n-1} k^2 \\
& \approx \quad < \sum_{k=0}^{n-1} k \approx n^2/2; \sum_{k=0}^{n-1} k^2 \approx k^3/3 > \\
& 2n^3 - 4 \frac{n^3}{3} \\
& = \quad < \text{simplify} > \\
& \frac{2}{3}n^3.
\end{aligned}$$

5.5.1.5 Variant 5: Classical Gaussian elimination

Consider final loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} - L_{BL}U_{TR} \end{array} \right) \wedge \frac{L_{TL}U_{TL} = \widehat{A}_{TL}}{L_{BL}U_{TL} = \widehat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \widehat{A}_{TR}}{L_{BL}U_{TR} = \widehat{A}_{BL}}$$

At the top of the loop, after repartitioning, A then contains

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|ccc} L \setminus U_{00} & u_{01} & & U_{02} \\ \hline l_{10}^T & \widehat{\alpha}_{11} - l_{10}^T u_{01} & \widehat{a}_{12}^T - l_{10}^T U_{02} & \\ L_{20} & \widehat{a}_{21} - L_{20} u_{01} & \widehat{A}_{22} - L_{20} U_{02} & \\ \hline \end{array} \right)$$

$$\wedge \left. \begin{array}{c|c} \frac{L_{00}U_{00} = \widehat{A}_{00}}{l_{10}^T U_{00} = \widehat{a}_{10}^T} & L_{00}u_{01} = \widehat{a}_{01} \\ \hline L_{20}U_{00} = \widehat{A}_{20} & L_{20}u_{01} = \widehat{a}_{21} \end{array} \right|$$

while after updating A it must contain

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|ccc} L \setminus U_{00} & u_{01} & & U_{02} \\ \hline l_{10}^T & v_{11} & & u_{12}^T \\ L_{20} & l_{21} & & \widehat{A}_{22} - l_{21} u_{12}^T, \\ \hline \end{array} \right)$$

$$\wedge \underbrace{\left(\begin{array}{c|c} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c|c} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right) = \left(\begin{array}{c|c} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{array} \right)}_{\left(\begin{array}{c|c} L_{20} & l_{21} \\ 0 & v_{11} \end{array} \right) \left(\begin{array}{c|c} U_{00} & u_{01} \\ 0 & v_{11} \end{array} \right) = \left(\begin{array}{c|c} \widehat{A}_{20} & \widehat{a}_{21} \end{array} \right)} \left(\begin{array}{c|c} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c|c} U_{02} \\ u_{12}^T \end{array} \right) = \left(\begin{array}{c|c} \widehat{A}_{02} \\ \widehat{a}_{12}^T \end{array} \right)}$$

$$\left. \begin{array}{c|c} \overbrace{L_{00}U_{00} = \widehat{A}_{00} \\ l_{10}^T U_{00} = \widehat{a}_{10}^T}^{L_{20}U_{00} = \widehat{A}_{20}} & \overbrace{L_{00}u_{01} = \widehat{a}_{01} \\ l_{10}^T u_{01} + v_{11} = \widehat{\alpha}_{11}}^{L_{20}u_{01} + l_{21}v_{11} = \widehat{a}_{21}} \\ \hline L_{00}U_{02} = \widehat{A}_{02} & l_{10}^T U_{02} + u_{12}^T = \widehat{a}_{12}^T \end{array} \right|$$

for the loop invariant to again hold after the iteration. With this, we can compute the desired parts of L and U :

- $\alpha_{11} := v_{11} = \widehat{\alpha}_{11} - l_{10}^T u_{01} = \alpha_{11}$ (no-op).
(α_{11} already equals $\widehat{\alpha}_{11} - l_{01}^T u_{01}$.)
- $a_{12}^T := u_{12}^T = \widehat{a}_{12}^T - l_{10}^T U_{02} = a_{12}^T$ (no-op).
(a_{12}^T already equals $\widehat{a}_{12}^T - l_{01}^T U_{02}$.)
- Update $a_{21} := (\widehat{a}_{21} - L_{20}u_{01})/v_{11} = a_{21}/\alpha_{11}$.
(a_{21} already equals $\widehat{a}_{21} - L_{20}u_{01}$.)
- Update $A_{22} := \widehat{A}_{22} - L_{20}U_{02} - l_{21}u_{12}^T = A_{22} - a_{21}a_{12}^T$
(A_{22} already equals $\widehat{A}_{22} - L_{20}U_{02}$.)

The resulting algorithm is captured in [Figure 5.5.1.7](#).

$A = \text{LU-var5}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$\underline{a_{21} := l_{21} = a_{21}/\alpha_{11}}$
$\underline{A_{22} := A_{22} - a_{21}a_{12}^T}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$
endwhile

Figure 5.5.1.7 Variant 5 (classical Gaussian elimination) LU factorization algorithm.

Homework 5.5.1.5 If A is $n \times n$, show the cost of Variant 5 is approximately $\frac{2}{3}n^3$.

Solution. During the k th iteration, A_{00} is $k \times k$, for $k = 0, \dots, n-1$. Then the (approximate) cost of each of the steps is given by

- Update $a_{21} := l_{21} = a_{21}/\alpha_{11}$. Approximate cost: k flops.
- Update $A_{22} := A_{22} - l_{21}u_{12}^T = A_{22} - a_{21}a_{12}^T$. Approximate cost: $2(n-k-1)(n-k-1)$ flops.

Thus, ignoring $n-k-1$ flops for multiplying with $1/\alpha_{11}$ in each iteration, the total cost is approximately given by

$$\begin{aligned} & \sum_{k=0}^{n-1} 2(n-k-1)^2 \\ &= <\text{change of variable } j = n-k-1> \\ & 2 \sum_{j=0}^{n-1} j^2 \\ & \approx <\sum_{k=0}^{n-1} k^2 \approx k^3/3> \\ & 2 \frac{n^3}{3} \end{aligned}$$

5.5.1.6 Discussion

Remark 5.5.1.8 For a discussion of the different LU factorization algorithms that also gives a historic perspective, we recommend "Matrix Algorithms Volume 1" by G.W. Stewart [\[37\]](#).

5.5.2 Blocked LU factorization

Recall from [Subsection 3.3.4](#) that casting computation in terms of matrix-matrix multiplication facilitates high performance. In this unit we very briefly illustrate how the right-looking LU factorization can be reformulated as such a "blocked" algorithm. For details on other blocked LU factorization algorithms and blocked Cholesky factorization algorithms, we once again refer the interested reader to our Massive Open Online Course titled "LAFF-On Programming for Correctness" [\[28\]](#). We will revisit these kinds of issues in the final week of this course.

Consider $A = LU$ and partition these matrices as

$$A \rightarrow \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right), L \rightarrow \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right), U \rightarrow \left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array} \right),$$

where A_{11} , L_{11} , and U_{11} are $b \times b$ submatrices. Then

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right) \left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array} \right) = \left(\begin{array}{c|c} L_{11}U_{11} & L_{11}A_{12} \\ \hline A_{21}U_{11} & A_{22} - L_{21}U_{12} \end{array} \right).$$

From this we conclude that

$$\begin{array}{c|c} A_{11} = L_{11}U_{11} & A_{12} = L_{11}U_{12} \\ \hline A_{21} = L_{21}U_{11} & A_{22} - L_{21}U_{12} = L_{22}U_{22}. \end{array}$$

This suggests the following steps:

- Compute the LU factorization of A_{11} (e.g., using any of the "unblocked" algorithms from [Subsection 5.5.1](#)).

$$A_{11} = L_{11}U_{11},$$

overwriting A_{11} with the factors.

- Solve

$$L_{11}U_{12} = A_{12}$$

for U_{12} , overwriting A_{12} with the result. This is known as a "triangular solve with multiple right-hand sides." This comes from the fact that solving

$$LX = B,$$

where L is lower triangular, can be reformulated by partitioning X and B by columns,

$$\underbrace{\left(\begin{array}{c|c|c} x_0 & x_1 & \cdots \\ \hline Lx_0 & Lx_1 & \cdots \end{array} \right)}_{L\left(\begin{array}{c|c|c} x_0 & x_1 & \cdots \\ \hline Lx_0 & Lx_1 & \cdots \end{array} \right)} = \left(\begin{array}{c|c|c} b_0 & b_1 & \cdots \end{array} \right),$$

which exposes that for each pair of columns we must solve the unit lower triangular system $Lx_j = b_j$.

- Solve

$$L_{21}U_{11} = A_{21}$$

for L_{21} , overwriting A_{21} with the result. This is also a "triangular solve with multiple right-hand sides" since we can instead view it as solving the lower triangular system with multiple right-hand sides

$$U_{11}^T L_{21}^T = A_{21}^T.$$

(In practice, the matrices are *not* transposed.)

- Update

$$A_{22} := A_{22} - L_{21}U_{12}.$$

- Proceed by computing the LU factorization of the updated A_{22} .

This motivates the algorithm in [Figure 5.5.2.1](#).

$A = \text{LU-blk-var5}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{array} \right)$
$A_{11} := \text{LU}(A_{11})$ L_{11} and U_{11} overwrite A_{11}
Solve $L_{11}U_{12} = A_{12}$ overwriting A_{12} with U_{12}
Solve $L_{21}U_{11} = A_{21}$ overwriting A_{21} with L_{21}
$A_{22} := A_{22} - A_{21}A_{12}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.5.2.1 Blocked Variant 5 (classical Gaussian elimination) LU factorization algorithm.

The important observation is that if A is $m \times m$ and b is much smaller than m , then most of the computation is in the matrix-matrix multiplication $A_{22} := A_{22} - A_{21}A_{12}$.

Remark 5.5.2.2 For each (unblocked) algorithm in [Subsection 5.5.1](#), there is a corresponding blocked algorithm.

5.6 Wrap Up

5.6.1 Additional homework

In this chapter, we discussed how the LU factorization (with pivoting) can be used to solve $Ax = y$. Why don't we instead discuss how to compute the inverse of the matrix A and compute $x = A^{-1}y$? Through a sequence of exercises, we illustrate why one should (almost) never compute the inverse of a matrix.

Homework 5.6.1.1 Let $A \in \mathbb{C}^{m \times m}$ be nonsingular and B its inverse. We know that $AB = I$ and hence

$$A \left(\begin{array}{c|c|c} b_0 & \cdots & b_{m-1} \end{array} \right) = \left(\begin{array}{c|c|c} e_0 & \cdots & e_{m-1} \end{array} \right),$$

where e_j can be thought of as the standard basis vector indexed with j or the column of I indexed with j .

1. Justify the following algorithm for computing B :

```

for  $j = 0, \dots, m - 1$ 
    Compute the LU factorization with pivoting :  $P(p)A = LU$ 
    Solve  $Lz = P(p)e_j$ 
    Solve  $Ub_j = z$ 
endfor

```

2. What is the cost, in flops, of the above algorithm?
3. How can we reduce the cost in the most obvious way and what is the cost of this better algorithm?
4. If we want to solve $Ax = y$ we can now instead compute $x = By$. What is the cost of this multiplication and how does this cost compare with the cost of computing it via the LU factorization, once the LU factorization has already been computed:

```

Solve  $Lz = P(p)y$ 
Solve  $Ux = z$ 

```

What do we conclude about the wisdom of computing the inverse?

Homework 5.6.1.2 Let L be a unit lower triangular matrix. Partition

$$L = \left(\begin{array}{c|c} 1 & 0 \\ l_{21} & L_{22} \end{array} \right).$$

1. Show that

$$L^{-1} = \left(\begin{array}{c|c} 1 & 0 \\ -L_{22}^{-1}l_{21} & L_{22}^{-1} \end{array} \right).$$

2. Use the insight from the last part to complete the following algorithm for computing the inverse of a unit lower triangular matrix:

$[L] = \text{inv}(L)$
$L \rightarrow \left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right)$
L_{TL} is 0×0
while $n(L_{TL}) < n(L)$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right)$
$l_{21} :=$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right)$
endwhile

3. The correct algorithm in the last part will avoid inverting matrices and will require, approximately, $\frac{1}{3}m^3$ flops. Analyze the cost of your algorithm.

Homework 5.6.1.3 LINPACK, the first software package for computing various operations related to solving (dense) linear systems, includes routines for inverting a matrix. When a survey was conducted to see what routines were in practice most frequently used, to the dismay of the developers, it was discovered that routine for inverting matrices was among them. To solve $Ax = y$ users were inverting A and then computing $x = A^{-1}y$. For this reason, the successor to LINPACK, LAPACK, does not even include a routine for inverting a matrix. Instead, if a user wants to compute the inverse, the user must go through the steps.

Compute the LU factorization with pivoting : $P(p)A = LU$

Invert L , overwriting L with the result

Solve $UX = L$ for X

Compute $A^{-1} := XP(p)$ (permuting the columns of X)

1. Justify that the described steps compute A^{-1} .
2. Propose an algorithm for computing X that solves $UX = L$. Be sure to take advantage of the triangular structure of U and L .
3. Analyze the cost of the algorithm in the last part of this question. If you did it right, it should require, approximately, m^3 operations.
4. What is the total cost of inverting the matrix?

5.6.2 Summary

The process known as Gaussian elimination is equivalent to computing the LU factorization of the matrix $A \in \mathbb{C}^{m \times m}$

$$: A = LU,$$

where L is a unit lower triangular matrix and U is an upper triangular matrix.

Definition 5.6.2.1 Given a matrix $A \in \mathbb{C}^{m \times n}$ with $m \geq n$, its LU factorization is given by $A = LU$ where $L \in \mathbb{C}^{m \times n}$ is unit lower trapezoidal and $U \in \mathbb{C}^{n \times n}$ is upper triangular with nonzeros on its diagonal. \diamond

Definition 5.6.2.2 Principal leading submatrix. For $k \leq n$, the $k \times k$ principal leading submatrix of a matrix A is defined to be the square matrix $A_{TL} \in \mathbb{C}^{k \times k}$ such that $A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$. \diamond

Lemma 5.6.2.3 Let $L \in \mathbb{C}^{n \times n}$ be a unit lower triangular matrix and $U \in \mathbb{C}^{n \times n}$ be an upper triangular matrix. Then $A = LU$ is nonsingular if and only if U has no zeroes on its diagonal.

Theorem 5.6.2.4 Existence of the LU factorization. Let $A \in \mathbb{C}^{m \times n}$ and $m \geq n$ have linearly independent columns. Then A has a (unique) LU factorization if and only if all its principal leading submatrices are nonsingular.

$A = \text{LU-right-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$a_{21} := a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{12}^T$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.6.2.5 Right-looking LU factorization algorithm.

The right-looking algorithm performs the same computations as the algorithm

```

for  $j := 0, \dots, n - 1$ 
  for  $i := j + 1, \dots, n - 1$ 
     $\lambda_{i,j} := \alpha_{i,j}/\alpha_{j,j}$ 
     $\alpha_{i,j} := 0$ 
  endfor
  for  $i := j + 1, \dots, n - 1$ 
    for  $k = j + 1, \dots, n - 1$ 
       $\alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j}\alpha_{j,k}$ 
    endfor
  endfor
endfor

```

$A = \text{LU-left-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
Solve $L_{00}u_{01} = a_{01}$ overwriting a_{01} with u_{01}
$\alpha_{11} := v_{11} = \alpha_{11} - a_{10}^T a_{01}$
$a_{21} := a_{21} - A_{20}a_{01}$
$a_{21} := l_{21} = a_{21}/\alpha_{11}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$
endwhile

Figure 5.6.2.6 Left-looking LU factorization algorithm. L_{00} is the unit lower triangular matrix stored in the strictly lower triangular part of A_{00} (with the diagonal implicitly stored).

Solving $Ax = b$ via LU factorization:

- Compute the LU factorization $A = LU$.
- Solve $Lz = b$.
- Solve $Ux = z$.

Cost of LU factorization: Starting with an $m \times n$ matrix A , LU factorization requires approximately $mn^2 - \frac{1}{3}n^3$ flops. If $m = n$ this becomes

$$\frac{2}{3}n^3 \text{ flops.}$$

Definition 5.6.2.7 A matrix L_k of the form

$$L_k = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array} \right),$$

where I_k is the $k \times k$ identity matrix and I is an identity matrix "of appropriate size" is called a Gauss transform. \diamond

$$L_k^{-1} = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array} \right)^{-1} = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right).$$

Definition 5.6.2.8 Given

$$p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{n-1} \end{pmatrix},$$

where $\{\pi_0, \pi_1, \dots, \pi_{n-1}\}$ is a permutation (rearrangement) of the integers $\{0, 1, \dots, n-1\}$, we define the permutation matrix $P(p)$ by

$$P(p) = \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}.$$

\diamond

If P is a permutation matrix then $P^{-1} = P^T$.

Definition 5.6.2.9 Elementary pivot matrix. Given $\pi \in \{0, \dots, n-1\}$ define the elementary pivot matrix

$$\tilde{P}(\pi) = \begin{pmatrix} e_\pi^T \\ \hline e_1^T \\ \vdots \\ \hline e_{\pi-1}^T \\ \hline e_0^T \\ \hline e_{\pi+1}^T \\ \vdots \\ e_{n-1}^T \end{pmatrix}$$

or, equivalently,

$$\tilde{P}(\pi) = \begin{cases} I_n & \text{if } \pi = 0 \\ \left(\begin{array}{c|cc|c} 0 & 0 & 1 & 0 \\ \hline 0 & I_{\pi-1} & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & I_{n-\pi-1} \end{array} \right) & \text{otherwise,} \end{cases}$$

where n is the size of the permutation matrix. \diamond

$[A, p] = \text{LU piv-right-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right)$
A_{TL} is 0×0 , p_T has 0 elements
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ p_2 \end{array} \right)$
$\pi_1 := \max_i \left(\frac{\alpha_{11}}{a_{21}} \right)$
$\left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right) \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$a_{21} := a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{12}^T$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ p_2 \end{array} \right)$
endwhile

Figure 5.6.2.10 Right-looking LU factorization algorithm with partial pivoting.Solving $Ax = b$ via LU factorization: with row pivoting:

- Compute the LU factorization with pivoting $PA = LU.k$
- Apply the row exchanges to the right-hand side: $y = Pb$.
- Solve $Lz = y$.
- Solve $Ux = z$.

Solve $Lz = y$, overwriting y with z (Variant 1)
$L \rightarrow \left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right)$
L_{TL} is 0×0 and y_T has 0 elements
while $n(L_{TL}) < n(L)$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array} \right)$
$y_2 := y_2 - \psi_1 l_{21}$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array} \right)$
endwhile

Figure 5.6.2.11 Lower triangular solve (with unit lower triangular matrix), Variant 1

Solve $Lz = y$, overwriting y with z (Variant 2)
$L \rightarrow \left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$
L_{TL} is 0×0 and y_T has 0 elements
while $n(L_{TL}) < n(L)$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$
$\underline{\psi_1 := \psi_1 - l_{10}^T y_0}$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$
endwhile

Figure 5.6.2.12 Lower triangular solve (with unit lower triangular matrix), Variant 2

Solve $Ux = z$, overwriting z with x (Variant 1)
$U \rightarrow \left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right), z \rightarrow \left(\begin{array}{c} z_T \\ z_B \end{array} \right)$
U_{BR} is 0×0 and z_B has 0 elements
while $n(U_{BR}) < n(U)$
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \rightarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$
$\underline{\zeta_1 := \zeta_1/v_{11}}$
$\underline{z_0 := z_0 - \zeta_1 u_{01}}$
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \leftarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$
endwhile

Figure 5.6.2.13 Upper triangular solve Variant 1

Solve $Ux = z$, overwriting z with x (Variant 2)
$U \rightarrow \left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right), z \rightarrow \left(\begin{array}{c} z_T \\ z_B \end{array} \right)$
U_{BR} is 0×0 and z_B has 0 elements
while $n(U_{BR}) < n(U)$
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \rightarrow \left(\begin{array}{cc c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \rightarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$
$\zeta_1 := \zeta_1 - u_{12}^T z_2$
$\zeta_1 := \zeta_1 / v_{11}$
$\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \left(\begin{array}{c} z_T \\ z_B \end{array} \right) \leftarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array} \right)$
endwhile

Figure 5.6.2.14 Upper triangular solve Variant 2

Cost of triangular solve Starting with an $n \times n$ (upper or lower) triangular matrix T , solving $Tx = b$ requires approximately n^2 flops.

Provided the solution of $Ax = b$ yields some accuracy in the solution, that accuracy can be improved through a process known as **iterative refinement**.

- Let \hat{x} is an approximate solution to $Ax = b$.
- Let $\hat{\delta}x$ is an approximate solution to $A\delta x = b - A\hat{x}$,
- Then $\hat{x} + \hat{\delta}x$, is an improved approximation.
- This process can be repeated until the accuracy in the computed solution is as good as warranted by the conditioning of A and the accuracy in b .

Definition 5.6.2.15 Hermitian positive definite matrix. A matrix $A \in \mathbb{C}^{n \times n}$ is Hermitian positive definite (HPD) if and only if it is Hermitian ($A^H = A$) and for all nonzero vectors $x \in \mathbb{C}^n$ it is the case that $x^H Ax > 0$. If in addition $A \in \mathbb{R}^{n \times n}$ then A is said to be symmetric positive definite (SPD). \diamond

Some insights regarding HPD matrices:

- B has linearly independent columns if and only if $A = B^H B$ is HPD.
- A diagonal matrix has only positive values on its diagonal if and only if it is HPD.
- If A is HPD, then its diagonal elements are all real-valued and positive.
- If $A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$, where A_{TL} is square, is HPD, then A_{TL} and A_{BR} are HPD.

Theorem 5.6.2.16 Cholesky Factorization Theorem. Given an HPD matrix A there exists a lower triangular matrix L such that $A = LL^H$. If the diagonal elements of L are

restricted to be positive, L is unique.

$A = \text{Chol-right-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$\alpha_{11} := \lambda_{11} = \sqrt{\alpha_{11}}$
$a_{21} := l_{21} = a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{12}^T$ (syr: update only lower triangular part)
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

Figure 5.6.2.17 Cholesky factorization algorithm (right-looking variant). The operation "syr" refers to "symmetric rank-1 update", which performs a rank-1 update, updating only the lower triangular part of the matrix in this algorithm.

Lemma 5.6.2.18 Let $A = \left(\begin{array}{c|c} \alpha_{11} & a_{21}^H \\ \hline a_{21} & A_{22} \end{array} \right), \in \mathbb{C}^{n \times n}$ be HPD and $l_{21} = a_{21}/\sqrt{\alpha_{11}}$. Then $A_{22} - l_{21}l_{21}^H$ is HPD.

Let $\hat{x} \in \mathbb{C}^n$ equal the solution to the linear least-squares (LLS) problem

$$\|b - A\hat{x}\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2, \quad (5.6.1)$$

where A has linearly independent columns, equals the solution to the normal equations

$$\underbrace{A^H A}_B \hat{x} = \underbrace{A^H b}_y.$$

This solution can be computed via the steps

- Form $B = A^H A$. Cost: approximately mn^2 flops.
- Factor $B = LL^H$ (Cholesky factorization). Cost: approximately $n^3/3$ flops.
- Compute $y = A^H b$. Cost: approximately $2mn$ flops.
- Solve $Lz = y$. Cost: approximately n^2 flops.
- Solve $L^H \hat{x} = z$. Cost: approximately n^2 flops.

for a total of, approximately, $mn^2 + n^3/3$ flops.

Week 6

Numerical Stability

The material in this chapter has been adapted from

- [6] Paolo Bientinesi, Robert A. van de Geijn, Goal-Oriented and Modular Stability Analysis, SIAM Journal on Matrix Analysis and Applications , Volume 32 Issue 1, February 2011.

and the technical report version of that paper (which includes exercises)

- [7] Paolo Bientinesi, Robert A. van de Geijn, The Science of Deriving Stability Analyses, FLAME Working Note #33. Aachen Institute for Computational Engineering Sciences, RWTH Aachen. TR AICES-2008-2. November 2008.

We recommend the technical report version for those who want to gain a deep understanding.

In this chapter, we focus on computation with real-valued scalars, vectors, and matrices.

6.1 Opening Remarks

6.1.1 Whose problem is it anyway?

Ponder This 6.1.1.1 What if we solve $Ax = b$ on a computer and the result is an approximate solution \hat{x} due to roundoff error that is incurred. If we don't know x , how do we check that \hat{x} approximates x with a small relative error? Should we check the residual $b - A\hat{x}$?

Solution.

- If

$$\frac{\|b - A\hat{x}\|}{\|b\|}$$

is small, then we cannot necessarily conclude that

$$\frac{\|\hat{x} - x\|}{\|x\|}$$

is small (in other words: that \hat{x} is relatively close to x).

- If

$$\frac{\|b - A\hat{x}\|}{\|b\|}$$

is small, then we *can* conclude that \hat{x} solves a nearby problem, provided we trust whatever routine computes $A\hat{x}$. After all, it solves

$$A\hat{x} = \hat{b}$$

where

$$\frac{\|b - \hat{b}\|}{\|b\|}$$

is small.

So, $\|b - A\hat{x}\|/\|b\|$ being small is a *necessary* condition, but not a *sufficient* condition. If $\|b - A\hat{x}\|/\|b\|$ is small, then \hat{x} is as good an answer as the problem warrants, since a small error in the right-hand side is to be expected either because data inherently has error in it or because in storing the right-hand side the input was inherently rounded.

In the presence of roundoff error, it is hard to determine whether an implementation is correct. Let's examine a few scenarios.

Homework 6.1.1.2 You use some linear system solver and it gives the wrong answer. In other words, you solve $Ax = b$ on a computer, computing \hat{x} , and somehow you determine that

$$\|x - \hat{x}\|$$

is large. Which of the following is a possible cause (identify all):

- There is a bug in the code. In other words, the algorithm that is used is sound (gives the right answer in exact arithmetic) but its implementation has an error in it.
- The linear system is ill-conditioned. A small relative error in the right-hand side can amplify into a large relative error in the solution.
- The algorithm you used accumulates a significant roundoff error.
- All is well: $\|\hat{x} - x\|$ is large but the relative error $\|\hat{x} - x\|/\|x\|$ is small.

Solution. All are possible causes. This week, we will delve into this.

6.1.2 Overview

- 6.1 Opening Remarks
 - 6.1.1 Whose problem is it anyway?
 - 6.1.2 Overview
 - 6.1.3 What you will learn

- 6.2 Floating Point Arithmetic
 - 6.2.1 Storing real numbers as floating point numbers
 - 6.2.2 Error in storing a real number as a floating point number
 - 6.2.3 Models of floating point computation
 - 6.2.4 Stability of a numerical algorithm
 - 6.2.5 Conditioning versus stability
 - 6.2.6 Absolute value of vectors and matrices
- 6.3 Error Analysis for Basic Linear Algebra Algorithms
 - 6.3.1 Initial insights
 - 6.3.2 Backward error analysis of dot product: general case
 - 6.3.3 Dot product: error results
 - 6.3.4 Matrix-vector multiplication
 - 6.3.5 Matrix-matrix multiplication
- 6.4 Error Analysis for Solving Linear Systems
 - 6.4.1 Numerical stability of triangular solve
 - 6.4.2 Numerical stability of LU factorization
 - 6.4.3 Numerical stability of linear solve via LU factorization
 - 6.4.4 Numerical stability of linear solve via LU factorization with partial pivoting
 - 6.4.5 Is LU with Partial Pivoting Stable?
- 6.5 Enrichments
 - 6.5.1 Systematic derivation of backward error analyses
 - 6.5.2 LU factorization with pivoting can fail in practice
- 6.6 Wrap Up
 - 6.6.1 Additional homework
 - 6.6.2 Summary

6.1.3 What you will learn

This week, you explore how roundoff error when employing floating point computation affect correctness.

Upon completion of this week, you should be able to

- Recognize how floating point numbers are stored.

- Employ strategies for avoiding unnecessary overflow and underflow that can occur in intermediate computations.
- Compute the machine epsilon (also called the unit roundoff) for a given floating point representation.
- Quantify errors in storing real numbers as floating point numbers and bound the incurred relative error in terms of the machine epsilon.
- Analyze error incurred in floating point computation using the Standard Computation Model (SCM) and the Alternative Computation Model (ACM) to determine their forward and backward results.
- Distinguish between conditioning of a problem and stability of an algorithm.
- Derive error results for simple linear algebra computations.
- State and interpret error results for solving linear systems.
- Argue how backward error can affect the relative error in the solution of a linear system.

6.2 Floating Point Arithmetic

6.2.1 Storing real numbers as floating point numbers



YouTube: <https://www.youtube.com/watch?v=sWcdwmCdVOU>

Only a finite number of (binary) digits can be used to store a real number in the memory of a computer. For so-called single-precision and double-precision floating point numbers, 32 bits and 64 bits are typically employed, respectively.

Recall that any real number can be written as $\mu \times \beta^e$, where β is the base (an integer greater than one), $\mu \in [-1, 1]$ is the mantissa, and e is the exponent (an integer). For our discussion, we will define the set of floating point numbers, F , as the set of all numbers $\chi = \mu \times \beta^e$ such that

- $\beta = 2$,
- $\mu = \pm.\delta_0\delta_1 \dots \delta_{t-1}$ (μ has only t (binary) digits), where $\delta_j \in \{0, 1\}$),

- $\delta_0 = 0$ iff $\mu = 0$ (the mantissa is normalized), and
- $-L \leq e \leq U$.

With this, the elements in F can be stored with a finite number of (binary) digits.

Example 6.2.1.1 Let $\beta = 2$, $t = 3$, $\mu = .101$, and $e = 1$. Then

$$\begin{aligned} \mu \times \beta^e &= \\ &= .101 \times 2^1 \\ &= \\ &= (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}) \times 2^1 \\ &= \\ &= \left(\frac{1}{2} + \frac{0}{4} + \frac{1}{8}\right) \times 2 \\ &= \\ &= 1.25 \end{aligned}$$

□

Observe that

- There is a largest number (in absolute value) that can be stored. Any number with larger magnitude "overflows". Typically, this causes a value that denotes a NaN (Not-a-Number) to be stored.
- There is a smallest number (in absolute value) that can be stored. Any number that is smaller in magnitude "underflows". Typically, this causes a zero to be stored.

In practice, one needs to be careful to consider overflow and underflow. The following example illustrates the importance of paying attention to this.

Example 6.2.1.2 Computing the (Euclidean) length of a vector is an operation we will frequently employ. Careful attention must be paid to overflow and underflow when computing it.

Given $x \in \mathbb{R}^n$, consider computing

$$\|x\|_2 = \sqrt{\sum_{i=0}^{n-1} \chi_i^2}. \quad (6.2.1)$$

Notice that

$$\|x\|_2 \leq \sqrt{n} \max_{i=0}^{n-1} |\chi_i|$$

and hence, unless some χ_i is close to overflowing, the result will not overflow. The problem is that if some element χ_i has the property that χ_i^2 overflows, intermediate results in the computation in (6.2.1) will overflow. The solution is to determine k such that

$$|\chi_k| = \max_{i=0}^{n-1} |\chi_i|$$

and to then instead compute

$$\|x\|_2 = |\chi_k| \sqrt{\sum_{i=0}^{n-1} \left(\frac{\chi_i}{\chi_k} \right)^2}.$$

It can be argued that the same approach also avoids underflow if underflow can be avoided. \square

In our discussion, we mostly ignore this aspect of floating point computation.

Remark 6.2.1.3 Any time a real number is stored in our computer, it is stored as a nearby floating point number (element in F) (either through rounding or truncation). Nearby, of course, could mean that it is stored as the exact number if it happens to also be a floating point number.

6.2.2 Error in storing a real number as a floating point number



YouTube: <https://www.youtube.com/watch?v=G2jawQW5WPc>

Remark 6.2.2.1 We consider the case where a real number is truncated to become the stored floating point number. This makes the discussion a bit simpler.

Let positive χ be represented by

$$\chi = .\delta_0\delta_1 \cdots \times 2^e,$$

where δ_i are binary digits and $\delta_0 = 1$ (the mantissa is normalized). If t binary digits are stored by our floating point system, then

$$\check{\chi} = .\delta_0\delta_1 \cdots \delta_{t-1} \times 2^e$$

is stored (if truncation is employed). If we let $\delta\chi = \chi - \check{\chi}$. Then

$$\begin{aligned} \delta\chi &= \underbrace{.\delta_0\delta_1 \cdots \delta_{t-1}\delta_t \cdots \times 2^e}_{\chi} - \underbrace{.\delta_0\delta_1 \cdots \delta_{t-1} \times 2^e}_{\check{\chi}} \\ &= \underbrace{.0 \cdots 00}_{t} \delta_t \cdots \times 2^e \\ &< \underbrace{.0 \cdots 01}_{t} \times 2^e = 2^{-t}2^e. \end{aligned}$$

Since χ is positive and $\delta_0 = 1$,

$$\chi = .\delta_0\delta_1 \cdots \times 2^e \geq \frac{1}{2} \times 2^e.$$

Thus,

$$\frac{\delta\chi}{\chi} \leq \frac{2^{-t}2^e}{\frac{1}{2}2^e} = 2^{-(t-1)},$$

which can also be written as

$$\delta\chi \leq 2^{-(t-1)}\chi.$$

A careful analysis of what happens when χ equals zero or is negative yields

$$|\delta\chi| \leq 2^{-(t-1)}|\chi|.$$

Example 6.2.2.2 The number $4/3 = 1.3333\dots$ can be written as

$$\begin{aligned} & 1.3333\dots \\ &= \\ & 1 + \frac{0}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} + \dots \\ &= \quad < \text{convert to binary representation} > \\ & 1.0101\dots \times 2^0 \\ &= \quad < \text{normalize} > \\ & .10101\dots \times 2^1 \end{aligned}$$

Now, if $t = 4$ then this would be truncated to

$$.1010 \times 2^1,$$

which equals the number

$$\begin{aligned} & .101 \times 2^1 = \\ & \frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{0}{16} \times 2^1 \\ &= \\ & 0.625 \times 2 = \quad < \text{convert to decimal} > \\ & 1.25 \end{aligned}$$

The relative error equals

$$\frac{1.333\dots - 1.25}{1.333\dots} = 0.0625.$$

□

If $\check{\chi}$ is computed by rounding instead of truncating, then

$$|\delta\chi| \leq 2^{-t}|\chi|.$$

We can abstract away from the details of the base that is chosen and whether rounding or truncation is used by stating that storing χ as the floating point number $\check{\chi}$ obeys

$$|\delta\chi| \leq \epsilon_{\text{mach}}|\chi|$$

where ϵ_{mach} is known as the *machine epsilon* or *unit roundoff*. When single precision floating point numbers are used $\epsilon_{\text{mach}} \approx 10^{-8}$, yielding roughly eight decimal digits of accuracy in the

stored value. When double precision floating point numbers are used $\epsilon_{\text{mach}} \approx 10^{-16}$, yielding roughly sixteen decimal digits of accuracy in the stored value.

Example 6.2.2.3 The number $4/3 = 1.3333\cdots$ can be written as

$$\begin{aligned} & 1.3333\cdots \\ &= \\ & 1 + \frac{0}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} + \cdots \\ &= \quad < \text{convert to binary representation} > \\ & 1.0101\cdots \times 2^0 \\ &= \quad < \text{normalize} > \\ & .10101\cdots \times 2^1 \end{aligned}$$

Now, if $t = 4$ then this would be rounded to

$$.1011 \times 2^1,$$

which is equals the number

$$\begin{aligned} & .1011 \times 2^1 = \\ & \frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} \times 2^1 \\ &= \\ & 0.6875 \times 2 = \quad < \text{convert to decimal} > \\ & 1.375 \end{aligned}$$

The relative error equals

$$\frac{|1.333\cdots - 1.375|}{1.333\cdots} = 0.03125.$$

□

Definition 6.2.2.4 Machine epsilon (unit roundoff). The machine epsilon (unit round-off), ϵ_{mach} , is defined as the smallest positive floating point number χ such that the floating point number that represents $1 + \chi$ is greater than one. ◇

Remark 6.2.2.5 The quantity ϵ_{mach} is machine dependent. It is a function of the parameters characterizing how a specific architecture converts reals to floating point numbers.

Homework 6.2.2.1 Assume a floating point number system with $\beta = 2$, a mantissa with t digits, and truncation when storing.

- Write the number 1 as a floating point number in this system.
- What is the ϵ_{mach} for this system?

Solution.

- Write the number 1 as a floating point number.

Answer:

$$\underbrace{.10\cdots 0}_{t \text{ digits}} \times 2^1.$$

- What is the ϵ_{mach} for this system?

Answer:

$$\underbrace{\underbrace{.10\cdots 0}_{t \text{ digits}} \times 2^1}_{1} + \underbrace{\underbrace{.00\cdots 1}_{t \text{ digits}} \times 2^1}_{2^{-(t-1)}} = \underbrace{\underbrace{.10\cdots 1}_{t \text{ digits}} \times 2^1}_{> 1}$$

and

$$\underbrace{\underbrace{.10\cdots 0}_{t \text{ digits}} \times 2^1}_{1} + \underbrace{\underbrace{.00\cdots 0}_{t \text{ digits}} 11\cdots \times 2^1}_{< 2^{-(t-1)}} = \underbrace{\underbrace{.10\cdots 0}_{t \text{ digits}} 11\cdots \times 2^1}_{\text{truncates to } 1}$$

Notice that

$$\underbrace{.00\cdots 1}_{t \text{ digits}} \times 2^1$$

can be represented as

$$\underbrace{.10\cdots 0}_{t \text{ digits}} \times 2^{-(t-2)}$$

and

$$\underbrace{.00\cdots 0}_{t \text{ digits}} 11\cdots \times 2^1$$

as

$$\underbrace{.11\cdots 1}_{t \text{ digits}} \times 2^{-(t-1)}$$

Hence $\epsilon_{\text{mach}} = 2^{-(t-1)}$.

6.2.3 Models of floating point computation

When computing with floating point numbers on a target computer, we will assume that all (floating point) arithmetic that is performed is in terms of additions, subtractions, multiplications, and divisions: $\{+, -, \times, /\}$.

6.2.3.1 Notation

In our discussions, we will distinguish between exact and computed quantities. The function fl(expression) returns the result of the evaluation of expression, where every operation is executed in floating point arithmetic. For example, given $\chi, \psi, \zeta, \omega \in F$ and assuming that

the expressions are evaluated from left to right and order of operations is obeyed,

$$\text{fl}(\chi + \psi + \zeta/\omega)$$

is equivalent to

$$\text{fl}(\text{fl}(\chi + \psi) + \text{fl}(\zeta/\omega)).$$

Equality between the quantities *lhs* and *rhs* is denoted by *lhs* = *rhs*. Assignment of *rhs* to *lhs* is denoted by *lhs* := *rhs* (*lhs* becomes *rhs*). In the context of a program, the statements *lhs* := *rhs* and *lhs* := fl(*rhs*) are equivalent. Given an assignment

$$\kappa := \text{expression},$$

we use the notation $\check{\kappa}$ (pronounced "check kappa") to denote the quantity resulting from fl(expression), which is actually stored in the variable κ :

$$\check{\kappa} = \text{fl}(\text{expression}).$$

Remark 6.2.3.1 In future discussion, we will use the notation $[\cdot]$ as shorthand for fl(\cdot).

6.2.3.2 Standard Computational Model (SCM)



YouTube: <https://www.youtube.com/watch?v=RIsljyjFbonU>

The Standard Computational Model (SCM) assumes that, for any two floating point numbers χ and ψ , the basic arithmetic operations satisfy the equality

$$\text{fl}(\chi \text{ op } \psi) = (\chi \text{ op } \psi)(1 + \epsilon), |\epsilon| \leq \epsilon_{\text{mach}}, \text{ and } \text{op} \in \{+, -, *, /\}.$$

The quantity ϵ is a function of χ, ψ and op . Sometimes we add a subscript $(\epsilon_+, \epsilon_*, \dots)$ to indicate what operation generated the $(1 + \epsilon)$ error factor. We always assume that all the input variables to an operation are floating point numbers.

Remark 6.2.3.2 We can interpret the SCM as follows: These operations are performed exactly and it is only in storing the result that a roundoff error occurs.

What really happens is that enough digits of the result are computed so that the net effect is as if the result of the exact operation was stored.

Given $\chi, \psi \in F$, performing any operation $\text{op} \in \{+, -, *, /\}$ with χ and ψ in floating point arithmetic, $\text{fl}(\chi \text{ op } \psi)$ yields a result that is correct up to machine precision: Let $\zeta = \chi \text{ op } \psi$ and $\check{\zeta} = \zeta + \delta\zeta = \text{fl}(\chi \text{ op } \psi)$. Then $|\delta\zeta| \leq \epsilon_{\text{mach}}|\zeta|$ and hence $\check{\zeta}$ is close to ζ (it has k correct binary digits).

Example 6.2.3.3 Consider the operation

$$\kappa = 4/3,$$

where we notice that both 4 and 3 can be exactly represented in our floating point system with $\beta = 2$ and $t = 4$. Recall that the real number $4/3 = 1.3333\cdots$ is stored as $.1010 \times 2^1$, if $t = 4$ and truncation is employed. This equals 1.25 in decimal representation. The relative error was 0.0625. Now

$$\begin{aligned} \check{\kappa} &= \\ &\text{fl}(4/3) \\ &= \\ &1.25 \\ &= \\ &1.333\cdots + (-0.0833\cdots) \\ &= \\ &1.333\cdots \times \left(1 + \frac{-0.0833\cdots}{1.333\cdots}\right) \\ &= \\ &4/3 \times (1 + (-0.0625)) \\ &= \\ &\kappa(1 + \epsilon_\gamma), \end{aligned}$$

where

$$|\epsilon_\gamma| = 0.0625 \leq \underbrace{0.125}_{\epsilon_{\text{mach}} = 2^{-(t-1)}}.$$

□

6.2.3.3 Alternative Computational Model (ACM)



YouTube: <https://www.youtube.com/watch?v=6jBxznXcivg>

For certain problems it is convenient to use the Alternative Computational Model (ACM) [21], which also assumes for the basic arithmetic operations that

$$\text{fl}(\chi \text{ op } \psi) = \frac{\chi \text{ op } \psi}{1 + \epsilon}, |\epsilon| \leq \epsilon_{\text{mach}}, \text{ and } \text{op} \in \{+, -, *, /\}.$$

As for the standard computation model, the quantity ϵ is a function of χ, ψ and op . Note that the ϵ 's produced using the standard and alternative models are generally not equal.

The Taylor series expansion of $1/(1 + \epsilon)$ is given by

$$\frac{1}{1 + \epsilon} = 1 + (-\epsilon) + O(\epsilon^2),$$

which explains how the SCM and ACM are related.

The ACM is useful when analyzing algorithms that involve division. In this course, we don't analyze in detail any such algorithms. We include this discussion of ACM for completeness.

Remark 6.2.3.4 Sometimes it is more convenient to use the SCM and sometimes the ACM. Trial and error, and eventually experience, will determine which one to use.

6.2.4 Stability of a numerical algorithm



YouTube: https://www.youtube.com/watch?v=_AoelpfTLhI

Correctness in the presence of error (e.g., when floating point computations are performed) takes on a different meaning. For many problems for which computers are used, there is one correct answer and we expect that answer to be computed by our program. The problem is that most real numbers cannot be stored exactly in a computer memory. They are stored as approximations, floating point numbers, instead. Hence storing them and/or computing with them inherently incurs error. The question thus becomes "When is a program correct in the presence of such errors?"

Let us assume that we wish to evaluate the mapping $f : \mathcal{D} \rightarrow \mathcal{R}$ where $\mathcal{D} \subset \mathbb{R}^n$ is the domain and $\mathcal{R} \subset \mathbb{R}^m$ is the range (codomain). Now, we will let $\check{f} : \mathcal{D} \rightarrow \mathcal{R}$ denote a computer implementation of this function. Generally, for $x \in \mathcal{D}$ it is the case that $f(x) \neq \check{f}(x)$. Thus, the computed value is not "correct". From earlier discussions about how the condition number of a matrix can amplify relative error, we know that it may not be the case that $\check{f}(x)$ is "close to" $f(x)$: even if \check{f} is an exact implementation of f , the mere act of storing x may introduce a small error δx and $f(x + \delta x)$ may be far from $f(x)$ if f is ill-conditioned.

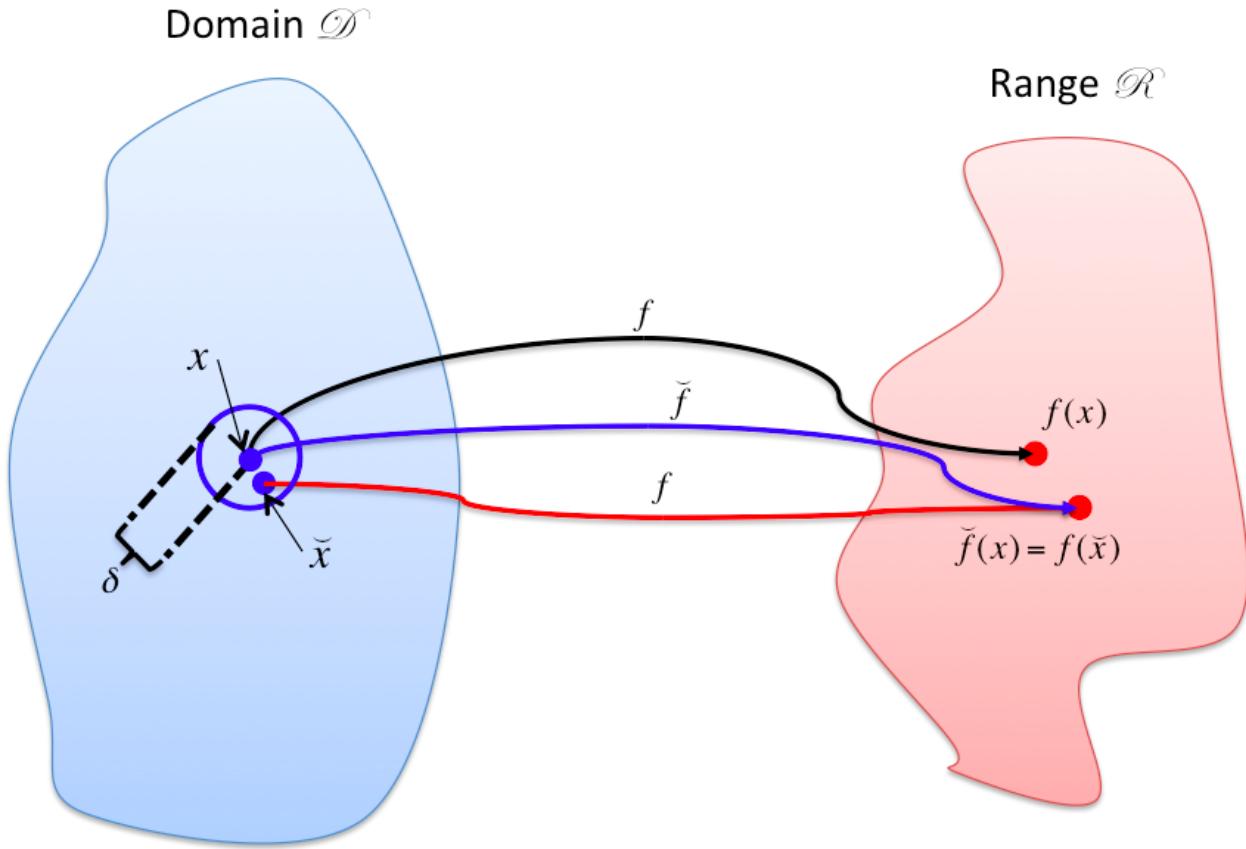


Figure 6.2.4.1 In this illustration, $f : \mathcal{D} \rightarrow \mathcal{R}$ is a function to be evaluated. The function \check{f} represents the implementation of the function that uses floating point arithmetic, thus incurring errors. The fact that for a nearby value, \check{x} , the computed value equals the exact function applied to the slightly perturbed input x , that is,

$$f(\check{x}) = \check{f}(x),$$

means that the error in the computation can be attributed to a small change in the input. If this is true, then \check{f} is said to be a (numerically) stable implementation of f for input x .

The following defines a property that captures correctness in the presence of the kinds of errors that are introduced by computer arithmetic:

Definition 6.2.4.2 Backward stable implementation. Given the mapping $f : D \rightarrow R$, where $D \subset \mathbb{R}^n$ is the domain and $R \subset \mathbb{R}^m$ is the range (codomain), let $\check{f} : D \rightarrow R$ be a computer implementation of this function. We will call \check{f} a backward stable (also called "numerically stable") implementation of f on domain \mathcal{D} if for all $x \in D$ there exists a \check{x} "close" to x such that $\check{f}(x) = f(\check{x})$. \diamond

In other words, \check{f} is a stable implementation if the error that is introduced is similar to that introduced when f is evaluated with a slightly changed input. This is illustrated in Figure 6.2.4.1 for a specific input x . If an implementation is not stable, it is numerically

unstable.

The algorithm is said to be forward stable on domain \mathcal{D} if for all $x \in \mathcal{D}$ it is the case that $\tilde{f}(x) \approx f(x)$. In other words, the computed result equals a slight perturbation of the exact result.

Example 6.2.4.3 Under the SCM from the last unit, floating point addition, $\kappa := \chi + \psi$, is a backward stable operation.

Solution.

$$\begin{aligned}
 \check{\kappa} &= <\text{computed value for } \kappa> \\
 [\chi + \psi] &= <\text{SCM}> \\
 (\chi + \psi)(1 + \epsilon_+) &= <\text{distribute}> \\
 \chi(1 + \epsilon_+) + \psi(1 + \epsilon_+) &= \\
 &= \\
 (\chi + \delta\chi) + (\psi + \delta\psi) &
 \end{aligned}$$

where

- $|\epsilon_+| \leq \epsilon_{\text{mach}}$,
- $\delta\chi = \chi\epsilon_+$,
- $\delta\psi = \psi\epsilon_+$.

Hence $\check{\kappa}$ equals the exact result when adding nearby inputs. \square

Homework 6.2.4.1

- ALWAYS/SOMETIMES/NEVER: Under the SCM from the last unit, floating point subtraction, $\kappa := \chi - \psi$, is a backward stable operation.
- ALWAYS/SOMETIMES/NEVER: Under the SCM from the last unit, floating point multiplication, $\kappa := \chi \times \psi$, is a backward stable operation.
- ALWAYS/SOMETIMES/NEVER: Under the SCM from the last unit, floating point division, $\kappa := \chi/\psi$, is a backward stable operation.

Answer.

- ALWAYS: Under the SCM from the last unit, floating point subtraction, $\kappa := \chi - \psi$, is a backward stable operation.
- ALWAYS: Under the SCM from the last unit, floating point multiplication, $\kappa := \chi \times \psi$, is a backward stable operation.
- ALWAYS: Under the SCM from the last unit, floating point division, $\kappa := \chi/\psi$, is a backward stable operation.

Now prove it!

Solution.

- ALWAYS: Under the SCM from the last unit, floating point subtraction, $\kappa := \chi - \psi$, is a backward stable operation.

$$\begin{aligned}
 \check{\kappa} &= <\text{computed value for } \kappa> \\
 [\chi - \psi] &= <\text{SCM}> \\
 (\chi - \psi)(1 + \epsilon_-) &= <\text{distribute}> \\
 \chi(1 + \epsilon_-) - \psi(1 + \epsilon_-) &= \\
 &= \\
 (\chi + \delta\chi) - (\psi + \delta\psi) &
 \end{aligned}$$

where

- $|\epsilon_-| \leq \epsilon_{\text{mach}}$,
- $\delta\chi = \chi\epsilon_-$,
- $\delta\psi = \psi\epsilon_-$.

Hence $\check{\kappa}$ equals the exact result when subtracting nearby inputs.

- ALWAYS: Under the SCM from the last unit, floating point multiplication, $\kappa := \chi \times \psi$, is a backward stable operation.

$$\begin{aligned}
 \check{\kappa} &= <\text{computed value for } \kappa> \\
 [\chi \times \psi] &= <\text{SCM}> \\
 (\chi \times \psi)(1 + \epsilon_\times) &= <\text{associative property}> \\
 \chi \times \psi(1 + \epsilon_\times) &= \\
 &= \\
 \chi(\psi + \delta\psi) &
 \end{aligned}$$

where

- $|\epsilon_\times| \leq \epsilon_{\text{mach}}$,
- $\delta\psi = \psi\epsilon_\times$.

Hence $\check{\kappa}$ equals the exact result when multiplying nearby inputs.

- **ALWAYS:** Under the SCM from the last unit, floating point division, $\kappa := \chi/\psi$, is a backward stable operation.

$$\begin{aligned}
 \check{\kappa} &= <\text{computed value for } \kappa> \\
 [\chi/\psi] &< \\
 &= \text{SCM} > \\
 (\chi/\psi)(1 + \epsilon_f) & \\
 &= <\text{commutative property}> \\
 \chi(1 + \epsilon_f)/\psi & \\
 &= \\
 (\chi + \delta\chi)/\psi &
 \end{aligned}$$

where

- $|\epsilon_f| \leq \epsilon_{\text{mach}}$,
- $\delta\chi = \chi\epsilon_f$,

Hence $\check{\kappa}$ equals the exact result when dividing nearby inputs.

Ponder This 6.2.4.2 In the last homework, we showed that floating point division is backward stable by showing that $[\chi/\psi] = (\chi + \delta\chi)/\psi$ for suitably small $\delta\chi$.

How would one show that $[\chi/\psi] = \chi/(\psi + \delta\psi)$ for suitably small $\delta\psi$?

6.2.5 Conditioning versus stability



YouTube: <https://www.youtube.com/watch?v=e29Yk4XCyLs>

It is important to keep conditioning versus stability straight:

- *Conditioning* is a property of the problem you are trying to solve. A problem is well-conditioned if a small change in the input is guaranteed to only result in a small change in the output. A problem is ill-conditioned if a small change in the input can result in a large change in the output.
- *Stability* is a property of an implementation. If the implementation, when executed with an input always yields an output that can be attributed to slightly changed input, then the implementation is backward stable.

In other words, in the presence of roundoff error, computing a wrong answer may be due to the problem (if it is ill-conditioned), the implementation (if it is numerically unstable), or a programming bug (if the implementation is sloppy). Obviously, it can be due to some combination of these.

Now,

- If you compute the solution to a well-conditioned problem with a numerically stable implementation, then you will get an answer that is close to the actual answer.
- If you compute the solution to a well-conditioned problem with a numerically unstable implementation, then you may or may not get an answer that is close to the actual answer.
- If you compute the solution to an ill-conditioned problem with a numerically stable implementation, then you may or may not get an answer that is close to the actual answer.

Yet another way to look at this: A numerically stable implementation will yield an answer that is as accurate as the conditioning of the problem warrants.

6.2.6 Absolute value of vectors and matrices

In the above discussion of error, the vague notions of "near" and "slightly perturbed" are used. Making these notions exact usually requires the introduction of measures of size for vectors and matrices, i.e., norms. When analyzing the stability of algorithms, we instead give all bounds in terms of the absolute values of the individual elements of the vectors and/or matrices. While it is easy to convert such bounds to bounds involving norms, the converse is not true.

Definition 6.2.6.1 **Absolute value of vector and matrix.** Given $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$,

$$|x| = \begin{pmatrix} |\chi_0| \\ |\chi_1| \\ \vdots \\ |\chi_{n-1}| \end{pmatrix} \quad \text{and} \quad |A| = \begin{pmatrix} |\alpha_{0,0}| & |\alpha_{0,1}| & \dots & |\alpha_{0,n-1}| \\ |\alpha_{1,0}| & |\alpha_{1,1}| & \dots & |\alpha_{1,n-1}| \\ \vdots & \vdots & \ddots & \vdots \\ |\alpha_{m-1,0}| & |\alpha_{m-1,1}| & \dots & |\alpha_{m-1,n-1}| \end{pmatrix}.$$

◊

Definition 6.2.6.2 Let $\Delta \in \{<, \leq, =, \geq, >\}$ and $x, y \in \mathbb{R}^n$. Then

$$|x| \Delta |y| \quad \text{iff} \quad |\chi_i| \Delta |\psi_i|,$$

for all $i = 0, \dots, n - 1$. Similarly, given A and $B \in \mathbb{R}^{m \times n}$,

$$|A| \Delta |B| \quad \text{iff} \quad |\alpha_{ij}| \Delta |\beta_{ij}|,$$

for all $i = 0, \dots, m - 1$ and $j = 0, \dots, n - 1$.

◊

The next Lemma is exploited in later sections:

Homework 6.2.6.1 Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$.

ALWAYS/SOMETIMES/NEVER: $|AB| \leq |A||B|$.

Answer. ALWAYS

Now prove it.

Solution. Let $C = AB$. Then the (i, j) entry in $|C|$ is given by

$$|\gamma_{i,j}| = \left| \sum_{p=0}^{k-1} \alpha_{i,p} \beta_{p,j} \right| \leq \sum_{p=0}^{k-1} |\alpha_{i,p} \beta_{p,j}| = \sum_{p=0}^{k-1} |\alpha_{i,p}| |\beta_{p,j}|$$

which equals the (i, j) entry of $|A||B|$. Thus $|AB| \leq |A||B|$.

The fact that the bounds that we establish can be easily converted into bounds involving norms is a consequence of the following theorem, where $\| \cdot \|_F$ indicates the Frobenius matrix norm.

Theorem 6.2.6.3 Let $A, B \in \mathbb{R}^{m \times n}$. If $|A| \leq |B|$ then $\|A\|_F \leq \|B\|_F$, $\|A\|_1 \leq \|B\|_1$, and $\|A\|_\infty \leq \|B\|_\infty$.

Homework 6.2.6.2 Prove Theorem 6.2.6.3

Solution.

- Show that if $|A| \leq |B|$ then $\|A\|_F \leq \|B\|_F$:

$$\|A\|_F^2 = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2 \leq \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\beta_{i,j}|^2 = \|B\|_F^2.$$

Hence $\|A\|_F \leq \|B\|_F$.

- Show that if $|A| \leq |B|$ then $\|A\|_1 \leq \|B\|_1$:

Let

$$A = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right) \quad \text{and} \quad B = \left(\begin{array}{c|c|c} b_0 & \cdots & b_{n-1} \end{array} \right).$$

Then

```

 $\|A\|_1$ 
    =      < alternate way of computing 1-norm >
 $\max_{0 \leq j < n} \|a_j\|_1$ 
    =      < expose individual entries of  $a_j$  >
 $\max_{0 \leq j < n} \left( \sum_{i=0}^{m-1} |\alpha_{i,j}| \right)$ 
    =      < choose  $k$  to be the index that maximizes >
 $\left( \sum_{i=0}^{m-1} |\alpha_{i,k}| \right)$ 
    ≤      < entries of  $B$  bound corresponding entries of  $A$  >
 $\left( \sum_{i=0}^{m-1} |\beta_{i,k}| \right)$ 
    =      < express sum as 1-norm of column indexed with  $k$  >
 $\|b_k\|_1$ 
    ≤      < take max over all columns >
 $\max_{0 \leq j < n} \|b_j\|_1$ 
    =      < definition of 1-norm >
 $\|B\|_1.$ 

```

- Show that if $|A| \leq |B|$ then $\|A\|_\infty \leq \|B\|_\infty$:

Note:

- $\|A\|_\infty = \|A^T\|_1$ and $\|B\|_\infty = \|B^T\|_1$.
 - If $|A| \leq |B|$ then, clearly, $|A^T| \leq |B^T|$.

Hence

$$\|A\|_\infty = \|A^T\|_1 < \|B^T\|_1 = \|B\|_\infty.$$

6.3 Error Analysis for Basic Linear Algebra Algorithms

6.3.1 Initial insights



YouTube: <https://www.youtube.com/watch?v=OHqdJ3hjHFY>

Before giving a general result, let us focus on the case where the vectors x and y have only a few elements:

Example 6.3.1.1 Consider

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} \text{ and } y = \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix}$$

and the computation

$$\kappa := x^T y.$$

Under the SCM given in Subsubsection 6.2.3.2, the computed result, $\check{\kappa}$, satisfies

$$\check{\kappa} = \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)}) & 0 \\ 0 & (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix}. \quad (6.3.1)$$

Solution.

$$\begin{aligned} \check{\kappa} &= < \check{\kappa} = [x^T y] > \\ &= \left[\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right] \\ &= < \text{definition of } x^T y > \\ &= [\chi_0 \psi_0 + \chi_1 \psi_1] \\ &= < \text{each suboperation is performed in floating point arithmetic} > \\ &= [[\chi_0 \psi_0] + [\chi_1 \psi_1]] \\ &= < \text{apply SCM multiple times} > \\ &= [\chi_0 \psi_0 (1 + \epsilon_*^{(0)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)})] \\ &= < \text{apply SCM} > \\ &= (\chi_0 \psi_0 (1 + \epsilon_*^{(0)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)})) (1 + \epsilon_+^{(1)}) \\ &= < \text{distribute} > \\ &= \chi_0 \psi_0 (1 + \epsilon_*^{(0)}) (1 + \epsilon_+^{(1)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)}) (1 + \epsilon_+^{(1)}) \\ &= < \text{commute} > \\ &= \chi_0 (1 + \epsilon_*^{(0)}) (1 + \epsilon_+^{(1)}) \psi_0 + \chi_1 (1 + \epsilon_*^{(1)}) (1 + \epsilon_+^{(1)}) \psi_1 \\ &= < \text{(perhaps too) slick way of expressing the final result} > \\ &= \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)}) & 0 \\ 0 & (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \\ , \end{aligned}$$

where $|\epsilon_*^{(0)}|, |\epsilon_*^{(1)}|, |\epsilon_+^{(1)}| \leq \epsilon_{\text{mach}}$. \square

An important insight from this example is that the result in (6.3.1) can be manipulated to associate the accumulated error with vector x as in

$$\check{\kappa} = \begin{pmatrix} \chi_0 (1 + \epsilon_*^{(0)}) (1 + \epsilon_+^{(1)}) \\ \chi_1 (1 + \epsilon_*^{(1)}) (1 + \epsilon_+^{(1)}) \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix}$$

or with vector y

$$\check{\kappa} = \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0(1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)}) \\ \psi_1(1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)}) \end{pmatrix}.$$

This will play a role when we later analyze algorithms that use the dot product.

Homework 6.3.1.1 Consider

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} \text{ and } y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \end{pmatrix}$$

and the computation

$$\kappa := x^T y$$

computed in the order indicated by

$$\kappa := (\chi_0\psi_0 + \chi_1\psi_1) + \chi_2\psi_2.$$

Employ the SCM given in Subsubsection 6.2.3.2, to derive a result similar to that given in (6.3.1).

Answer.

$$\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix}^T \begin{pmatrix} (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)})(1 + \epsilon_+^{(2)}) & 0 & 0 \\ 0 & (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)})(1 + \epsilon_+^{(2)}) & 0 \\ 0 & 0 & (1 + \epsilon_*^{(2)})(1 + \epsilon_+^{(2)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \end{pmatrix},$$

where $|\epsilon_*^{(0)}|, |\epsilon_*^{(1)}|, |\epsilon_+^{(1)}|, |\epsilon_*^{(2)}|, |\epsilon_+^{(2)}| \leq \epsilon_{\text{mach}}$.

Solution. Here is a solution that builds on the last example and paves the path toward

the general solution presented in the next unit.

$$\begin{aligned}
 \check{\kappa} &= <\check{\kappa} = [x^T y]> \\
 &= <(\chi_0\psi_0 + \chi_1\psi_1) + \chi_2\psi_2]> \\
 &= <\text{each suboperation is performed in floating point arithmetic}> \\
 &= [[\chi_0\psi_0 + \chi_1\psi_1] + [\chi_2\psi_2]] \\
 &= <\text{reformulate so we can use result from Example 6.3.1.1}> \\
 &= \left[\left[\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right] + [\chi_2\psi_2] \right] \\
 &= <\text{use Example 6.3.1.1; twice SCM}> \\
 &= \left(\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)}) & 0 \\ 0 & (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right. \\
 &\quad \left. + \chi_2\psi_2(1 + \epsilon_*^{(2)}) \right) (1 + \epsilon_+^{(2)}) \\
 &= <\text{distribute, commute}> \\
 &= \left(\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)}) & 0 \\ 0 & (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)}) \end{pmatrix} (1 + \epsilon_+^{(2)}) \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right. \\
 &\quad \left. + \chi_2(1 + \epsilon_*^{(2)})(1 + \epsilon_+^{(2)})\psi_2 \right) \\
 &= <\text{(perhaps too) slick way of expressing the final result}> \\
 &= \left(\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix}^T \begin{pmatrix} (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)})(1 + \epsilon_+^{(2)}) & 0 & 0 \\ 0 & (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)})(1 + \epsilon_+^{(2)}) & 0 \\ 0 & 0 & (1 + \epsilon_*^{(2)})(1 + \epsilon_+^{(2)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \end{pmatrix} \right) \\
 &,
 \end{aligned}$$

where $|\epsilon_*^{(0)}|, |\epsilon_*^{(1)}|, |\epsilon_+^{(1)}|, |\epsilon_*^{(2)}|, |\epsilon_+^{(2)}| \leq \epsilon_{\text{mach}}$.

6.3.2 Backward error analysis of dot product: general case



YouTube: <https://www.youtube.com/watch?v=PmFUqJXogm8>

Consider now

$$\kappa := x^T y = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-2} \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix} = \left(((\chi_0 \psi_0 + \chi_1 \psi_1) + \dots) + \chi_{n-2} \psi_{n-2} \right) + \chi_{n-1} \psi_{n-1}.$$

Under the computational model given in [Subsection 6.2.3](#) the computed result, $\check{\kappa}$, satisfies

$$\begin{aligned} \check{\kappa} &= \left(\left((\chi_0 \psi_0 (1 + \epsilon_*^{(0)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)})) (1 + \epsilon_+^{(1)}) + \dots \right) (1 + \epsilon_+^{(n-2)}) \right. \\ &\quad \left. + \chi_{n-1} \psi_{n-1} (1 + \epsilon_*^{(n-1)}) \right) (1 + \epsilon_+^{(n-1)}) \\ &= \chi_0 \psi_0 (1 + \epsilon_*^{(0)}) (1 + \epsilon_+^{(1)}) (1 + \epsilon_+^{(2)}) \dots (1 + \epsilon_+^{(n-1)}) \\ &\quad + \chi_1 \psi_1 (1 + \epsilon_*^{(1)}) (1 + \epsilon_+^{(1)}) (1 + \epsilon_+^{(2)}) \dots (1 + \epsilon_+^{(n-1)}) \\ &\quad + \chi_2 \psi_2 (1 + \epsilon_*^{(2)}) \quad (1 + \epsilon_+^{(2)}) \dots (1 + \epsilon_+^{(n-1)}) \\ &\quad + \dots \\ &\quad + \chi_{n-1} \psi_{n-1} (1 + \epsilon_*^{(n-1)}) \quad (1 + \epsilon_+^{(n-1)}) \\ &= \sum_{i=0}^{n-1} \left(\chi_i \psi_i (1 + \epsilon_*^{(i)}) \prod_{j=i}^{n-1} (1 + \epsilon_+^{(j)}) \right) \end{aligned}$$

so that

$$\check{\kappa} = \sum_{i=0}^{n-1} \left(\chi_i \psi_i (1 + \epsilon_*^{(i)}) \prod_{j=i}^{n-1} (1 + \epsilon_+^{(j)}) \right), \quad (6.3.2)$$

where $\epsilon_+^{(0)} = 0$ and $|\epsilon_*^{(0)}|, |\epsilon_*^{(j)}|, |\epsilon_+^{(j)}| \leq \epsilon_{\text{mach}}$ for $j = 1, \dots, n-1$.

Clearly, a notation to keep expressions from becoming unreadable is desirable. For this reason we introduce the symbol θ_j :



YouTube: <https://www.youtube.com/watch?v=6qnYXaw4Bms>

Lemma 6.3.2.1 *Let $\epsilon_i \in \mathbb{R}$, $0 \leq i \leq n-1$, $n\epsilon_{\text{mach}} < 1$, and $|\epsilon_i| \leq \epsilon_{\text{mach}}$. Then $\exists \theta_n \in \mathbb{R}$ such that*

$$\prod_{i=0}^{n-1} (1 + \epsilon_i)^{\pm 1} = 1 + \theta_n,$$

with $|\theta_n| \leq n\epsilon_{\text{mach}}/(1 - n\epsilon_{\text{mach}})$.

Here the ± 1 means that on an individual basis, the term is either used in a multiplication or a division. For example

$$(1 + \epsilon_0)^{\pm 1}(1 + \epsilon_1)^{\pm 1}$$

might stand for

$$(1 + \epsilon_0)(1 + \epsilon_1) \quad \text{or} \quad \frac{(1 + \epsilon_0)}{(1 + \epsilon_1)} \quad \text{or} \quad \frac{(1 + \epsilon_1)}{(1 + \epsilon_0)} \quad \text{or} \quad \frac{1}{(1 + \epsilon_1)(1 + \epsilon_0)}$$

so that this lemma can accommodate an analysis that involves a mixture of the Standard and Alternative Computational Models (SCM and ACM).

Proof. By Mathematical Induction.

- Base case. $n = 1$. Trivial.
- Inductive Step. The Inductive Hypothesis (I.H.) tells us that for all $\epsilon_i \in \mathbb{R}$, $0 \leq i \leq n - 1$, $n\epsilon_{\text{mach}} < 1$, and $|\epsilon_i| \leq \epsilon_{\text{mach}}$, there exists a $\theta_n \in \mathbb{R}$ such that

$$\prod_{i=0}^{n-1} (1 + \epsilon_i)^{\pm 1} = 1 + \theta_n, \text{ with } |\theta_n| \leq n\epsilon_{\text{mach}}/(1 - n\epsilon_{\text{mach}}).$$

We will show that if $\epsilon_i \in \mathbb{R}$, $0 \leq i \leq n$, $(n + 1)\epsilon_{\text{mach}} < 1$, and $|\epsilon_i| \leq \epsilon_{\text{mach}}$, then there exists a $\theta_{n+1} \in \mathbb{R}$ such that

$$\prod_{i=0}^n (1 + \epsilon_i)^{\pm 1} = 1 + \theta_{n+1}, \text{ with } |\theta_{n+1}| \leq (n + 1)\epsilon_{\text{mach}}/(1 - (n + 1)\epsilon_{\text{mach}}).$$

- Case 1: The last term comes from the application of the SCM.
 $\prod_{i=0}^n (1 + \epsilon_i)^{\pm 1} = \prod_{i=0}^{n-1} (1 + \epsilon_i)^{\pm 1}(1 + \epsilon_n)$. See [Ponder This 6.3.2.1](#).
- Case 2: The last term comes from the application of the ACM.
 $\prod_{i=0}^n (1 + \epsilon_i)^{\pm 1} = (\prod_{i=0}^{n-1} (1 + \epsilon_i)^{\pm 1})/(1 + \epsilon_n)$. By the I.H. there exists a θ_n such that $(1 + \theta_n) = \prod_{i=0}^{n-1} (1 + \epsilon_i)^{\pm 1}$ and $|\theta_n| \leq n\epsilon_{\text{mach}}/(1 - n\epsilon_{\text{mach}})$. Then

$$\frac{\prod_{i=0}^{n-1} (1 + \epsilon_i)^{\pm 1}}{1 + \epsilon_n} = \frac{1 + \theta_n}{1 + \epsilon_n} = 1 + \underbrace{\frac{\theta_n - \epsilon_n}{1 + \epsilon_n}}_{\theta_{n+1}},$$

which tells us how to pick θ_{n+1} . Now

$$\begin{aligned}
 |\theta_{n+1}| &= <\text{definition of } \theta_{n+1}> \\
 |(\theta_n - \epsilon_n)/(1 + \epsilon_n)| &\leq <|\theta_n - \epsilon_n| \leq |\theta_n| + |\epsilon_n| \leq |\theta_n| + \epsilon_{\text{mach}}> \\
 (|\theta_n| + \epsilon_{\text{mach}})/(1 + \epsilon_n) &\leq <|1 + \epsilon_n| \geq 1 - |\epsilon_n| \geq 1 - \epsilon_{\text{mach}}> \\
 (|\theta_n| + \epsilon_{\text{mach}})/(1 - \epsilon_{\text{mach}}) &\leq <\text{bound } |\theta_n| \text{ using I.H.}> \\
 (\frac{n\epsilon_{\text{mach}}}{1-n\epsilon_{\text{mach}}} + \epsilon_{\text{mach}})/(1 - \epsilon_{\text{mach}}) &= <\text{algebra}> \\
 (n\epsilon_{\text{mach}} + (1 - n\epsilon_{\text{mach}})\epsilon_{\text{mach}})/((1 - n\epsilon_{\text{mach}})(1 - \epsilon_{\text{mach}})) \\
 &= <\text{algebra}> \\
 ((n + 1)\epsilon_{\text{mach}} - n\epsilon_{\text{mach}}^2)/(1 - (n + 1)\epsilon_{\text{mach}} + n\epsilon_{\text{mach}}^2) &\leq <\text{increase numerator; decrease denominator}> \\
 ((n + 1)\epsilon_{\text{mach}})/(1 - (n + 1)\epsilon_{\text{mach}}).
 \end{aligned}$$

- By the Principle of Mathematical Induction, the result holds.

■

Ponder This 6.3.2.1 Complete the proof of [Lemma 6.3.2.1](#).

Remark 6.3.2.2 The quantity θ_n will be used throughout these notes. It is not intended to be a specific number. Instead, it is an order of magnitude identified by the subscript n , which indicates the number of error factors of the form $(1 + \epsilon_i)$ and/or $(1 + \epsilon_i)^{-1}$ that are grouped together to form $(1 + \theta_n)$.

Since we will often encounter the bound on $|\theta_n|$ that appears in [Lemma 6.3.2.1](#) we assign it a symbol as follows:

Definition 6.3.2.3 For all $n \geq 1$ and $n\epsilon_{\text{mach}} < 1$, define

$$\gamma_n = n\epsilon_{\text{mach}}/(1 - n\epsilon_{\text{mach}}).$$

◇

With this notation, (6.3.2) simplifies to

$$\begin{aligned}
 \check{\kappa} &= \\
 &= \chi_0\psi_0(1 + \theta_n) + \chi_1\psi_1(1 + \theta_n) + \cdots + \chi_{n-1}\psi_{n-1}(1 + \theta_2) \\
 &= \\
 &\left(\begin{array}{c} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{array} \right)^T \left(\begin{array}{ccccc} (1 + \theta_n) & 0 & 0 & \cdots & 0 \\ 0 & (1 + \theta_n) & 0 & \cdots & 0 \\ 0 & 0 & (1 + \theta_{n-1}) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (1 + \theta_2) \end{array} \right) \left(\begin{array}{c} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{array} \right) \\
 &= \\
 &\left(\begin{array}{c} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{array} \right)^T \underbrace{\left(I + \left(\begin{array}{ccccc} \theta_n & 0 & 0 & \cdots & 0 \\ 0 & \theta_n & 0 & \cdots & 0 \\ 0 & 0 & \theta_{n-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \theta_2 \end{array} \right) \right)}_{I + \Sigma^{(n)}} \left(\begin{array}{c} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{array} \right) \\
 &= \\
 &x^T(I + \Sigma^{(n)})y,
 \end{aligned} \tag{6.3.3}$$

where $|\theta_j| \leq \gamma_j$, $j = 2, \dots, n$.

Remark 6.3.2.4 Two instances of the symbol θ_n , appearing even in the same expression, typically do not represent the same number. For example, in (6.3.3) a $(1 + \theta_n)$ multiplies each of the terms $\chi_0\psi_0$ and $\chi_1\psi_1$, but these two instances of θ_n , as a rule, do not denote the same quantity. In particular, one should be careful when factoring out such quantities.



YouTube: <https://www.youtube.com/watch?v=Uc6NuDZMakE>

As part of the analyses the following bounds will be useful to bound error that accumulates:

Lemma 6.3.2.5 If $n, b \geq 1$ then $\gamma_n \leq \gamma_{n+b}$ and $\gamma_n + \gamma_b + \gamma_n\gamma_b \leq \gamma_{n+b}$.

This lemma will be invoked when, for example, we want to bound $|\epsilon|$ such that $1 + \epsilon = (1 + \epsilon_1)(1 + \epsilon_2) = 1 + (\epsilon_1 + \epsilon_2 + \epsilon_1\epsilon_2)$ knowing that $|\epsilon_1| \leq \gamma_n$ and $|\epsilon_2| \leq \gamma_b$.

Homework 6.3.2.2 Prove Lemma 6.3.2.5.**Solution.**

$$\begin{aligned}
 \gamma_n &= <\text{definition}> \\
 (n\epsilon_{\text{mach}})/(1 - n\epsilon_{\text{mach}}) &\leq <b \geq 1> \\
 ((n+b)\epsilon_{\text{mach}})/(1 - n\epsilon_{\text{mach}}) &\leq <1/(1 - n\epsilon_{\text{mach}}) \leq 1/(1 - (n+b)\epsilon_{\text{mach}}) \text{ if } (n+b)\epsilon_{\text{mach}} < 1> \\
 ((n+b)\epsilon_{\text{mach}})/(1 - (n+b)\epsilon_{\text{mach}}) &= <\text{definition}> \\
 \gamma_{n+b}.
 \end{aligned}$$

and

$$\begin{aligned}
 \gamma_n + \gamma_b + \gamma_n \gamma_b &= <\text{definition}> \\
 \frac{n\epsilon_{\text{mach}}}{1-n\epsilon_{\text{mach}}} + \frac{b\epsilon_{\text{mach}}}{1-b\epsilon_{\text{mach}}} + \frac{n\epsilon_{\text{mach}}}{(1-n\epsilon_{\text{mach}})} \frac{b\epsilon_{\text{mach}}}{(1-b\epsilon_{\text{mach}})} &= <\text{algebra}> \\
 \frac{n\epsilon_{\text{mach}}(1-b\epsilon_{\text{mach}})+(1-n\epsilon_{\text{mach}})b\epsilon_{\text{mach}}+b^2n\epsilon_{\text{mach}}^2}{(1-n\epsilon_{\text{mach}})(1-b\epsilon_{\text{mach}})} &= <\text{algebra}> \\
 \frac{n\epsilon_{\text{mach}}-b^2n\epsilon_{\text{mach}}^2+b\epsilon_{\text{mach}}-b^2n\epsilon_{\text{mach}}^2+b^2n\epsilon_{\text{mach}}^2}{1-(n+b)\epsilon_{\text{mach}}+b^2n\epsilon_{\text{mach}}^2} &= <\text{algebra}> \\
 \frac{(n+b)\epsilon_{\text{mach}}-b^2n\epsilon_{\text{mach}}^2}{1-(n+b)\epsilon_{\text{mach}}+b^2n\epsilon_{\text{mach}}^2} &\leq < b^2n\epsilon_{\text{mach}}^2 > 0 \\
 \frac{(n+b)\epsilon_{\text{mach}}}{1-(n+b)\epsilon_{\text{mach}}+b^2n\epsilon_{\text{mach}}^2} &\leq < b^2n\epsilon_{\text{mach}}^2 > 0 \\
 \frac{(n+b)\epsilon_{\text{mach}}}{1-(n+b)\epsilon_{\text{mach}}} &= <\text{definition}> \\
 \gamma_{n+b}.
 \end{aligned}$$

6.3.3 Dot product: error resultsYouTube: <https://www.youtube.com/watch?v=QxUCV4k8Gu8>

It is of interest to accumulate the roundoff error encountered during computation as a perturbation of input and/or output parameters:

- $\check{\kappa} = (x + \delta x)^T y$;
($\check{\kappa}$ is the exact output for a slightly perturbed x)
- $\check{\kappa} = x^T(y + \delta y)$;
($\check{\kappa}$ is the exact output for a slightly perturbed y)
- $\check{\kappa} = x^T y + \delta \kappa$.
($\check{\kappa}$ equals the exact result plus an error)

The first two are backward error results (error is accumulated onto input parameters, showing that the algorithm is numerically stable since it yields the exact output for a slightly perturbed input) while the last one is a forward error result (error is accumulated onto the answer). We will see that in different situations, a different error result may be needed by analyses of operations that require a dot product.

Let us focus on the second result. Ideally one would show that each of the entries of y is slightly perturbed relative to that entry:

$$\delta y = \begin{pmatrix} \sigma_0 \psi_0 \\ \vdots \\ \sigma_{n-1} \psi_{n-1} \end{pmatrix} = \begin{pmatrix} \sigma_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{n-1} \end{pmatrix} \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \Sigma y,$$

where each σ_i is "small" and $\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{n-1})$. The following special structure of Σ , inspired by (6.3.3) will be used in the remainder of this note:

$$\Sigma^{(n)} = \begin{cases} 0 \times 0 \text{ matrix} & \text{if } n = 0 \\ \theta_1 & \text{if } n = 1 \\ \text{diag}(\theta_n, \theta_n, \theta_{n-1}, \dots, \theta_2) & \text{otherwise} . \end{cases}$$

Recall that θ_j is an order of magnitude variable with $|\theta_j| \leq \gamma_j$.

Homework 6.3.3.1 Let $k \geq 0$ and assume that $|\epsilon_1|, |\epsilon_2| \leq \epsilon_{\text{mach}}$, with $\epsilon_1 = 0$ if $k = 0$. Show that

$$\begin{pmatrix} I + \Sigma^{(k)} & 0 \\ 0 & (1 + \epsilon_1) \end{pmatrix} (1 + \epsilon_2) = (I + \Sigma^{(k+1)}).$$

Hint: Reason the cases where $k = 0$ and $k = 1$ separately from the case where $k > 1$.

Solution. Case: $k = 0$.

Then

$$\begin{aligned}
 & \left(\begin{array}{cc} I + \Sigma^{(k)} & 0 \\ 0 & (1 + \epsilon_1) \end{array} \right) (1 + \epsilon_2) \\
 & = \quad < k = 0 \text{ means } (I + \Sigma^{(k)}) \text{ is } 0 \times 0 \text{ and } (1 + \epsilon_1) = (1 + 0) > \\
 & (1 + 0)(1 + \epsilon_2) \\
 & = \\
 & (1 + \epsilon_2) \\
 & = \\
 & (1 + \theta_1) \\
 & = \\
 & (I + \Sigma^{(1)}).
 \end{aligned}$$

Case: $k = 1$.

Then

$$\begin{aligned}
 & \left(\begin{array}{cc} I + \Sigma^{(k)} & 0 \\ 0 & (1 + \epsilon_1) \end{array} \right) (1 + \epsilon_2) \\
 & = \\
 & \left(\begin{array}{cc} 1 + \theta_1 & 0 \\ 0 & (1 + \epsilon_1) \end{array} \right) (1 + \epsilon_2) \\
 & = \\
 & \left(\begin{array}{cc} (1 + \theta_1)(1 + \epsilon_2) & 0 \\ 0 & (1 + \epsilon_1)(1 + \epsilon_2) \end{array} \right) \\
 & = \\
 & \left(\begin{array}{cc} (1 + \theta_2) & 0 \\ 0 & (1 + \theta_2) \end{array} \right) \\
 & = \\
 & (I + \Sigma^{(2)}).
 \end{aligned}$$

Case: $k > 1$.

Notice that

$$\begin{aligned}
 & (I + \Sigma^{(k)})(1 + \epsilon_2) \\
 & = \\
 & \left(\begin{array}{cccc} 1 + \theta_k & 0 & 0 & \cdots & 0 \\ 0 & 1 + \theta_k & 0 & \cdots & 0 \\ 0 & 0 & 1 + \theta_{k-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 + \theta_2 \end{array} \right) (1 + \epsilon_2) \\
 & = \\
 & \left(\begin{array}{cccc} 1 + \theta_{k+1} & 0 & 0 & \cdots & 0 \\ 0 & 1 + \theta_{k+1} & 0 & \cdots & 0 \\ 0 & 0 & 1 + \theta_k & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 + \theta_3 \end{array} \right)
 \end{aligned}$$

Then

$$\begin{aligned}
 & \left(\begin{array}{cc} I + \Sigma^{(k)} & 0 \\ 0 & (1 + \epsilon_1) \end{array} \right) (1 + \epsilon_2) \\
 &= \left(\begin{array}{cc} (I + \Sigma^{(k)})(1 + \epsilon_2) & 0 \\ 0 & (1 + \epsilon_1)(1 + \epsilon_2) \end{array} \right) \\
 &= \left(\begin{array}{cc} \left(\begin{array}{ccccc} 1 + \theta_{k+1} & 0 & 0 & \cdots & 0 \\ 0 & 1 + \theta_{k+1} & 0 & \cdots & 0 \\ 0 & 0 & 1 + \theta_k & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 + \theta_3 \end{array} \right) & 0 \\ 0 & (1 + \theta_2) \end{array} \right) \\
 &= (I + \Sigma^{(k+1)}).
 \end{aligned}$$

We state a theorem that captures how error is accumulated by the algorithm.

Theorem 6.3.3.1 Let $x, y \in \mathbb{R}^n$ and let $\kappa := x^T y$ be computed in the order indicated by

$$(\cdots ((\chi_0 \psi_0 + \chi_1 \psi_1) + \chi_2 \psi_2) + \cdots) + \chi_{n-1} \psi_{n-1}.$$

Then

$$\check{\kappa} = [x^T y] = x^T (I + \Sigma^{(n)}) y.$$

Proof.

Proof by Mathematical Induction on n , the size of vectors x and y .

- Base case.

$m(x) = m(y) = 0$. Trivial!

- Inductive Step.

Inductive Hypothesis (I.H.): Assume that if $x_T, y_T \in \mathbb{R}^k$, $k > 0$, then

$$\text{fl}(x_T^T y_T) = x_T^T (I + \Sigma_T) y_T, \text{ where } \Sigma_T = \Sigma^{(k)}.$$

We will show that when $x_T, y_T \in \mathbb{R}^{k+1}$, the equality $\text{fl}(x_T^T y_T) = x_T^T (I + \Sigma_T) y_T$ holds true again. Assume that $x_T, y_T \in \mathbb{R}^{k+1}$, and partition $x_T \rightarrow \begin{pmatrix} x_0 \\ \chi_1 \end{pmatrix}$ and $y_T \rightarrow \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}$.

Then

$$\begin{aligned}
 & \text{fl}\left(\begin{pmatrix} x_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}\right) \\
 &= \quad <\text{definition}> \\
 & \text{fl}((\text{fl}(x_0^T y_0) + \text{fl}(\chi_1 \psi_1))) \\
 &= \quad <\text{I.H. with } x_T = x_0, y_T = y_0, \text{ and } \Sigma_0 = \Sigma^{(k)}> \\
 & \text{fl}(x_0^T (I + \Sigma_0) y_0 + \text{fl}(\chi_1 \psi_1)) \\
 &= \quad <\text{SCM, twice}> \\
 & \left(x_0^T (I + \Sigma_0) y_0 + \chi_1 \psi_1 (1 + \epsilon_*)\right) (1 + \epsilon_+) \\
 &= \quad <\text{rearrangement}> \\
 & \begin{pmatrix} x_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (I + \Sigma_0) & 0 \\ 0 & (1 + \epsilon_*) \end{pmatrix} (1 + \epsilon_+) \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \\
 &= \quad <\text{renaming}> \\
 & x_T^T (I + \Sigma_T) y_T
 \end{aligned}$$

where $|\epsilon_*|, |\epsilon_+| \leq \epsilon_{\text{mach}}$, $\epsilon_+ = 0$ if $k = 0$, and

$$(I + \Sigma_T) = \begin{pmatrix} (I + \Sigma_0) & 0 \\ 0 & (1 + \epsilon_*) \end{pmatrix} (1 + \epsilon_+)$$

so that $\Sigma_T = \Sigma^{(k+1)}$.

- By the Principle of Mathematical Induction, the result holds.

■

A number of useful consequences of [Theorem 6.3.3.1](#) follow. These will be used later as an inventory (library) of error results from which to draw when analyzing operations and algorithms that utilize a dot product.

Corollary 6.3.3.2 *Under the assumptions of [Theorem 6.3.3.1](#) the following relations hold:*

R-1B $\check{\kappa} = (x + \delta x)^T y$, where $|\delta x| \leq \gamma_n |x|$,

R-2B $\check{\kappa} = x^T (y + \delta y)$, where $|\delta y| \leq \gamma_n |y|$;

R-1F $\check{\kappa} = x^T y + \delta \kappa$, where $|\delta \kappa| \leq \gamma_n |x|^T |y|$.

Proof. R-1B

We leave the proof of [Corollary 6.3.3.2](#) R-1B as an exercise.

R-2B

The proof of [Corollary 6.3.3.2](#) R-2B is, of course, just a minor modification of the proof of [Corollary 6.3.3.2](#) R-1B.

R-1F

For Corollary 6.3.3.2 R-1F, let $\delta\kappa = x^T \Sigma^{(n)} y$, where $\Sigma^{(n)}$ is as in Theorem 6.3.3.1. Then

$$\begin{aligned} |\delta\kappa| &= |x^T \Sigma^{(n)} y| \\ &\leq |\chi_0||\theta_n||\psi_0| + |\chi_1||\theta_n||\psi_1| + \cdots + |\chi_{n-1}||\theta_2||\psi_{n-1}| \\ &\leq \gamma_n|\chi_0||\psi_0| + \gamma_n|\chi_1||\psi_1| + \cdots + \gamma_2|\chi_{n-1}||\psi_{n-1}| \\ &\leq \gamma_n|x|^T|y|. \end{aligned}$$

■

Homework 6.3.3.2 Prove Corollary 6.3.3.2 R1-B.

Solution. From Theorem 6.3.3.1 we know that

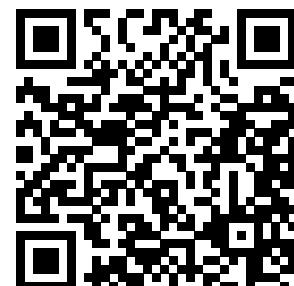
$$\check{\kappa} = x^T(I + \Sigma^{(n)})y = (x + \underbrace{\Sigma^{(n)}x}_{\delta x})^T y.$$

Then

$$\begin{aligned} |\delta x| &= |\Sigma^{(n)}x| = \left\| \begin{pmatrix} \theta_n\chi_0 \\ \theta_n\chi_1 \\ \theta_{n-1}\chi_2 \\ \vdots \\ \theta_2\chi_{n-1} \end{pmatrix} \right\| = \left\| \begin{pmatrix} |\theta_n\chi_0| \\ |\theta_n\chi_1| \\ |\theta_{n-1}\chi_2| \\ \vdots \\ |\theta_2\chi_{n-1}| \end{pmatrix} \right\| = \left\| \begin{pmatrix} |\theta_n||\chi_0| \\ |\theta_n||\chi_1| \\ |\theta_{n-1}||\chi_2| \\ \vdots \\ |\theta_2||\chi_{n-1}| \end{pmatrix} \right\| \\ &\leq \left\| \begin{pmatrix} |\theta_n||\chi_0| \\ |\theta_n||\chi_1| \\ |\theta_n||\chi_2| \\ \vdots \\ |\theta_n||\chi_{n-1}| \end{pmatrix} \right\| \leq \left\| \begin{pmatrix} \gamma_n|\chi_0| \\ \gamma_n|\chi_1| \\ \gamma_n|\chi_2| \\ \vdots \\ \gamma_n|\chi_{n-1}| \end{pmatrix} \right\| = \gamma_n|x|. \end{aligned}$$

(Note: strictly speaking, one should probably treat the case $n = 1$ separately.)

6.3.4 Matrix-vector multiplication



YouTube: <https://www.youtube.com/watch?v=q7rACP0u4ZQ>

Assume $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$. Partition

$$A = \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix}.$$

Then computing $y := Ax$ can be orchestrated as

$$\begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} := \begin{pmatrix} \tilde{a}_0^T x \\ \tilde{a}_1^T x \\ \vdots \\ \tilde{a}_{m-1}^T x \end{pmatrix}. \quad (6.3.4)$$

From R-1B 6.3.3.2 regarding the dot product we know that

$$\begin{aligned} \check{y} &= \begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} (\tilde{a}_0 + \delta\tilde{a}_0)^T x \\ (\tilde{a}_1 + \delta\tilde{a}_1)^T x \\ \vdots \\ (\tilde{a}_{m-1} + \delta\tilde{a}_{m-1})^T x \end{pmatrix} \\ &= \left(\begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} + \begin{pmatrix} \delta\tilde{a}_0^T \\ \delta\tilde{a}_1^T \\ \vdots \\ \delta\tilde{a}_{m-1}^T \end{pmatrix} \right) x = (A + \Delta A)x, \end{aligned}$$

where $|\delta\tilde{a}_i| \leq \gamma_n |\tilde{a}_i|$, $i = 0, \dots, m-1$, and hence $|\Delta A| \leq \gamma_n |A|$.

Also, from Corollary 6.3.3.2 R-1F regarding the dot product we know that

$$\check{y} = \begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} \tilde{a}_0^T x + \delta\psi_0 \\ \tilde{a}_1^T x + \delta\psi_1 \\ \vdots \\ \tilde{a}_{m-1}^T x + \delta\psi_{m-1} \end{pmatrix} = \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} x + \begin{pmatrix} \delta\psi_0 \\ \delta\psi_1 \\ \vdots \\ \delta\psi_{m-1} \end{pmatrix} = Ax + \delta y.$$

where $|\delta\psi_i| \leq \gamma_n |\tilde{a}_i|^T |x|$ and hence $|\delta y| \leq \gamma_n |A| |x|$.

The above observations can be summarized in the following theorem:

Theorem 6.3.4.1 *Error results for matrix-vector multiplication. Let $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ and consider the assignment $y := Ax$ implemented via dot products as expressed in (6.3.4). Then these equalities hold:*

R-1B $\check{y} = (A + \Delta A)x$, where $|\Delta A| \leq \gamma_n |A|$.

R-1F $\check{y} = Ax + \delta y$, where $|\delta y| \leq \gamma_n |A| |x|$.

Ponder This 6.3.4.1 In the above theorem, could one instead prove the result

$$\check{y} = A(x + \delta x),$$

where δx is "small"?

Solution. The answer is "sort of". The reason is that for each individual element of y

$$\check{\psi}_i = \tilde{a}_i^T(x + \delta x)$$

which would appear to support that

$$\begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} \tilde{a}_0^T(x + \delta x) \\ \tilde{a}_1^T(x + \delta x) \\ \vdots \\ \tilde{a}_{m-1}^T(x + \delta x) \end{pmatrix}.$$

However, the δx for each entry $\check{\psi}_i$ is different, meaning that we cannot factor out $x + \delta x$ to find that $\check{y} = A(x + \delta x)$.

However, one could argue that we know that $\check{y} = Ax + \delta y$ where $|\delta y| \leq \gamma_n |A| \|x\|$. Hence if $A\delta x = \delta y$ then $A(x + \delta x) = \check{y}$. This would mean that δy is in the column space of A . (For example, if A is nonsingular). However, that is not quite what we are going for here.

6.3.5 Matrix-matrix multiplication



YouTube: <https://www.youtube.com/watch?v=pvBMuIzIob8>



The idea behind backward error analysis is that the computed result is the exact result when computing with changed inputs. Let's consider matrix-matrix multiplication:

$$C := AB.$$

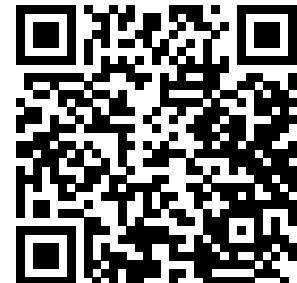
What we would like to be able to show is that there exist ΔA and ΔB such that the computed result, \check{C} , satisfies

$$\check{C} := (A + \Delta A)(B + \Delta B).$$

Let's think about this...

Ponder This 6.3.5.1 Can one find matrices ΔA and ΔB such that

$$\check{C} = (A + \Delta A)(B + \Delta B)?$$



YouTube: <https://www.youtube.com/watch?v=3d6kQ6rnRhA>

For matrix-matrix multiplication, it *is* possible to "throw" the error onto the result, as summarized by the following theorem:

Theorem 6.3.5.1 Forward error for matrix-matrix multiplication. *Let $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$ and consider the assignment $C := AB$ implemented via matrix-vector multiplication. Then there exists $\Delta C \in \mathbb{R}^{m \times n}$ such that*

$$\check{C} = AB + \Delta C, \text{ where } |\Delta C| \leq \gamma_k |A| |B|.$$

Homework 6.3.5.2 Prove Theorem 6.3.5.1.

Solution. Partition

$$C = \left(\begin{array}{c|c|c|c} c_0 & c_1 & \cdots & c_{n-1} \end{array} \right) \quad \text{and} \quad B = \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right).$$

Then

$$\left(\begin{array}{c|c|c|c} c_0 & c_1 & \cdots & c_{n-1} \end{array} \right) := \left(\begin{array}{c|c|c|c} Ab_0 & Ab_1 & \cdots & Ab_{n-1} \end{array} \right).$$

From R-1F 6.3.4.1 regarding matrix-vector multiplication we know that

$$\begin{aligned} \left(\begin{array}{c|c|c|c} \check{c}_0 & \check{c}_1 & \cdots & \check{c}_{n-1} \end{array} \right) &= \left(\begin{array}{c|c|c|c} Ab_0 + \delta c_0 & Ab_1 + \delta c_1 & \cdots & Ab_{n-1} + \delta c_{n-1} \end{array} \right) \\ &= \left(\begin{array}{c|c|c|c} Ab_0 & Ab_1 & \cdots & Ab_{n-1} \end{array} \right) + \left(\begin{array}{c|c|c|c} \delta c_0 & \delta c_1 & \cdots & \delta c_{n-1} \end{array} \right) \\ &= AB + \Delta C. \end{aligned}$$

where $|\delta c_j| \leq \gamma_k |A| |b_j|$, $j = 0, \dots, n - 1$, and hence $|\Delta C| \leq \gamma_k |A| |B|$.



YouTube: <https://www.youtube.com/watch?v=rxKba-pnquQ>

Remark 6.3.5.2 In practice, matrix-matrix multiplication is often the parameterized operation $C := \alpha AB + \beta C$. A consequence of Theorem 6.3.5.1 is that for $\beta \neq 0$, the error *can* be attributed to a change in parameter C , which means the error has been "thrown back" onto an input parameter.

6.4 Error Analysis for Solving Linear Systems

6.4.1 Numerical stability of triangular solve



YouTube: https://www.youtube.com/watch?v=ayj_rNkSMig

We now use the error results for the dot product to derive a backward error result for solving $Lx = y$, where L is an $n \times n$ lower triangular matrix, via the algorithm in [Figure 6.4.1.1](#), a variation on the algorithm in [Figure 5.3.5.1](#) that stores the result in vector x and does not assume that L is unit lower triangular.

Solve $Lx = y$
$L \rightarrow \left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right), x \rightarrow \left(\begin{array}{c} x_T \\ x_B \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$
L_{TL} is 0×0 and y_T, x_T have 0 elements
while $n(L_{TL}) < n(L)$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} x_T \\ x_B \end{array} \right) \rightarrow \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_2 \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$
$\underline{\chi_1 := (\psi_1 - l_{10}^T x_0) / \lambda_{11}}$
$\left(\begin{array}{c c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} x_T \\ x_B \end{array} \right) \leftarrow \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_2 \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$
endwhile

Figure 6.4.1.1 Dot product based lower triangular solve algorithm.

To establish the backward error result for this algorithm, we need to understand the error incurred in the key computation

$$\chi_1 := (\psi_1 - l_{10}^T x_0) / \lambda_{11}.$$

The following theorem gives the required (forward error) result, abstracted away from the specifics of how it occurs in the lower triangular solve algorithm.

Lemma 6.4.1.2 *Let $n \geq 1$, $\lambda, \nu \in \mathbb{R}$ and $x, y \in \mathbb{R}^n$. Assume $\lambda \neq 0$ and consider the computation*

$$\nu := (\alpha - x^T y) / \lambda,$$

Then

$$(\lambda + \delta\lambda)\check{\nu} = \alpha - (x + \delta x)^T y, \text{ where } |\delta x| \leq \gamma_n |x| \text{ and } |\delta\lambda| \leq \gamma_2 |\lambda|.$$

Homework 6.4.1.1 Prove Lemma 6.4.1.2

Hint. Use the Alternative Computations Model ([Subsubsection 6.2.3.3](#)) appropriately.

Solution. We know that

- From [Corollary 6.3.3.2](#) R-1B: if $\beta = x^T y$ then $\check{\beta} = (x + \delta x)^T y$ where $|\delta x| \leq \gamma_n |x|$.
- From the ACM ([Subsubsection 6.2.3.3](#)): If $\nu = (\alpha - \beta)/\lambda$ then

$$\check{\nu} = \frac{\alpha - \beta}{\lambda} \frac{1}{(1 + \epsilon_-)(1 + \epsilon_\beta)},$$

where $|\epsilon_-| \leq \epsilon_{\text{mach}}$ and $|\epsilon_\beta| \leq \epsilon_{\text{mach}}$.

Hence

$$\check{\nu} = \frac{\alpha - (x + \delta x)^T y}{\lambda} \frac{1}{(1 + \epsilon_-)(1 + \epsilon_\beta)},$$

or, equivalently,

$$\lambda(1 + \epsilon_-)(1 + \epsilon_\beta)\check{\nu} = \alpha - (x + \delta x)^T y,$$

or,

$$\lambda(1 + \theta_2)\check{\nu} = \alpha - (x + \delta x)^T y,$$

where $|\theta_2| \leq \gamma_2$, which can also be written as

$$(\lambda + \delta\lambda)\check{\nu} = \alpha - (x + \delta x)^T y,$$

where $\delta\lambda = \theta_2\lambda$ and hence $|\delta\lambda| \leq \gamma_2 \|\lambda\|$.

The error result for the algorithm in [Figure 6.4.1.1](#) is given by

Theorem 6.4.1.3 *Let $L \in \mathbb{R}^{n \times n}$ be a nonsingular lower triangular matrix and let \check{x} be the computed result when executing [Figure 6.4.1.1](#) to solve $Lx = y$ under the computation model from [Subsection 6.2.3](#). Then there exists a matrix ΔL such that*

$$(L + \Delta L)\check{x} = y \text{ where } |\Delta L| \leq \max(\gamma_2, \gamma_{n-1})|L|.$$

The reasoning behind the result is that one expects the maximal error to be incurred during the final iteration when computing $\chi_1 := (\psi_1 - l_{10}^T x_0)/\lambda_{11}$. This fits [Lemma 6.4.1.2](#), except that this assignment involves a dot product with vectors of length $n - 1$ rather than of length n .

You now prove [Theorem 6.4.1.3](#) by first proving the special cases where $n = 1$ and $n = 2$, and then the general case.

Homework 6.4.1.2 Prove Theorem 6.4.1.3 for the case where $n = 1$.

Solution. Case 1: $n = 1$.

The system looks like $\lambda_{11}\chi_1 = \psi_1$ so that

$$\chi_1 = \psi_1 / \lambda_{11}$$

and

$$\check{\chi}_1 = \psi_1 / \lambda_{11} \frac{1}{1 + \epsilon_\gamma}$$

Rearranging gives us

$$\lambda_{11}\check{\chi}_1(1 + \epsilon_\gamma) = \psi_1$$

or

$$(\lambda_{11} + \delta\lambda_{11})\check{\chi}_1 = \psi_1$$

where $\delta\lambda_{11} = \epsilon_\gamma/\lambda_{11}$ and hence

$$\begin{aligned} |\delta\lambda_{11}| &= |\epsilon_\gamma|/|\lambda_{11}| \\ &\leq \gamma_1|\lambda_{11}| \\ &\leq \gamma_2|\lambda_{11}| \\ &\leq \max(\gamma_2, \gamma_{n-1})|\lambda_{11}|. \end{aligned}$$

Homework 6.4.1.3 Prove Theorem 6.4.1.3 for the case where $n = 2$.

Solution. Case 2: $n = 2$.

The system now looks like

$$\left(\begin{array}{c|c} \lambda_{00} & 0 \\ \hline \lambda_{10} & \lambda_{11} \end{array} \right) \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix}.$$

From the proof of Case 1 we know that

$$(\lambda_{00} + \delta\lambda_{00})\check{\chi}_0 = \psi_0, \text{ where } |\delta\lambda_{00}| \leq \gamma_1|\lambda_{00}|. \quad (6.4.1)$$

Since $\chi_1 = (\psi_1 - \lambda_{10}\check{\chi}_0)/\lambda_{11}$, Lemma 6.4.1.2 tells us that

$$(\lambda_{10} + \delta\lambda_{10})\check{\chi}_0 + (\lambda_{11} + \delta\lambda_{11})\check{\chi}_1 = \psi_1, \quad (6.4.2)$$

where

$$|\delta\lambda_{10}| \leq \gamma_1|\lambda_{10}| \text{ and } |\delta\lambda_{11}| \leq \gamma_2|\lambda_{11}|.$$

(6.4.1) and (6.4.2) can be combined into

$$\left(\begin{array}{c|c} \lambda_{00} + \delta\lambda_{00} & 0 \\ \hline \lambda_{10} + \delta\lambda_{10} & \lambda_{11} + \delta\lambda_{11} \end{array} \right) \begin{pmatrix} \check{\chi}_0 \\ \check{\chi}_1 \end{pmatrix} = \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix},$$

where

$$\left(\begin{array}{c|c} |\delta\lambda_{00}| & 0 \\ \hline |\delta\lambda_{10}| & |\delta\lambda_{11}| \end{array} \right) \leq \left(\begin{array}{c|c} \gamma_1|\lambda_{00}| & 0 \\ \hline \gamma_1|\lambda_{10}| & \gamma_2|\lambda_{11}| \end{array} \right).$$

Since $\gamma_1 \leq \gamma_2$

$$\left| \left(\begin{array}{c|c} \delta\lambda_{00} & 0 \\ \hline \delta\lambda_{10} & \delta\lambda_{11} \end{array} \right) \right| \leq \max(\gamma_2, \gamma_{n-1}) \left| \left(\begin{array}{c|c} \lambda_{00} & 0 \\ \hline \lambda_{10} & \lambda_{11} \end{array} \right) \right|.$$

Homework 6.4.1.4 Prove Theorem 6.4.1.3 for $n \geq 1$.

Solution. We will utilize a proof by induction.

- Case 1: $n = 1$.

See Homework 6.4.1.2.

- Case 2: $n = 2$.

See Homework 6.4.1.3.

- Case 3: $n > 2$.

The system now looks like

$$\left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array} \right) \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right) = \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right), \quad (6.4.3)$$

where $L_{00} \in \mathbb{R}^{(n-1) \times (n-1)}$, and the inductive hypothesis states that

$$(L_{00} + \Delta L_{00})\check{x}_0 = y_0 \text{ where } |\Delta L_{00}| \leq \max(\gamma_2, \gamma_{n-2})|L_{00}|.$$

Since $\chi_1 = (\psi_1 - l_{10}^T \check{x}_0)/\lambda_{11}$, Lemma 6.4.1.2 tells us that

$$(l_{10} + \delta l_{10})^T \check{x}_0 + (\lambda_{11} + \delta \lambda_{11})\check{\chi}_1 = \psi_1, \quad (6.4.4)$$

where $|\delta l_{10}| \leq \gamma_{n-1}|l_{10}|$ and $|\delta \lambda_{11}| \leq \gamma_2|\lambda_{11}|$.

(6.4.3) and (6.4.4) can be combined into

$$\left(\begin{array}{c|c} L_{00} + \delta L_{00} & 0 \\ \hline (l_{10} + \delta l_{10})^T & \lambda_{11} + \delta \lambda_{11} \end{array} \right) \left(\begin{array}{c} \check{x}_0 \\ \check{\chi}_1 \end{array} \right) = \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right),$$

where

$$\left(\begin{array}{c|c} |\delta L_{00}| & 0 \\ \hline |\delta l_{10}^T| & |\delta \lambda_{11}| \end{array} \right) \leq \left(\begin{array}{c|c} \max(\gamma_2, \gamma_{n-2})|L_{00}| & 0 \\ \hline \gamma_{n-1}|l_{10}^T| & \gamma_2|\lambda_{11}| \end{array} \right)$$

and hence

$$\left| \left(\begin{array}{c|c} \delta L_{00} & 0 \\ \hline \delta l_{10}^T & \delta \lambda_{11} \end{array} \right) \right| \leq \max(\gamma_2, \gamma_{n-1}) \left| \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array} \right) \right|.$$

- By the Principle of Mathematical Induction, the result holds for all $n \geq 1$.



YouTube: https://www.youtube.com/watch?v=GB7wj7_dhCE

A careful examination of the solution to [Homework 6.4.1.2](#), together with the fact that $\gamma_{n-1} \leq \gamma_n$ allows us to state a slightly looser, but cleaner, result of [Theorem 6.4.1.3](#):

Corollary 6.4.1.4 *Let $L \in \mathbb{R}^{n \times n}$ be a nonsingular lower triangular matrix and let \check{x} be the computed result when executing [Figure 6.4.1.1](#) to solve $Lx = y$ under the computation model from [Subsection 6.2.3](#). Then there exists a matrix ΔL such that*

$$(L + \Delta L)\check{x} = y \text{ where } |\Delta L| \leq \gamma_n |L|.$$

6.4.2 Numerical stability of LU factorization



YouTube: <https://www.youtube.com/watch?v=fds-FeL28ok>

The numerical stability of various LU factorization algorithms as well as the triangular solve algorithms can be found in standard graduate level numerical linear algebra texts [19] [21]. Of particular interest may be the analysis of the [Crout variant of LU factorization 5.5.1.4](#) in

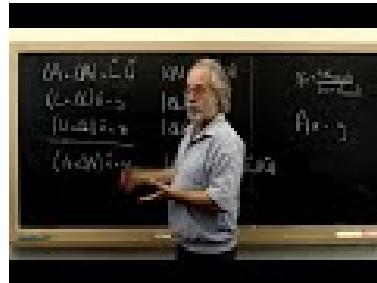
- [6] Paolo Bientinesi, Robert A. van de Geijn, Goal-Oriented and Modular Stability Analysis, SIAM Journal on Matrix Analysis and Applications , Volume 32 Issue 1, February 2011.
- [7] Paolo Bientinesi, Robert A. van de Geijn, The Science of Deriving Stability Analyses, FLAME Working Note #33. Aachen Institute for Computational Engineering Sciences, RWTH Aachen. TR AICES-2008-2. November 2008. (Technical report version with exercises.)

since these papers use the same notation as we use in our notes. Here is the pertinent result from those papers:

Theorem 6.4.2.1 Backward error of Crout variant for LU factorization. *Let $A \in \mathbb{R}^{n \times n}$ and let the LU factorization of A be computed via the Crout variant, yielding approximate factors \check{L} and \check{U} . Then*

$$(A + \Delta A) = \check{L}\check{U} \quad \text{with} \quad |\Delta A| \leq \gamma_n |\check{L}| |\check{U}|.$$

6.4.3 Numerical stability of linear solve via LU factorization



YouTube: <https://www.youtube.com/watch?v=c1NsTSCpe1k>

Let us now combine the results from [Subsection 6.4.1](#) and [Subsection 6.4.2](#) into a backward error result for solving $Ax = y$ via LU factorization and two triangular solves.

Theorem 6.4.3.1 Let $A \in \mathbb{R}^{n \times n}$ and $x, y \in \mathbb{R}^n$ with $Ax = y$. Let \check{x} be the approximate solution computed via the following steps:

- Compute the LU factorization, yielding approximate factors \check{L} and \check{U} .
- Solve $\check{L}z = y$, yielding approximate solution \check{z} .
- Solve $\check{U}\check{x} = \check{z}$, yielding approximate solution \check{x} .

Then

$$(A + \Delta A)\check{x} = y \quad \text{with} \quad |\Delta A| \leq (3\gamma_n + \gamma_n^2)|\check{L}||\check{U}|.$$

We refer the interested learner to the proof in the previously mentioned papers [\[6\]](#) [\[7\]](#).

Homework 6.4.3.1 The question left is how a change in a nonsingular matrix affects the accuracy of the solution of a linear system that involves that matrix. We saw in [Subsection 1.4.1](#) that if

$$Ax = y \text{ and } A(x + \delta x) = y + \delta y$$

then

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\delta y\|}{\|y\|}$$

when $\|\cdot\|$ is a subordinate norm. But what we want to know is how a change in A affects the solution:

$$Ax = y \text{ and } (A + \Delta A)(x + \delta x) = y$$

then

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A) \frac{\|\Delta A\|}{\|A\|}}{1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}}.$$

Prove this!

Solution.

$$Ax = y \text{ and } (A + \Delta A)(x + \delta x) = y$$

implies that

$$(A + \Delta A)(x + \delta x) = Ax$$

or, equivalently,

$$\Delta Ax + A\delta x + \Delta A\delta x = 0.$$

We can rewrite this as

$$\delta x = A^{-1}(-\Delta Ax - \Delta A\delta x)$$

so that

$$\|\delta x\| = \|A^{-1}(-\Delta Ax - \Delta A\delta x)\| \leq \|A^{-1}\| \|\Delta A\| \|x\| + \|A^{-1}\| \|\Delta A\| \|\delta x\|.$$

This can be rewritten as

$$(1 - \|A^{-1}\| \|\Delta A\|) \|\delta x\| \leq \|A^{-1}\| \|\Delta A\| \|x\|$$

and finally

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\| \|\Delta A\|}{1 - \|A^{-1}\| \|\Delta A\|}$$

and finally

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A\| \|A^{-1}\| \frac{\|\Delta A\|}{\|A\|}}{1 - \|A\| \|A^{-1}\| \frac{\|\Delta A\|}{\|A\|}}.$$

The last homework brings up a good question: If A is nonsingular, how small does ΔA need to be for it to be nonsingular?

Theorem 6.4.3.2 *Let A be nonsingular, $\|\cdot\|$ be a subordinate norm, and*

$$\frac{\|\Delta A\|}{\|A\|} < \frac{1}{\kappa(A)}.$$

Then $A + \Delta A$ is nonsingular.

Proof. Proof by contradiction.

Assume that A is nonsingular,

$$\frac{\|\Delta A\|}{\|A\|} < \frac{1}{\kappa(A)}.$$

and $A + \Delta A$ is singular. We will show this leads to a contradiction.

Since $A + \Delta A$ is singular, there exists $x \neq 0$ such that $(A + \Delta A)x = 0$. We can rewrite this as

$$x = -A^{-1}\Delta Ax$$

and hence

$$\|x\| = \|A^{-1}\Delta Ax\| \leq \|A^{-1}\| \|\Delta A\| \|x\|.$$

Dividing both sides by $\|x\|$ yields

$$1 \leq \|A^{-1}\| \|\Delta A\|$$

and hence $\frac{1}{\|A^{-1}\|} \leq \|\Delta A\|$ and finally

$$\frac{1}{\|A\| \|A^{-1}\|} \leq \frac{\|\Delta A\|}{\|A\|},$$

which is a contradiction. ■

6.4.4 Numerical stability of linear solve via LU factorization with partial pivoting



YouTube: <https://www.youtube.com/watch?v=n95C8qjMBcI>

The analysis of LU factorization without partial pivoting is related to that of LU factorization with partial pivoting as follows:

- We have shown that LU factorization with partial pivoting is equivalent to the LU factorization without partial pivoting on a pre-permuted matrix: $PA = LU$, where P is a permutation matrix.
- The permutation (exchanging of rows) doesn't involve any floating point operations and therefore does not generate error.

It can therefore be argued that, as a result, the error that is accumulated is equivalent with or without partial pivoting

More slowly, what if we took the following approach to LU factorization with partial pivoting:

- Compute the LU factorization with partial pivoting yielding the pivot matrix P , the unit lower triangular matrix L , and the upper triangular matrix U . In exact arithmetic this would mean these matrices are related by $PA = LU$.
- In practice, no error exists in P (except that a wrong index of a row with which to pivot may result from roundoff error in the intermediate results in matrix A) and approximate factors \check{L} and \check{U} are computed.
- If we now took the pivot matrix P and formed $B = PA$ (without incurring error since rows are merely swapped) and then computed the LU factorization of B , then the computed L and U would equal exactly the \check{L} and \check{U} that resulted from computing the LU factorization with row pivoting with A in floating point arithmetic. Why? Because the exact same computations are performed although possibly with data that is temporarily in a different place in the matrix at the time of that computation.
- We know that therefore \check{L} and \check{U} satisfy

$$B + \Delta B = \check{L} \check{U}, \text{ where } |\Delta B| \leq \gamma_n |\check{L}| |\check{U}| ..$$

We conclude that

$$PA + \Delta B = \check{L}\check{U}, \text{ where } |\Delta B| \leq \gamma_n |\check{L}| |\check{U}|$$

or, equivalently,

$$P(A + \Delta A) = \check{L}\check{U}, \text{ where } P|\Delta A| \leq \gamma_n |\check{L}| |\check{U}|$$

where $\Delta B = P\Delta A$ and we note that $P|\Delta A| = |P\Delta A|$ (taking the absolute value of a matrix and then swapping rows yields the same matrix as when one first swaps the rows and then takes the absolute value).

6.4.5 Is LU with Partial Pivoting Stable?



YouTube: <https://www.youtube.com/watch?v=TdLM41LCma4>

The last unit gives a backward error result regarding LU factorization (and, by extension, LU factorization with pivoting):

$$(A + \Delta A) = \check{L}\check{U} \quad \text{with} \quad |\Delta A| \leq \gamma_n |\check{L}| |\check{U}|.$$

The question now is: does this mean that LU factorization with partial pivoting is stable? In other words, is ΔA , which we bounded with $|\Delta A| \leq \gamma_n |\check{L}| |\check{U}|$, always small relative to the entries of $|A|$? The following exercise gives some insight:

Homework 6.4.5.1 Apply LU with partial pivoting to

$$A = \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & 1 \\ -1 & -1 & 1 \end{pmatrix}.$$

Pivot only when necessary.

Solution. Notice that no pivoting is necessary. Eliminating the entries below the diagonal in the first column yields:

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & -1 & 2 \end{pmatrix}.$$

Eliminating the entries below the diagonal in the second column again does not require pivoting and yields:

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 4 \end{pmatrix}.$$

Homework 6.4.5.2 Generalize the insights from the last homework to a $n \times n$ matrix. What is the maximal element growth that is observed?

Solution. Consider

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & \cdots & 1 & 1 & \\ -1 & -1 & \cdots & -1 & 1 & \end{pmatrix}.$$

Notice that no pivoting is necessary when LU factorization with pivoting is performed.

Eliminating the entries below the diagonal in the first column yields:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 2 \\ 0 & -1 & 1 & \cdots & 0 & 2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & -1 & \cdots & 1 & 2 & \\ 0 & -1 & \cdots & -1 & 2 & \end{pmatrix}.$$

Eliminating the entries below the diagonal in the second column again does not require pivoting and yields:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 2 \\ 0 & 0 & 1 & \cdots & 0 & 4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 4 & \\ 0 & 0 & \cdots & -1 & 4 & \end{pmatrix}.$$

Continuing like this for the remaining columns, eliminating the entries below the diagonal leaves us with the upper triangular matrix

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 2 \\ 0 & 0 & 1 & \cdots & 0 & 4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 2^{n-2} & \\ 0 & 0 & \cdots & 0 & 2^{n-1} & \end{pmatrix}.$$

From these exercises we conclude that even LU factorization with partial pivoting can yield large (exponential) element growth in U .

In practice, this does not seem to happen and LU factorization is considered to be stable.

6.5 Enrichments

6.5.1 Systematic derivation of backward error analyses

Throughout the course, we have pointed out that the FLAME notation facilitates the systematic derivation of linear algebra algorithms. The papers

- [6] Paolo Bientinesi, Robert A. van de Geijn, Goal-Oriented and Modular Stability Analysis, SIAM Journal on Matrix Analysis and Applications , Volume 32 Issue 1, February 2011.
- [7] Paolo Bientinesi, Robert A. van de Geijn, The Science of Deriving Stability Analyses, FLAME Working Note #33. Aachen Institute for Computational Engineering Sciences, RWTH Aachen. TR AICES-2008-2. November 2008. (Technical report version of the SIAM paper, but with exercises.)

extend this to the systematic derivation of the backward error analysis of algorithms. Other publications and texts present error analyses on a case-by-case basis (much like we do in these materials) rather than as a systematic and comprehensive approach.

6.5.2 LU factorization with pivoting can fail in practice

While LU factorization with pivoting is considered to be a numerically stable approach to solving linear systems, the following paper discusses cases where it may fail in practice:

- [18] Leslie V. Foster, Gaussian elimination with partial pivoting can fail in practice, SIAM Journal on Matrix Analysis and Applications, 15 (1994), pp. 1354–1362.

Also of interest may be the paper

- [46] Stephen J. Wright, A Collection of Problems for Which Gaussian Elimination with Partial Pivoting is Unstable, SIAM Journal on Scientific Computing, Vol. 14, No. 1, 1993.

which discusses a number of (not necessarily practical) examples where LU factorization with pivoting fails.

6.6 Wrap Up

6.6.1 Additional homework

Homework 6.6.1.1 In Units 6.3.1-3 we analyzed how error accumulates when computing a dot product of x and y of size m in the order indicated by

$$\kappa = ((\cdots ((\chi_0 \psi_0 + \chi_1 \psi_1) + \chi_2 \psi_2) + \cdots) + \chi_{m-1} \psi_{m-1}).$$

Let's illustrate an alternative way of computing the dot product:

- For $m = 2$:

$$\kappa = \chi_0\psi_0 + \chi_1\psi_1$$

- For $m = 4$:

$$\kappa = (\chi_0\psi_0 + \chi_1\psi_1) + (\chi_2\psi_2 + \chi_3\psi_3)$$

- For $m = 8$:

$$\kappa = ((\chi_0\psi_0 + \chi_1\psi_1) + (\chi_2\psi_2 + \chi_3\psi_3)) + ((\chi_4\psi_4 + \chi_5\psi_5) + (\chi_6\psi_6 + \chi_7\psi_7))$$

and so forth. Analyze how under the SCM error accumulates and state backward stability results. You may assume that m is a power of two.

6.6.2 Summary

In our discussions, the set of floating point numbers, F , is the set of all numbers $\chi = \mu \times \beta^e$ such that

- $\beta = 2$,
- $\mu = \pm.\delta_0\delta_1 \cdots \delta_{t-1}$ (μ has only t (binary) digits), where $\delta_j \in \{0, 1\}$),
- $\delta_0 = 0$ iff $\mu = 0$ (the mantissa is normalized), and
- $-L \leq e \leq U$.

Definition 6.6.2.1 Machine epsilon (unit roundoff). The machine epsilon (unit round-off), ϵ_{mach} , is defined as the smallest positive floating point number χ such that the floating point number that represents $1 + \chi$ is greater than one. \diamond

$$\text{fl(expression)} = [\text{expression}]$$

equals the result when computing $\{\text{rm expression}\}$ using floating point computation (rounding or truncating as every intermediate result is stored). If

$$\kappa = \text{expression}$$

in exact arithmetic, then we done the associated floating point result with

$$\check{\kappa} = [\text{expression}].$$

The Standard Computational Model (SCM) assumes that, for any two floating point numbers χ and ψ , the basic arithmetic operations satisfy the equality

$$\text{fl}(\chi \text{ op } \psi) = (\chi \text{ op } \psi)(1 + \epsilon), |\epsilon| \leq \epsilon_{\text{mach}}, \text{ and op } \in \{+, -, *, /\}.$$

The Alternative Computational Model (ACM) assumes for the basic arithmetic operations that

$$\text{fl}(\chi \text{ op } \psi) = \frac{\chi \text{ op } \psi}{1 + \epsilon}, |\epsilon| \leq \epsilon_{\text{mach}}, \text{ and op } \in \{+, -, *, /\}.$$

Definition 6.6.2.2 Backward stable implementation. Given the mapping $f : D \rightarrow R$, where $D \subset \mathbb{R}^n$ is the domain and $R \subset \mathbb{R}^m$ is the range (codomain), let $\check{f} : D \rightarrow R$ be a computer implementation of this function. We will call \check{f} a backward stable (also called "numerically stable") implementation of f on domain \mathcal{D} if for all $x \in D$ there exists a \check{x} "close" to x such that $\check{f}(x) = f(\check{x})$. \diamond

- *Conditioning* is a property of the problem you are trying to solve. A problem is well-conditioned if a small change in the input is guaranteed to only result in a small change in the output. A problem is ill-conditioned if a small change in the input can result in a large change in the output.
- *Stability* is a property of an implementation. If the implementation, when executed with an input always yields an output that can be attributed to slightly changed input, then the implementation is backward stable.

Definition 6.6.2.3 Absolute value of vector and matrix. Given $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$,

$$|x| = \begin{pmatrix} |\chi_0| \\ |\chi_1| \\ \vdots \\ |\chi_{n-1}| \end{pmatrix} \quad \text{and} \quad |A| = \begin{pmatrix} |\alpha_{0,0}| & |\alpha_{0,1}| & \dots & |\alpha_{0,n-1}| \\ |\alpha_{1,0}| & |\alpha_{1,1}| & \dots & |\alpha_{1,n-1}| \\ \vdots & \vdots & \ddots & \vdots \\ |\alpha_{m-1,0}| & |\alpha_{m-1,1}| & \dots & |\alpha_{m-1,n-1}| \end{pmatrix}.$$

 \diamond

Definition 6.6.2.4 Let $\Delta \in \{<, \leq, =, \geq, >\}$ and $x, y \in \mathbb{R}^n$. Then

$$|x| \Delta |y| \quad \text{iff} \quad |\chi_i| \Delta |\psi_i|,$$

with $i = 0, \dots, n-1$. Similarly, given A and $B \in \mathbb{R}^{m \times n}$,

$$|A| \Delta |B| \quad \text{iff} \quad |\alpha_{ij}| \Delta |\beta_{ij}|,$$

with $i = 0, \dots, m-1$ and $j = 0, \dots, n-1$. \diamond

Theorem 6.6.2.5 Let $A, B \in \mathbb{R}^{m \times n}$. If $|A| \leq |B|$ then $\|A\|_1 \leq \|B\|_1$, $\|A\|_\infty \leq \|B\|_\infty$, and $\|A\|_F \leq \|B\|_F$.

Consider

$$\kappa := x^T y = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-2} \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix} = \left((\chi_0 \psi_0 + \chi_1 \psi_1) + \dots \right) + \chi_{n-2} \psi_{n-2} + \chi_{n-1} \psi_{n-1}.$$

Under the computational model given in Subsection 6.2.3 the computed result, $\check{\kappa}$, satisfies

$$\check{\kappa} = \sum_{i=0}^{n-1} \left(\chi_i \psi_i (1 + \epsilon_*^{(i)}) \prod_{j=i}^{n-1} (1 + \epsilon_+^{(j)}) \right),$$

where $\epsilon_+^{(0)} = 0$ and $|\epsilon_*^{(0)}|, |\epsilon_*^{(j)}|, |\epsilon_+^{(j)}| \leq \epsilon_{\text{mach}}$ for $j = 1, \dots, n-1$.

Lemma 6.6.2.6 *Let $\epsilon_i \in \mathbb{R}$, $0 \leq i \leq n-1$, $n\epsilon_{\text{mach}} < 1$, and $|\epsilon_i| \leq \epsilon_{\text{mach}}$. Then $\exists \theta_n \in \mathbb{R}$ such that*

$$\prod_{i=0}^{n-1} (1 + \epsilon_i)^{\pm 1} = 1 + \theta_n,$$

with $|\theta_n| \leq n\epsilon_{\text{mach}}/(1 - n\epsilon_{\text{mach}})$.

Here the ± 1 means that on an individual basis, the term is either used in a multiplication or a division. For example

$$(1 + \epsilon_0)^{\pm 1}(1 + \epsilon_1)^{\pm 1}$$

might stand for

$$(1 + \epsilon_0)(1 + \epsilon_1) \quad \text{or} \quad \frac{(1 + \epsilon_0)}{(1 + \epsilon_1)} \quad \text{or} \quad \frac{(1 + \epsilon_1)}{(1 + \epsilon_0)} \quad \text{or} \quad \frac{1}{(1 + \epsilon_1)(1 + \epsilon_0)}$$

so that this lemma can accommodate an analysis that involves a mixture of the Standard and Alternative Computational Models (SCM and ACM).

Definition 6.6.2.7 For all $n \geq 1$ and $n\epsilon_{\text{mach}} < 1$, define

$$\gamma_n = n\epsilon_{\text{mach}}/(1 - n\epsilon_{\text{mach}}).$$

◇

simplifies to

$$\begin{aligned} \check{\kappa} &= \\ &= \chi_0 \psi_0 (1 + \theta_n) + \chi_1 \psi_1 (1 + \theta_n) + \cdots + \chi_{n-1} \psi_{n-1} (1 + \theta_2) \\ &= \\ &= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} (1 + \theta_n) & 0 & 0 & \cdots & 0 \\ 0 & (1 + \theta_n) & 0 & \cdots & 0 \\ 0 & 0 & (1 + \theta_{n-1}) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (1 + \theta_2) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix} \\ &= \\ &= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \left(I + \begin{pmatrix} \theta_n & 0 & 0 & \cdots & 0 \\ 0 & \theta_n & 0 & \cdots & 0 \\ 0 & 0 & \theta_{n-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \theta_2 \end{pmatrix} \right) \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix}, \end{aligned}$$

where $|\theta_j| \leq \gamma_j$, $j = 2, \dots, n$.

Lemma 6.6.2.8 *If $n, b \geq 1$ then $\gamma_n \leq \gamma_{n+b}$ and $\gamma_n + \gamma_b + \gamma_n \gamma_b \leq \gamma_{n+b}$.*

Theorem 6.6.2.9 Let $x, y \in \mathbb{R}^n$ and let $\kappa := x^T y$ be computed in the order indicated by

$$(\cdots ((\chi_0 \psi_0 + \chi_1 \psi_1) + \chi_2 \psi_2) + \cdots) + \chi_{n-1} \psi_{n-1}.$$

Then

$$\check{\kappa} = [x^T y] = x^T(I + \Sigma^{(n)})y.$$

Corollary 6.6.2.10 Under the assumptions of [Theorem 6.6.2.9](#) the following relations hold:

R-1B $\check{\kappa} = (x + \delta x)^T y$, where $|\delta x| \leq \gamma_n |x|$,

R-2B $\check{\kappa} = x^T(y + \delta y)$, where $|\delta y| \leq \gamma_n |y|$;

R-1F $\check{\kappa} = x^T y + \delta \kappa$, where $|\delta \kappa| \leq \gamma_n |x|^T |y|$.

Theorem 6.6.2.11 Error results for matrix-vector multiplication. Let $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ and consider the assignment $\check{y} := Ax$ implemented via dot products as expressed in [\(6.3.4\)](#). Then these equalities hold:

R-1B $\check{y} = (A + \Delta A)x$, where $|\Delta A| \leq \gamma_n |A|$.

R-1F $\check{y} = Ax + \delta y$, where $|\delta y| \leq \gamma_n |A| |x|$.

Theorem 6.6.2.12 Forward error for matrix-matrix multiplication. Let $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$ and consider the assignment $C := AB$ implemented via matrix-vector multiplication. Then there exists $\Delta C \in \mathbb{R}^{m \times n}$ such that

$$\check{C} = AB + \Delta C, \text{ where } |\Delta C| \leq \gamma_k |A| |B|.$$

Lemma 6.6.2.13 Let $n \geq 1$, $\lambda, \nu \in \mathbb{R}$ and $x, y \in \mathbb{R}^n$. Assume $\lambda \neq 0$ and consider the computation

$$\nu := (\alpha - x^T y) / \lambda,$$

Then

$$(\lambda + \delta \lambda) \check{\nu} = \alpha - (x + \delta x)^T y, \text{ where } |\delta \lambda| \leq \gamma_2 |\lambda| \text{ and } |\delta x| \leq \gamma_n |x|.$$

Theorem 6.6.2.14 Let $L \in \mathbb{R}^{n \times n}$ be a nonsingular lower triangular matrix and let \check{x} be the computed result when executing [Figure 6.4.1.1](#) to solve $Lx = y$ under the computation model from [Subsection 6.2.3](#). Then there exists a matrix ΔL such that

$$(L + \Delta L) \check{x} = y \text{ where } |\Delta L| \leq \max(\gamma_2, \gamma_{n-1}) |L|.$$

Corollary 6.6.2.15 Let $L \in \mathbb{R}^{n \times n}$ be a nonsingular lower triangular matrix and let \check{x} be the computed result when executing [Figure 6.4.1.1](#) to solve $Lx = y$ under the computation model from [Subsection 6.2.3](#). Then there exists a matrix ΔL such that

$$(L + \Delta L) \check{x} = y \text{ where } |\Delta L| \leq \gamma_n |L|.$$

Theorem 6.6.2.16 Backward error of Crout variant for LU factorization. Let $A \in \mathbb{R}^{n \times n}$ and let the LU factorization of A be computed via the Crout variant, yielding

approximate factors \check{L} and \check{U} . Then

$$(A + \Delta A) = \check{L} \check{U} \quad \text{with} \quad |\Delta A| \leq \gamma_n |\check{L}| |\check{U}|.$$

Theorem 6.6.2.17 Let $A \in \mathbb{R}^{n \times n}$ and $x, y \in \mathbb{R}^n$ with $Ax = y$. Let \check{x} be the approximate solution computed via the following steps:

- Compute the LU factorization, yielding approximate factors \check{L} and \check{U} .
- Solve $\check{L}z = y$, yielding approximate solution \check{z} .
- Solve $\check{U}\check{x} = \check{z}$, yielding approximate solution \check{x} .

Then

$$(A + \Delta A)\check{x} = y \quad \text{with} \quad |\Delta A| \leq (3\gamma_n + \gamma_n^2)|\check{L}||\check{U}|.$$

Theorem 6.6.2.18 Let A and $A + \Delta A$ be nonsingular and

$$Ax = y \text{ and } (A + \Delta A)(x + \delta x) = y$$

then

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A) \frac{\|\Delta A\|}{\|A\|}}{1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}}.$$

Theorem 6.6.2.19 Let A be nonsingular, $\|\cdot\|$ be a subordinate norm, and

$$\frac{\|\Delta A\|}{\|A\|} < \frac{1}{\kappa(A)}.$$

Then $A + \Delta A$ is nonsingular.

An important example that demonstrates how LU with partial pivoting can incur "element growth":

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & & \cdots & 1 & 1 \\ -1 & -1 & & \cdots & -1 & 1 \end{pmatrix}.$$

Week 7

Solving Sparse Linear Systems

7.1 Opening Remarks

7.1.1 Where do sparse linear systems come from?



YouTube: https://www.youtube.com/watch?v=Qq_cQbVQA5Y

Many computational engineering and science applications start with some law of physics that applies to some physical problem. This is mathematically expressed as a Partial Differential Equation (PDE). We here will use one of the simplest of PDEs, Poisson's equation on the domain Ω in two dimensions:

$$-\Delta u = f.$$

In two dimensions this is alternatively expressed as

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y) \quad (7.1.1)$$

with Dirichlet boundary condition $\partial\Omega = 0$ (meaning that $u(x, y) = 0$ on the boundary of domain Ω). For example, the domain may be the square $0 \leq x, y \leq 1$, $\partial\Omega$ its boundary, and the question may be a membrane with f being some load from, for example, a sound wave.

Since this course does not require a background in the mathematics of PDEs, let's explain the gist of all this in layman's terms.

- We want to find the function u that satisfies the conditions specified by (7.1.1). It is assumed that u is appropriately differentiable.

- For simplicity, let's assume the domain is the square with $0 \leq x \leq 1$ and $0 \leq y \leq 1$ so that the boundary Ω is the boundary of this square. We assume that on the boundary the function equals zero.
- It is usually difficult to analytically determine the continuous function u that solves such a "boundary value problem" (except for very simple examples).
- To solve the problem computationally, the problem is "discretized". What this means for our example is that a mesh is laid over the domain, values for the function u at the mesh points are approximated, and the operator is approximated. In other words, the continuous domain is viewed as a mesh instead, as illustrated in Figure 7.1.1.1 (Left). We will assume an $N \times N$ mesh of equally spaced points, where the distance between two adjacent points is $h = 1/(N+1)$. This means the mesh consists of points $\{(\chi_i, \psi_j)\}$ with $\chi_i = (i+1)h$ for $i = 0, 1, \dots, N-1$ and $\psi_j = (j+1)h$ for $j = 0, 1, \dots, N-1$.

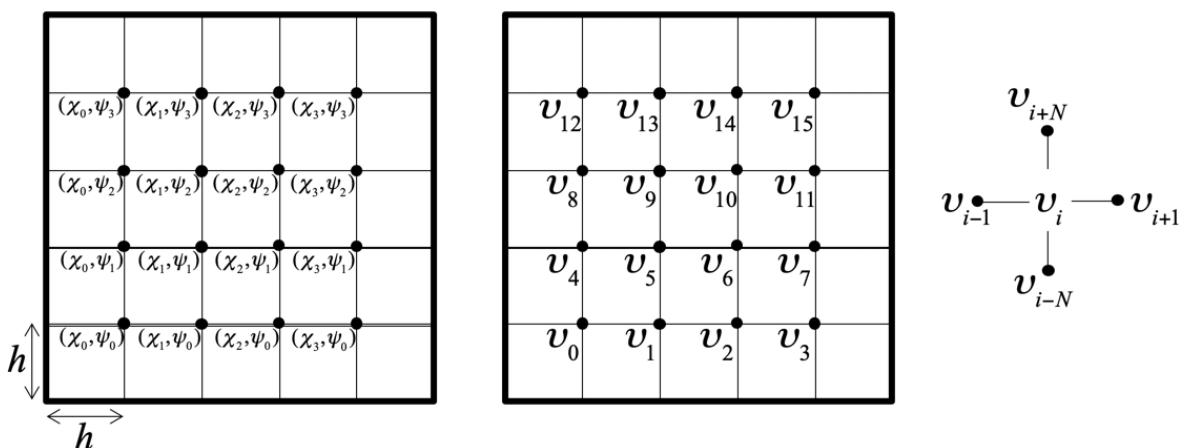


Figure 7.1.1.1 2D mesh.

- If you do the math, details of which can be found in Subsection 7.4.1, you find that the problem in (7.1.1) can be approximated with a linear equation at each mesh point:

$$\frac{-u(\chi_i, \psi_{j-1}) - u(\chi_{i-1}, \psi_j) + 4u(\chi_i, \psi_j) - u(x_{i+1}, \psi_j) - u(x_i, \psi_{j+1})}{h^2} = f(\chi_i, \psi_j).$$

The values in this equation come from the "five point stencil" illustrated in Figure 7.1.1.1 (Right).



YouTube: <https://www.youtube.com/watch?v=GvdBA5emnSs>

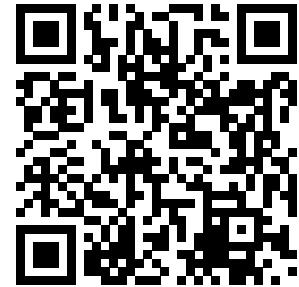
- If we number the values at the grid points, $u(\chi_i, \psi_j)$ in what is called the "natural ordering" as illustrated in Figure 7.1.1.1 (Middle), then we can write all these insights, together with the boundary condition, as

$$-v_{i-N} - v_{i-1} + 4v_i - v_{i+1} - v_{i+N} = h^2\phi_i$$

or, equivalently,

$$v_i = \frac{h^2\phi_i + v_{i-N} + v_{i-1} + v_{i+1} + v_{i+N}}{4}$$

with appropriate modifications for the case where i places the point that yielded the equation on the bottom, left, right, and/or top of the mesh.



YouTube: <https://www.youtube.com/watch?v=VYMBSJQaUM>

All these insights can be put together into a system of linear equations:

$$\begin{array}{ccccccccc} 4v_0 & -v_1 & & -v_4 & & & & & = h^2\phi_0 \\ -v_0 & +4v_1 & -v_2 & & -v_5 & & & & = h^2\phi_1 \\ & -v_1 & +4v_2 & -v_3 & & -v_6 & & & = h^2\phi_2 \\ & & -v_2 & +4v_3 & & & -v_7 & & = h^2\phi_3 \\ -v_0 & & & +4v_4 & -v_5 & & -v_8 & & = h^2\phi_4 \\ & \ddots & & \ddots & \ddots & & \ddots & & \vdots \end{array}$$

where $\phi_i = f(\chi_i, \psi_j)$ if (χ_i, ψ_j) is the point associated with value v_i . In matrix notation this becomes

$$\left(\begin{array}{cccc|ccc|c} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & -1 & & -1 & & & \\ & & -1 & 4 & & & -1 & & \\ \hline -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & \ddots & \\ & & -1 & & -1 & 4 & -1 & & \\ \hline & & & -1 & & 4 & \ddots & & \\ & & & & \ddots & \ddots & \ddots & & \end{array} \right) \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{pmatrix} = \begin{pmatrix} h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ h^2\phi_8 \end{pmatrix}.$$

This demonstrates how solving the discretized Poisson's equation boils down to the solution of a linear system $Au = h^2f$, where A has a distinct sparsity pattern (pattern of nonzeros).

Homework 7.1.1.1 The observations in this unit suggest the following way of solving (7.1.1):

- Discretize the domain $0 \leq \chi, \psi \leq 1$ by creating an $(N + 2) \times (N + 2)$ mesh of points.
- An $(N + 2) \times (N + 2)$ array U holds the values $u(\chi_i, \psi_i)$ plus the boundary around it.
- Create an $(N+2) \times (N+2)$ array F that holds the values $f(\chi_i, \psi_j)$ (plus, for convenience, extra values that correspond to the boundary).
- Set all values in U to zero. This initializes the last rows and columns to zero, which captures the boundary condition, and initializes the rest of the values at the mesh points to zero.
- Repeatedly update all interior points with the formula

$$U(i, j) = \frac{(h^2) \underbrace{F(i, j)}_{f(\chi_i, \psi_j)} + \underbrace{U(i, j - 1)}_{u(\chi_i, \psi_{j-1})} + \underbrace{U(i - 1, j)}_{u(\chi_{i-1}, \psi_j)} + \underbrace{U(i + 1, j)}_{u(\chi_{i+1}, \psi_j)} + \underbrace{U(i, j + 1)}_{u(\chi_i, \psi_{j+1})})}{4}$$

until the values converge.

- Bingo! You have written your first iterative solver for a sparse linear system.
- Test your solver with the problem where $f(\chi, \psi) = (\alpha + \beta)\pi^2 \sin(\alpha\pi\chi) \sin(\beta\pi\psi)$.
- Hint: if x and y are arrays with the vectors x and y (with entries χ_i and ψ_j), then $\text{mesh}(x, y, U)$ plots the values in U .

Hint. An outline for a matlab script can be found in [Assignments/Week07/matlab/Poisson_Jacobi_iteration.m](#). When you execute the script, in the COMMAND WINDOW enter "RETURN" to advance to the next iteration.

Solution. [Assignments/Week07/answers/Poisson_Jacobi_iteration.m](#). When you execute the script, in the COMMAND WINDOW enter "RETURN" to advance to the next iteration.

Remark 7.1.1.2 In [Homework 7.2.1.4](#) we store the vectors u and f as they appear in [Figure 7.1.1.1](#) as 2D arrays. This captures the fact that a 2d array of numbers isn't necessarily a matrix. In this case, it is a vector that is stored as a 2D array because it better captures how the values to be computed relate to the physical problem from which they arise.



YouTube: <https://www.youtube.com/watch?v=j-ELcqx3bRo>

Remark 7.1.1.3 The point of this launch is that many problems that arise in computational science require the solution to a system of linear equations $Ax = b$ where A is a (very) sparse matrix. Often, the matrix does not even need to be explicitly formed and stored.

Remark 7.1.1.4 Wilkinson defined a sparse matrix as any matrix with enough zeros that it pays to take advantage of them.

7.1.2 Overview

- 7.1 Opening
 - 7.1.1 Where do sparse linear systems come from?
 - 7.1.2 Overview
 - 7.1.3 What you will learn
- 7.2 Direct Solution
 - 7.2.1 Banded matrices
 - 7.2.2 Nested dissection
 - 7.2.3 Observations
- 7.3 Iterative Solution
 - 7.3.1 Jacobi iteration
 - 7.3.2 Gauss-Seidel iteration
 - 7.3.3 Convergence of splitting methods
 - 7.3.4 Successive Over-Relaxation (SOR)
- 7.4 Enrichments
 - 7.4.1 Details!
 - 7.4.2 Parallelism in splitting methods
 - 7.4.3 Dr. SOR
- 7.5 Wrap Up
 - 7.5.1 Additional homework
 - 7.5.2 Summary

7.1.3 What you will learn

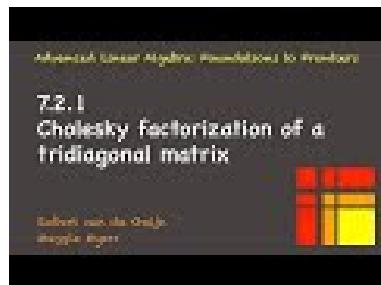
This week is all about solving nonsingular linear systems with matrices that are sparse (have enough zero entries that it is worthwhile to exploit them).

Upon completion of this week, you should be able to

- Exploit sparsity when computing the Cholesky factorization and related triangular solves of a banded matrix.
- Derive the cost for a Cholesky factorization and related triangular solves of a banded matrix.
- Utilize nested dissection to reduce fill-in when computing the Cholesky factorization and related triangular solves of a sparse matrix.
- Connect sparsity patterns in a matrix to the graph that describes that sparsity pattern.
- Relate computations over discretized domains to the Jacobi, Gauss-Seidel, Successive Over-Relaxation (SOR) and Symmetric Successive Over-Relaxation (SSOR) iterations.
- Formulate the Jacobi, Gauss-Seidel, Successive Over-Relaxation (SOR) and Symmetric Successive Over-Relaxation (SSOR) iterations as splitting methods.
- Analyze the convergence of splitting methods.

7.2 Direct Solution

7.2.1 Banded matrices



YouTube: https://www.youtube.com/watch?v=UX6Z6q1_prs

It is tempting to simply use a dense linear solver to compute the solution to $Ax = b$ via, for example, LU or Cholesky factorization, even when A is sparse. This would require $O(n^3)$ operations, where n equals the size of matrix A . What we see in this unit is that we can take advantage of a "banded" structure in the matrix to greatly reduce the computational cost.

Homework 7.2.1.1 The 1D equivalent of the example from Subsection 7.1.1 is given by the

tridiagonal linear system

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}. \quad (7.2.1)$$

Prove that this linear system is nonsingular.

Hint. Consider $Ax = 0$. We need to prove that $x = 0$. If you instead consider the equivalent problem

$$\left(\begin{array}{c|ccc|c} 1 & 0 & & & 0 \\ \hline -1 & \left(\begin{array}{cccc} 2 & -1 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{array} \right) & | & 0 \\ 0 & & & & 0 \\ \vdots & & & & \vdots \\ 0 & & & & 0 \\ 0 & & & & -1 \\ \hline 0 & 0 & & & 1 \end{array} \right) \begin{pmatrix} \chi_{-1} \\ \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-2} \\ \chi_{n-1} \\ \hline \chi_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \hline 0 \end{pmatrix}$$

that introduces two extra variables $\chi_{-1} = 0$ and $\chi_n = 0$, the problem for all χ_i , $0 \leq i < n$, becomes

$$-\chi_{i-1} + 2\chi_i - \chi_{i+1} = 0.$$

or, equivalently,

$$\chi_i = \frac{\chi_{i-1} + \chi_{i+1}}{2}.$$

Reason through what would happen if any χ_i is not equal to zero.

Solution. Building on the hint: Let's say that $\chi_i \neq 0$ while $\chi_{-1}, \dots, \chi_{i-1}$ are. Then

$$\chi_i = \frac{\chi_{i-1} + \chi_{i+1}}{2} = \frac{1}{2}\chi_{i+1}$$

and hence

$$\chi_{i+1} = 2\chi_i > 0..$$

Next,

$$\chi_{i+1} = \frac{\chi_i + \chi_{i+2}}{2} = 2\chi_i$$

and hence

$$\chi_{i+2} = 4\chi_i - \chi_i = 3\chi_i > 0.$$

Continuing this argument, the solution to the recurrence relation is $\chi_n = (n - i + 1)\chi_i$ and you find that $\chi_n > 0$ which is a contradiction.

This course covers topics in a "circular" way, where sometimes we introduce and use results that we won't formally cover until later in the course. Here is one such situation. In

a later week you will prove these relevant results involving eigenvalues:

- A symmetric matrix is symmetric positive definite (SPD) if and only if its eigenvalues are positive.
- The Gershgorin Disk Theorem tells us that the matrix in (7.2.1) has nonnegative eigenvalues.
- A matrix is singular if and only if it has zero as an eigenvalue.

These insights, together with [Homework 7.2.1.1](#), tell us that the matrix in (7.2.1) is SPD.

Homework 7.2.1.2 Compute the Cholesky factor of

$$A = \begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 5 & -2 & 0 \\ 0 & -2 & 10 & 6 \\ 0 & 0 & 6 & 5 \end{pmatrix}.$$

Answer.

$$L = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & -1 & 3 & 0 \\ 0 & 0 & 2 & 1 \end{pmatrix}.$$

Homework 7.2.1.3 Let $A \in \mathbb{R}^{n \times n}$ be tridiagonal and SPD so that

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{1,0} & & & & \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{2,1} & & & \\ & \ddots & \ddots & \ddots & & \\ & & & \alpha_{n-2,n-3} & \alpha_{n-2,n-2} & \alpha_{n-1,n-2} \\ & & & & \alpha_{n-1,n-2} & \alpha_{n-1,n-1} \end{pmatrix}. \quad (7.2.2)$$

- Propose a Cholesky factorization algorithm that exploits the structure of this matrix.
- What is the cost? (Count square roots, divides, multiplies, and subtractions.)
- What would have been the (approximate) cost if we had not taken advantage of the tridiagonal structure?

Solution.

- If you play with a few smaller examples, you can conjecture that the Cholesky factor of (7.2.2) is a bidiagonal matrix (the main diagonal plus the first subdiagonal). Thus,

$A = LL^T$ translates to

$$\begin{aligned}
 & \left(\begin{array}{ccc} \alpha_{0,0} & \alpha_{1,0} & \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{2,1} \\ \ddots & \ddots & \ddots \\ & \alpha_{n-2,n-3} & \alpha_{n-2,n-2} & \alpha_{n-1,n-2} \\ & & \alpha_{n-1,n-2} & \alpha_{n-1,n-1} \end{array} \right) \\
 = & \left(\begin{array}{ccc} \lambda_{0,0} & & \\ \lambda_{1,0} & \lambda_{1,1} & \\ \ddots & \ddots & \\ & \lambda_{n-2,n-3} & \lambda_{n-2,n-2} \\ & & \lambda_{n-1,n-2} & \lambda_{n-1,n-1} \end{array} \right) \left(\begin{array}{ccc} \lambda_{0,0} & \lambda_{1,0} & \\ \lambda_{1,1} & \lambda_{2,1} & \\ \ddots & \ddots & \\ & \lambda_{n-2,n-2} & \lambda_{n-1,n-2} \\ & & \lambda_{n-1,n-1} \end{array} \right) \\
 = & \left(\begin{array}{ccc} \lambda_{0,0}\lambda_{0,0} & \lambda_{0,0}\lambda_{1,0} & \\ \lambda_{1,0}\lambda_{0,0} & \lambda_{1,0}\lambda_{0,1} + \lambda_{1,1}\lambda_{1,1} & \lambda_{1,1}\lambda_{2,1} \\ & \lambda_{21}\lambda_{11} & \ddots \\ & & \ddots & \star\star \\ & & & \lambda_{n-2,n-2}\lambda_{n-1,n-2} \\ & & & \lambda_{n-1,n-2}\lambda_{n-2,n-2} & \lambda_{n-1,n-2}\lambda_{n-1,n-2} \\ & & & & + \lambda_{n-1,n-1}\lambda_{n-1,n-1} \end{array} \right),
 \end{aligned}$$

where $\star\star = \lambda_{n-3,n-2}\lambda_{n-3,n-2} + \lambda_{n-2,n-2}\lambda_{n-2,n-2}$. With this insight, the algorithm that overwrites A with its Cholesky factor is given by

```

for  $i = 0, \dots, n - 2$ 
   $\alpha_{i,i} := \sqrt{\alpha_{i,i}}$ 
   $\alpha_{i+1,i} := \alpha_{i+1,i}/\alpha_{i,i}$ 
   $\alpha_{i+1,i+1} := \alpha_{i+1,i+1} - \alpha_{i+1,i}\alpha_{i+1,i}$ 
endfor
   $\alpha_{n-1,n-1} := \sqrt{\alpha_{n-1,n-1}}$ 
```

- A cost analysis shows that this requires n square roots, $n - 1$ divides, $n - 1$ multiplies, and $n - 1$ subtracts.
- The cost, had we not taken advantage of the special structure, would have been (approximately) $\frac{1}{3}n^3$.

Homework 7.2.1.4 Propose an algorithm for overwriting y with the solution to $Ax = y$ for the SPD matrix in [Homework 7.2.1.3](#).

Solution.

- Use the algorithm from [Homework 7.2.1.3](#) to overwrite A with its Cholesky factor.
- Since $A = LL^T$, we need to solve $Lz = y$ and then $L^Tx = z$.
 - Overwriting y with the solution of $Lz = y$ (forward substitution) is accomplished

by the following algorithm (here L had overwritten A):

```

for  $i = 0, \dots, n - 2$ 
     $\psi_i := \psi_i / \alpha_{i,i}$ 
     $\psi_{i+1} := \psi_{i+1} - \alpha_{i+1,i}\psi_i$ 
endfor
 $\psi_{n-1} := \psi_{n-1} / \alpha_{n-1,n-1}$ 

```

- Overwriting y with the solution of $Lx = z$ (where z has overwritten y (back substitution) is accomplished by the following algorithm (here L had overwritten A):

```

for  $n - 1 = 0, \dots, 1$ 
     $\psi_i := \psi_i / \alpha_{i,i}$ 
     $\psi_{i-1} := \psi_{i-1} - \alpha_{i,i-1}\psi_i$ 
endfor
 $\psi_0 := \psi_0 / \alpha_{0,0}$ 

```

The last exercises illustrate how special structure (in terms of patterns of zeroes and nonzeros) can often be exploited to reduce the cost of factoring a matrix and solving a linear system.



YouTube: <https://www.youtube.com/watch?v=kugJ2NljC2U>

The bandwidth of a matrix is defined as the smallest integer b such that all elements on the j th superdiagonal and subdiagonal of the matrix equal zero if $j > b$.

- A diagonal matrix has bandwidth 1.
- A tridiagonal matrix has bandwidth 2.
- And so forth.

Let's see how to take advantage of the zeroes in a matrix with bandwidth b , focusing on SPD matrices.

Definition 7.2.1.1 The half-band width of a symmetric matrix equals the number of sub-diagonals beyond which all the matrix contains only zeroes. For example, a diagonal matrix has half-band width of zero and a tridiagonal matrix has a half-band width of one. ◇

Homework 7.2.1.5 Assume the SPD matrix $A \in \mathbb{R}^{m \times m}$ has a bandwidth of b . Propose a

modification of the right-looking Cholesky factorization from [Figure 5.4.3.1](#)

$A = \text{Chol-right-looking}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$\alpha_{11} := \sqrt{\alpha_{11}}$
$a_{21} := a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{21}^T$ (updating only the lower triangular part)
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
endwhile

that takes advantage of the zeroes in the matrix. (You will want to draw yourself a picture.) What is its approximate cost in flops (when m is large)?

Solution. See the below video.



YouTube: <https://www.youtube.com/watch?v=AoldARTix5Q>

Ponder This 7.2.1.6 Propose a modification of the FLAME notation that allows one to elegantly express the algorithm you proposed for [Homework 7.2.1.5](#)

Ponder This 7.2.1.7 Another way of looking at an SPD matrix $A \in \mathbb{R}^{n \times n}$ with bandwidth b is to block it

$$A = \begin{pmatrix} A_{0,0} & A_{1,0}^T & & & & \\ A_{1,0} & A_{1,1} & A_{2,1}^T & & & \\ & \ddots & \ddots & \ddots & & \\ & & A_{n-2,n-3} & A_{n-2,n-2} & A_{n-1,n-2}^T & \\ & & & A_{n-1,n-2} & A_{n-1,n-1} & \end{pmatrix}.$$

where, $A_{i,j} \in \mathbb{R}^{b \times b}$ and for simplicity we assume that n is a multiple of b . Propose an algorithm for computing its Cholesky factorization that exploits this block structure. What special structure do matrices $A_{i+1,i}$ have? Can you take advantage of this structure?

Analyze the cost of your proposed algorithm.

7.2.2 Nested dissection



YouTube: <https://www.youtube.com/watch?v=r1P4Ze7Yqe0>

The purpose of the game is to limit **fill-in**, which happens when zeroes turn into non-zeroes. With an example that would result from, for example, Poisson's equation, we will illustrate the basic techniques, which are known as "nested dissection."

If you consider the mesh that results from the discretization of, for example, a square domain, the numbering of the mesh points does not need to be according to the "natural ordering" we chose to use before. As we number the mesh points, we reorder (permute) both the columns of the matrix (which correspond to the elements v_i to be computed) and the equations that tell one how v_i is computed from its neighbors. If we choose a **separator**, the points highlighted in red in Figure 7.2.2.1 (Top-Left), and order the mesh points to its left first, then the ones to its right, and finally the points in the separator, we create a pattern of zeroes, as illustrated in Figure 7.2.2.1 (Top-Right).

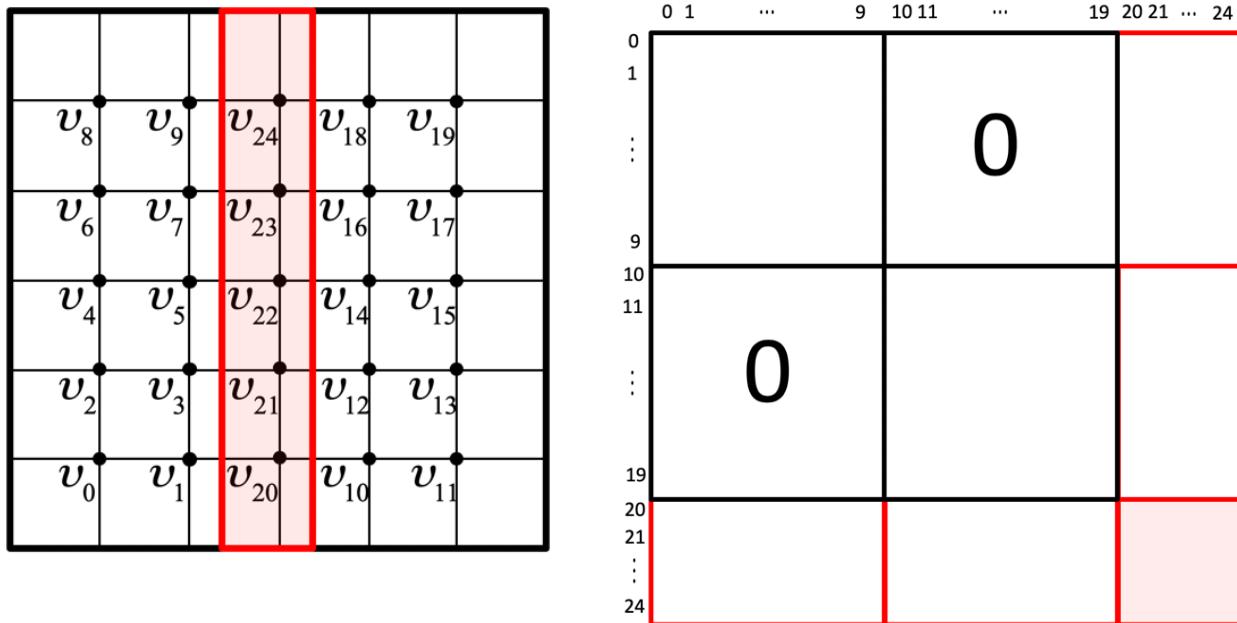


Figure 7.2.2.1 An illustration of nested dissection.

Homework 7.2.2.1 Consider the SPD matrix

$$A = \left(\begin{array}{c|c|c} A_{00} & 0 & A_{20}^T \\ \hline 0 & A_{11} & A_{21}^T \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right).$$

- What special structure does the Cholesky factor of this matrix have?
- How can the different parts of the Cholesky factor be computed in a way that takes advantage of the zero blocks?
- How do you take advantage of the zero pattern when solving with the Cholesky factors?

Solution.

- The Cholesky factor of this matrix has the structure

$$L = \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline 0 & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array} \right).$$

- We notice that $A = LL^T$ means that

$$\left(\begin{array}{c|c|c} A_{00} & 0 & A_{20}^T \\ \hline 0 & A_{11} & A_{21}^T \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right) = \underbrace{\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline 0 & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array} \right)}_{\left(\begin{array}{c|c|c} L_{00}L_{00}^T & 0 & * \\ \hline 0 & L_{11}L_{11}^T & * \\ \hline L_{20}L_{00}^T & L_{21}L_{11}^T & L_{20}L_{20}^T + L_{21}L_{21}^T + L_{22}L_{22}^T \end{array} \right)}^T,$$

where the $*$ s indicate "symmetric parts" that don't play a role. We deduce that the following steps will yield the Cholesky factor:

- Compute the Cholesky factor of A_{00} :

$$A_{00} = L_{00}L_{00}^T,$$

overwriting A_{00} with the result.

- Compute the Cholesky factor of A_{11} :

$$A_{11} = L_{11}L_{11}^T,$$

overwriting A_{11} with the result.

- Solve

$$XL_{00}^T = A_{20}$$

for X , overwriting A_{20} with the result. (This is a triangular solve with multiple right-hand sides in disguise.)

- o Solve

$$XL_{11}^T = A_{21}$$

for X , overwriting A_{21} with the result. (This is a triangular solve with multiple right-hand sides in disguise.)

- o Update the lower triangular part of A_{22} with

$$A_{22} - L_{20}L_{20}^T - L_{21}L_{21}^T.$$

- o Compute the Cholesky factor of A_{22} :

$$A_{22} = L_{22}L_{22}^T,$$

overwriting A_{22} with the result.

- If we now want to solve $Ax = y$, we can instead first solve $Lz = y$ and then $L^Tx = z$. Consider

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline 0 & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array} \right) \begin{pmatrix} z_0 \\ z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}.$$

This can be solved via the steps

- o Solve $L_{00}z_0 = y_0$.
- o Solve $L_{11}z_1 = y_1$.
- o Solve $L_{22}z_2 = y_2 - L_{20}z_0 - L_{21}z_1$.

Similarly,

$$\left(\begin{array}{c|c|c} L_{00}^T & 0 & L_{20}^T \\ \hline 0 & L_{11}^T & L_{21}^T \\ \hline 0 & 0 & L_{22}^T \end{array} \right)^T \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \end{pmatrix}.$$

can be solved via the steps

- o Solve $L_{22}^Tx_2 = z_2$.
- o Solve $L_{11}^Tx_1 = z_1 - L_{21}^Tx_2$.
- o Solve $L_{00}^Tx_0 = z_0 - L_{20}^Tx_2$.



YouTube: <https://www.youtube.com/watch?v=mwX0wPRdw7U>

Each of the three subdomains that were created in Figure 7.2.2.1 can themselves be reordered by identifying separators. In Figure 7.2.2.2 we illustrate this only for the left and right subdomains. This creates a recursive structure in the matrix. Hence, the name **nested dissection** for this approach.

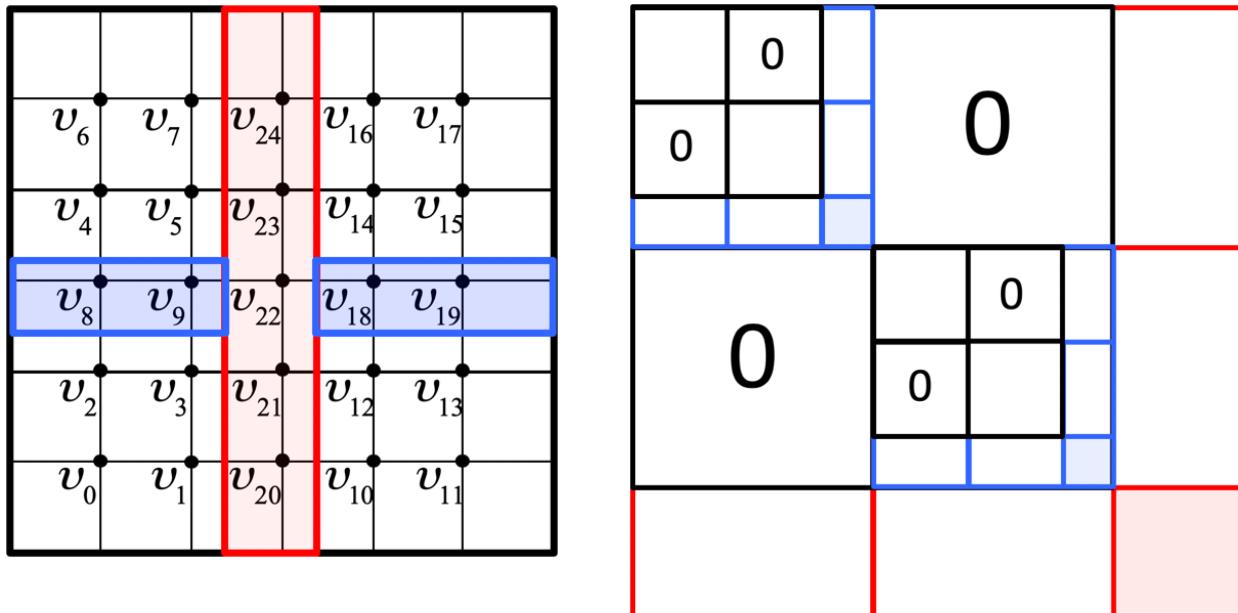


Figure 7.2.2.2 A second level of nested dissection.

7.2.3 Observations

Through an example, we have illustrated the following insights regarding the direct solution of sparse linear systems:

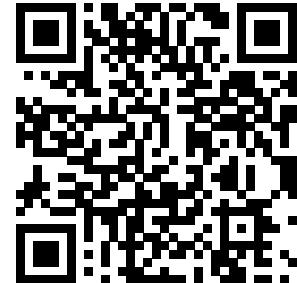
- There is a one-to-one correspondence between links in the graph that shows how mesh points are influenced by other mesh points (connectivity) and nonzeros in the matrix. If the graph is undirected, then the sparsity in the matrix is symmetric (provided the unknowns are ordered in the same order as the equations that relate the unknowns to their neighbors). If the graph is directed, then the matrix has a nonsymmetric sparsity pattern.
- Renumbering the mesh points is equivalent to correspondingly permuting the columns of the matrix and the solution vector. Reordering the corresponding equations is equivalent to permuting the rows of the matrix.

These observations relate the problem of reducing fill-in to the problem of partitioning the graph by identifying a separator. The smaller the number of mesh points in the separator (the interface), the smaller the submatrix that corresponds to it and the less fill-in will occur related to this dissection.

Remark 7.2.3.1 Importantly: one can start with a mesh and manipulate it into a matrix or one can start with a matrix and have its sparsity pattern prescribe the graph.

7.3 Iterative Solution

7.3.1 Jacobi iteration



YouTube: <https://www.youtube.com/watch?v=0Mbvk1ihIFo>

Let's review what we saw in [Subsection 7.1.1](#). The linear system $Au = f$

$$\begin{array}{ccccccccc}
 4v_0 & -v_1 & & -v_4 & & & & = h^2\phi_0 \\
 -v_0 & +4v_1 & -v_2 & & -v_5 & & & = h^2\phi_1 \\
 & -v_1 & +4v_2 & -v_3 & & -v_6 & & = h^2\phi_2 \\
 & & -v_2 & +4v_3 & & & -v_7 & = h^2\phi_3 \\
 -v_0 & & & +4v_4 & -v_5 & & -v_8 & = h^2\phi_4 \\
 & \ddots & & \ddots & \ddots & & \ddots & \vdots
 \end{array}$$

which can be written in matrix form as

$$\left(\begin{array}{cccc|ccc|c}
 4 & -1 & & -1 & & & & & \\
 -1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & -1 & & -1 & & & \\
 & & -1 & 4 & & & -1 & & \\
 \hline
 -1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & \ddots & \\
 & & -1 & & -1 & 4 & -1 & & \\
 & & & -1 & & -1 & 4 & & \\
 \hline
 & & & -1 & & & 4 & \ddots & \\
 & & & & \ddots & & \ddots & & \vdots
 \end{array} \right) \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{pmatrix} = \begin{pmatrix} h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots \end{pmatrix}.$$

was solved by repeatedly updating

$$v_i = \frac{h^2\phi_i + v_{i-N} + v_{i-1} + v_{i+1} + v_{i+N}}{4}$$

modified appropriately for points adjacent to the boundary. Let's label the value of v_i during the k th iteration with $v_i^{(k)}$ and state the algorithm more explicitly as

```

for  $k = 0, \dots$ , convergence
  for  $i = 0, \dots, N \times N - 1$ 
     $v_i^{(k+1)} = (h^2\phi_i + v_{i-N}^{(k)} + v_{i-1}^{(k)} + v_{i+1}^{(k)} + v_{i+N}^{(k)})/4$ 
  endfor
endfor

```

again, modified appropriately for points adjacent to the boundary. The superscripts are there to emphasize the iteration during which a value is updated. In practice, only the values for iteration k and $k + 1$ need to be stored. We can also capture the algorithm with a vector and matrix as

$$\begin{array}{rcl}
 4v_0^{(k+1)} & = & v_1^{(k)} & + v_4^{(k)} & + h^2\phi_0 \\
 4v_1^{(k+1)} & = & v_0^{(k)} & + v_2^{(k)} & + v_5^{(k)} & + h^2\phi_1 \\
 4v_2^{(k+1)} & = & v_1^{(k)} & + v_3^{(k)} & + v_6^{(k)} & + h^2\phi_2 \\
 4v_3^{(k+1)} & = & & v_2^{(k)} & + v_7^{(k)} & + h^2\phi_3 \\
 4v_4^{(k+1)} & = & v_0^{(k)} & & + v_5^{(k)} & - v_8^{(k)} & + h^2\phi_4 \\
 \vdots & & \ddots & & \ddots & \ddots & \vdots
 \end{array}$$

which can be written in matrix form as

$$\begin{aligned}
 & \left(\begin{array}{c|c|c|c}
 4 & & & \\
 & 4 & & \\
 & & 4 & \\
 & & & 4 \\ \hline
 & | & | & | \\
 & 4 & 4 & 4 & 4 \\ \hline
 & | & | & | & | \\
 & & & & \ddots \\ \hline
 & | & | & | & | \\
 & & & & \ddots \\ \hline
 & | & | & | & | \\
 & & & & \ddots \\ \hline
 \end{array} \right) \begin{pmatrix} v_0^{(k+1)} \\ v_1^{(k+1)} \\ v_2^{(k+1)} \\ v_3^{(k+1)} \\ \hline v_4^{(k+1)} \\ v_5^{(k+1)} \\ v_6^{(k+1)} \\ v_7^{(k+1)} \\ \hline v_8^{(k+1)} \\ \vdots \end{pmatrix} \\
 & = \left(\begin{array}{cc|cc|c}
 0 & 1 & 1 & & \\
 1 & 0 & 1 & 1 & \\
 & 1 & 0 & 1 & \\
 & & 1 & 0 & \\ \hline
 1 & & 0 & 1 & 1 \\
 & 1 & 1 & 0 & 1 \\
 & & 1 & 0 & 1 \\
 & & & 1 & 0 \\ \hline
 & | & | & | & | \\
 & 1 & 1 & \ddots & 1 \\
 & & & \ddots & \ddots \\ \hline
 & | & | & | & | \\
 & 1 & \ddots & \ddots & \ddots \\
 & & \ddots & \ddots & \ddots \\ \hline
 & | & | & | & | \\
 & 1 & & 0 & \ddots \\
 & & \ddots & \ddots & \ddots \\ \hline
 \end{array} \right) \begin{pmatrix} v_0^{(k)} \\ v_1^{(k)} \\ v_2^{(k)} \\ v_3^{(k)} \\ \hline v_4^{(k)} \\ v_5^{(k)} \\ v_6^{(k)} \\ v_7^{(k)} \\ \hline v_8^{(k)} \\ \vdots \end{pmatrix} + \begin{pmatrix} h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots \end{pmatrix}. \tag{7.3.1}
 \end{aligned}$$



YouTube: https://www.youtube.com/watch?v=7rDvET9_nek

How can we capture this more generally?

- We wish to solve $Ax = y$.

We write A as the difference of its diagonal, $M = D$, and the negative of its off-diagonal part, $N = D - A$ so that

$$A = D - (D - A) = M - N.$$

In our example, $M = 4I$ and $N = 4I - A$.

- We then notice that

$$Ax = y$$

can be rewritten as

$$(M - N)x = y$$

or, equivalently,

$$Mx = Nx + y.$$

If you think about it carefully, this captures (7.3.1) for our example. Finally,

$$x = M^{-1}(Nx + y).$$

- If we now let $x^{(k)}$ be the values of our vector x in the current step. Then the values after all elements have been updated are given by the vector

$$x^{(k+1)} = M^{-1}(Nx^{(k)} + y).$$

- All we now need is an initial guess for the solution, $x^{(0)}$, and we are ready to iteratively solve the linear system by computing $x^{(1)}$, $x^{(2)}$, etc., until we (approximately) reach a fixed point where $x^{(k+1)} = M^{-1}(Nx^{(k)} + y) \approx x^{(k)}$.

The described method, where M equals the diagonal of A and $N = D - A$, is known as the **Jacobi iteration**.

Remark 7.3.1.1 The important observation is that the computation involves a matrix-vector multiplication with a sparse matrix, $N = D - A$, and a solve with a diagonal matrix, $M = D$.

7.3.2 Gauss-Seidel iteration



YouTube: <https://www.youtube.com/watch?v=ufMUh01vDew>

A variation on the Jacobi iteration is the Gauss-Seidel iteration. It recognizes that since values at points are updated in some order, if a neighboring value has already been updated earlier in the current step, then you might as well use that updated value. For our example from [Subsection 7.1.1](#) this is captured by the algorithm

```

for  $k = 0, \dots$ , convergence
  for  $i = 0, \dots, N \times N - 1$ 
     $v_i^{(k+1)} = (h^2\phi_i + v_{i-N}^{(k+1)} + v_{i-1}^{(k+1)} + v_{i+1}^{(k)} + v_{i+N}^{(k)})/4$ 
  endfor
endfor

```

modified appropriately for points adjacent to the boundary. This algorithm exploits the fact that $v_{i-N}^{(k+1)}$ and $v_{i-1}^{(k+1)}$ have already been computed by the time $v_i^{(k+1)}$ is updated. Once again, the superscripts are there to emphasize the iteration during which a value is updated. In practice, the superscripts can be dropped because of the order in which the computation happens.

Homework 7.3.2.1 Modify the code for [Homework 7.1.1.1](#) (what you now know as the Jacobi iteration) to implement the Gauss-Seidel iteration.

Solution. [Assignments/Week07/answers/Poisson_GS_iteration.m](#).

When you execute the script, in the COMMAND WINDOW enter "RETURN" to advance to the next iteration.

You may also want to observe the Jacobi and Gauss-Seidel iterations in action side-by-side in [Assignments/Week07/answers/Poisson_Jacobi_vs_GS_iteration.m](#).

Homework 7.3.2.2 Here we repeat [\(7.3.1\)](#) for Jacobi's iteration applied to the example in

Subsection 7.1.1:

$$\begin{aligned}
 & \left(\begin{array}{ccc|cc|c} 4 & & & & & \\ & 4 & & & & \\ & & 4 & & & \\ \hline & & & 4 & & \\ & & & & 4 & \\ & & & & & 4 \\ \hline & & & & & \ddots \end{array} \right) \begin{pmatrix} v_0^{(k+1)} \\ v_1^{(k+1)} \\ v_2^{(k+1)} \\ v_3^{(k+1)} \\ \hline v_4^{(k+1)} \\ v_5^{(k+1)} \\ v_6^{(k+1)} \\ v_7^{(k+1)} \\ \hline v_8^{(k+1)} \\ \vdots \end{pmatrix} \\
 = & \left(\begin{array}{ccc|cc|c} 0 & 1 & & 1 & & \\ 1 & 0 & 1 & 1 & 1 & \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & & \\ \hline 1 & & & 0 & 1 & 1 \\ & 1 & & 1 & 0 & 1 \\ & & 1 & & 1 & 0 \\ \hline & & & 1 & \ddots & 0 \\ & & & & \ddots & \ddots \end{array} \right) \begin{pmatrix} v_0^{(k)} \\ v_1^{(k)} \\ v_2^{(k)} \\ v_3^{(k)} \\ \hline v_4^{(k)} \\ v_5^{(k)} \\ v_6^{(k)} \\ v_7^{(k)} \\ \hline v_8^{(k)} \\ \vdots \end{pmatrix} + \begin{pmatrix} h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots \end{pmatrix}. \tag{7.3.2}
 \end{aligned}$$

Modify this to reflect the Gauss-Seidel iteration.

Solution.

$$\begin{array}{c}
 \left(\begin{array}{cccc|cc|c} 4 & & & & & & \\ -1 & 4 & & & & & \\ & -1 & 4 & & & & \\ & & -1 & 4 & & & \\ \hline -1 & & & 4 & & & \\ & -1 & & & 4 & & \\ & & -1 & & & 4 & \\ & & & -1 & & & \\ \hline & & & & -1 & & 4 \\ & & & & & \ddots & \\ & & & & & & \ddots \end{array} \right) \begin{pmatrix} v_0^{(k+1)} \\ v_1^{(k+1)} \\ v_2^{(k+1)} \\ v_3^{(k+1)} \\ \hline v_4^{(k+1)} \\ v_5^{(k+1)} \\ v_6^{(k+1)} \\ v_7^{(k+1)} \\ \hline v_8^{(k+1)} \\ \vdots \end{pmatrix} \\
 := \left(\begin{array}{ccc|cc|c} 0 & 1 & & 1 & & \\ & 0 & 1 & & 1 & \\ & & 0 & 1 & & \\ & & & 0 & 1 & \\ \hline & & & & 1 & \\ & & & & & \ddots \\ & & & & & \\ & & & & & \end{array} \right) \begin{pmatrix} v_0^{(k)} \\ v_1^{(k)} \\ v_2^{(k)} \\ v_3^{(k)} \\ \hline v_4^{(k)} \\ v_5^{(k)} \\ v_6^{(k)} \\ v_7^{(k)} \\ \hline v_8^{(k)} \\ \vdots \end{pmatrix} + \begin{pmatrix} h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots \end{pmatrix}.
 \end{array}$$

This homework suggests the following:

- We wish to solve $Ax = y$.

We write symmetric A as

$$A = \underbrace{(D - L)}_M - \underbrace{(L^T)}_N,$$

where $-L$ equals the strictly lower triangular part of A and D is its diagonal.

- We then notice that

$$Ax = y$$

can be rewritten as

$$(D - L - L^T)x = y$$

or, equivalently,

$$(D - L)x = L^T x + y.$$

If you think about it carefully, this captures (7.3.2) for our example. Finally,

$$x = (D - L)^{-1}(L^T x + y).$$

- If we now let $x^{(k)}$ be the values of our vector x in the current step. Then the values after all elements have been updated are given by the vector

$$x^{(k+1)} = (D - L)^{-1}(L^T x^{(k)} + y).$$

Homework 7.3.2.3 When the Gauss-Seidel iteration is used to solve $Ax = y$, where $A \in \mathbb{R}^{n \times n}$, it computes entries of $x^{(k+1)}$ in the forward order $\chi_0^{(k+1)}, \chi_1^{(k+1)}, \dots$. If $A = D - L - L^T$, this is captured by

$$(D - L)x^{(k+1)} = L^T x^{(k)} + y. \quad (7.3.3)$$

Modify (7.3.3) to yield a "reverse" Gauss-Seidel method that computes the entries of vector $x^{(k+1)}$ in the order $\chi_{n-1}^{(k+1)}, \chi_{n-2}^{(k+1)}, \dots$

Solution. The reverse order is given by $\chi_{n-1}^{(k+1)}, \chi_{n-2}^{(k+1)}, \dots$. This corresponds to the splitting $M = D - L^T$ and $N = L$ so that

$$(D - L^T)x^{(k+1)} = Lx^{(k)} + y.$$

Homework 7.3.2.4 A "symmetric" Gauss-Seidel iteration to solve symmetric $Ax = y$, where $A \in \mathbb{R}^{n \times n}$, alternates between computing entries in forward and reverse order. In other words, if $A = M_F - N_F$ for the forward Gauss-Seidel method and $A = M_R - N_R$ for the reverse Gauss-Seidel method, then

$$\begin{aligned} M_F x^{(k+\frac{1}{2})} &= N_F x^{(k)} + y \\ M_R x^{(k+1)} &= N_R x^{(k+\frac{1}{2})} + y \end{aligned}$$

constitutes one iteration of this symmetric Gauss-Seidel iteration. Determine M and N such that

$$Mx^{(k+1)} = Nx^{(k)} + y$$

equals one iteration of the symmetric Gauss-Seidel iteration.

(You may want to follow the hint...)

Hint.

- From this unit and the last homework, we know that $M_F = (D - L)$, $N_F = L^T$, $M_R = (D - L^T)$, and $N_R = L$.
- Show that

$$(D - L^T)x^{(k+1)} = L(D - L)^{-1}L^T x^{(k)} + (I + L(D - L)^{-1})y.$$

- Show that $I + L(D - L)^{-1} = D(D - L)^{-1}$.
- Use these insights to determine M and N .

Solution.

- From this unit and the last homework, we know that $M_F = (D - L)$, $N_F = L^T$, $M_R = (D - L^T)$, and $N_R = L$.

- Show that

$$(D - L^T)x^{(k+1)} = L(D - L)^{-1}L^T x^{(k)} + (I + L(D - L)^{-1})y.$$

We show this by substituting M_R and N_R :

$$(D - L^T)x^{(k+1)} = Lx^{(k+\frac{1}{2})} + y$$

and then substituting in for $x^{(k+\frac{1}{2})}$, M_F and N_F :

$$(D - L^T)x^{(k+1)} = L((D - L)^{-1}L^T x^{(k)} + y) + y.$$

Multiplying out the right-hand side and factoring out y yields the desired result.

- Show that $I + L(D - L)^{-1} = D(D - L)^{-1}$.

We show this by noting that

$$\begin{aligned} I + L(D - L)^{-1} &= \\ &= (D - L)(D - L)^{-1} + L(D - L)^{-1} = \\ &= (D - L + L)(D - L)^{-1} = \\ &= D(D - L)^{-1}. \end{aligned}$$

- Use these insights to determine M and N .

We now notice that

$$(D - L^T)x^{(k+1)} = L(D - L)^{-1}L^T x^{(k)} + (I + L(D - L)^{-1})y$$

can be rewritten as (Someone check this... My brain hurts.)

$$\begin{aligned} (D - L^T)x^{(k+1)} &= L(D - L)^{-1}L^T x^{(k)} + D(D - L)^{-1}y \\ \underbrace{(D - L)D^{-1}(D - L^T)}_M x^{(k+1)} &= \underbrace{(D - L)D^{-1}L(D - L)^{-1}L^T}_N x^{(k)} + y \end{aligned}$$

7.3.3 Convergence of splitting methods



YouTube: <https://www.youtube.com/watch?v=L6PZhc-G7cE>

The Jacobi and Gauss-Seidel iterations can be generalized as follows. Split matrix $A = M - N$ where M is nonsingular. Now,

$$(M - N)x = y$$

is equivalent to

$$Mx = Nx + y$$

and

$$x = M^{-1}(Nx + y).$$

This is an example of a fixed-point equation: Plug x into $M^{-1}(Nx + y)$ and the result is again x . The iteration is then created by viewing the vector on the left as the next approximation to the solution given the current approximation x on the right:

$$x^{(k+1)} = M^{-1}(Nx^{(k)} + y).$$

Let $A = (D - L - U)$ where $-L$, D , and $-U$ are the strictly lower triangular, diagonal, and strictly upper triangular parts of A .

- For the Jacobi iteration, $M = D$ and $N = (L + U)$.
- For the Gauss-Seidel iteration, $M = (D - L)$ and $N = U$.

In practice, M is not inverted. Instead, the iteration is implemented as

$$Mx^{(k+1)} = Nx^{(k)} + y,$$

with which we emphasize that we solve with M rather than inverting it.

Homework 7.3.3.1 Why are the choices of M and N used by the Jacobi iteration and Gauss-Seidel iteration convenient?

Solution. Both methods have two advantages:

- The multiplication $Nu^{(k)}$ can exploit sparsity in the original matrix A .
- Solving with M is relatively cheap. In the case of the Jacobi iteration ($M = D$) it is trivial. In the case of the Gauss-Seidel iteration ($M = (D - L)$), the lower triangular system inherits the sparsity pattern of the corresponding part of A .

Homework 7.3.3.2 Let $A = M - N$ be a splitting of matrix A . Let $x^{(k+1)} = M^{-1}(Nx^{(k)} + y)$. Show that

$$x^{(k+1)} = x^{(k)} + M^{-1}r^{(k)}, \text{ where } r^{(k)} = y - Ax^{(k)}.$$

Solution.

$$\begin{aligned}
 & x^{(k)} + M^{-1}r^{(k)} \\
 &= \\
 & x^{(k)} + M^{-1}(y - Ax^{(k)}) \\
 &= \\
 & x^{(k)} + M^{-1}(y - (M - N)x^{(k)}) \\
 &= \\
 & x^{(k)} + M^{-1}y - M^{-1}(M - N)x^{(k)} \\
 &= \\
 & x^{(k)} + M^{-1}y - (I - M^{-1}N)x^{(k)} \\
 &= \\
 & M^{-1}(Nx^{(k)} + y)
 \end{aligned}$$

This last exercise provides an important link between iterative refinement, discussed in Subsection 5.3.7, and splitting methods. Let us revisit this, using the notation from this section.

If $Ax = y$ and $x^{(k)}$ is a (current) approximation to x , then

$$r^{(k)} = y - Ax^{(k)}$$

is the (current) residual. If we solve

$$A\delta x^{(k)} = r^{(k)}$$

or, equivalently, compute

$$\delta x = A^{-1}r^{(k)}$$

then

$$x = x^{(k)} + \delta x$$

is the solution to $Ax = y$. Now, if we merely compute an approximation,

$$\delta x^{(k)} \approx A^{-1}r^{(k)},$$

then

$$x^{(k+1)} = x^{(k)} + \delta x^{(k)}$$

is merely a (hopefully better) approximation to x . If $M \approx A$ then

$$\delta x^{(k)} = M^{-1}r^{(k)} \approx A^{-1}r^{(k)}.$$

So, the better M approximates A , the faster we can expect $x^{(k)}$ to converge to x .

With this in mind, we notice that if $A = D - L - U$, where D , $-L$, and $-U$ equals its diagonal, strictly lower triangular, and strictly upper triangular part, and we split $A = M - N$, then $M = D - L$ is a better approximation to matrix A than is $M = D$.

Ponder This 7.3.3.3 Given these insights, why might the symmetric Gauss-Seidel method discussed in Homework 7.3.2.4 have benefits over the regular Gauss-Seidel method?

Loosely speaking, a sequence of numbers, $\chi^{(k)}$ is said to converge to the number χ if $|\chi^{(k)} - \chi|$ eventually becomes arbitrarily close to zero. This is written as

$$\lim_{k \rightarrow \infty} \chi^{(k)} = \chi.$$

A sequence of vectors, $x^{(k)}$, converges to the vector x if for some norm $\|\cdot\|$

$$\lim_{k \rightarrow \infty} \|x^{(k)} - x\| = 0.$$

Because of the equivalence of norms, if the sequence converges in one norm, it converges in all norms. In particular, it means it converges in the ∞ -norm, which means that $\max_i |\chi_i^{(k)} - \chi_i|$ converges to zero, and hence for all entries $|\chi_i^{(k)} - \chi_i|$ eventually becomes arbitrarily small. Finally, a sequence of matrices, $A^{(k)}$, converges to the matrix A if for some norm $\|\cdot\|$

$$\lim_{k \rightarrow \infty} \|A^{(k)} - A\|.$$

Again, if it converges for one norm, it converges for all norms and the individual elements of $A^{(k)}$ converge to the corresponding elements of A .

Let's now look at the convergence of splitting methods. If x solves $Ax = y$ and $x^{(k)}$ is the sequence of vectors generated starting with $x^{(0)}$, then

$$\begin{aligned} Mx &= Nx + y \\ Mx^{(k+1)} &= Nx^{(k)} + y \end{aligned}$$

so that

$$M(x^{(k+1)} - x) = N(x^{(k)} - x)$$

or, equivalently,

$$x^{(k+1)} - x = (M^{-1}N)(x^{(k)} - x).$$

This, in turn, means that

$$x^{(k+1)} - x = (M^{-1}N)^{k+1}(x^{(0)} - x).$$

If $\|\cdot\|$ is a vector norm and its induced matrix norm, then

$$\|x^{(k+1)} - x\| = \|(M^{-1}N)^{k+1}(x^{(0)} - x)\| \leq \|M^{-1}N\|^{k+1}\|x^{(0)} - x\|.$$

Hence, if $\|M^{-1}N\| < 1$ in that norm, then $\lim_{i \rightarrow \infty} \|M^{-1}N\|^i = 0$ and hence $x^{(k)}$ converges to x . We summarize this in the following theorem:

Theorem 7.3.3.1 *Let $A \in \mathbb{R}^{n \times n}$ be nonsingular and $x, y \in \mathbb{R}^n$ so that $Ax = y$. Let $A = M - N$ be a splitting of A , $x^{(0)}$ be given (an initial guess), and $x^{(k+1)} = M^{-1}(Nx^{(k)} + y)$. If $\|M^{-1}N\| < 1$ for some matrix norm induced by the $\|\cdot\|$ vector norm, then $x^{(k)}$ will converge to the solution x .*

Because of the equivalence of matrix norms, if we can find *any* matrix norm $\|\cdot\|$ such that $\|M^{-1}N\| < 1$, the sequence of vectors converges.

Ponder This 7.3.3.4 Contemplate the finer points of the last argument about the convergence of $(M^{-1}N)^i$



YouTube: https://www.youtube.com/watch?v=uv8cMeR9u_U

Understanding the following observation will have to wait until after we cover eigenvalues and eigenvectors, later in the course. For splitting methods, it is the spectral radius of a matrix (the magnitude of the eigenvalue with largest magnitude), $\rho(B)$, that often gives us insight into whether the method converges. This, once again, requires us to use a result from a future week in this course: It can be shown that for all $B \in \mathbb{R}^{m \times m}$ and $\epsilon > 0$ there exists a norm $\|\cdot\|_{B,\epsilon}$ such that $\|B\|_{B,\epsilon} \leq \rho(B) + \epsilon$. What this means is that if we can show that $\rho(M^{-1}N) < 1$, then the splitting method converges for the given matrix A .

Homework 7.3.3.5 Given nonsingular $A \in \mathbb{R}^{n \times n}$, what splitting $A = M - N$ will give the fastest convergence to the solution of $Ax = y$?

Solution. $M = A$ and $N = 0$. Then, regardless of the initial vector $x^{(0)}$,

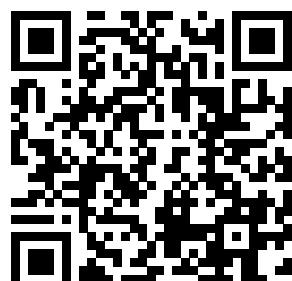
$$x^{(1)} := M^{-1}(Nx^{(0)} + y) = A^{-1}(0x^{(0)} + y) = A^{-1}y.$$

Thus, convergence occurs after a single iteration.



YouTube: <https://www.youtube.com/watch?v=lKlsk0qdCu0>

7.3.4 Successive Over-Relaxation (SOR)



YouTube: <https://www.youtube.com/watch?v=t9Bl9z7HPTQ>

Recall that if $A = D - L - U$ where $-L$, D , and $-U$ are the strictly lower triangular, diagonal, and strictly upper triangular parts of A , then the Gauss-Seidel iteration for solving $Ax = y$ can be expressed as $x^{(k+1)} = (D - L)^{-1}(Ux + y)$ or, equivalently, $\chi_i^{(k+1)}$ solves

$$\sum_{j=0}^{i-1} \alpha_{i,j} \chi_j^{(k+1)} + \alpha_{i,i} \chi_i^{(k+1)} = - \sum_{j=i+1}^{n-1} \alpha_{i,j} \chi_j^{(k)} + \psi_i.$$

where any term involving a zero is skipped. We label this $\chi_i^{(k+1)}$ with $\chi_i^{\text{GS}(i+1)}$ in our subsequent discussion.

What if we pick our next value a bit further:

$$\chi_i^{(k+1)} = \omega \chi_i^{\text{GS}(k+1)} + (1 - \omega) \chi_i^{(k)},$$

where $\omega \geq 1$. This is known as **over-relaxation**. Then

$$\chi_i^{\text{GS}(k+1)} = \frac{1}{\omega} \chi_i^{(k+1)} - \frac{1 - \omega}{\omega} \chi_i^{(k)}$$

and

$$\sum_{j=0}^{i-1} \alpha_{i,j} \chi_j^{(k+1)} + \alpha_{i,i} \left[\frac{1}{\omega} \chi_i^{(k+1)} - \frac{1 - \omega}{\omega} \chi_i^{(k)} \right] = - \sum_{j=i+1}^{n-1} \alpha_{i,j} \chi_j^{(k)} + \psi_i$$

or, equivalently,

$$\sum_{j=0}^{i-1} \alpha_{i,j} \chi_j^{(k+1)} + \frac{1}{\omega} \alpha_{i,i} \chi_i^{(k+1)} = \frac{1 - \omega}{\omega} \alpha_{i,i} \chi_i^{(k)} - \sum_{j=i+1}^{n-1} \alpha_{i,j} \chi_j^{(k)} + \psi_i.$$

This is equivalent to splitting

$$A = \underbrace{\left(\frac{1}{\omega} D - L \right)}_M - \underbrace{\left(\frac{1 - \omega}{\omega} D + U \right)}_N,$$

an iteration known as successive over-relaxation (SOR). The idea now is that the relaxation parameter ω can often be chosen to improve (reduce) the spectral radius of $M^{-1}N$, thus accelerating convergence.

We continue with $A = D - L - U$, where $-L$, D , and $-U$ are the strictly lower triangular, diagonal, and strictly upper triangular parts of A . Building on SOR where

$$A = \underbrace{\left(\frac{1}{\omega} D - L \right)}_{M_F} - \underbrace{\left(\frac{1 - \omega}{\omega} D + U \right)}_{N_F},$$

where the F stands for "Forward." Now, an alternative would be to compute the elements of x in reverse order, using the latest available values. This is equivalent to splitting

$$A = \underbrace{\left(\frac{1}{\omega} D - U \right)}_{M_R} - \underbrace{\left(\frac{1 - \omega}{\omega} D + L \right)}_{N_R},$$

where the R stands for "Reverse." The symmetric successive over-relaxation (SSOR) iteration combines the "forward" SOR with a "reverse" SOR, much like the symmetric Gauss-Seidel does:

$$\begin{aligned} x^{(k+\frac{1}{2})} &= M_F^{-1}(N_F x^{(k)} + y) \\ x^{(k+1)} &= M_R^{-1}(N_R x^{(k+\frac{1}{2})} + y). \end{aligned}$$

This can be expressed as splitting $A = M - N$. The details are a bit messy, and we will skip them.

7.4 Enrichments

7.4.1 Details!

To solve the problem computationally the problem is again discretized. Relating back to the problem of the membrane on the unit square in the previous section, this means that the continuous domain is viewed as a mesh instead, as illustrated in [Figure 7.4.1.1](#).

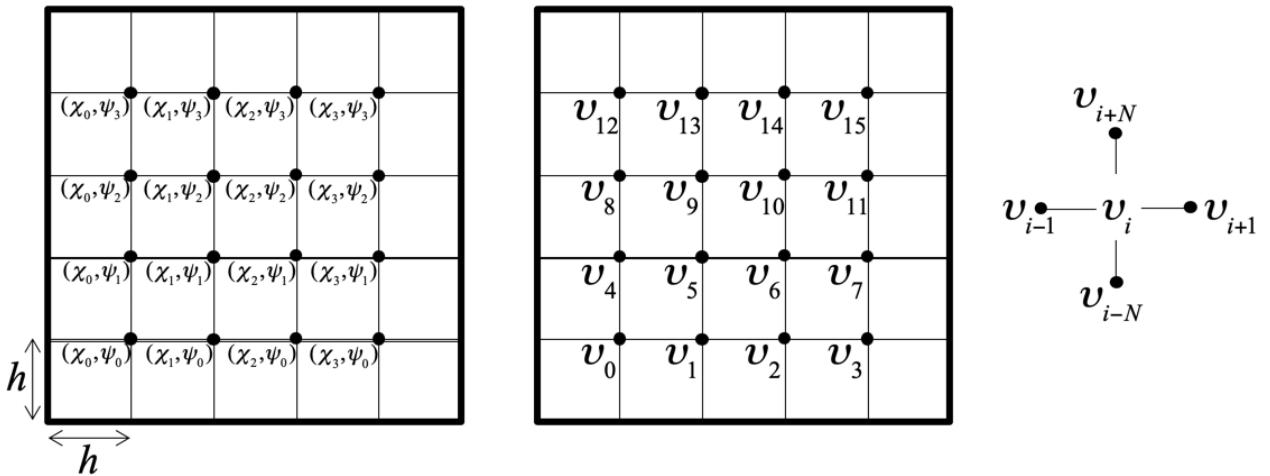


Figure 7.4.1.1 2D mesh.

In that figure, v_i equals, for example, the displacement from rest of the point on the membrane.

Now, let ϕ_i be the value of $f(x, y)$ at the mesh point i . One can approximate

$$\frac{\partial^2 u(x, y)}{\partial x^2} \approx \frac{u(x-h, y) - 2u(x, y) + u(x+h, y)}{h^2}$$

and

$$\frac{\partial^2 u(x, y)}{\partial y^2} \approx \frac{u(x, y-h) - 2u(x, y) + u(x, y+h)}{h^2}$$

so that

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

becomes

$$\frac{-u(x-h, y) + 2u(x, y) - u(x+h, y)}{h^2} + \frac{-u(x, y-h) + 2u(x, y) - u(x, y+h)}{h^2} = f(x, y)$$

or, equivalently,

$$\frac{-u(x-h, y) - u(x, y-h) + 4u(x, y) - u(x+h, y) - u(x, y+h)}{h^2} = f(x, y).$$

If (x, y) corresponds to the point i in a mesh where the interior points form an $N \times N$ grid, this translates to the system of linear equations

$$-v_{i-N} - v_{i-1} + 4v_i - v_{i+1} - v_{i+N} = h^2\phi_i.$$

This can be rewritten as

$$v_i = \frac{h^2\phi_i + v_{i-N} + v_{i-1} + v_{i+1} + v_{i+N}}{4}$$

or

$$\begin{array}{ccccccccc} 4v_0 & -v_1 & & -v_4 & & & & = h^2\phi_0 \\ -v_0 & +4v_1 & -v_2 & & -v_5 & & & = h^2\phi_1 \\ & -v_1 & +4v_2 & -v_3 & & -v_6 & & = h^2\phi_2 \\ & & -v_2 & +4v_3 & & & -v_7 & = h^2\phi_3 \\ -v_0 & & & +4v_4 & -v_5 & & -v_8 & = h^2\phi_4 \\ & v_1 & & \ddots & \ddots & \ddots & \ddots & \vdots \end{array}$$

In matrix notation this becomes

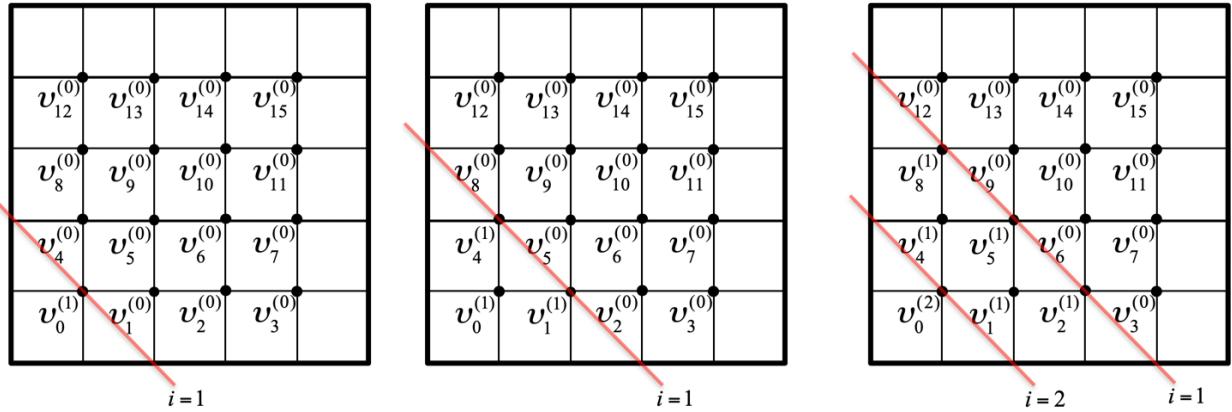
$$\left(\begin{array}{cccc|ccc|c} 4 & -1 & & & -1 & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & -1 & & -1 & & & \\ & & -1 & 4 & & & -1 & & \\ \hline -1 & & & & 4 & -1 & & -1 & \\ & -1 & & & -1 & 4 & -1 & & \\ & & -1 & & -1 & 4 & -1 & & \\ & & & -1 & & -1 & 4 & & \\ \hline & & & & -1 & & 4 & & \\ & & & & & \ddots & \ddots & & \\ & & & & & & \ddots & & \vdots \end{array} \right) \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{pmatrix} = \begin{pmatrix} h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ h^2\phi_8 \end{pmatrix}. \quad (7.4.1)$$

This demonstrates how solving the discretized Poisson's equation boils down to the solution of a linear system $Au = h^2f$, where A has a distinct sparsity pattern (pattern of nonzeros).

7.4.2 Parallelism in splitting methods

One of the advantages of, for example, the Jacobi iteration over the Gauss-Seidel iteration is that the values at all mesh points can be updated simultaneously. This comes at the expense of slower convergence to the solution.

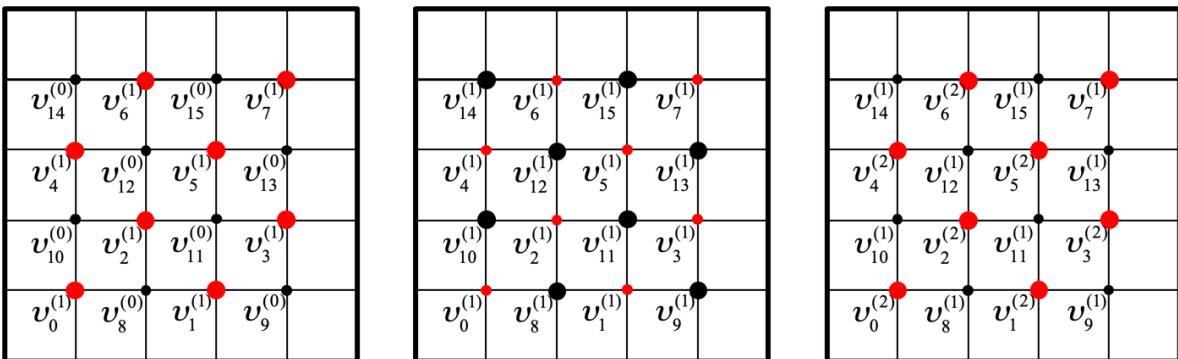
There is actually quite a bit of parallelism to be exploited in the Gauss-Seidel iteration as well. Consider our example of a mesh on a square domain as illustrated by



- First $v_0^{(1)}$ is computed from $v_1^{(0)}$ and $v_N^{(0)}$.
- Second, simultaneously,
 - $v_1^{(1)}$ can be computed from $v_0^{(1)}$, $v_2^{(0)}$, and $v_{N+1}^{(0)}$.
 - $v_N^{(1)}$ can be computed from $v_0^{(1)}$, $v_{N+1}^{(0)}$, and $v_{2N}^{(0)}$.
- Third, simultaneously,
 - $v_2^{(1)}$ can be computed from $v_1^{(1)}$, $v_2^{(0)}$, and $v_{N+2}^{(0)}$.
 - $v_{N+1}^{(1)}$ can be computed from $v_1^{(1)}$, $v_N^{(0)}$, and $v_{N+2}^{(0)}$, and $v_{2N+1}^{(0)}$.
 - $v_{2N}^{(1)}$ can be computed from $v_N^{(1)}$, and $v_{2N+1}^{(0)}$, and $v_{3N}^{(0)}$.
 - AND $v_0^{(2)}$ can be computed from $v_1^{(1)}$ and $v_N^{(1)}$, which starts a new "wave."

What we notice is that taking the opportunity to update when data is ready creates wavefronts through the mesh, where each wavefront corresponds to computation related to a different iteration.

Alternatively, extra parallelism can be achieved by ordering the mesh points using what is called a **red-black ordering**. Again focusing on our example of a mesh placed on a domain, the idea is to partition the mesh points into two groups, where each group consists of points that are not adjacent in the mesh: the red points and the black points.



The iteration then proceeds by alternating between (simultaneously) updating all values at the red points and (simultaneously) updating all values at the black points, always using the most updated values.

7.4.3 Dr. SOR



YouTube: <https://www.youtube.com/watch?v=WDsF7gaj4E4>

SOR was first proposed in 1950 by David M. Young and Stanley P. Frankel. David Young (1923-2008) was a colleague of ours at UT-Austin. His vanity license plate read "Dr. SOR."

7.5 Wrap Up

7.5.1 Additional homework

Homework 7.5.1.1 In Subsection 7.3.4 we discussed SOR and SSOR. Research how to choose the relaxation parameter ω and then modify your implementation of Gauss-Seidel from Homework 7.3.2.1 to investigate the benefits.

7.5.2 Summary

Let $A \in \mathbb{R}^{n \times n}$ be tridiagonal and SPD so that

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{1,0} & & & \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{2,1} & & \\ & \ddots & \ddots & \ddots & \\ & & \alpha_{n-2,n-3} & \alpha_{n-2,n-2} & \alpha_{n-1,n-2} \\ & & & \alpha_{n-1,n-2} & \alpha_{n-1,n-1} \end{pmatrix}.$$

Then its Cholesky factor is given by

$$\begin{pmatrix} \lambda_{0,0} & & & & \\ \lambda_{1,0} & \lambda_{1,1} & & & \\ & \ddots & \ddots & & \\ & & \lambda_{n-2,n-3} & \lambda_{n-2,n-2} & \\ & & & \lambda_{n-1,n-2} & \lambda_{n-1,n-1} \end{pmatrix}.$$

An algorithm for computing it is given by

```

for  $i = 0, \dots, n - 2$ 
     $\alpha_{i,i} := \sqrt{\alpha_{i,i}}$ 
     $\alpha_{i+1,i} := \alpha_{i+1,i}/\alpha_{i,i}$ 
     $\alpha_{i+1,i+1} := \alpha_{i+1,i+1} - \alpha_{i+1,i}\alpha_{i+1,i}$ 
endfor
 $\alpha_{n-1,n-1} := \sqrt{\alpha_{n-1,n-1}}$ 

```

It requires n square roots, $n - 1$ divides, $n - 1$ multiplies, and $n - 1$ subtracts. An algorithm for overwriting y with the solution to $Ax = y$ given its Cholesky factor is given by

- Overwrite y with the solution of $Lz = y$ (forward substitution) is accomplished by the following algorithm (here L has overwritten A):

```

for  $i = 0, \dots, n - 2$ 
     $\psi_i := \psi_i/\alpha_{i,i}$ 
     $\psi_{i+1} := \psi_{i+1} - \alpha_{i+1,i}\psi_i$ 
endfor
 $\psi_{n-1} := \psi_{n-1}/\alpha_{n-1,n-1}$ 

```

- Overwriting y with the solution of $L^T z = y$, (where z has overwritten y (back substitution)).

```

for  $i = n - 1, \dots, 1$ 
     $\psi_i := \psi_i/\alpha_{i,i}$ 
     $\psi_{i-1} := \psi_{i-1} - \alpha_{i,i-1}\psi_i$ 
endfor
 $\psi_0 := \psi_0/\alpha_{0,0}$ 

```

Definition 7.5.2.1 The half-band width of a symmetric matrix equals the number of sub-diagonals beyond which all the matrix contains only zeroes. For example, a diagonal matrix has half-band width of zero and a tridiagonal matrix has a half-band width of one. ◇

Nested dissection: a hierarchical partitioning of the graph that captures the sparsity of a matrix in an effort to reorder the rows and columns of that matrix so as to reduce fill-in (the overwriting of zeroes in the matrix with nonzeros).

Splitting methods: The system of linear equations $Ax = y$, splitting methods view A as $A = M - N$ and then, given an initial approximation $x^{(0)}$, create a sequence of approximations, $x^{(k)}$ that under mild conditions converge to x by solving

$$Mx^{(k+1)} = Nx^{(k)} + b$$

or, equivalently, computing

$$x^{(k+1)} = M^{-1}(Nx^{(k)} + b).$$

This method converges to x if for some norm $\|\cdot\|$

$$\|M^{-1}N\| < 1.$$

Given $A = D - L - U$ where $-L$, D , and $-U$ equal the strictly lower triangular, diagonal, and strictly upper triangular parts of A , commonly used splitting methods are

- Jacobi iteration: $A = \underbrace{D}_M - \underbrace{(L + U)}_N$.
- Gauss-Seidel iteration: $A = \underbrace{D - L}_M - \underbrace{U}_N$.
- Successive Over-Relaxation (SOR): $A = \underbrace{\frac{1}{\omega}D - L}_M - \underbrace{\left(\frac{1-\omega}{\omega}D + U\right)}_N$, where ω is the relaxation parameter.
- Symmetric Successive Over-Relaxation (SSOR).

Week 8

Descent Methods

8.1 Opening

8.1.1 Solving linear systems by solving a minimization problem



YouTube: <https://www.youtube.com/watch?v=--WEfBpj1Ts>

Consider the quadratic polynomial

$$f(\chi) = \frac{1}{2}\alpha\chi^2 - \beta\chi.$$

Finding the value $\hat{\chi}$ that minimizes this polynomial can be accomplished via the steps:

- Compute the derivative and set it to zero:

$$f'(\hat{\chi}) = \alpha\hat{\chi} - \beta = 0.$$

We notice that computing $\hat{\chi}$ is equivalent to solving the linear system (of one equation)

$$\alpha\hat{\chi} = \beta.$$

- It is a minimum if $\alpha > 0$ (the quadratic polynomial is concaved up).

Obviously, you can turn this around: in order to solve $\alpha\hat{\chi} = \beta$ where $\alpha > 0$, we can instead minimize the polynomial

$$f(\chi) = \frac{1}{2}\alpha\chi^2 - \beta\chi.$$

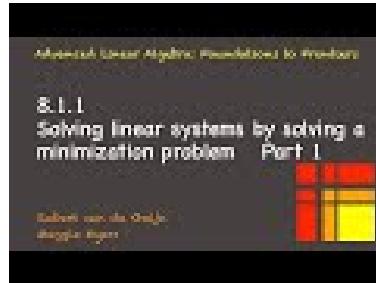
This course does not have multivariate calculus as a prerequisite, so we will walk you through the basic results we will employ. We will focus on finding a solution to $Ax = b$ where A is symmetric positive definite (SPD). (In our discussions we will just focus on real-valued problems). Now, if

$$f(x) = \frac{1}{2}x^T Ax - x^T b,$$

then its **gradient** equals

$$\nabla f(x) = Ax - b.$$

The function $f(x)$ is minimized (when A is SPD) when its gradient equals zero, which allows us to compute the vector for which the function achieves its minimum. The basic insight is that in order to solve $A\hat{x} = b$ we can instead find the vector \hat{x} that minimizes the function $f(x) = \frac{1}{2}x^T Ax - x^T b$.



YouTube: <https://www.youtube.com/watch?v=rh9GhwU1fuU>

Theorem 8.1.1.1 Let A be SPD and assume that $A\hat{x} = b$. Then the vector \hat{x} minimizes the function $f(x) = \frac{1}{2}x^T Ax - x^T b$.

Proof. This proof does not employ multivariate calculus!

Let $A\hat{x} = b$. Then

$$\begin{aligned} f(x) &= <\text{definition of } f(x)> \\ \frac{1}{2}x^T Ax - x^T b &= < A\hat{x} = b > \\ \frac{1}{2}x^T Ax - x^T A\hat{x} &= <\text{algebra}> \\ \frac{1}{2}x^T Ax - x^T A\hat{x} + \underbrace{\frac{1}{2}\hat{x}^T A\hat{x} - \frac{1}{2}\hat{x}^T A\hat{x}}_0 &= <\text{factor out}> \\ \frac{1}{2}(x - \hat{x})^T A(x - \hat{x}) - \frac{1}{2}\hat{x}^T A\hat{x}. & \end{aligned}$$

Since $\hat{x}^T A\hat{x}$ is independent of x , and A is SPD, this is clearly minimized when $x = \hat{x}$. ■

8.1.2 Overview

- 8.1 Opening
 - 8.1.1 Solving linear systems by solving a minimization problem

- 8.1.2 Overview
- 8.1.3 What you will learn
- 8.2 Search directions
 - 8.2.1 Basics of descent methods
 - 8.2.2 Toward practical descent methods
 - 8.2.3 Relation to Splitting Methods
 - 8.2.4 Method of Steepest Descent
 - 8.2.5 Preconditioning
- 8.3 The Conjugate Gradient Method
 - 8.3.1 A-conjugate directions
 - 8.3.2 Existence of A-conjugate search directions
 - 8.3.3 Conjugate Gradient Method Basics
 - 8.3.4 Technical details
 - 8.3.5 Practical Conjugate Gradient Method algorithm
 - 8.3.6 Final touches for the Conjugate Gradient Method
- 8.4 Enrichments
 - 8.4.1 Conjugate Gradient Method: Variations on a theme
- 8.5 Wrap Up
 - 8.5.1 Additional homework
 - 8.5.2 Summary

8.1.3 What you will learn

This week, you are introduced to additional techniques for solving sparse linear systems (or any linear system where computing a matrix-vector multiplication with the matrix is cheap). We discuss descent methods in general and the Conjugate Gradient Method in particular, which is the most important member of this family of algorithms.

Upon completion of this week, you should be able to

- Relate solving a linear system of equations $Ax = b$, where A is symmetric positive definite (SPD), to finding the minimum of the function $f(x) = \frac{1}{2}x^T Ax + x^T b$.
- Solve $Ax = b$ via descent methods including the Conjugate Gradient Method.
- Exploit properties of A-conjugate search directions to morph the Method of Steepest Descent into a practical Conjugate Gradient Method.

- Recognize that while in exact arithmetic the Conjugate Gradient Method solves $Ax = b$ in a finite number of iterations, in practice it is an iterative method due to error introduced by floating point arithmetic.
- Accelerate the Method of Steepest Descent and Conjugate Gradient Method by applying a preconditioner implicitly defines a new problem with the same solution and better condition number.

8.2 Search directions

8.2.1 Basics of descent methods



YouTube: <https://www.youtube.com/watch?v=V7Cvihzs-n4>

Remark 8.2.1.1 In the video, the quadratic polynomial pictured takes on the value $-\hat{x}A\hat{x}$ at \hat{x} and that minimum is below the x-axis. This does not change the conclusions that are drawn in the video.

The basic idea behind a descent method is that at the k th iteration one has an approximation to x , $x^{(k)}$, and one would like to create a better approximation, $x^{(k+1)}$. To do so, the method picks a **search direction**, $p^{(k)}$, and chooses the next approximation by taking a step from the current approximate solution in the direction of $p^{(k)}$:

$$x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}.$$

In other words, one searches for a minimum along a line defined by the current iterate, $x^{(k)}$, and the search direction, $p^{(k)}$. One then picks α_k so that, preferably, $f(x^{(k+1)}) \leq f(x^{(k)})$. This is summarized in [Figure 8.2.1.2](#).

```

Given :  $A, b, x^{(0)}$ 
 $r^{(0)} := b - Ax^{(0)}$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
     $p^{(k)} :=$  next direction
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$  for some scalar  $\alpha_k$ 
     $r^{(k+1)} := b - Ax^{(k+1)}$ 
     $k := k + 1$ 
endwhile

```

Figure 8.2.1.2 Outline for a descent method.

To this goal, typically, an **exact descent method** picks α_k to exactly minimize the function along the line from the current approximate solution in the direction of $p^{(k)}$.



YouTube: <https://www.youtube.com/watch?v=01Slxl3oAc8>

Now,

$$\begin{aligned}
& f(x^{(k+1)}) \\
& = < x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} > \\
& f(x^{(k)} + \alpha_k p^{(k)}) \\
& = < \text{evaluate} > \\
& \frac{1}{2} (x^{(k)} + \alpha_k p^{(k)})^T A (x^{(k)} + \alpha_k p^{(k)}) - (x^{(k)} + \alpha_k p^{(k)})^T b \\
& = < \text{multiply out} > \\
& \frac{1}{2} x^{(k)T} A x^{(k)} + \alpha_k p^{(k)T} A x^{(k)} + \frac{1}{2} \alpha_k^2 p^{(k)T} A p^{(k)} - x^{(k)T} b - \alpha_k p^{(k)T} b \\
& = < \text{rearrange} > \\
& \frac{1}{2} x^{(k)T} A x^{(k)} - x^{(k)T} b + \frac{1}{2} \alpha_k^2 p^{(k)T} A p^{(k)} + \alpha_k p^{(k)T} A x^{(k)} - \alpha_k p^{(k)T} b \\
& = < \text{substitute } f(x^{(k)}) \text{ and factor out common terms} > \\
& f(x^{(k)}) + \frac{1}{2} \alpha_k^2 p^{(k)T} A p^{(k)} + \alpha_k p^{(k)T} (A x^{(k)} - b) \\
& = < \text{substitute } r^{(k)} \text{ and commute to expose polynomial in } \alpha_k > \\
& \frac{1}{2} p^{(k)T} A p^{(k)} \alpha_k^2 - p^{(k)T} r^{(k)} \alpha_k + f(x^{(k)}),
\end{aligned}$$

where $r^{(k)} = b - Ax^{(k)}$ is the **residual**. This is a quadratic polynomial in the scalar α_k (since this is the only free variable).



YouTube: https://www.youtube.com/watch?v=SA_VrhP7EZg

Minimizing

$$f(x^{(k+1)}) = \frac{1}{2} p^{(k)T} A p^{(k)} \alpha_k^2 - p^{(k)T} r^{(k)} \alpha_k + f(x^{(k)})$$

exactly requires the derivative with respect to α_k to be zero:

$$0 = \frac{df(x^{(k)} + \alpha_k p^{(k)})}{d\alpha_k} = p^{(k)T} A p^{(k)} \alpha_k - p^{(k)T} r^{(k)}.$$

Hence, for a given choice of p_k

$$\alpha_k = \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}} \quad \text{and} \quad x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}.$$

provides the next approximation to the solution. This leaves us with the question of how to pick the search directions $\{p^{(0)}, p^{(1)}, \dots\}$.

A basic decent method based on these ideas is given in [Figure 8.2.1.3](#).

```

Given :  $A, b, x^{(0)}$ 
 $r^{(0)} := b - Ax^{(0)}$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
     $p^{(k)} :=$  next direction
     $\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := b - Ax^{(k+1)}$ 
     $k := k + 1$ 
endwhile

```

Figure 8.2.1.3 Basic descent method.

Homework 8.2.1.1 The cost of an iterative method is a combination of how many iterations it takes to convergence and the cost per iteration. For the loop in [Figure 8.2.1.3](#), count the number of matrix-vector multiplications, dot products, and "axpy" operations (not counting the cost of determining the next descent direction).

Solution.

$$\begin{aligned}\alpha_k &:= \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}} && 1 \text{ mvmult, 2 dot products} \\ x^{(k+1)} &:= x^{(k)} + \alpha_k p^{(k)} && 1 \text{ axpy} \\ r^{(k+1)} &:= b - Ax^{(k+1)} && 1 \text{ mvmult}\end{aligned}$$

Total: 2 matrix-vector multiplies (mvmults), 2 dot products, 1 axpy.

8.2.2 Toward practical descent methods



YouTube: https://www.youtube.com/watch?v=aBTI_EEQNKE

Even though matrices are often highly sparse, a major part of the cost of solving $Ax = b$ via descent methods is in the matrix-vector multiplication (a cost that is proportional to the number of nonzeros in the matrix). For this reason, reducing the number of these is an important part of the design of the algorithm.

Homework 8.2.2.1 Let

$$\begin{aligned}x^{(k+1)} &= x^{(k)} + \alpha_k p^{(k)} \\ r^{(k)} &= b - Ax^{(k)} \\ r^{(k+1)} &= b - Ax^{(k+1)}\end{aligned}$$

Show that

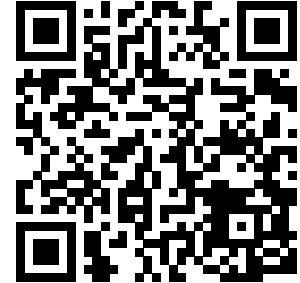
$$r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}.$$

Solution.

$$\begin{aligned}r^{(k+1)} &= b - Ax^{(k+1)} \\ &= \langle r^{(k)} = b - Ax^{(k)} \rangle \\ r^{(k+1)} &= r^{(k)} + Ax^{(k)} - Ax^{(k+1)} \\ &= \langle \text{rearrange, factor} \rangle \\ r^{(k+1)} &= r^{(k)} - A(x^{(k+1)} - x^{(k)}) \\ &= \langle x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \rangle \\ r^{(k+1)} &= r^{(k)} - \alpha_k A p^{(k)}\end{aligned}$$

Alternatively:

$$\begin{aligned}
 r^{(k+1)} &= b - Ax^{(k+1)} \\
 &= \quad < x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} > \\
 r^{(k+1)} &= b - A(x^{(k)} + \alpha_k p^{(k)}) \\
 &= \quad < \text{distribute} > \\
 r^{(k+1)} &= b - Ax^{(k)} - \alpha_k Ap^{(k)} \\
 &= \quad < \text{definition of } r^{(k)} > \\
 r^{(k+1)} &= r^{(k)} - \alpha_k Ap^{(k)}
 \end{aligned}$$



YouTube: <https://www.youtube.com/watch?v=j00GS9mTgd8>

With the insights from this last homework, we can reformulate our basic descent method into one with only one matrix-vector multiplication, as illustrated in Figure 8.2.2.1.

Given : $A, b, x^{(0)}$
 $r^{(0)} := b - Ax^{(0)}$
 $k := 0$
while $r^{(k)} \neq 0$
 $\quad p^{(k)} = \text{next direction}$
 $\quad \alpha_k := \frac{p^{(k) T} r^{(k)}}{p^{(k) T} A p^{(k)}}$
 $\quad x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
 $\quad r^{(k+1)} := b - Ax^{(k+1)}$
 $\quad k := k + 1$
endwhile

Given : $A, b, x^{(0)}$
 $r^{(0)} := b - Ax^{(0)}$
 $k := 0$
while $r^{(k)} \neq 0$
 $\quad p^{(k)} := \text{next direction}$
 $\quad \alpha_k := \frac{p^{(k) T} r^{(k)}}{p^{(k) T} A p^{(k)}}$
 $\quad x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
 $\quad r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$
 $\quad k := k + 1$
endwhile

Given : $A, b, x^{(0)}$
 $r^{(0)} := b - Ax^{(0)}$
 $k := 0$
while $r^{(k)} \neq 0$
 $\quad p^{(k)} := \text{next direction}$
 $\quad q^{(k)} := A p^{(k)}$
 $\quad \alpha_k := \frac{p^{(k) T} r^{(k)}}{p^{(k) T} q^{(k)}}$
 $\quad x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
 $\quad r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$
 $\quad k := k + 1$
endwhile

Figure 8.2.2.1 Left: Basic descent method from last unit. Middle: Minor modification that recasts the computation of the residual $r^{(k+1)}$ as an update of the previous residual $r^{(k)}$. Right: modification that reduces the number of matrix-vector multiplications by introducing temporary vector $q^{(k)}$.

Homework 8.2.2.2 For loops in the algorithm in Figure 8.2.2.1 (Right), count the number of matrix-vector multiplications, dot products, and "axy" operations (not counting the cost of determining the next descent direction).

Solution.

$$\begin{aligned}
 q^{(k)} &:= Ap^{(k)} && 1 \text{ mvmult} \\
 \alpha_k &:= \frac{p^{(k)T} r^{(k)}}{p^{(k)T} q x^{(k)}} && 2 \text{ dot products} \\
 x^{(k+1)} &:= x^{(k)} + \alpha_k p^{(k)} && 1 \text{ axpy} \\
 r^{(k+1)} &:= r^{(k)} - \alpha_k q^{(k)} && 1 \text{ axpy}
 \end{aligned}$$

Total: 1 mvmults, 2 dot products, 2 axpys



YouTube: https://www.youtube.com/watch?v=0GqV_hfaxJA

We finish our discussion regarding basic descent methods by observing that we don't need to keep the history of vectors, $x^{(k)}$, $p^{(k)}$, $r^{(k)}$, $q^{(k)}$, and scalar α_k that were computed as long as they are not needed to compute the next search direction, leaving us with the algorithm

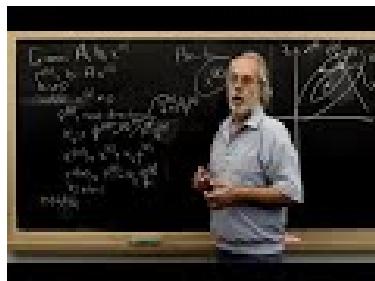
```

Given :  $A, b, x$ 
 $r := b - Ax$ 
while  $r^{(k)} \neq 0$ 
     $p :=$  next direction
     $q := Ap$ 
     $\alpha := \frac{p^T r}{p^T q}$ 
     $x := x + \alpha p$ 
     $r := r - \alpha q$ 
endwhile

```

Figure 8.2.2.2 The algorithm from [Figure 8.2.2.1](#) (Right) storing only the most current vectors and scalar.

8.2.3 Relation to Splitting Methods



YouTube: <https://www.youtube.com/watch?v=ifwail0B1EI>

Let us pick some really simple search directions in the right-most algorithm in [Home-](#)

work 8.2.2.2: $p^{(k)} = e_i \bmod n$, which cycles through the standard basis vectors.

Homework 8.2.3.1 For the right-most algorithm in [Homework 8.2.2.2](#), show that if $p^{(0)} = e_0$, then

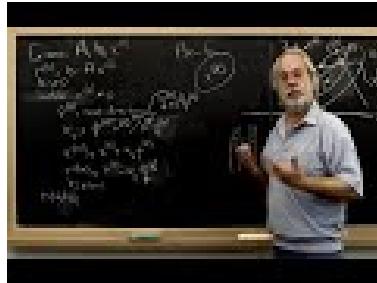
$$\chi_0^{(1)} = \chi_0^{(0)} + \frac{1}{\alpha_{0,0}} \left(\beta_0 - \sum_{j=0}^{n-1} \alpha_{0,j} \chi_j^{(0)} \right) = \frac{1}{\alpha_{0,0}} \left(\beta_0 - \sum_{j=1}^{n-1} \alpha_{0,j} \chi_j^{(0)} \right).$$

Solution.

- $p^{(0)} = e_0$.
- $p^{(0)T} A p^{(0)} = e_0^T A e_0 = \alpha_{0,0}$ (the $(0, 0)$ element in A , not to be mistaken for α_0).
- $r^{(0)} = Ax^{(0)} - b$.
- $p^{(0)T} r^{(0)} = e_0^T (b - Ax^{(0)}) = e_0^T b - e_0^T Ax^{(0)} = \beta_0 - \tilde{a}_0^T x^{(0)}$, where \tilde{a}_k^T denotes the k th row of A .
- $x^{(1)} = x^{(0)} + \alpha_0 p^{(0)} = x^{(0)} + \frac{p^{(0)T} r^{(0)}}{p^{(0)T} A p^{(0)}} e_0 = x^{(0)} + \frac{\beta_0 - \tilde{a}_0^T x^{(0)}}{\alpha_{0,0}} e_0$. This means that only the first element of $x^{(0)}$ changes, and it changes to

$$\chi_0^{(1)} = \chi_0^{(0)} + \frac{1}{\alpha_{0,0}} \left(\beta_0 - \sum_{j=0}^{n-1} \alpha_{0,j} \chi_j^{(0)} \right) = \frac{1}{\alpha_{0,0}} \left(\beta_0 - \sum_{j=1}^{n-1} \alpha_{0,j} \chi_j^{(0)} \right).$$

This looks familiar...



YouTube: <https://www.youtube.com/watch?v=karx3stbVdE>

Careful contemplation of the last homework reveals that this is exactly how the first element in vector x , χ_0 , is changed in the Gauss-Seidel method!

Ponder This 8.2.3.2 Continue the above argument to show that this choice of descent directions yields the Gauss-Seidel iteration.

8.2.4 Method of Steepest Descent



YouTube: <https://www.youtube.com/watch?v=t0qAd10hIw>

For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that we are trying to minimize, for a given x , the direction in which the function most rapidly increases in value at x is given by its gradient,

$$\nabla f(x).$$

Thus, the direction in which it decreases most rapidly is

$$-\nabla f(x).$$

For our function

$$f(x) = \frac{1}{2}x^T Ax - x^T b$$

this direction of steepest descent is given by

$$-\nabla f(x) = -(Ax - b) = b - Ax,$$

which we recognize as the residual. Thus, recalling that $r^{(k)} = b - Ax^{(k)}$, the direction of steepest descent at $x^{(k)}$ is given by $p^{(k)} = r^{(k)} = b - Ax^{(k)}$. These insights motivate the algorithms in Figure 8.2.4.1.

Given : $A, b, x^{(0)}$

$$r^{(0)} := b - Ax^{(0)}$$

$$k := 0$$

while $r^{(k)} \neq 0$

$$p^{(k)} := r^{(k)}$$

$$q^{(k)} := Ap^{(k)}$$

$$\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} q^{(k)}}$$

$$x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$$

$$r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$$

$$k := k + 1$$

endwhile

Given : A, b, x

$$k := 0$$

$$r := b - Ax$$

while $r \neq 0$

$$p := r$$

$$q := Ap$$

$$\alpha := \frac{p^T r}{p^T q}$$

$$x := x + \alpha p$$

$$r := r - \alpha q$$

$$k := k + 1$$

endwhile

Figure 8.2.4.1 Steepest descent algorithm, with indices and without indices.

8.2.5 Preconditioning



YouTube: <https://www.youtube.com/watch?v=i-83HdtrI1M>

For a general (appropriately differential) nonlinear function $f(x)$, using the direction of steepest descent as the search direction is often a reasonable choice. For our problem, especially if A is relatively ill-conditioned, we can do better.

Here is the idea: Let $A = Q\Sigma Q^T$ be the SVD of SPD matrix A (or, equivalently for SPD matrices, its spectral decomposition, which we will discuss in [Subsection 9.2.4](#)). Then

$$f(x) = \frac{1}{2}x^T Ax - x^T b = \frac{1}{2}x^T Q\Sigma Q^T x - x^T QQ^T b.$$

Using the change of basis $y = Q^T x$ and $\hat{b} = Q^T b$, then

$$g(y) = \frac{1}{2}y^T \Sigma y - y^T \hat{b}.$$

How this relates to the convergence of the Method of Steepest Descent is discussed (informally) in the video. The key insight is that if $\kappa(A) = \sigma_0/\sigma_{n-1}$ (the ratio between the largest and smallest eigenvalues or, equivalently, the ratio between the largest and smallest singular value) is large, then convergence can take many iterations.

What would happen if instead $\sigma_0 = \dots = \sigma_{n-1}$? Then $A = Q\Sigma Q^T$ is the SVD/Spectral decomposition of A and $A = Q(\sigma_0 I)Q^T$. If we then perform the Method of Steepest Descent with y (the transformed vector x) and \hat{b} (the transformed right-hand side), then

$$\begin{aligned} y^{(1)} &= \\ &= y^{(0)} - \frac{r^{(0)T} r^{(0)}}{r^{(0)T} \sigma_0 I r^{(0)}} r^{(0)} \\ &= y^{(0)} - \frac{1}{\sigma_0} r^{(0)} \\ &= y^{(0)} - \frac{1}{\sigma_0} (\sigma_0 y^{(0)} - \hat{b}) \\ &= \frac{1}{\sigma_0} \hat{b}, \end{aligned}$$

which is the solution to $\sigma_0 I = \hat{b}$. Thus, the iteration converges in one step. The point we are trying to (informally) make is that if A is well-conditioned, then the Method of Steepest Descent converges faster.



Now, $Ax = b$ is equivalent to $M^{-1}Ax = M^{-1}b$. Hence, one can define a new problem with the same solution and, hopefully, a better condition number by letting $\tilde{A} = M^{-1}A$ and $\tilde{b} = M^{-1}b$. A better condition number results if $M \approx A$ since then $M^{-1}A \approx A^{-1}A \approx I$. A constraint is that M should be chosen so that solving with it is easy/cheap. The matrix M is called a **preconditioner**.

A problem is that, in our discussion of descent methods, we restrict ourselves to the case where the matrix is SPD. Generally speaking, $M^{-1}A$ will not be SPD. To fix this, choose $M \approx A$ to be SPD and let $M = L_M L_M^T$ equal its Cholesky factorization. If $A = LL^T$ is the Cholesky factorization of A , then $L_M^{-1}AL_M^{-T} \approx L_M^{-1}LL^T L_M^{-T} \approx I$. With this, we can transform our linear system $Ax = b$ in to one that has the same solution:

$$\underbrace{L_M^{-1}AL_M^{-T}}_{\tilde{A}} \underbrace{L_M^T x}_{\tilde{x}} = \underbrace{L_M^{-1}b}_{\tilde{b}}.$$

We note that \tilde{A} is SPD and hence one can apply the Method of Steepest Descent to $\tilde{A}\tilde{x} = \tilde{b}$, where $\tilde{A} = L_M^{-1}AL_M^{-T}$, $\tilde{x} = L_M^T x$, and $\tilde{b} = L_M^{-1}b$. Once the method converges to the solution \tilde{x} , one can transform that solution of this back to solution of the original problem by solving $L_M^T x = \tilde{x}$. If M is chosen carefully, $\kappa(L_M^{-1}AL_M^{-T})$ can be greatly improved. The best choice would be $M = A$, of course, but that is not realistic. The point is that in our case where A is SPD, ideally the preconditioner should be SPD.

Some careful rearrangement takes the method of steepest descent on the transformed problem to the much simpler preconditioned algorithm on the right in [Figure 8.2.5.1](#).

Given : $A, b, x^{(0)}$

$$r^{(0)} := b - Ax^{(0)}$$

$$k := 0$$

while $r^{(k)} \neq 0$

$$p^{(k)} := r^{(k)}$$

$$q^{(k)} := Ap^{(k)}$$

$$\alpha_k := \frac{p^{(k)T}r^{(k)}}{p^{(k)T}q^{(k)}}$$

$$x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$$

$$r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$$

$$k := k + 1$$

endwhile

Given : $A, b, x^{(0)}, M$

$$M = LL^T$$

$$\tilde{A} = L^{-1}AL^{-T}$$

$$\tilde{b} = L^{-1}b$$

$$\tilde{x}^{(0)} = L^T x^{(0)}$$

$$\tilde{r}^{(0)} := \tilde{b} - \tilde{A}\tilde{x}^{(0)}$$

$$k := 0$$

while $\tilde{r}^{(k)} \neq 0$

$$\tilde{p}^{(k)} := \tilde{r}^{(k)}$$

$$\tilde{q}^{(k)} := \tilde{A}\tilde{p}^{(k)}$$

$$\tilde{\alpha}_k := \frac{\tilde{p}^{(k)T}\tilde{r}^{(k)}}{\tilde{p}^{(k)T}\tilde{q}^{(k)}}$$

$$\tilde{x}^{(k+1)} := \tilde{x}^{(k)} + \tilde{\alpha}_k \tilde{p}^{(k)}$$

$$\tilde{r}^{(k+1)} := \tilde{r}^{(k)} - \tilde{\alpha}_k \tilde{q}^{(k)}$$

$$x^{(k+1)} = L^{-T}\tilde{x}^{(k+1)}$$

$$k := k + 1$$

endwhile

Given : $A, b, x^{(0)}, M$

$$r^{(0)} := b - Ax^{(0)}$$

$$k := 0$$

while $r^{(k)} \neq 0$

$$p^{(k)} := M^{-1}r^{(k)}$$

$$q^{(k)} := Ap^{(k)}$$

$$\alpha_k := \frac{p^{(k)T}r^{(k)}}{p^{(k)T}q^{(k)}}$$

$$x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$$

$$r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$$

$$x^{(k+1)} = L^{-T}\tilde{x}^{(k+1)}$$

$$k := k + 1$$

endwhile

Figure 8.2.5.1 Left: method of steepest descent. Middle: method of steepest descent with transformed problem. Right: preconditioned method of steepest descent. It can be checked that the $x^{(k)}$ computed by the middle algorithm is exactly the $x^{(k)}$ computed by the one on the right. Of course, the computation $x^{(k+1)} = L^{-T}\tilde{x}^{(k+1)}$ needs only be done once, after convergence, in the algorithm in the middle. We state it this way to facilitate [Homework 8.2.5.1](#).

Homework 8.2.5.1 Show that the algorithm in [Figure 8.2.5.1](#) (Middle) computes the same values for $x^{(k)}$ as does the algorithm to its right.

Hint. You will want to do a prove by induction. To start, conjecture a relationship between $\tilde{r}^{(k)}$ and $r^{(k)}$ and then prove that that relationship, and the relationship $x^{(k)} = L^{-T}\tilde{x}^{(k)}$ hold for all k , where $r^{(k)}$ and $x^{(k)}$ are as computed by the algorithm on the right.

Solution 1. Notice that $\tilde{A} = L^{-1}AL^{-T}$ implies that $\tilde{A}L^T = L^{-1}A$. We will show that for all $k \geq 0$

- $\tilde{x}^{(k)} = L^T x^{(k)}$

- $\tilde{r}^{(k)} = L^{-1}r^{(k)},$

- $\tilde{p}^{(k)} = L^T p^{(k)},$

- $\tilde{\alpha}_k = \alpha_k$

via a proof by induction.

- Base case: $k = 0$.

- $\tilde{x}^{(0)}$ is initialized as $\tilde{x}^{(0)} := L^T x^{(0)}$.

- $\tilde{r}^{(0)}$
 - = < algorithm on left >
 - $\tilde{b} - \tilde{A}\tilde{x}^{(0)}$
 - = < initialization of \tilde{b} and $\tilde{x}^{(0)}$ >
 - $L^{-1}b - \tilde{A}L^T x^{(0)}$
 - = < initialization of \tilde{A} >
 - $L^{-1}b - L^{-1}Ax^{(0)}$
 - = < factor out and initialization of $r^{(0)}$ >
 - $L^{-1}r^{(0)}$

- $\tilde{p}^{(0)}$
 - = < initialization in algorithm >
 - $\tilde{r}^{(0)}$
 - = < $\tilde{r}^{(0)} = L^{-1}r^{(0)}$ >
 - $L^{-1}r^{(0)}$
 - = < from right algorithm: $r^{(k)} = Mp^{(k)}$ and $M = LL^T$ >
 - $L^{-1}LL^Tp^{(0)}$
 - = < $L^{-1}L = I$ >
 - $= L^Tp^{(0)}.$

- $\tilde{\alpha}_0$
 - = < middle algorithm >
 - $\frac{\tilde{p}^{(0)T}\tilde{r}^{(0)}}{\tilde{p}^{(0)T}\tilde{A}\tilde{p}^{(0)}}$
 - = < $\tilde{p}^{(0)} = L^Tp^{(0)}$ etc. >
 - $\frac{(L^Tp^{(0)})^T L^{-1}r^{(0)}}{(L^Tp^{(0)})^T L^{-1}AL^{-T}L^Tp^{(0)}}$
 - = < transpose and cancel >
 - $\frac{p^{(0)T}r^{(0)}}{p^{(0)T}Ap^{(0)}}$
 - = < right algorithm >
 - $\alpha_0.$

- Inductive Step: Assume that $\tilde{x}^{(k)} = L^T x^{(k)}$, $\tilde{r}^{(k)} = L^{-1}r^{(k)}$, $\tilde{p}^{(k)} = L^T p^{(k)}$, and $\tilde{\alpha}_k = \alpha_k$. Show that $\tilde{x}^{(k+1)} = L^T x^{(k+1)}$, $\tilde{r}^{(k+1)} = L^{-1}r^{(k+1)}$, $\tilde{p}^{(k+1)} = L^T p^{(k+1)}$, and $\tilde{\alpha}_{k+1} = \alpha_{k+1}$.

- $\tilde{x}^{(k+1)}$
 - = middle algorithm
 - $\tilde{x}^{(k)} + \tilde{\alpha}_k \tilde{p}^{(k)}$
 - = < I.H. >
 - $L^T x^{(k)} + \alpha_k L^T p^{(k)}$
 - = < factor out; right algorithm >
 - $L^T x^{(k+1)}$

- $\tilde{r}^{(k+1)}$

$$\begin{aligned}
 &= \quad < \text{middle algorithm} > \\
 \tilde{r}^{(k)} - \tilde{\alpha}_k \tilde{A} \tilde{p}^{(k)} &= \quad < \text{I.H.} > \\
 L^{-1} r^{(k)} - \alpha_k L^{-1} A L^{-T} L^T p^{(k)} &= \quad < L^{-T} L^T = I; \text{ factor out; right algorithmn} > \\
 L^{-1} r^{(k+1)} &
 \end{aligned}$$
- $\tilde{p}^{(k+1)}$

$$\begin{aligned}
 &= \quad < \text{middle algorithm} > \\
 \tilde{r}^{(k+1)} &= \quad < \tilde{r}^{(k+1)} = L^{-1} r^{(k+1)} > \\
 L^{-1} r^{(k+1)} &= \quad < \text{from right algorithm: } r^{(k+1)} = M p^{(k+1)} \text{ and } M = LL^T > \\
 L^{-1} L L^T p^{(k+1)} &= \quad < L^{-1} L = I > \\
 &= L^T p^{(k+1)}. \\
 \end{aligned}$$
- $\tilde{\alpha}_{k+1}$

$$\begin{aligned}
 &= \quad < \text{middle algorithm} > \\
 \frac{\tilde{p}^{(k+1) T} \tilde{r}^{(k+1)}}{\tilde{p}^{(k+1) T} \tilde{A} \tilde{p}^{(k+1)}} &= \quad < \tilde{p}^{(k+1)} = L^T p^{(k+1)} \text{ etc.} > \\
 \frac{(L^T p^{(k+1)})^T L^{-1} r^{(k+1)}}{(L^T p^{(k+1)})^T L^{-1} A L^{-T} L^T p^{(k+1)}} &= \quad < \text{transpose and cancel} > \\
 \frac{p^{(k+1) T} r^{(k+1)}}{p^{(k+1) T} A p^{(k+1)}} &= \quad < \text{right algorithm} > \\
 &\alpha_{k+1}.
 \end{aligned}$$

- By the Principle of Mathematical Induction the result holds.

Solution 2 (Constructive solution). Let's start with the algorithm in the middle:

```

Given :  $A, b, x^{(0)}$ ,
 $M = LL^T$ 
 $\tilde{A} = L^{-1}AL^{-T}$ 
 $\tilde{b} = L^{-1}b$ 
 $\tilde{x}^{(0)} = L^T x^{(0)}$ 
 $\tilde{r}^{(0)} := \tilde{b} - \tilde{A}\tilde{x}^{(0)}$ 
 $k := 0$ 
while  $\tilde{r}^{(k)} \neq 0$ 
     $\tilde{p}^{(k)} := \tilde{r}^{(k)}$ 
     $\tilde{q}^{(k)} := \tilde{A}\tilde{p}^{(k)}$ 
     $\tilde{\alpha}_k := \frac{\tilde{p}^{(k)T}\tilde{r}^{(k)}}{\tilde{p}^{(k)T}\tilde{q}^{(k)}}$ 
     $\tilde{x}^{(k+1)} := \tilde{x}^{(k)} + \tilde{\alpha}_k\tilde{p}^{(k)}$ 
     $\tilde{r}^{(k+1)} := \tilde{r}^{(k)} - \tilde{\alpha}_k\tilde{q}^{(k)}$ 
     $x^{(k+1)} = L^{-T}\tilde{x}^{(k+1)}$ 
     $k := k + 1$ 
endwhile

```

We now notice that $\tilde{A} = L^{-1}AL^{-T}$ and we can substitute this into the algorithm:

```

Given :  $A, b, x^{(0)}$ ,
 $M = LL^T$ 
 $\tilde{b} = L^{-1}b$ 
 $\tilde{x}^{(0)} = L^T x^{(0)}$ 
 $\tilde{r}^{(0)} := \tilde{b} - L^{-1}AL^{-T}\tilde{x}^{(0)}$ 
 $k := 0$ 
while  $\tilde{r}^{(k)} \neq 0$ 
     $\tilde{p}^{(k)} := \tilde{r}^{(k)}$ 
     $\tilde{q}^{(k)} := L^{-1}AL^{-T}\tilde{p}^{(k)}$ 
     $\tilde{\alpha}_k := \frac{\tilde{p}^{(k)T}\tilde{r}^{(k)}}{\tilde{p}^{(k)T}\tilde{q}^{(k)}}$ 
     $\tilde{x}^{(k+1)} := \tilde{x}^{(k)} + \tilde{\alpha}_k\tilde{p}^{(k)}$ 
     $\tilde{r}^{(k+1)} := \tilde{r}^{(k)} - \tilde{\alpha}_k\tilde{q}^{(k)}$ 
     $x^{(k+1)} = L^{-T}\tilde{x}^{(k+1)}$ 
     $k := k + 1$ 
endwhile

```

Next, we notice that $x^{(k+1)} = L^{-T}\tilde{x}^{(k+1)}$ or, equivalently,

$$\tilde{x}^{(k)} = L^T x^{(k)}.$$

We substitute that

Given : $A, b, x^{(0)}$,
 $M = LL^T$
 $\tilde{b} = L^{-1}b$
 $L^T x^{(0)} = L^T x^{(0)}$
 $\tilde{r}^{(0)} := \tilde{b} - L^{-1}AL^{-T}L^T x^{(0)}$
 $k := 0$
while $\tilde{r}^{(k)} \neq 0$
 $\tilde{p}^{(k)} := \tilde{r}^{(k)}$
 $\tilde{q}^{(k)} := L^{-1}AL^{-T}\tilde{p}^{(k)}$
 $\tilde{\alpha}_k := \frac{\tilde{p}^{(k)T}\tilde{r}^{(k)}}{\tilde{p}^{(k)T}\tilde{q}^{(k)}}$
 $L^T x^{(k+1)} := L^T x^{(k)} + \tilde{\alpha}_k \tilde{p}^{(k)}$
 $\tilde{r}^{(k+1)} := \tilde{r}^{(k)} - \tilde{\alpha}_k \tilde{q}^{(k)}$
 $x^{(k+1)} = L^{-T}\tilde{x}^{(k+1)}$
 $k := k + 1$
endwhile

or, equivalently **Given :** $A, b, x^{(0)}$,
 $M = LL^T$
 $\tilde{b} = L^{-1}b$
 $\tilde{r}^{(0)} := \tilde{b} - L^{-1}Ax^{(0)}$
 $k := 0$
while $\tilde{r}^{(k)} \neq 0$
 $\tilde{p}^{(k)} := \tilde{r}^{(k)}$
 $\tilde{q}^{(k)} := L^{-1}AL^{-T}\tilde{p}^{(k)}$
 $\tilde{\alpha}_k := \frac{\tilde{p}^{(k)T}\tilde{r}^{(k)}}{\tilde{p}^{(k)T}\tilde{q}^{(k)}}$
 $x^{(k+1)} := x^{(k)} + \tilde{\alpha}_k L^{-T}\tilde{p}^{(k)}$
 $\tilde{r}^{(k+1)} := \tilde{r}^{(k)} - \tilde{\alpha}_k \tilde{q}^{(k)}$
 $k := k + 1$
endwhile

Now, we exploit that $\tilde{b} = L^{-1}b$ and $\tilde{r}^{(k)}$ equals the residual $\tilde{b} - \tilde{A}\tilde{x}^{(k)} = L^{-1}b - L^{-1}AL^{-T}L^T x^{(k)} = L^{-1}(b - Ax^{(k)}) = L^{-1}r^{(k)}$. Substituting these insights in gives us

Given : $A, b, x^{(0)}$,
 $M = LL^T$
 $L^{-1}b = L^{-1}b$
 $L^{-1}r^{(0)} := L^{-1}(b - Ax^{(0)})$
 $k := 0$
while $L^{-1}r^{(k)} \neq 0$
 $\tilde{p}^{(k)} := L^{-1}r^{(k)}$
 $\tilde{q}^{(k)} := L^{-1}AL^{-T}\tilde{p}^{(k)}$
 $\tilde{\alpha}_k := \frac{\tilde{p}^{(k)T}L^{-1}r^{(k)}}{\tilde{p}^{(k)T}\tilde{q}^{(k)}}$
 $x^{(k+1)} := x^{(k)} + \tilde{\alpha}_k L^{-T}\tilde{p}^{(k)}$
 $L^{-1}r^{(k+1)} := L^{-1}r^{(k)} - \tilde{\alpha}_k \tilde{q}^{(k)}$
 $k := k + 1$
endwhile

or, equivalently **Given :** $A, b, x^{(0)}$,
 $M = LL^T$
 $r^{(0)} := b - Ax^{(0)}$
 $k := 0$
while $r^{(k)} \neq 0$
 $\tilde{p}^{(k)} := L^{-1}r^{(k)}$
 $\tilde{q}^{(k)} := L^{-1}AL^{-T}\tilde{p}^{(k)}$
 $\tilde{\alpha}_k := \frac{\tilde{p}^{(k)T}L^{-1}r^{(k)}}{\tilde{p}^{(k)T}\tilde{q}^{(k)}}$
 $x^{(k+1)} := x^{(k)} + \tilde{\alpha}_k L^{-T}\tilde{p}^{(k)}$
 $r^{(k+1)} := r^{(k)} - \tilde{\alpha}_k \tilde{q}^{(k)}$
 $k := k + 1$
endwhile

Now choose $\tilde{p}^{(k)} = L^T p^{(k)}$ so that $AL^{-T}\tilde{p}^{(k)}$ becomes $Ap^{(k)}$:

Given : $A, b, x^{(0)},$ $M = LL^T$ $r^{(0)} := b - Ax^{(0)}$ $k := 0$ while $r^{(k)} \neq 0$ $p^{(k)} := L^{-T}L^{-1}r^{(k)}$ $\tilde{q}^{(k)} := L^{-1}Ap^{(k)}$ $\tilde{\alpha}_k := \frac{(L^T p^{(k)})^T L^{-1} r^{(k)}}{(L^T p^{(k)})^T \tilde{q}^{(k)}}$ $x^{(k+1)} := x^{(k)} + \tilde{\alpha}_k L^{-T} L^T p^{(k)}$ $r^{(k+1)} := r^{(k)} - \tilde{\alpha}_k L \tilde{q}^{(k)}$ $k := k + 1$ endwhile	or, equivalently Given : $A, b, x^{(0)},$ $M = LL^T$ $r^{(0)} := b - Ax^{(0)}$ $k := 0$ while $r^{(k)} \neq 0$ $p^{(k)} := M^{-1}r^{(k)}$ $\tilde{q}^{(k)} := L^{-1}Ap^{(k)}$ $\tilde{\alpha}_k := \frac{p^{(k) T} r^{(k)}}{p^{(k) T} L \tilde{q}^{(k)}}$ $x^{(k+1)} := x^{(k)} + \tilde{\alpha}_k p^{(k)}$ $r^{(k+1)} := r^{(k)} - \tilde{\alpha}_k L \tilde{q}^{(k)}$ $k := k + 1$ endwhile
--	--

Finally, if we choose $L\tilde{q}^{(k)} = q^{(k)}$ and $\tilde{\alpha}_k = \alpha_k$ we end up with

Given : $A, b, x^{(0)},$ $M = LL^T$ $r^{(0)} := b - Ax^{(0)}$ $k := 0$ while $r^{(k)} \neq 0$ $p^{(k)} := M^{-1}r^{(k)}$ $q^{(k)} := Ap^{(k)}$ $\alpha_k := \frac{p^{(k) T} r^{(k)}}{p^{(k) T} q^{(k)}}$ $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ $r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$ $k := k + 1$ endwhile
--

8.3 The Conjugate Gradient Method

8.3.1 A-conjugate directions



YouTube: <https://www.youtube.com/watch?v=9-SyyJv0XuU>

Let's start our generic descent method algorithm with $x^{(0)} = 0$. Here we do not use the temporary vector $q^{(k)} = Ap^{(k)}$ so that later we can emphasize how to cast the Conjugate Gradient Method in terms of as few matrix-vector multiplication as possible (one to be exact).

Given : A, b $x^{(0)} := 0$ $r^{(0)} := b - Ax^{(0)} (= b)$ $k := 0$ while $r^{(k)} \neq 0$ $p^{(k)} :=$ next direction $\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$ $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$ $k := k + 1$ endwhile	Given : A, b $x := 0$ $r := b$ while $r \neq 0$ $p :=$ next direction $\alpha := \frac{p^T r}{p^T A p}$ $x := x + \alpha p$ $r := r - \alpha A p$ endwhile
--	---

Figure 8.3.1.1 Generic descent algorithm started with $x^{(0)} = 0$. Left: with indices. Right: without indices.

Now, since $x^{(0)} = 0$, clearly

$$x^{(k+1)} = \alpha_0 p^{(0)} + \cdots + \alpha_k p^{(k)}.$$

Thus, $x^{(k+1)} \in \text{Span}(p^{(0)}, \dots, p^{(k)})$.

It would be nice if after the k th iteration

$$f(x^{(k+1)}) = \min_{x \in \text{Span}(p^{(0)}, \dots, p^{(k)})} f(x) \quad (8.3.1)$$

and the search directions were linearly independent. Then, the resulting descent method, in exact arithmetic, is guaranteed to complete in at most n iterations. This is because then

$$\text{Span}(p^{(0)}, \dots, p^{(n-1)}) = \mathbb{R}^n$$

so that

$$f(x^{(n)}) = \min_{x \in \text{Span}(p^{(0)}, \dots, p^{(n-1)})} f(x) = \min_{x \in \mathbb{R}^n} f(x)$$

and hence $Ax^{(n)} = b$.

Unfortunately, the Method of Steepest Descent does not have this property. The next approximation to the solution, $x^{(k+1)}$ minimizes $f(x)$ where x is constrained to be on the line $x^{(k)} + \alpha p^{(k)}$. Because in each step $f(x^{(k+1)}) \leq f(x^{(k)})$, a slightly stronger result holds: It also minimizes $f(x)$ where x is constrained to be on the union of lines $x^{(j)} + \alpha p^{(j)}$, $j = 0, \dots, k$. However, unless we pick the search directions very carefully, that is not the same as it minimizing over all vectors in $\text{Span}(p^{(0)}, \dots, p^{(k)})$.



YouTube: <https://www.youtube.com/watch?v=j8uNP7zjdv8>

We can write (8.3.1) more concisely: Let

$$P^{(k-1)} = \begin{pmatrix} p^{(0)} & p^{(1)} & \cdots & p^{(k-1)} \end{pmatrix}$$

be the matrix that holds the history of all search directions so far (as its columns). Then, letting

$$a^{(k-1)} = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{k-1} \end{pmatrix},$$

we notice that

$$x^{(k)} = \begin{pmatrix} p^{(0)} & \cdots & p^{(k-1)} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{k-1} \end{pmatrix} = P^{(k-1)} a^{(k-1)}. \quad (8.3.2)$$

Homework 8.3.1.1 Let $p^{(k)}$ be a new search direction that is linearly independent of the columns of $P^{(k-1)}$, which themselves are linearly independent. Show that

$$\begin{aligned} \min_{x \in \text{Span}(p^{(0)}, \dots, p^{(k-1)}, p^{(k)})} f(x) &= \min_y f(P^{(k)}y) \\ &= \min_y \left[\frac{1}{2} y_0^T P^{(k-1)T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)T} b \right. \\ &\quad \left. + \psi_1 y_0^T P^{(k-1)T} A p^{(k)} + \frac{1}{2} \psi_1^2 p^{(k)T} A p^{(k)} - \psi_1 p^{(k)T} b \right], \end{aligned}$$

where $y = \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \in \mathbb{R}^{k+1}$.

Hint.

$$x \in \text{Span}(p^{(0)}, \dots, p^{(k-1)}, p^{(k)})$$

if and only if there exists

$$y = \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \in \mathbb{R}^{k+1} \text{ such that } x = \begin{pmatrix} P^{(k-1)} & | & p^{(k)} \end{pmatrix} \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}.$$

Solution.

$$\begin{aligned}
 & \min_{x \in \text{Span}(p^{(0)}, \dots, p^{(k-1)}, p^{(k)})} f(x) \\
 &= \quad < \text{equivalent formulation} > \\
 & \min_y f(\left(\begin{array}{|c} P^{(k-1)} | p^{(k)} \end{array} \right) y) \\
 &= \quad < \text{partition } y = \left(\frac{y_0}{\psi_1} \right) > \\
 & \min_y f(\left(\begin{array}{|c} P^{(k-1)} | p^{(k)} \end{array} \right) \left(\frac{y_0}{\psi_1} \right)) \\
 &= \quad < \text{instantiate } f > \\
 & \min_y \left[\frac{1}{2} \left[\left(\begin{array}{|c} P^{(k-1)} | p^{(k)} \end{array} \right) \left(\frac{y_0}{\psi_1} \right) \right]^T A \left(\begin{array}{|c} P^{(k-1)} | p^{(k)} \end{array} \right) \left(\frac{y_0}{\psi_1} \right) \right. \\
 & \quad \left. - \left[\left(\begin{array}{|c} P^{(k-1)} | p^{(k)} \end{array} \right) \left(\frac{y_0}{\psi_1} \right) \right]^T b \right] . \\
 &= \quad < \text{multiply out} > \\
 & \min_y \left[\frac{1}{2} \left[y_0^T P^{(k-1)T} + \psi_1 p^{(k)T} \right] A \left[P^{(k-1)} y_0 + \psi_1 p^{(k)} \right] - y_0^T P^{(k-1)T} b - \psi_1 p^{(k)T} b \right] \\
 &= \quad < \text{multiply out some more} > \\
 & \min_y \left[\frac{1}{2} y_0^T P^{(k-1)T} A P^{(k-1)} y_0 + \psi_1 y_0^T P^{(k-1)T} A p^{(k)} \right. \\
 & \quad \left. + \frac{1}{2} \psi_1^2 p^{(k)T} A p^{(k)} - y_0^T P^{(k-1)T} b - \psi_1 p^{(k)T} b \right] \\
 &= \quad < \text{rearrange} > \\
 & \min_y \left[\frac{1}{2} y_0^T P^{(k-1)T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)T} b + \psi_1 y_0^T P^{(k-1)T} A p^{(k)} \right. \\
 & \quad \left. + \frac{1}{2} \psi_1^2 p^{(k)T} A p^{(k)} - \psi_1 p^{(k)T} b \right] .
 \end{aligned}$$



YouTube: <https://www.youtube.com/watch?v=5eNmr776GJY>

Now, if

$$P^{(k-1)T} A p^{(k)} = 0$$

then

$$\begin{aligned}
 & \min_{x \in \text{Span}(p^{(0)}, \dots, p^{(k-1)}, p^{(k)})} f(x) \\
 &= < \text{from before} > \\
 & \min_y \left[\frac{1}{2} y_0^T P^{(k-1)T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)T} b \right. \\
 & \quad \left. + \underbrace{\psi_1 y_0^T P^{(k-1)T} A p^{(k)}}_0 + \frac{1}{2} \psi_1^2 p^{(k)T} A p^{(k)} - \psi_1 p^{(k)T} b \right] \\
 &= < \text{remove zero term} > \\
 & \min_y \left[\frac{1}{2} y_0^T P^{(k-1)T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)T} b \right. \\
 & \quad \left. + \frac{1}{2} \psi_1^2 p^{(k)T} A p^{(k)} - \psi_1 p^{(k)T} b \right] \\
 &= < \text{split into two terms that can be minimized separately} > \\
 & \min_{y_0} \left[\frac{1}{2} y_0^T P^{(k-1)T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)T} b \right] + \min_{\psi_1} \left[\frac{1}{2} \psi_1^2 p^{(k)T} A p^{(k)} - \psi_1 p^{(k)T} b \right] \\
 &= < \text{recognize first set of terms as } f(P^{(k-1)} y_0) > \\
 & \min_{x \in \text{Span}(p^{(0)}, \dots, p^{(k-1)})} f(x) + \min_{\psi_1} \left[\frac{1}{2} \psi_1^2 p^{(k)T} A p^{(k)} - \psi_1 p^{(k)T} b \right].
 \end{aligned}$$

The minimizing ψ_1 is given by

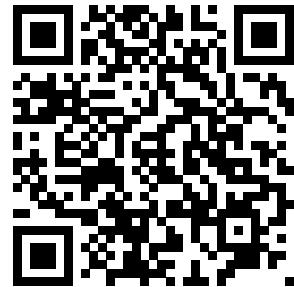
$$\psi_1 = \frac{p^{(k)T} b}{p^{(k)T} A p^{(k)}}.$$

If we pick $p^{(k)} = p^{(k)}$ and $\alpha_k = \psi_1$ then

$$x^{(k+1)} = P^{(k-1)} y_0 + \psi_1 p^{(k)} = \alpha_0 p^{(0)} + \dots + \alpha_{k-1} p^{(k-1)} + \alpha_k p^{(k)} = x^{(k)} + \alpha_k p^{(k)}.$$

A sequence of such directions is said to be A-conjugate.

Definition 8.3.1.2 A-conjugate directions. Let A be SPD. A sequence $p^{(0)}, \dots, p^{(k-1)} \in \mathbb{R}^n$ such that $p^{(j)T} A p^{(i)} = 0$ if and only if $j \neq i$ is said to be A-conjugate. \diamond



YouTube: <https://www.youtube.com/watch?v=70t6zgeMHs8>

Homework 8.3.1.2 Let $A \in \mathbb{R}^{n \times n}$ be SPD.

ALWAYS/SOMETIMES/NEVER: The columns of $P \in \mathbb{R}^{n \times k}$ are A-conjugate if and only if $P^T A P = D$ where D is diagonal and has positive values on its diagonal.

Answer. ALWAYS

Now prove it.

Solution.

$$\begin{aligned}
 P^T AP &= \quad < \text{partition } P \text{ by columns} > \\
 &\left(p_0 \mid \cdots \mid p_{k-1} \right)^T A \left(p_0 \mid \cdots \mid p_{k-1} \right) \\
 &= \quad < \text{transpose} > \\
 &\left(\begin{array}{c} p_0^T \\ \vdots \\ p_{k-1}^T \end{array} \right) A \left(p_0 \mid \cdots \mid p_{k-1} \right) \\
 &= \quad < \text{multiply out} > \\
 &\left(\begin{array}{c} p_0^T \\ \vdots \\ p_{k-1}^T \end{array} \right) \left(Ap_0 \mid \cdots \mid Ap_{k-1} \right) \\
 &= \quad < \text{multiply out} > \\
 &\left(\begin{array}{c|c|c|c} p_0^T Ap_0 & p_0^T Ap^{(k)} & \cdots & p_0^T Ap_{k-1} \\ \hline p^{(k)T} Ap_0 & p^{(k)T} Ap^{(k)} & \cdots & p^{(k)T} Ap_{k-1} \\ \hline \vdots & & \vdots & \\ \hline p_{k-1}^T Ap_0 & p_{k-1}^T Ap^{(k)} & \cdots & p_{k-1}^T Ap_{k-1} \end{array} \right) \\
 &= \quad < \text{multiply out} > \\
 &\left(\begin{array}{c|c|c|c} p_0^T Ap_0 & 0 & \cdots & 0 \\ \hline 0 & p^{(k)T} Ap^{(k)} & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & p_{k-1}^T Ap_{k-1} \end{array} \right),
 \end{aligned}$$

which is a diagonal matrix and its diagonal elements are positive since $\$ A \$$ is SPD.

Homework 8.3.1.3 Let $A \in \mathbb{R}^{n \times n}$ be SPD and the columns of $P \in \mathbb{R}^{n \times k}$ be A-conjugate.

ALWAYS/SOMETIMES/NEVER: The columns of P are linearly independent.

Answer. ALWAYS

Now prove it!

Solution. We employ a proof by contradiction. Suppose the columns of P are not linearly independent. Then there exists $y \neq 0$ such that $Py = 0$. Let $D = P^T AP$. From the last homework we know that D is diagonal and has positive diagonal elements. But then

$$\begin{aligned}
 0 &= \quad < Py = 0 > \\
 (Py)^T A (Py) &= \quad < \text{multiply out} > \\
 y^T P^T A P y &= \quad < P^T A P = D > \\
 y^T D y &> \quad < D \text{ is SPD} > \\
 0,
 \end{aligned}$$

which is a contradiction. Hence, the columns of P are linearly independent.

The above observations leaves us with a descent method that picks the search directions to be A-conjugate, given in [Figure 8.3.1.3](#).

```

Given :  $A, b$ 
 $x^{(0)} := 0$ 
 $r^{(0)} = b$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
    Choose  $p^{(k)}$  such that  $p^{(k)T} AP^{(k-1)} = 0$  and  $p^{(k)T} r^{(k)} \neq 0$ 
     $\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} Ap^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$ 
     $k := k + 1$ 
endwhile

```

Figure 8.3.1.3 Basic method that chooses the search directions to be A-conjugate.

Remark 8.3.1.4 The important observation is that if $p^{(0)}, \dots, p^{(k)}$ are chosen to be A-conjugate, then $x^{(k+1)}$ minimizes not only

$$f(x^{(k)} + \alpha p^{(k)})$$

but also

$$\min_{x \in \text{Span}(p^{(0)}, \dots, p^{(k-1)})} f(x).$$

8.3.2 Existence of A-conjugate search directions



YouTube: <https://www.youtube.com/watch?v=yXfR71mJ64w>

The big question left dangling at the end of the last unit was whether there exists a direction $p^{(k)}$ that is A-orthogonal to all previous search directions and that is not orthogonal to $r^{(k)}$. Let us examine this:

- Assume that all prior search directions $p^{(0)}, \dots, p^{(k-1)}$ were A-conjugate.
- Consider all vectors $p \in \mathbb{R}^n$ that are A-conjugate to $p^{(0)}, \dots, p^{(k-1)}$. A vector p has this property if and only if $p \perp \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})$.

- For $p \perp \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})$ we notice that

$$p^T r^{(k)} = p^T(b - Ax^{(k)}) = p^T(b - AP^{(k-1)}a^{(k-1)})$$

where we recall from (8.3.2) that

$$P^{(k-1)} = \begin{pmatrix} p^{(0)} & \cdots & p^{(k-1)} \end{pmatrix} \text{ and } a^{(k-1)} = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{k-1} \end{pmatrix}.$$

- If all vectors p that are A-conjugate to $p^{(0)}, \dots, p^{(k-1)}$ are orthogonal to the current residual, $p^T r^{(k)} = 0$ for all p with $P^{(k-1)T}Ap = 0$, then

$$0 = p^T b - p^T AP^{(k-1)}a^{(k-1)} = p^T b \text{ for all } p \perp \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)}).$$

Let's think about this: b is orthogonal to all vectors that are orthogonal to $\text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})$. This means that

$$b \in \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)}).$$

- Hence $b = AP^{(k-1)}z$ for some $z \in \mathbb{R}^k$. It also means that $x = P^{(k-1)}z$ solves $Ax = b$.
- We conclude that our method must already have found the solution since $x^{(k)}$ minimizes $f(x)$ over all vectors in $\text{Span}(p^{(0)}, \dots, p^{(k-1)})$. Thus $Ax^{(k)} = b$ and $r^{(k)} = 0$.

We conclude that there exist descent methods that leverage A-conjugate search directions as described in Figure 8.3.1.3. The question now is how to find a new A-conjugate search direction at every step.

8.3.3 Conjugate Gradient Method Basics



YouTube: <https://www.youtube.com/watch?v=OWnTq1PIFnQ>

The idea behind the Conjugate Gradient Method is that in the current iteration we have an approximation, $x^{(k)}$ to the solution to $Ax = b$. By construction, since $x^{(0)} = 0$,

$$x^{(k)} = \alpha_0 p^{(0)} + \cdots + \alpha_{k-1} p^{(k-1)}.$$

Also, the residual

$$\begin{aligned}
 r^{(k)} &= \\
 &= b - Ax^{(k)} \\
 &= \\
 &= b - A(\alpha_0 p^{(0)} + \cdots + \alpha_{k-1} p^{(k-1)}) \\
 &= \\
 &= b - \alpha_0 Ap^{(0)} - \cdots - \alpha_{k-1} Ap^{(k-1)} \\
 &= \\
 r^{(k-1)} &- \alpha_{k-1} Ap^{(k-1)}.
 \end{aligned}$$

If $r^{(k)} = 0$, then we know that $x^{(k)}$ solves $Ax = b$, and we are done.

Assume that $r^{(k)} \neq 0$. The question now is "How should we construct a new $p^{(k)}$ that is A-conjugate to the previous search directions and so that $p^{(k)T}r^{(k)} \neq 0?$ " Here are some thoughts:

- We like the direction of steepest descent, $r^{(k)} = b - Ax^{(k)}$, because it is the direction in which $f(x)$ decreases most quickly.
- Let us chose $p^{(k)}$ to be the vector that is A-conjugate to $p^{(0)}, \dots, p^{(k-1)}$ and closest to the direction of steepest descent, $r^{(k)}$:

$$\|p^{(k)} - r^{(k)}\|_2 = \min_{p \perp \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})} \|r^{(k)} - p\|_2.$$

This yields the algorithm in [Figure 8.3.3.1](#).

```

Given :  $A, b$ 
 $x^{(0)} := 0$ 
 $r^{(0)} := b$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
    if  $k = 0$ 
         $p^{(k)} = r^{(0)}$ 
    else
         $p^{(k)}$  minimizes  $\min_{p \perp \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})} \|r^{(k)} - p\|_2$ 
    endif
     $\alpha_k := \frac{p^{(k)T}r^{(k)}}{p^{(k)T}Ap^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := r^{(k)} - \alpha_k Ap^{(k)}$ 
     $k := k + 1$ 
endwhile

```

[Figure 8.3.3.1](#) Basic Conjugate Gradient Method.

8.3.4 Technical details

This unit is probably the most technically difficult unit in the course. We give the details here for completeness, but you will likely live a happy and productive research life without worrying about them too much... The important part is the final observation: that the next search direction computed by the Conjugate Gradient Method is a linear combination of the current residual (the direction of steepest descent) and the last search direction.



YouTube: <https://www.youtube.com/watch?v=i5MoVhNsXYU>

Let's look more carefully at $p^{(k)}$ that satisfies

$$\|r^{(k)} - p^{(k)}\|_2 = \min_{p \perp \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})} \|r^{(k)} - p\|_2.$$

Notice that

$$r^{(k)} = v + p^{(k)}$$

where v is the orthogonal projection of $r^{(k)}$ onto $\text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})$

$$\|r^{(k)} - v\|_2 = \min_{w \in \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})} \|r^{(k)} - w\|_2$$

which can also be formulated as $v = AP^{(k-1)}z^{(k)}$, where

$$\|r^{(k)} - AP^{(k-1)}z^{(k)}\|_2 = \min_{z \in \mathbb{R}^k} \|r^{(k)} - AP^{(k-1)}z\|_2.$$

This can be recognized as a standard linear least squares problem. This allows us to make a few important observations:



YouTube: <https://www.youtube.com/watch?v=ye1FuJixbHQ>

Theorem 8.3.4.1 In Figure 8.3.3.1,

- $P^{(k-1)T}r^{(k)} = 0$.

- $\text{Span}(p^{(0)}, \dots, p^{(k-1)}) = \text{Span}(r^{(0)}, \dots, r^{(k-1)}) = \text{Span}(b, Ab, \dots, A^{k-1}b)$.

Proof.

- Proving that

$$P^{(k-1)T}r^{(k)} = 0.$$

starts by considering that

$$\begin{aligned} & f(P^{(k-1)}y) \\ &= \frac{1}{2}(P^{(k-1)}y)^T A(P^{(k-1)}y) - (P^{(k-1)}y)^T b \\ &= \frac{1}{2}y^T (P^{(k-1)T}AP^{(k-1)})y - y^T P^{(k-1)T}b \end{aligned}$$

is minimized by y_0 that satisfies

$$(P^{(k-1)T}AP^{(k-1)})y_0 = P^{(k-1)T}b.$$

Since $x^{(k)}$ minimizes

$$\min_{x \in \text{Span}(p^{(0)}, \dots, p^{(k-1)})} f(x)$$

we conclude that $x = P^{(k-1)}y_0$. But then

$$0 = P^{(k-1)T}b - (P^{(k-1)T}Ax^{(k)}) = P^{(k-1)T}(b - Ax^{(k)}) = P^{(k-1)T}r^{(k)}.$$

- Show that $\text{Span}(p^{(0)}, \dots, p^{(k-1)}) = \text{Span}(r^{(0)}, \dots, r^{(k-1)}) = \text{Span}(b, Ab, \dots, A^{k-1}b)$.

Proof by induction on k .

- Base case: $k = 1$.

The result clearly holds since $p^{(0)} = r^{(0)} = b$.

- Inductive Hypothesis: Assume the result holds for $n \leq k$.

Show that the result holds for $k = n + 1$.

■ If $k = n + 1$ then $r^{(k-1)} = r^{(n)} = r^{(n-1)} - \alpha_{n-1}Ap^{(n-1)}$. By I.H.

$$r^{(n-1)} \in \text{Span}(b, Ab, \dots, A^{n-1}b)$$

and

$$p^{(n-1)} \in \text{Span}(b, Ab, \dots, A^{n-1}b).$$

But then

$$Ap^{(n-1)} \in \text{Span}(Ab, A^2b, \dots, A^n b)$$

and hence

$$r^{(n)} \in \text{Span}(b, Ab, A^2b, \dots, A^n b).$$

■ $p^{(n)} = r^{(n)} - AP^{(n-1)}y_0$ and hence

$$p^{(n)} \in \text{Span}(b, Ab, A^2b, \dots, A^n b)$$

since

$$r^{(n)} \in \text{Span}(b, Ab, A^2b, \dots, A^n b)$$

and

$$AP^{n-1}y_0 \in \text{Span}(Ab, A^2b, \dots, A^n b).$$

- We complete the inductive step by noting that all three subspaces have the same dimension and hence must be the same subspace.
- By the Principle of Mathematical Induction, the result holds.

■

Definition 8.3.4.2 Krylov subspace. The subspace

$$\mathcal{K}_k(A, b) = \text{Span}(b, Ab, \dots, A^{k-1}b)$$

is known as the **order-k Krylov subspace**. ◇

The next technical detail regards the residuals that are computed by the Conjugate Gradient Method. They are mutually orthogonal, and hence we, once again, conclude that the method must compute the solution (in exact arithmetic) in at most n iterations. It will also play an important role in reducing the number of matrix-vector multiplications needed to implement the final version of the Conjugate Gradient Method.

Theorem 8.3.4.3 *The residual vectors $r^{(k)}$ are mutually orthogonal.*

Proof. In [Theorem 8.3.4.1](#) we established that

$$\text{Span}(p^{(0)}, \dots, p^{(j-1)}) = \text{Span}(r^{(0)}, \dots, r^{(j-1)})$$

and hence

$$\text{Span}(r^{(0)}, \dots, r^{(j-1)}) \subset \text{Span}(p^{(0)}, \dots, p^{(k-1)}) =$$

for $j < k$. Hence $r^{(j)} = P^{(k-1)}t^{(j)}$ for some vector $t^{(j)} \in \mathbb{R}^k$. Then

$$r^{(k)T}r^{(j)} = r^{(k)T}P^{(k-1)}t^{(j)} = 0.$$

Since this holds for all k and $j < k$, the desired result is established. ■

Next comes the most important result. We established that

$$p^{(k)} = r^{(k)} - AP^{(k-1)}z^{(k-1)} \tag{8.3.3}$$

where $z^{(k)}$ solves

$$\min_{z \in \mathbb{R}^k} \|r^{(k)} - AP^{(k-1)}z\|_2.$$

What we are going to show is that in fact the next search direction equals a linear combination of the current residual and the previous search direction.

Theorem 8.3.4.4 For $k \geq 1$, the search directions generated by the Conjugate Gradient Method satisfy

$$p^{(k)} = r^{(k)} + \gamma_k p^{(k-1)}$$

for some constant γ_k .

Proof. This proof has a lot of very technical details. No harm done if you only pay cursory attention to those details.

Partition $z^{(k-1)} = \begin{pmatrix} z_0 \\ \zeta_1 \end{pmatrix}$ and recall that $r^{(k)} = r^{(k-1)} - \gamma_{k-1} A p^{(k-1)}$ so that

$$\begin{aligned} p^{(k)} &= \underbrace{r^{(k)} - AP^{(k-1)}z^{(k-1)}}_{\text{by (8.3.3)}} \\ &= \underbrace{r^{(k)} - AP^{(k-1)}z^{(k-1)}}_{<z^{(k-1)}>} \\ &= r^{(k)} - AP^{(k-2)}z_0 + \zeta_1 Ap^{(k-1)} \\ &= r^{(k)} - \left(AP^{(k-2)}z_0 + \zeta_1(r^{(k)} - r^{(k-1)})/\alpha_{k-1} \right) \\ &= \underbrace{\left(1 - \frac{\zeta_1}{\alpha_{k-1}} \right) r^{(k)} + \underbrace{\left(\frac{\zeta_1}{\alpha_{k-1}} r^{(k-1)} - AP^{(k-2)}z_0 \right)}_{s^{(k)}}}_{<>} \\ &= \left(1 - \frac{\zeta_1}{\alpha_{k-1}} \right) r^{(k)} + s^{(k)}. \end{aligned}$$

We notice that $r^{(k)}$ and $s^{(k)}$ are orthogonal. Hence

$$\|p^{(k)}\|_2^2 = \left(1 + \frac{\zeta_1}{\alpha_{k-1}} \right) \|r^{(k)}\|_2^2 + \|s^{(k)}\|_2^2$$

and minimizing $p^{(k)}$ means minimizing the two separate parts. Since $r^{(k)}$ is fixed, this means minimizing $\|s^{(k)}\|_2^2$. An examination of $s^{(k)}$ exposes that

$$s^{(k)} = \frac{\zeta_1}{\alpha_{k-1}} r^{(k-1)} - AP^{(k-2)}z_0 = -\frac{\zeta_1}{\alpha_{k-1}} \left(r^{(k-1)} - AP^{(k-2)}w_0 \right)$$

where $w_0 = -(\alpha_{k-1}/\zeta_1)z_0$. We recall that

$$\|r^{(k-1)} - p^{(k-1)}\|_2 = \min_{p \perp \text{Span}(p^{(0)}, \dots, p^{(k-2)})} \|r^{(k-1)} - Ap\|_2$$

and hence we conclude that s_k is a vector the direction of $p^{(k-1)}$. Since we are only interested in the direction of $p^{(k)}$, $\frac{\zeta_1}{\alpha_{k-1}}$ is not relevant. The upshot of this lengthy analysis is that

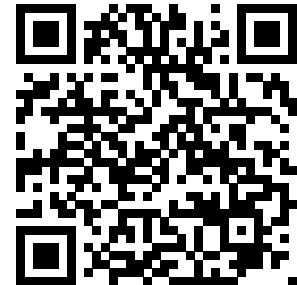
$$p^{(k)} = r^{(k)} + \gamma_k p^{(k-1)}.$$

■

This implies that while the Conjugate Gradient Method is an A-conjugate method and hence leverages a "memory" of all previous search directions,

$$f(x^{(k)}) = \min_{x \in \text{Span}(p^{(0)}, \dots, p^{(k-1)})} f(x),$$

only the last search direction is needed to compute the current one. This reduces the cost of computing the current search direction and means we don't have to store all previous ones.



YouTube: <https://www.youtube.com/watch?v=jHBK10QE01s>

Remark 8.3.4.5 This is a very, very, very big deal...

8.3.5 Practical Conjugate Gradient Method algorithm



YouTube: <https://www.youtube.com/watch?v=FVWgZKJQjz0>

We have noted that $p^{(k)} = r^{(k)} + \gamma_k p^{(k-1)}$. Since $p^{(k)}$ is A-conjugate to $p^{(k-1)}$ we find that

$$p^{(k-1)T} A p^{(k)} = p^{(k-1)T} A r^{(k)} + \gamma_k p^{(k-1)T} A p^{(k-1)}$$

so that

$$\gamma_k = -p^{(k-1)T} A r^{(k)} / p^{(k-1)T} A p^{(k-1)}.$$

This yields the first practical instantiation of the Conjugate Gradient method, given in Figure 8.3.5.1.

```

Given :  $A, b$ 
 $x^{(0)} := 0$ 
 $r^{(0)} := b$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
  if  $k = 0$ 
     $p^{(k)} = r^{(0)}$ 
  else
     $\gamma_k := -p^{(k-1)T} A r^{(k)} / p^{(k-1)T} A p^{(k-1)}$ 
     $p^{(k)} := r^{(k)} + \gamma_k p^{(k-1)}$ 
  endif
   $\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$ 
   $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
   $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$ 
   $k := k + 1$ 
endwhile

```

Figure 8.3.5.1 Conjugate Gradient Method.

Homework 8.3.5.1 In [Figure 8.3.5.1](#) we compute

$$\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}.$$

Show that an alternative formula for α_k is given by

$$\alpha_k := \frac{r^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}.$$

Hint. Use the fact that $p^{(k)} = r^{(k)} + \gamma_k p^{(k-1)}$ and the fact that $r^{(k)}$ is orthogonal to all previous search directions to show that $p^{(k)T} r^{(k)} = r^{(k)T} r^{(k)}$.

Solution. We need to show that $p^{(k)T} r^{(k)} = r^{(k)T} r^{(k)}$.

$$\begin{aligned}
& p^{(k)T} r^{(k)} \\
&= < r^{(k)} + \gamma_k p^{(k-1)} > \\
& (r^{(k)} + \gamma_k p^{(k-1)})^T r^{(k)} \\
&= < \text{distribute} > \\
& r^{(k)T} r^{(k)} + \gamma_k p^{(k-1)T} r^{(k)} \\
&= < p^{(k-1)T} r^{(k)} = 0 > \\
& r^{(k)T} r^{(k)}.
\end{aligned}$$

The last homework justifies the refined Conjugate Gradient Method in [Figure 8.3.5.2](#) (Left).

Given : A, b $x^{(0)} := 0$ $r^{(0)} := b$ $k := 0$ while $r^{(k)} \neq 0$ if $k = 0$ $p^{(k)} = r^{(0)}$ else $\gamma_k := -(p^{(k-1)T} A r^{(k)}) / (p^{(k-1)T} A p^{(k-1)})$ $p^{(k)} := r^{(k)} + \gamma_k p^{(k-1)}$ endif $\alpha_k := \frac{r^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$ $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$ $k := k + 1$ endwhile	Given : A, b $x^{(0)} := 0$ $r^{(0)} := b$ $k := 0$ while $r^{(k)} \neq 0$ if $k = 0$ $p^{(k)} = r^{(0)}$ else $\gamma_k := (r^{(k)T} r^{(k)}) / (r^{(k-1)T} r^{(k-1)})$ $p^{(k)} := r^{(k)} + \gamma_k p^{(k-1)}$ endif $\alpha_k := \frac{r^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$ $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$ $k := k + 1$ endwhile
--	---

Figure 8.3.5.2 Alternative Conjugate Gradient Method algorithms.

Homework 8.3.5.2 For the Conjugate Gradient Method discussed so far,

- Show that

$$r^{(k)T} r^{(k)} = -\alpha_{k-1} r^{(k)T} A p^{k-1}.$$

- Show that

$$p^{(k-1)T} A p^{(k-1)} = r^{(k-1)T} r^{(k-1)} / \alpha_{k-1}.$$

Hint. Recall that

$$r^{(k)} = r^{(k-1)} - \alpha_{k-1} A p^{(k-1)}. \quad (8.3.4)$$

and rewrite (8.3.4) as

$$A p^{(k-1)} = (r^{(k-1)} - r^{(k)}) / \alpha_{k-1}.$$

and recall that in the previous iteration

$$p^{(k-1)} = r^{(k-1)} - \gamma_{k-1} p^{(k-2)}.$$

Solution.

$$r^{(k)T} r^{(k)} = r^{(k)T} r^{(k-1)} - \alpha_{k-1} r^{(k)T} A p^{k-1} = -\alpha_{k-1} r^{(k)T} A p^{k-1}.$$

$$\begin{aligned}
 & p^{(k-1)T} A p^{(k-1)} \\
 &= (r^{(k-1)} - \gamma_{k-1} p^{(k-2)})^T A p^{(k-1)} \\
 &= r^{(k-1)T} A p^{(k-1)} \\
 &= r^{(k-1)T} (r^{(k-1)} - r^{(k)}) / \alpha_{k-1} \\
 &= r^{(k-1)T} r^{(k-1)} / \alpha_{k-1}.
 \end{aligned}$$

From the last homework we conclude that

$$\gamma_k = -(p^{(k-1)T} A r^{(k)}) / (p^{(k-1)T} A p^{(k-1)}) = r^{(k)T} r^{(k)} / r^{(k-1)T} r^{(k-1)}.$$

This is summarized in on the right in [Figure 8.3.5.2](#).

8.3.6 Final touches for the Conjugate Gradient Method



YouTube: <https://www.youtube.com/watch?v=f3rLky6mA4>

We finish our discussion of the Conjugate Gradient Method by revisiting the stopping criteria and preconditioning.

8.3.6.1 Stopping criteria

In theory, the Conjugate Gradient Method requires at most n iterations to achieve the condition where the residual is zero so that $x^{(k)}$ equals the exact solution. In practice, it is an iterative method due to the error introduced by floating point arithmetic. For this reason, the iteration proceeds while $\|r^{(k)}\|_2 \geq \epsilon_{\text{mach}} \|b\|_2$ and some maximum number of iterations is not yet performed.

8.3.6.2 Preconditioning

In [Subsection 8.2.5](#) we noted that the method of steepest Descent can be greatly accelerated by employing a preconditioner. The Conjugate Gradient Method can be greatly accelerated. While in theory the method requires at most n iterations when A is $n \times n$, in practice a preconditioned Conjugate Gradient Method requires very few iterations.

Homework 8.3.6.1 Add preconditioning to the algorithm in [Figure 8.3.5.2](#) (right).

Solution. To add preconditioning to

$$Ax = b$$

we pick a SPD preconditioner $M = \tilde{L}\tilde{L}^T$ and instead solve the equivalent problem

$$\underbrace{\tilde{L}^{-1}A\tilde{L}^{-T}}_{\tilde{A}} \underbrace{\tilde{L}^T x}_{\tilde{x}} = \underbrace{\tilde{L}^{-1}b}_{\tilde{b}}$$

This changes the algorithm in [Figure 8.3.5.2](#) (right) to

```

Given :  $A, b, M = \tilde{L}\tilde{L}^T$ 
 $\tilde{x}^{(0)} := 0$ 
 $\tilde{A} = \tilde{L}^{-1}A\tilde{L}^{-T}$ 
 $\tilde{r}^{(0)} := \tilde{L}^{-1}b$ 
 $k := 0$ 
while  $\tilde{r}^{(k)} \neq 0$ 
  if  $k = 0$ 
     $\tilde{p}^{(k)} = \tilde{r}^{(0)}$ 
  else
     $\tilde{\gamma}_k := (\tilde{r}^{(k)T}\tilde{r}^{(k)}) / (\tilde{r}^{(k-1)T}\tilde{r}^{(k-1)})$ 
     $\tilde{p}^{(k)} := \tilde{r}^{(k)} + \tilde{\gamma}_k \tilde{p}^{(k-1)}$ 
  endif
   $\tilde{\alpha}_k := \frac{\tilde{r}^{(k)T}\tilde{r}^{(k)}}{\tilde{p}^{(k)T}\tilde{A}\tilde{p}^{(k)}}$ 
   $\tilde{x}^{(k+1)} := \tilde{x}^{(k)} + \tilde{\alpha}_k \tilde{p}^{(k)}$ 
   $\tilde{r}^{(k+1)} := \tilde{r}^{(k)} - \tilde{\alpha}_k \tilde{A}\tilde{p}^{(k)}$ 
   $k := k + 1$ 
endwhile

```

Now, much like we did in the constructive solution to [Homework 8.2.5.1](#) we now morph this into an algorithm that more directly computes $x^{(k+1)}$. We start by substituting

$$\tilde{A} = \tilde{L}^{-1}A\tilde{L}^{-T}, \tilde{x}^{(k)} = \tilde{L}^T x^{(k)}, \tilde{r}^{(k)} = \tilde{L}^{-1}r^{(k)}, \tilde{p}^{(k)} = \tilde{L}^T p^{(k)},$$

which yields

```

Given :  $A, b, M = \tilde{L}\tilde{L}^T$ 
 $\tilde{L}^T x^{(0)} := 0$ 
 $\tilde{L}^{-1}r^{(0)} := \tilde{L}^{-1}b$ 
 $k := 0$ 
while  $\tilde{L}^{-1}r^{(k)} \neq 0$ 
  if  $k = 0$ 
     $\tilde{L}^T p^{(k)} = \tilde{L}^{-1}r^{(0)}$ 
  else
     $\tilde{\gamma}_k := ((\tilde{L}^{-1}r^{(k)})^T \tilde{L}^{-1}r^{(k)}) / (\tilde{L}^{-1}r^{(k-1)})^T \tilde{L}^{-1}r^{(k-1)}$ 
     $\tilde{L}^T p^{(k)} := \tilde{L}^{-1}r^{(k)} + \tilde{\gamma}_k \tilde{L}^T p^{(k-1)}$ 
  endif
   $\tilde{\alpha}_k := \frac{(\tilde{L}^{-1}r^{(k)})^T \tilde{L}^{-1}r^{(k)}}{((\tilde{L}^T p^{(k)})^T \tilde{L}^{-1}A\tilde{L}^{-T}\tilde{L}^T p^{(k)})}$ 
   $\tilde{L}^T x^{(k+1)} := \tilde{L}^T x^{(k)} + \tilde{\alpha}_k \tilde{L}^T p^{(k)}$ 
   $\tilde{L}^{-1}r^{(k+1)} := \tilde{L}^{-1}r^{(k)} - \tilde{\alpha}_k \tilde{L}^{-1}\tilde{L}^{-1}A\tilde{L}^{-T}\tilde{L}^{-T}\tilde{L}^T p^{(k)}$ 
   $k := k + 1$ 
endwhile

```

If we now simplify and manipulate various parts of this algorithm we get

```

Given :  $A, b, M = \tilde{L}\tilde{L}^T$ 
 $x^{(0)} := 0$ 
 $r^{(0)} := b$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
  if  $k = 0$ 
     $p^{(k)} = M^{-1}r^{(0)}$ 
  else
     $\tilde{\gamma}_k := (r^{(k)T} M^{-1} r^{(k)}) / (r^{(k-1)T} M^{-1} r^{(k-1)})$ 
     $p^{(k)} := M^{-1}r^{(k)} + \tilde{\gamma}_k p^{(k-1)}$ 
  endif
   $\tilde{\alpha}_k := \frac{r^{(k)T} M^{-1} r^{(k)}}{p^{(k)T} A p^{(k)}}$ 
   $x^{(k+1)} := x^{(k)} + \tilde{\alpha}_k p^{(k)}$ 
   $r^{(k+1)} := r^{(k)} - \tilde{\alpha}_k A p^{(k)}$ 
   $k := k + 1$ 
endwhile

```

Finally, we avoid the recomputing of $M^{-1}r^{(k)}$ and $Ap^{(k)}$ by introducing $z^{(k)}$ and $q^{(k)}$:

```

Given :  $A, b, M = \tilde{L}\tilde{L}^T$ 
 $x^{(0)} := 0$ 
 $r^{(0)} := b$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
     $z^{(k)} := M^{-1}r^{(k)}$ 
    if  $k = 0$ 
         $p^{(k)} = z^{(0)}$ 
    else
         $\tilde{\gamma}_k := (r^{(k)T}z^{(k)}) / (r^{(k-1)T}z^{(k-1)})$ 
         $p^{(k)} := z^{(k)} + \tilde{\gamma}_k p^{(k-1)}$ 
    endif
     $q^{(k)} := Ap^{(k)}$ 
     $\tilde{\alpha}_k := \frac{r^{(k)T}z^{(k)}}{p^{(k)T}q^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \tilde{\alpha}_k p^{(k)}$ 
     $r^{(k+1)} := r^{(k)} - \tilde{\alpha}_k q^{(k)}$ 
     $k := k + 1$ 
endwhile

```

(Obviously, there are a few other things that can be done to avoid unnecessary recomputations of $r^{(k)T}z^{(k)}$.)

8.4 Enrichments

8.4.1 Conjugate Gradient Method: Variations on a theme

Many variations on the Conjugate Gradient Method exist, which are employed in different situations. A concise summary of these, including suggestions as to which one to use when, can be found in

- [2] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM Press, 1993. [[PDF](#)]

8.5 Wrap Up

8.5.1 Additional homework

Homework 8.5.1.1 When using iterative methods, the matrices are typically very sparse. The question then is how to store a sparse matrix and how to perform a matrix-vector multiplication with it. One popular way is known as **compressed row storage** that involves three arrays:

- 1D array `nzA` (nonzero A) which stores the nonzero elements of matrix A . In this array, first all nonzero elements of the first row are stored, then the second row, etc. It has size `nnzeroes` (number of nonzeros).
- 1D array `ir` which is an integer array of size $n + 1$ such that `ir(1)` equals the index in array `nzA` where the first element of the first row is stored. `ir(2)` then gives the index where the first element of the second row is stored, and so forth. `ir(n+1)` equals `nnzeroes + 1`. Having this entry is convenient when you implement a matrix-vector multiplication with array `nzA`.
- 1D array `ic` of size `nnzeroes` which holds the column indices of the corresponding elements in array `nzA`.

1. Write a function

```
[ nzA, ir, ic ] = Create_Poisson_problem_nzA( N )
```

that creates the matrix A in this sparse format.

2. Write a function

```
y = SparseMvMult( nzA, ir, ic, x )
```

that computes $y = Ax$ with the matrix A stored in the sparse format.

8.5.2 Summary

Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, its **gradient** is given by

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_0}(x) \\ \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_{n-1}}(x) \end{pmatrix}.$$

$\nabla f(x)$ equals the direction in which the function f increases most rapidly at the point x and $-\nabla f(x)$ equals the direction of steepest descent (the direction in which the function f decreases most rapidly at the point x).

In this summary, we will assume that $A \in \mathbb{R}^{n \times n}$ is symmetric positive definite (SPD) and

$$f(x) = \frac{1}{2}x^T Ax - x^T b.$$

The gradient of this function equals

$$\nabla f(x) = Ax - b$$

and \hat{x} minimizes the function if and only if

$$A\hat{x} = b.$$

If $x^{(k)}$ is an approximation to \hat{x} then $r^{(k)} = b - Ax^{(k)}$ equals the corresponding residual. Notice that $r^{(k)} = -\nabla f(x^{(k)})$ and hence $r^{(k)}$ is the direction of steepest descent .

A prototypical descent method is given by

```

Given :  $A, b, x^{(0)}$ 
 $r^{(0)} := b - Ax^{(0)}$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
     $p^{(k)} :=$  next direction
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$  for some scalar  $\alpha_k$ 
     $r^{(k+1)} := b - Ax^{(k+1)}$ 
     $k := k + 1$ 
endwhile
```

Here $p^{(k)}$ is the "current" search direction and in each iteration we create the next approximation to \hat{x} , $x^{(k+1)}$, along the line $x^{(k)} + \alpha p^{(k)}$.

If $x^{(k+1)}$ minimizes along that line, the method is an exact descent method and

$$\alpha_k = \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$$

so that a prototypical exact descent method is given by

```

Given :  $A, b, x^{(0)}$ 
 $r^{(0)} := b - Ax^{(0)}$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
     $p^{(k)} :=$  next direction
     $\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := b - Ax^{(k+1)}$ 
     $k := k + 1$ 
endwhile
```

Once α_k is determined,

$$r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}.$$

which saves a matrix-vector multiplication when incorporated into the prototypical exact descent method:

```
Given :  $A, b, x^{(0)}$ 
 $r^{(0)} := b - Ax^{(0)}$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
     $p^{(k)} :=$  next direction
     $q^{(k)} := Ap^{(k)}$ 
     $\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} q^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$ 
     $k := k + 1$ 
endwhile
```

The steepest descent algorithm chooses $p^{(k)} = -\nabla f(x^{(k)}) = b - Ax^{(k)} = r^{(k)}$:

```
Given :  $A, b, x^{(0)}$ 
 $r^{(0)} := b - Ax^{(0)}$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
     $p^{(k)} := r^{(k)}$ 
     $q^{(k)} := Ap^{(k)}$ 
     $\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} q^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$ 
     $k := k + 1$ 
endwhile
```

Convergence can be greatly accelerated by incorporating a preconditioner, M , where, ideally, $M \approx A$ is SPD and solving $Mz = y$ is easy (cheap).

```
Given :  $A, b, x^{(0)}, M$ 
 $r^{(0)} := b - Ax^{(0)}$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
     $p^{(k)} := M^{-1}r^{(k)}$ 
     $q^{(k)} := Ap^{(k)}$ 
     $\alpha_k := \frac{p^{(k)T} r^{(k)}}{p^{(k)T} q^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$ 
     $k := k + 1$ 
endwhile
```

Definition 8.5.2.1 A-conjugate directions. Let A be SPD. A sequence $p^{(0)}, \dots, p^{(k-1)} \in \mathbb{R}^n$ such that $p^{(j)T}Ap^{(i)} = 0$ if and only if $j \neq i$ is said to be A-conjugate. \diamond

The columns of $P \in \mathbb{R}^{n \times k}$ are A-conjugate if and only if $P^TAP = D$ where D is diagonal and has positive values on its diagonal.

A-conjugate vectors are linearly independent.

A descent method that chooses the search directions to be A-conjugate will find the solution of $Ax = b$, where $A \in \mathbb{R}^{n \times n}$ is SPD, in at most n iterations:

```

Given :  $A, b$ 
 $x^{(0)} := 0$ 
 $r^{(0)} = b$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
    Choose  $p^{(k)}$  such that  $p^{(k)T}AP^{(k-1)} = 0$  and  $p^{(k)T}r^{(k)} \neq 0$ 
     $\alpha_k := \frac{p^{(k)T}r^{(k)}}{p^{(k)T}Ap^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := r^{(k)} - \alpha_k Ap^{(k)}$ 
     $k := k + 1$ 
endwhile

```

The Conjugate Gradient Method chooses the search direction to equal the vector $p^{(k)}$ that is A-conjugate to all previous search directions and is closest to the direction of steepest descent:

```

Given :  $A, b$ 
 $x^{(0)} := 0$ 
 $r^{(0)} := b$ 
 $k := 0$ 
while  $r^{(k)} \neq 0$ 
    if  $k = 0$ 
         $p^{(k)} = r^{(0)}$ 
    else
         $p^{(k)}$  minimizes  $\min_{p \perp \text{Span}(Ap^{(0)}, \dots, Ap^{(k-1)})} \|r^{(k)} - p\|_2$ 
    endif
     $\alpha_k := \frac{p^{(k)T}r^{(k)}}{p^{(k)T}Ap^{(k)}}$ 
     $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ 
     $r^{(k+1)} := r^{(k)} - \alpha_k Ap^{(k)}$ 
endwhile

```

The various vectors that appear in the Conjugate Gradient Method have the following properties: If $P^{(p-1)} = \begin{pmatrix} p^{(0)} & \dots & p^{(k-1)} \end{pmatrix}$ then

- $P^{(k-1)T}r^{(k)} = 0$.
- $\text{Span}(p^{(0)}, \dots, p^{(k-1)}) = \text{Span}(r^{(0)}, \dots, r^{(k-1)}) = \text{Span}(b, Ab, \dots, A^{k-1}b)$.

- The residual vectors $r^{(k)}$ are mutually orthogonal.
- For $k \geq 1$

$$p^{(k)} = r^{(k)} - \gamma_k p^{(k-1)}$$

Definition 8.5.2.2 Krylov subspace. The subspace

$$\mathcal{K}_k(A, b) = \text{Span}(b, Ab, \dots, A^{k-1}b)$$

is known as the **order-k Krylov subspace**. ◊

Alternative Conjugate Gradient Methods are given by

Given : A, b $x^{(0)} := 0$ $r^{(0)} := b$ $k := 0$ while $r^{(k)} \neq 0$ if $k = 0$ $p^{(k)} = r^{(0)}$ else $\gamma_k := -(p^{(k-1)T} A r^{(k)}) / (p^{(k-1)T} A p^{(k-1)})$ $p^{(k)} := r^{(k)} + \gamma_k p^{(k-1)}$ endif $\alpha_k := \frac{r^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$ $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$ $k := k + 1$ endwhile	Given : A, b $x^{(0)} := 0$ $r^{(0)} := b$ $k := 0$ while $r^{(k)} \neq 0$ if $k = 0$ $p^{(k)} = r^{(0)}$ else $\gamma_k := (r^{(k)T} r^{(k)}) / (r^{(k-1)T} r^{(k-1)})$ $p^{(k)} := r^{(k)} + \gamma_k p^{(k-1)}$ endif $\alpha_k := \frac{r^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$ $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$ $k := k + 1$ endwhile
--	---

A practical stopping criteria for the Conjugate Gradient Method is to proceed while $\|r^{(k)}\|_2 \leq \epsilon_{\text{mach}} \|b\|_2$ and some maximum number of iterations is not yet performed.

The Conjugate Gradient Method can be accelerated by incorporating a preconditioned, M , where $M \approx A$ is SDP.

Part III

The Algebraic Eigenvalue Problem

Week 9

Eigenvalues and Eigenvectors

9.1 Opening

9.1.1 Relating diagonalization to eigenvalues and eigenvectors

You may want to start your exploration of eigenvalues and eigenvectors by watching the video

- [Eigenvectors and eigenvalues | Essence of linear algebra, chapter 14](#) from the 3Blue1Brown series. (We don't embed the video because we are not quite sure that the rules about doing so are.)

Here are the insights from that video in the terminology of this week.



YouTube: https://www.youtube.com/watch?v=S_0gLYAh2Jk

Homework 9.1.1.1 Eigenvalues and eigenvectors are all about finding scalars, λ , and nonzero vectors, x , such that

$$Ax = \lambda x.$$

To help you visualizing how a 2×2 real-valued matrix transforms a vector on the unit circle in general, and eigenvectors of unit length in particular, we have created the function [Assignments/Week09/matlab/showeig.m](#) (inspired by such a function that used to be part of Matlab). You may now want to do a "git pull" to update your local copy of the **Assignments** directory.

Once you have uploaded this function to Matlab, in the command window, first create a 2×2 matrix in array A and then execute `showeig(A)`.

Here are some matrices to try:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -0.5 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 1 \\ 0 & -0.5 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 1 \\ 1 & -0.5 \end{bmatrix}$$

```
theta = pi/4;
A = [ cos( theta) -sin( theta )
      sin( theta ) cos( theta ) ]
```

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 0.5 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 1.5 \\ 1 & -0.5 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & -1 \\ 1 & 0.5 \end{bmatrix}$$

Can you explain some of what you observe?

Solution.

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -0.5 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 1 \\ 0 & -0.5 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 1 \\ 1 & -0.5 \end{bmatrix}$$

If you try a few different symmetric matrices, you will notice that the eigenvectors are always mutually orthogonal.

```
theta = pi/4;
A = [ cos( theta) -sin( theta )
      sin( theta ) cos( theta ) ]
```

In the end, no vectors are displayed. This is because for real-valued vectors, there are no vectors such that the rotated vector is in the same direction as the original vector. The eigenvalues and eigenvectors of a real-valued rotation are complex-valued. Unless θ is an integer multiple of π .

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}$$

We will see later that this is an example of a Jordan block. There is only one linearly independent eigenvector associated with the eigenvalue 2. Notice that the two eigenvectors that are displayed are not linearly independent (they point in opposite directions).

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 0.5 \end{bmatrix}$$

This matrix has linearly dependent columns (it has a nonzero vector in the null space and hence 0 is an eigenvalue).

$$A = \begin{bmatrix} 2 & 1.5 \\ 1 & -0.5 \end{bmatrix}$$

If you look carefully, you notice that the eigenvectors are not mutually orthogonal.

$$A = \begin{bmatrix} 2 & -1 \\ 1 & 0.5 \end{bmatrix}$$

Another example of a matrix with complex-valued eigenvalues and eigenvectors.

9.1.2 Overview

- 9.1 Opening
 - 9.1.1 Relating diagonalization to eigenvalues and eigenvectors
 - 9.1.2 Overview
 - 9.1.3 What you will learn
- 9.2 Basics
 - 9.2.1 Singular matrices and the eigenvalue problem
 - 9.2.2 The characteristic polynomial
 - 9.2.3 More properties of eigenvalues and vectors
 - 9.2.4 The Schur and Spectral Decompositions
 - 9.2.5 Diagonalizing a matrix

- 9.2.6 Jordan Canonical Form
- 9.3 The Power Method and related approaches
 - 9.3.1 The Power Method
 - 9.3.2 The Power Method: Convergence
 - 9.3.3 The Inverse Power Method
 - 9.3.4 The Rayleigh Quotient Iteration
 - 9.3.5 Discussion
- 9.4 Enrichments
- 9.5 Wrap Up
 - 9.5.1 Additional homework
 - 9.5.2 Summary

9.1.3 What you will learn

This week, you are reintroduced to the theory of eigenvalues, eigenvectors, and diagonalization. Building on this, we start our discovery of practical algorithms.

Upon completion of this week, you should be able to

- Connect the algebraic eigenvalue problem to the various ways in which singular matrices can be characterized.
- Relate diagonalization of a matrix to the eigenvalue problem.
- Link the eigenvalue problem to the Schur and Spectral Decompositions of a matrix.
- Translate theoretical insights into a practical Power Method and related methods.
- Investigate the convergence properties of practical algorithms.

9.2 Basics

9.2.1 Singular matrices and the eigenvalue problem



YouTube: <https://www.youtube.com/watch?v=j85zII8u2-I>

Definition 9.2.1.1 Eigenvalue, eigenvector, and eigenpair. Let $A \in \mathbb{C}^{m \times m}$. Then $\lambda \in \mathbb{C}$ and nonzero $x \in \mathbb{C}^m$ are said to be an eigenvalue and corresponding eigenvector if $Ax = \lambda x$. The tuple (λ, x) is said to be an eigenpair. \diamond

$Ax = \lambda x$ means that the action of A on an eigenvector x is as if it were multiplied by a scalar. In other words, the direction does not change and only its length is scaled. "Scaling" and "direction" should be taken loosely here: an eigenvalue can be negative (in which case the vector ends up pointing in the opposite direction) or even complex-valued.

As part of an introductory course on linear algebra, you learned that the following statements regarding an $m \times m$ matrix A are all equivalent:

- A is nonsingular.
- A has linearly independent columns.
- There does not exist a nonzero vector x such that $Ax = 0$.
- $\mathcal{N}(A) = \{0\}$. (The null space of A is trivial.)
- $\dim(\mathcal{N}(A)) = 0$.
- $\det(A) \neq 0$.

Since $Ax = \lambda x$ can be rewritten as $(\lambda I - A)x = 0$, we note that the following statements are equivalent for a given $m \times m$ matrix A :

- There exists a vector $x \neq 0$ such that $(\lambda I - A)x = 0$.
- $(\lambda I - A)$ is singular.
- $(\lambda I - A)$ has linearly dependent columns.
- The null space of $\lambda I - A$ is nontrivial.
- $\dim(\mathcal{N}(\lambda I - A)) > 0$.
- $\det(\lambda I - A) = 0$.

It will become important in our discussions to pick the right equivalent statement in a given situation.



YouTube: <https://www.youtube.com/watch?v=K-yDVqijSYw>

We will often talk about "the set of all eigenvalues." This set is called the **spectrum** of a matrix.

Definition 9.2.1.2 Spectrum of a matrix. The set of all eigenvalues of A is denoted by $\Lambda(A)$ and is called the spectrum of A . \diamond

The magnitude of the eigenvalue that is largest in magnitude is known as the spectral radius. The reason is that all eigenvalues lie in the circle in the complex plane, centered at the origin, with that radius.

Definition 9.2.1.3 Spectral radius. The spectral radius of A , $\rho(A)$, equals the absolute value of the eigenvalue with largest magnitude:

$$\rho(A) = \max_{\lambda \in \Lambda(A)} |\lambda|.$$

\diamond

In Subsection 7.3.3 we used the spectral radius to argue that the matrix that comes up when finding the solution to Poisson's equation is nonsingular. Key in that argument is a result known as the Gershgorin Disk Theorem.



YouTube: <https://www.youtube.com/watch?v=r1a9Q4E6hVI>

Theorem 9.2.1.4 Gershgorin Disk Theorem. Let $A \in \mathbb{C}^{m \times m}$,

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,m-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,m-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,m-1} \end{pmatrix},$$

$$\rho_i(A) = \sum_{j \neq i} |\alpha_{i,j}|,$$

and

$$R_i(A) = \{x \text{ s.t. } |x - \alpha_{i,i}| \leq \rho_i\}.$$

In other words, $\rho_i(A)$ equals the sum of the absolute values of the off diagonal elements of A in row i and $R_i(A)$ equals the set of all points in the complex plane that are within a distance ρ_i of diagonal element $\alpha_{i,i}$. Then

$$\Lambda(A) \subset \cup_i R_i(A).$$

In other words, the eigenvalues lie in the union of these disks.

Proof. Let $\lambda \in \Lambda(A)$. Then $(\lambda I - A)x = 0$ for some nonzero vector x . W.l.o.g. assume that index i has the property that $1 = \chi_i \geq |\chi_j|$ for $j \neq i$. Then

$$-\alpha_{i,0}\chi_0 - \cdots - \alpha_{i,i-1}\chi_{i-1} + (\lambda - \alpha_{i,i}) - \alpha_{i,i+1}\chi_{i+1} - \alpha_{i,m-1}\chi_{m-1} = 0$$

or, equivalently,

$$\lambda - \alpha_{i,i} = \alpha_{i,0}\chi_0 + \cdots + \alpha_{i,i-1}\chi_{i-1} + \alpha_{i,i+1}\chi_{i+1} + \alpha_{i,m-1}\chi_{m-1}.$$

Hence

$$\begin{aligned} & |\lambda - \alpha_{i,i}| \\ &= \\ & |\alpha_{i,0}\chi_0 + \cdots + \alpha_{i,i-1}\chi_{i-1} + \alpha_{i,i+1}\chi_{i+1} + \alpha_{i,m-1}\chi_{m-1}| \\ &\leq \\ & |\alpha_{i,0}\chi_0| + \cdots + |\alpha_{i,i-1}\chi_{i-1}| + |\alpha_{i,i+1}\chi_{i+1}| + |\alpha_{i,m-1}\chi_{m-1}| \\ &= \\ & \leq \\ & |\alpha_{i,0}||\chi_0| + \cdots + |\alpha_{i,i-1}||\chi_{i-1}| + |\alpha_{i,i+1}||\chi_{i+1}| + |\alpha_{i,m-1}||\chi_{m-1}| \\ &\leq \\ & |\alpha_{i,0}| + \cdots + |\alpha_{i,i-1}| + |\alpha_{i,i+1}| + |\alpha_{i,m-1}| \\ &\leq \\ & \rho_i(A). \end{aligned}$$



YouTube: <https://www.youtube.com/watch?v=19FXch2X7sQ>

It is important to note that it is not necessarily the case that each such disk has exactly one eigenvalue in it. There is, however, a slightly stronger result than [Theorem 9.2.1.4](#).

Corollary 9.2.1.5 *Let A and $R_i(A)$ be as defined in [Theorem 9.2.1.4](#). Let K and K^C be disjoint subsets of $\{0, \dots, m-1\}$ such that $K \cup K^C = \{0, \dots, m-1\}$. In other words, let K and K^C partition $\{0, \dots, m-1\}$. If*

$$(\cup_{k \in K} R_k(A)) \cap (\cup_{j \in K^C} R_j(A)) = \emptyset$$

then $\cup_{k \in K} R_k(A)$ contains exactly $|K|$ eigenvalues of A (multiplicity counted). In other words, if $\cup_{k \in K} R_k(A)$ does not intersect with any of the other disks, then it contains as many eigenvalues of A (multiplicity counted) as there are elements of K .

Proof. The proof splits $A = D + (A - D)$ where D equals the diagonal of A and considers $A_\omega = D + \omega(A - D)$, which varies continuously with ω . One can argue that the disks $R_i(A_0)$ start with only one eigenvalue each and only when they start intersecting can an eigenvalue "escape" the disk in which it started. We skip the details since we won't need this result in this course. ■

Through a few homeworks, let's review basic facts about eigenvalues and eigenvectors.

Homework 9.2.1.1 Let $A \in \mathbb{C}^{m \times m}$.

TRUE/FALSE: $0 \in \Lambda(A)$ if and only A is singular.

Answer. TRUE

Now prove it!

Solution.

- (\Rightarrow): Assume $0 \in \Lambda(A)$. Let x be an eigenvector associated with eigenvalue 0. Then $Ax = 0x = 0$. Hence there exists a nonzero vector x such that $Ax = 0$. This implies A is singular.
- (\Leftarrow): Assume A is singular. Then there exists $x \neq 0$ such that $Ax = 0$. Hence $Ax = 0x$ and 0 is an eigenvalue of A .

Homework 9.2.1.2 Let $A \in \mathbb{C}^{m \times m}$ be Hermitian.

ALWAYS/SOMETIMES/NEVER: All eigenvalues of A are real-valued.

Answer. ALWAYS

Now prove it!

Solution. Let (λ, x) be an eigenpair of A . Then

$$Ax = \lambda x$$

and hence

$$x^H Ax = \lambda x^H x.$$

If we now conjugate both sides we find that

$$\overline{x^H Ax} = \overline{\lambda x^H x}$$

which is equivalent to

$$(x^H Ax)^H = (\lambda x^H x)^H$$

which is equivalent to

$$x^H Ax = \overline{\lambda} x^H x$$

since A is Hermitian. We conclude that

$$\overline{\lambda} = \frac{x^H Ax}{x^H x} = \lambda$$

(since $x^H x \neq 0$).

Homework 9.2.1.3 Let $A \in \mathbb{C}^{m \times m}$ be Hermitian positive definite (HPD).

ALWAYS/SOMETIMES/NEVER: All eigenvalues of A are positive.

Answer. ALWAYS

Now prove it!

Solution. Let (λ, x) be an eigenpair of A . Then

$$Ax = \lambda x$$

and hence

$$x^H Ax = \lambda x^H x$$

and finally (since $x \neq 0$)

$$\lambda = \frac{x^H Ax}{x^H x}.$$

Since A is HPD, both $x^H Ax$ and $x^H x$ are positive, which means λ is positive.

The converse is also always true, but we are not ready to prove that yet.

Homework 9.2.1.4 Let $A \in \mathbb{C}^{m \times m}$ be Hermitian, (λ, x) and (μ, y) be eigenpairs associated with A , and $\lambda \neq \mu$.

ALWAYS/SOMETIMES/NEVER: $x^H y = 0$

Answer. ALWAYS

Now prove it!

Solution. Since

$$Ax = \lambda x \text{ and } Ay = \mu y$$

we know that

$$y^H Ax = \lambda y^H x \text{ and } x^H Ay = \mu x^H y$$

and hence (remembering that the eigenvalues are real-valued)

$$\lambda y^H x = y^H Ax = \overline{x^H Ay} = \mu \overline{x^H y} = \mu y^H x.$$

We can rewrite this as

$$(\lambda - \mu) y^H x = 0.$$

Since $\lambda \neq \mu$ this implies that $y^H x = 0$ and hence $x^H y = 0$.

Homework 9.2.1.5 Let $A \in \mathbb{C}^{m \times m}$, (λ, x) and (μ, y) be eigenpairs, and $\lambda \neq \mu$. Prove that x and y are linearly independent.

Solution. Proof by contradiction: Under the assumptions of the homework, we will show that assuming that x and y are linearly dependent leads to a contradiction.

If nonzero x and nonzero y are linearly dependent, then there exists $\gamma \neq 0$ such that $y = \gamma x$. Then

$$Ay = \mu y$$

implies that

$$A(\gamma x) = \mu(\gamma x)$$

and hence

$$\gamma\lambda x = \mu\gamma x.$$

Rewriting this we get that

$$(\lambda - \mu)\gamma x = 0.$$

Since $\lambda \neq \mu$ and $\gamma \neq 0$ this means that $x = 0$ which contradicts that x is an eigenvector.

We conclude that x and y are linearly independent.

We now generalize this insight.

Homework 9.2.1.6 Let $A \in \mathbb{C}^{m \times m}$, $k \leq m$, and (λ_i, x_i) for $1 \leq i < k$ be eigenpairs of this matrix. Prove that if $\lambda_i \neq \lambda_j$ when $i \neq j$ then the eigenvectors x_i are linearly independent. In other words, given a set of distinct eigenvalues, a set of vectors created by taking one eigenvector per eigenvalue is linearly independent.

Hint. Prove by induction.

Solution. Proof by induction on k .

- Base Case: $k = 1$. This is trivially.
- Assume the result holds for $1 \leq k < m$. Show it holds for $k + 1$.

The I.H. means that x_0, \dots, x_{k-1} are linearly independent. We need to show that x_k is not a linear combination of x_0, \dots, x_{k-1} . We will do so via a proof by contradiction.

Assume x_k is a linear combination of x_0, \dots, x_{k-1} so that

$$x_k = \gamma_0 x_0 + \cdots + \gamma_{k-1} x_{k-1}$$

with at least one $\gamma_j \neq 0$. We know that $Ax_k = \lambda_k x_k$ and hence

$$A(\gamma_0 x_0 + \cdots + \gamma_{k-1} x_{k-1}) = \lambda_k (\gamma_0 x_0 + \cdots + \gamma_{k-1} x_{k-1}).$$

Since $Ax_i = \lambda_i x_i$, we conclude that

$$\gamma_0 \lambda_0 x_0 + \cdots + \gamma_{k-1} \lambda_{k-1} x_{k-1} = \gamma_0 \lambda_k x_0 + \cdots + \gamma_{k-1} \lambda_k x_{k-1}$$

or, equivalently,

$$\gamma_0 (\lambda_0 - \lambda_k) x_0 + \cdots + \gamma_{k-1} (\lambda_{k-1} - \lambda_k) x_{k-1} = 0.$$

Since at least one $\gamma_i \neq 0$ and $\lambda_i \neq \lambda_k$ for $0 \leq i < k$, we conclude that x_0, \dots, x_{k-1} are linearly dependent, which is a contradiction.

Hence, x_0, \dots, x_k are linearly independent.

- By the Principle of Mathematical Induction, the result holds for $1 \leq k \leq m$.

We now return to some of the matrices we saw in Week 7.

Homework 9.2.1.7 Consider the matrices

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

and

$$\left(\begin{array}{ccc|ccc|c} 4 & -1 & & -1 & & & \\ -1 & 4 & -1 & & -1 & & \\ & -1 & 4 & -1 & & -1 & \\ & & -1 & 4 & & & -1 \\ \hline -1 & & & 4 & -1 & & -1 \\ & -1 & & -1 & 4 & -1 & \dots \\ & & -1 & & -1 & 4 & \\ \hline & & & -1 & & 4 & \dots \end{array} \right)$$

ALWAYS/SOMETIMES/NEVER: All eigenvalues of these matrices are nonnegative.

ALWAYS/SOMETIMES/NEVER: All eigenvalues of the first matrix are positive.

Answer. ALWAYS: All eigenvalues of these matrices are nonnegative.

ALWAYS: All eigenvalues of the first matrix are positive. (So are all the eigenvalues of the second matrix, but proving that is a bit trickier.)

Now prove it!

Solution. For the first matrix, we can use the Gershgorin disk theorem to conclude that all eigenvalues of the matrix lie in the set $\{x \text{ s.t. } |x - 2| \leq 2\}$. We also notice that the matrix is symmetric, which means that its eigenvalues are real-valued. Hence the eigenvalues are nonnegative. A similar argument can be used for the second matrix.

Now, in [Homework 7.2.1.1](#) we showed that the first matrix is nonsingular. Hence, it cannot have an eigenvalue equal to zero. We conclude that its eigenvalues are all positive.

It can be shown that the second matrix is also nonsingular, and hence has positive eigenvalues. However, that is a bit nasty to prove...

9.2.2 The characteristic polynomial



YouTube: <https://www.youtube.com/watch?v=NUvfjg-JUjg>

We start by discussing how to further characterize eigenvalues of a given matrix A . We say "characterize" because none of the discussed insights lead to practical algorithms for computing them, at least for matrices larger than 4×4 .

Homework 9.2.2.1 Let

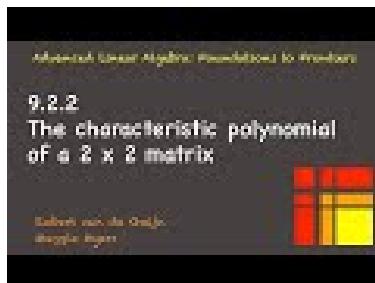
$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix}$$

be a nonsingular matrix. Show that

$$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix}^{-1} = \frac{1}{\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}} \begin{pmatrix} \alpha_{1,1} & -\alpha_{0,1} \\ -\alpha_{1,0} & \alpha_{0,0} \end{pmatrix}.$$

Solution. Recall that, for square matrices, $B = A^{-1}$ if and only if $AB = I$.

$$\begin{aligned} & \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix} \frac{1}{\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}} \begin{pmatrix} \alpha_{1,1} & -\alpha_{0,1} \\ -\alpha_{1,0} & \alpha_{0,0} \end{pmatrix} \\ &= \frac{1}{\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}} \begin{pmatrix} \alpha_{0,0}\alpha_{1,1} - \alpha_{0,1}\alpha_{1,0} & -\alpha_{0,0}\alpha_{0,1} + \alpha_{0,1}\alpha_{0,0} \\ \alpha_{1,0}\alpha_{0,1} + \alpha_{1,1}\alpha_{0,0} & -\alpha_{0,1}\alpha_{1,0} + \alpha_{0,0}\alpha_{1,1} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned}$$



YouTube: https://www.youtube.com/watch?v=WvwcrDM_K3k

What we notice from the last exercises is that $\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}$ characterizes whether

A has an inverse or not:

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix}$$

is nonsingular if and only if $\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1} \neq 0$. The expression $\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}$ is known as the determinant of the 2×2 matrix and denoted by $\det(A)$:

Definition 9.2.2.1 The determinant of

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix}$$

is given by

$$\det(A) = \alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}.$$

◇

Now, λ is an eigenvalue of A if and only if $\lambda I - A$ is singular. For our 2×2 matrix

$$\lambda I - A = \begin{pmatrix} \lambda - \alpha_{0,0} & -\alpha_{0,1} \\ -\alpha_{1,0} & \lambda - \alpha_{1,1} \end{pmatrix}$$

is singular if and only if

$$(\lambda - \alpha_{0,0})(\lambda - \alpha_{1,1}) - (-\alpha_{1,0})(-\alpha_{0,1}) = 0.$$

In other words, λ is an eigenvalue of this matrix if and only if λ is a root of

$$\begin{aligned} p_A(\lambda) &= (\lambda - \alpha_{0,0})(\lambda - \alpha_{1,1}) - (-\alpha_{1,0})(-\alpha_{0,1}) \\ &= \lambda^2 - (\alpha_{0,0} + \alpha_{1,1})\lambda + (\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}), \end{aligned}$$

which is a polynomial of degree two. A polynomial of degree two has two roots (counting multiplicity). This polynomial is known as the characteristic polynomial of the 2×2 matrix.

We now have a means for computing eigenvalues and eigenvectors of a 2×2 matrix:

- Form the characteristic polynomial $p_A(\lambda)$.
- Solve for its roots, λ_0 and λ_1 .
- Find nonzero vectors x_0 and x_1 in the null spaces of $\lambda_0 I - A$ and $\lambda_1 I - A$, respectively.



YouTube: <https://www.youtube.com/watch?v=FjoULa2dMC8>

The notion of a determinant of a matrix, $\det(A)$, generalizes to $m \times m$ matrices as does the fact that A is nonsingular if and only if $\det(A) \neq 0$. Similarly, the notion of a characteristic polynomial is then generalized to $m \times m$ matrices:

Definition 9.2.2.2 Characteristic polynomial. The characteristic polynomial of $m \times m$ matrix A is given by

$$p_A(\lambda) = \det(\lambda I - A).$$

◊

At some point in your education, you may have been taught how to compute the determinant of an arbitrary $m \times m$ matrix. *For this course*, such computations have no practical applicability, when matrices are larger than 3×3 or so, and hence we don't spend time on how to compute determinants. What *is* important to our discussions is that for an $m \times m$ matrix A the characteristic polynomial is a polynomial of degree m , a result we formalize in a theorem without giving a proof:

Theorem 9.2.2.3 *If $A \in \mathbb{C}^{m \times m}$ then $p_A(\lambda) = \det(\lambda I - A)$ is a polynomial of degree m .*

This insight now allows us to further characterize the set of all eigenvalues of a given matrix:

Theorem 9.2.2.4 *Let $A \in \mathbb{C}^{m \times m}$. Then $\lambda \in \Lambda(A)$ if and only if $p_A(\lambda) = 0$.*

Proof. This follows from the fact that a matrix has a nontrivial null space if and only if its determinant is zero. Hence, $p_A(\lambda) = 0$ if and only if there exists $x \neq 0$ such that $(\lambda I - A)x = 0$. $(\lambda I - A)x = 0$ if and only if $Ax = \lambda x$. ■

Recall that a polynomial of degree m ,

$$p_m(\chi) = \chi^m + \cdots + \gamma_1\chi + \gamma_0,$$

can be factored as

$$p_m(\chi) = (\chi - \chi_0)^{m_0} \cdots (\chi - \chi_{k-1})^{m_k},$$

where the χ_i are distinct roots, m_i equals the multiplicity of the root, and $m_0 + \cdots + m_{k-1} = m$. The concept of (algebraic) multiplicity carries over to eigenvalues.

Definition 9.2.2.5 Algebraic multiplicity of an eigenvalue. Let $A \in \mathbb{C}^{m \times m}$ and $p_m(\lambda)$ its characteristic polynomial. Then the (algebraic) multiplicity of eigenvalue λ_i equals the multiplicity of the corresponding root of the polynomial. ◊

Often we will list the eigenvalues of $A \in \mathbb{C}^{m \times m}$ as m eigenvalues $\lambda_0, \dots, \lambda_{m-1}$ even when some are equal (have algebraic multiplicity greater than one). In this case we say that we are counting multiplicity. In other words, we are counting each eigenvalue (root of the characteristic polynomial) separately, even if they are equal.

An immediate consequence is that A has m eigenvalues (multiplicity counted), since a polynomial of degree m has m roots (multiplicity counted), which is captured in the following lemma.

Lemma 9.2.2.6 *If $A \in \mathbb{C}^{m \times m}$ then A has m eigenvalues (multiplicity counted).*

The relation between eigenvalues and the roots of the characteristic polynomial yields a disconcerting insight: A general formula for the eigenvalues of an arbitrary $m \times m$ matrix with $m > 4$ does not exist. The reason is that "Galois theory" tells us that there is no general formula for the roots of a polynomial of degree $m > 4$ (details go beyond the scope of this

course). Given any polynomial $p_m(\chi)$ of degree m , an $m \times m$ matrix can be constructed such that its characteristic polynomial is $p_m(\lambda)$. In particular, if

$$p_m(\chi) = \chi^m + \alpha_{m-1}\chi^{m-1} + \cdots + \alpha_1\chi + \alpha_0$$

and

$$A = \begin{pmatrix} -\alpha_{n-1} & -\alpha_{n-2} & -\alpha_{n-3} & \cdots & -\alpha_1 & -\alpha_0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

then

$$p_m(\lambda) = \det(\lambda I - A).$$

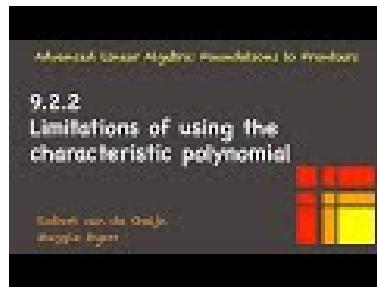
(Since we don't discuss how to compute the determinant of a general matrix, you will have to take our word for this fact.) Hence, we conclude that no general formula can be found for the eigenvalues for $m \times m$ matrices when $m > 4$. What we will see is that we will instead create algorithms that converge to the eigenvalues and/or eigenvectors of matrices.

Corollary 9.2.2.7 *If $A \in \mathbb{R}^{m \times m}$ is real-valued then some or all of its eigenvalues may be complex-valued. If eigenvalue λ is complex-valued, then its conjugate, $\bar{\lambda}$, is also an eigenvalue. Indeed, the complex eigenvalues of a real-valued matrix come in complex pairs.*

Proof. It can be shown that if A is real-valued, then the coefficients of its characteristic polynomial are all real-valued. Complex roots of a polynomial with real coefficients come in conjugate pairs. ■

The last corollary implies that if m is odd, then at least one eigenvalue of a real-valued $m \times m$ matrix must be real-valued.

Corollary 9.2.2.8 *If $A \in \mathbb{R}^{m \times m}$ is real-valued and m is odd, then at least one of the eigenvalues of A is real-valued.*



YouTube: <https://www.youtube.com/watch?v=BVqdIKTK1SI>

It would seem that the natural progression for computing eigenvalues and eigenvectors would be

- Form the characteristic polynomial $p_A(\lambda)$.
- Solve for its roots, $\lambda_0, \dots, \lambda_{m-1}$, which give us the eigenvalues of A .

- Find eigenvectors associated with the eigenvalues by finding bases for the null spaces of $\lambda_i I - A$.

However, as mentioned, finding the roots of a polynomial is a problem. Moreover, finding vectors in the null space is also problematic in the presence of roundoff error. For this reason, the strategy for computing eigenvalues and eigenvectors is going to be to compute approximations of eigenvectors hand in hand with the eigenvalues.

9.2.3 More properties of eigenvalues and vectors

No video for this unit

This unit reminds us of various properties of eigenvalue and eigenvectors through a sequence of homeworks.

Homework 9.2.3.1 Let λ be an eigenvalue of $A \in \mathbb{C}^{m \times m}$ and let

$$\mathcal{E}_\lambda(A) = \{x \in \mathbb{C}^m | Ax = \lambda x\}.$$

be the set of all eigenvectors of A associated with λ plus the zero vector (which is not considered an eigenvector). Show that $\mathcal{E}_\lambda(A)$ is a subspace.

Solution. A set $\mathcal{S} \subset \mathbb{C}^m$ is a subspace if and only if for all $\alpha \in \mathbb{C}$ and $x, y \in \mathbb{C}^m$ two conditions hold:

- $x \in \mathcal{S}$ implies that $\alpha x \in \mathcal{S}$.
- $x, y \in \mathcal{S}$ implies that $x + y \in \mathcal{S}$.
- $x \in \mathcal{E}_\lambda(A)$ implies $\alpha x \in \mathcal{E}_\lambda(A)$:

$x \in \mathcal{E}_\lambda(A)$ means that $Ax = \lambda x$. If $\alpha \in \mathbb{C}$ then $\alpha Ax = \alpha \lambda x$ which, by commutativity and associativity means that $A(\alpha x) = \lambda(\alpha x)$. Hence $(\alpha x) \in \mathcal{E}_\lambda(A)$.

- $x, y \in \mathcal{E}_\lambda(A)$ implies $x + y \in \mathcal{E}_\lambda(A)$:

$$A(x + y) = Ax + Ay = \lambda x + \lambda y = \lambda(x + y).$$

While there are an infinite number of eigenvectors associated with an eigenvalue, the fact that they form a subspace (provided the zero vector is added) means that they can be described by a finite number of vectors, namely a basis for that space.

Homework 9.2.3.2 Let $D \in \mathbb{C}^{m \times m}$ be a diagonal matrix. Give all eigenvalues of D . For each eigenvalue, give a convenient eigenvector.

Solution. Let

$$D = \begin{pmatrix} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_{m-1} \end{pmatrix}.$$

Then

$$\lambda I - D = \begin{pmatrix} \lambda - \delta_0 & 0 & \cdots & 0 \\ 0 & \lambda - \delta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda - \delta_{m-1} \end{pmatrix}$$

is singular if and only if $\lambda = \delta_i$ for some $i \in \{0, \dots, m-1\}$. Hence $\Lambda(D) = \{\delta_0, \delta_1, \dots, \delta_{m-1}\}$.

Now,

$$De_j = \text{the column of } D \text{ indexed with } j = \delta_j e_j$$

and hence e_j is an eigenvector associated with δ_j .

Homework 9.2.3.3 Compute the eigenvalues and corresponding eigenvectors of

$$A = \begin{pmatrix} -2 & 3 & -7 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

(Recall: the solution is not unique.)

Solution. The eigenvalues can be found on the diagonal: $\{-2, 1, 2\}$.

- To find an eigenvector associated with -2 , form

$$(-2)I - A = \begin{pmatrix} 0 & -3 & 7 \\ 0 & -3 & -1 \\ 0 & 0 & -4 \end{pmatrix}$$

and look for a vector in the null space of this matrix. By examination,

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

is in the null space of this matrix and hence an eigenvector of A .

- To find an eigenvector associated with 1 , form

$$(1)I - A = \begin{pmatrix} 3 & -3 & 7 \\ 0 & 0 & -1 \\ 0 & 0 & -1 \end{pmatrix}$$

and look for a vector in the null space of this matrix. Given where the zero appears on the diagonal, we notice that a vector of the form

$$\begin{pmatrix} \chi_0 \\ 1 \\ 0 \end{pmatrix}$$

is in the null space if χ_0 is chosen appropriately. This means that

$$3\chi_0 - 3(1) = 0$$

and hence $\chi_0 = 1$ so that

$$\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

is in the null space of this matrix and hence an eigenvector of A .

- To find an eigenvector associated with 2, form

$$(2)I - A = \begin{pmatrix} 4 & -3 & 7 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

and look for a vector in the null space of this matrix. Given where the zero appears on the diagonal, we notice that a vector of the form

$$\begin{pmatrix} \chi_0 \\ \chi_1 \\ 1 \end{pmatrix}$$

is in the null space if χ_0 and χ_1 are chosen appropriately. This means that

$$\chi_1 - 1(1) = 0$$

and hence $\chi_1 = 1$. Also,

$$4\chi_0 - 3(1) + 7(1) = 0$$

so that $\chi_0 = -1$,

$$\begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}$$

is in the null space of this matrix and hence an eigenvector of A .

Homework 9.2.3.4 Let $U \in \mathbb{C}^{m \times m}$ be an upper triangular matrix. Give all eigenvalues of U . For each eigenvalue, give a convenient eigenvector.

Solution. Let

$$U = \begin{pmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,m-1} \\ 0 & v_{1,1} & \cdots & v_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_{m-1,m-1} \end{pmatrix}.$$

Then

$$\lambda I - U = \begin{pmatrix} \lambda - v_{0,0} & -v_{0,1} & \cdots & -v_{0,m-1} \\ 0 & \lambda - v_{1,1} & \cdots & -v_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda - v_{m-1,m-1} \end{pmatrix}.$$

is singular if and only if $\lambda = v_{i,i}$ for some $i \in \{0, \dots, m-1\}$. Hence $\Lambda(U) = \{v_{0,0}, v_{1,1}, \dots, v_{m-1,m-1}\}$.

Let λ be an eigenvalue of U . Things get a little tricky if λ has multiplicity greater than one. Partition

$$U = \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ 0 & v_{11} & u_{12}^T \\ 0 & 0 & U_{22} \end{pmatrix}$$

where $v_{11} = \lambda$. We are looking for $x \neq 0$ such that $(\lambda I - U)x = 0$ or, partitioning x ,

$$\begin{pmatrix} v_{11}I - U_{00} & -u_{01} & -U_{02} \\ 0 & 0 & -u_{12}^T \\ 0 & 0 & v_{11}I - U_{22} \end{pmatrix} \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

If we choose $x_2 = 0$ and $\chi_1 = 1$, then

$$(v_{11}I - U_{00})x_0 - u_{01} = 0$$

and hence x_0 must satisfy

$$(v_{11}I - U_{00})x_0 = u_{01}.$$

If $v_{11}I - U_{00}$ is nonsingular, then there is a unique solution to this equation, and

$$\begin{pmatrix} (v_{11}I - U_{00})^{-1}u_{01} \\ 1 \\ 0 \end{pmatrix}$$

is the desired eigenvector. HOWEVER, this means that the partitioning

$$U = \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ 0 & v_{11} & u_{12}^T \\ 0 & 0 & U_{22} \end{pmatrix}$$

must be such that v_{11} is the FIRST diagonal element that equals λ .

In the next week, we will see that practical algorithms for computing the eigenvalues and eigenvectors of a square matrix morph that matrix into an upper triangular matrix via a sequence of transforms that preserve eigenvalues. The eigenvectors of that triangular matrix can then be computed using techniques similar to those in the solution to the last homework. Once those have been computed, they can be "back transformed" into the eigenvectors of the original matrix.

9.2.4 The Schur and Spectral Decompositions



YouTube: <https://www.youtube.com/watch?v=2AsK3KEtss0>

Practical methods for computing eigenvalues and eigenvectors transform a given matrix into a simpler matrix (diagonal or tridiagonal) via a sequence of transformations that preserve eigenvalues known as similarity transformations.

Definition 9.2.4.1 Given a nonsingular matrix Y , the transformation $Y^{-1}AY$ is called a similarity transformation (applied to matrix A). \diamond

Definition 9.2.4.2 Matrices A and B are said to be similar if there exists a nonsingular matrix Y such that $B = Y^{-1}AY$. \diamond

Homework 9.2.4.1 Let $A, B, Y \in \mathbb{C}^{m \times m}$, where Y is nonsingular, and (λ, x) an eigenpair of A .

Which of the follow is an eigenpair of $B = Y^{-1}AY$:

- (λ, x) .
- $(\lambda, Y^{-1}x)$.
- (λ, Yx) .
- $(1/\lambda, Y^{-1}x)$.

Answer. $(\lambda, Y^{-1}x)$.

Now justify your answer.

Solution. Since $Ax = \lambda x$ we know that

$$Y^{-1}AYY^{-1}x = \lambda Y^{-1}x.$$

Hence $(\lambda, Y^{-1}x)$ is an eigenpair of B .

The observation is that similarity transformations preserve the eigenvalues of a matrix, as summarized in the following theorem.

Theorem 9.2.4.3 Let $A, Y, B \in \mathbb{C}^{m \times m}$, assume Y is nonsingular, and let $B = Y^{-1}AY$. Then $\Lambda(A) = \Lambda(B)$.

Proof. Let $\lambda \in \Lambda(A)$ and x be an associated eigenvector. Then $Ax = \lambda x$ if and only if $Y^{-1}AYY^{-1}x = Y^{-1}\lambda x$ if and only if $B(Y^{-1}x) = \lambda(Y^{-1}x)$. \blacksquare

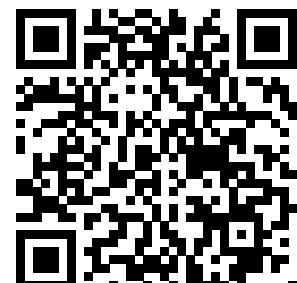
It is not hard to expand the last proof to show that if A is similar to B and $\lambda \in \Lambda(A)$ has algebraic multiplicity k then $\lambda \in \Lambda(B)$ has algebraic multiplicity k .



YouTube: <https://www.youtube.com/watch?v=n02VjGJX5CQ>

In Subsection 2.2.7, we argued that the application of unitary matrices is desirable, since they preserve length and hence don't amplify error. For this reason, unitary similarity transformations are our weapon of choice when designing algorithms for computing eigenvalues and eigenvectors.

Definition 9.2.4.4 Given a nonsingular matrix Q the transformation $Q^H A Q$ is called a unitary similarity transformation (applied to matrix A). ◇



YouTube: <https://www.youtube.com/watch?v=mJNM4EYB-9s>

The following is a fundamental theorem for the algebraic eigenvalue problem that is key to practical algorithms for finding eigenvalues and eigenvectors.

Theorem 9.2.4.5 Schur Decomposition Theorem. *Let $A \in \mathbb{C}^{m \times m}$. Then there exist a unitary matrix Q and upper triangular matrix U such that $A = QUQ^H$. This decomposition is called the Schur decomposition of matrix A .*

Proof. We will outline how to construct Q so that $Q^H A Q = U$, an upper triangular matrix.

Since a polynomial of degree m has at least one root, matrix A has at least one eigenvalue, λ_1 , and corresponding eigenvector q_1 , where we normalize this eigenvector to have length one.

Thus $Aq_1 = \lambda_1 q_1$. Choose Q_2 so that $Q = \begin{pmatrix} q_1 & | & Q_2 \end{pmatrix}$ is unitary. Then

$$\begin{aligned} Q^H A Q &= \\ &= \begin{pmatrix} q_1 & | & Q_2 \end{pmatrix}^H A \begin{pmatrix} q_1 & | & Q_2 \end{pmatrix} \\ &= \begin{pmatrix} q_1^H A q_1 & | & q_1^H A Q_2 \\ Q_2^H A q_1 & | & Q_2^H A Q_2 \end{pmatrix} \\ &= \begin{pmatrix} \lambda_1 & | & q_1^H A Q_2 \\ \lambda Q_2^H q_1 & | & Q_2^H A Q_2 \end{pmatrix} \\ &= \begin{pmatrix} \lambda_1 & | & w^T \\ 0 & | & B \end{pmatrix}, \end{aligned}$$

where $w^T = q_1^H A Q_2$ and $B = Q_2^H A Q_2$. This insight can be used to construct an inductive proof. \blacksquare

In other words: Given matrix A , there exists a unitary matrix Q such that applying the unitary similarity transformation $Q^H A Q$ yields an upper triangular matrix U . Since then $\Lambda(A) = \Lambda(U)$, the eigenvalues of A can be found on the diagonal of U . The eigenvectors of U can be computed and from those the eigenvectors of A can be recovered.

One should not mistake the above theorem and its proof for a constructive way to compute the Schur decomposition: finding an eigenvalue, λ_1 and/or the eigenvector associated with it, q_1 , is difficult. Also, completing the unitary matrix $\begin{pmatrix} q_1 & | & Q_2 \end{pmatrix}$ is expensive (requiring the equivalent of a QR factorization).

Homework 9.2.4.2 Let $A \in \mathbb{C}^{m \times m}$, $A = QUQ^H$ be its Schur decomposition, and $X^{-1}UX = \Lambda$, where Λ is a diagonal matrix and X is nonsingular.

- How are the elements of Λ related to the elements of U ?
- How are the columns of X related to the eigenvectors of A ?

Solution.

- How are the elements of Λ related to the elements of U ?

The diagonal elements of U equal the diagonal elements of Λ .

- How are the columns of X related to the eigenvectors of A ?

$$A = QUQ^H = QX\Lambda X^{-1}Q^H = (QX)\Lambda(QX)^{-1}.$$

Hence the columns of QX equal eigenvectors of A .



YouTube: https://www.youtube.com/watch?v=uV5-00_LBkA

If the matrix is Hermitian, then the Schur decomposition has the added property that U is diagonal. The resulting decomposition is known as the Spectral decomposition.

Theorem 9.2.4.6 Spectral Decomposition Theorem. *Let $A \in \mathbb{C}^{m \times m}$ be Hermitian. Then there exist a unitary matrix Q and diagonal matrix $D \in \mathbb{R}^{m \times m}$ such that $A = QDQ^H$. This decomposition is called the spectral decomposition of matrix A .*

Proof. Let $A = QUQ^H$ be the Schur decomposition of A . Then $U = Q^H A Q$. Since A is Hermitian, so is U since $U^H = (Q^H A Q)^H = Q^H A^H Q = Q^H A Q = U$. A triangular matrix that is Hermitian is diagonal. Any Hermitian matrix has a real-valued diagonal and hence D has real-valued on its diagonal. ■

In practical algorithms, it will often occur that an intermediate result can be partitioned into smaller subproblems. This is known as **deflating** the problem and builds on the following insights.

Theorem 9.2.4.7 *Let $A \in \mathbb{C}^{m \times m}$ be of form $A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right)$, where A_{TL} and A_{BR} are square submatrices. Then $\Lambda(A) = \Lambda(A_{TL}) \cup \Lambda(A_{BR})$.*

The proof of the above theorem follows from the next homework regarding how the Schur decomposition of A can be computed from the Schur decompositions of A_{TL} and A_{BR} .

Homework 9.2.4.3 Let $A \in \mathbb{C}^{m \times m}$ be of form

$$A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right),$$

where A_{TL} and A_{BR} are square submatrices with Schur decompositions

$$A_{TL} = Q_{TL}U_{TL}Q_{TL}^H \text{ and } A_{BR} = Q_{BR}U_{BR}Q_{BR}^H.$$

Give the Schur decomposition of A .

Solution.

$$\begin{aligned}
 A &= \\
 &\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right) \\
 &= \\
 &\left(\begin{array}{c|c} Q_{TL}U_{TL}Q_{TL}^H & A_{TR} \\ \hline 0 & Q_{BR}U_{BR}Q_{BR}^H \end{array} \right) \\
 &= \\
 &\underbrace{\left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right)}_Q \quad \underbrace{\left(\begin{array}{c|c} U_{TL} & Q_{TL}^H A_{TR} Q_{BR} \\ \hline 0 & U_{BR} \end{array} \right)}_U \quad \underbrace{\left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right)^H}_{Q^H}
 \end{aligned}$$

Homework 9.2.4.4 Generalize the result in the last homework for block upper triangular matrices:

$$A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ \hline 0 & A_{1,1} & \cdots & A_{1,N-1} \\ \hline 0 & 0 & \ddots & \vdots \\ \hline 0 & 0 & \cdots & A_{N-1,N-1} \end{array} \right).$$

Solution. For $i = 0, \dots, N-1$, let $A_{i,i} = Q_i U_{i,i} Q_i^H$ be the Schur decomposition of $A_{i,i}$.

Then

$$\begin{aligned}
 A &= \\
 &= \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ \hline 0 & A_{1,1} & \cdots & A_{1,N-1} \\ \hline 0 & 0 & \ddots & \vdots \\ \hline 0 & 0 & \cdots & A_{N-1,N-1} \end{array} \right) \\
 &= \\
 &= \left(\begin{array}{c|c|c|c} Q_0 U_{0,0} Q_0^H & A_{0,1} & \cdots & A_{0,N-1} \\ \hline 0 & Q_1 U_{1,1} Q_1^H & \cdots & A_{1,N-1} \\ \hline 0 & 0 & \ddots & \vdots \\ \hline 0 & 0 & \cdots & Q_{N-1} U_{N-1,N-1} Q_{N-1}^H \end{array} \right) \\
 &= \\
 &= \left(\begin{array}{c|c|c|c} Q_0 & 0 & \cdots & 0 \\ \hline 0 & Q_1 & \cdots & 0 \\ \hline 0 & 0 & \ddots & \vdots \\ \hline 0 & 0 & \cdots & Q_{N-1} \end{array} \right) \\
 &\quad \left(\begin{array}{c|c|c|c} U_{0,0} & Q_0^H A_{0,1} Q_1 & \cdots & Q_0^T A_{0,N-1} Q_{N-1} \\ \hline 0 & U_{1,1} & \cdots & Q_1^H A_{1,N-1} Q_{N-1} \\ \hline 0 & 0 & \ddots & \vdots \\ \hline 0 & 0 & \cdots & U_{N-1,N-1} \end{array} \right) \\
 &\quad \left(\begin{array}{c|c|c|c} Q_0 & 0 & \cdots & 0 \\ \hline 0 & Q_1 & \cdots & 0 \\ \hline 0 & 0 & \ddots & \vdots \\ \hline 0 & 0 & \cdots & Q_{N-1} \end{array} \right)^H
 \end{aligned}$$

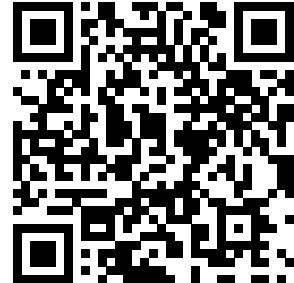
9.2.5 Diagonalizing a matrix



YouTube: <https://www.youtube.com/watch?v=dLaFK2TJ7y8>

The algebraic eigenvalue problem or, more simply, the computation of eigenvalues and eigenvectors is often presented as the problem of diagonalizing a matrix. We make that link in this unit.

Definition 9.2.5.1 Diagonalizable matrix. A matrix $A \in \mathbb{C}^{m \times m}$ is said to be diagonalizable if and only if there exists a nonsingular matrix X and diagonal matrix D such that $X^{-1}AX = D$. \diamond



YouTube: <https://www.youtube.com/watch?v=qW5lcD3K1RU>

Why is this important? Consider the equality $w = Av$. Notice that we can write w as a linear combination of the columns of X :

$$w = X \underbrace{(X^{-1}w)}_{\tilde{w}} .$$

In other words, $X^{-1}w$ is the vector of coefficients when w is written in terms of the basis that consists of the columns of X . Similarly, we can write v as a linear combination of the columns of X :

$$v = X \underbrace{(X^{-1}v)}_{\tilde{v}} .$$

Now, since X is nonsingular, $w = Av$ is equivalent to $X^{-1}w = X^{-1}AXX^{-1}v$, and hence $X^{-1}w = D(X^{-1}v)$.

Remark 9.2.5.2 We conclude that if we view the matrices in the right basis (namely the basis that consists of the columns of X), then the transformation $w := Av$ simplifies to $\tilde{w} := D\tilde{v}$. This is a really big deal.

How is diagonalizing a matrix related to eigenvalues and eigenvectors? Let's assume that $X^{-1}AX = D$. We can rewrite this as

$$AX = XD$$

and partition

$$A \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right) \left(\begin{array}{c|c|c|c} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_{m-1} \end{array} \right).$$

Multiplying this out yields

$$\left(\begin{array}{c|c|c|c} Ax_0 & Ax_1 & \cdots & Ax_{m-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} \delta_0 x_0 & \delta_1 x_1 & \cdots & \delta_{m-1} x_{m-1} \end{array} \right).$$

We conclude that

$$Ax_j = \delta_j x_j$$

which means that the entries on the diagonal of D are the eigenvalues of A and the corresponding eigenvectors are found as columns of X .

Homework 9.2.5.1 In [Homework 9.2.3.3](#), we computed the eigenvalues and corresponding eigenvectors of

$$A = \begin{pmatrix} -2 & 3 & -7 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}.$$

Use the answer to that question to give a matrix X such that $X^{-1}AX = \Lambda$. Check that $AX = X\Lambda$.

Solution. The eigenpairs computed for [Homework 9.2.3.3](#) were

$$(-2, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}), (1, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}), \text{ and } (2, \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}).$$

Hence

$$\begin{pmatrix} 1 & 1 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} -2 & 3 & -7 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 & 1 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

We can check this:

$$\underbrace{\begin{pmatrix} -2 & 3 & -7 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 & 1 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}}_{\begin{pmatrix} -2 & 1 & -2 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \end{pmatrix}} = \underbrace{\begin{pmatrix} 1 & 1 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}}_{\begin{pmatrix} -2 & 1 & -2 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \end{pmatrix}}.$$

Now assume that the eigenvalues of $A \in \mathbb{C}^{m \times m}$ are given by $\{\lambda_0, \lambda_1, \dots, \lambda_{m-1}\}$, where eigenvalues are repeated according to their algebraic multiplicity. Assume that there are m linearly independent vectors $\{x_0, x_1, \dots, x_{m-1}\}$ such that $Ax_j = \lambda_j x_j$. Then

$$A \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right) \left(\begin{array}{c|c|c|c} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{m-1} \end{array} \right).$$

Hence, if $X = \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right)$ and $D = \text{diag}(\lambda_0, \lambda_1, \dots, \lambda_{m-1})$ then $X^{-1}AX = D$. In other words, if A has m linearly independent eigenvectors, then A is diagonalizable.

These insights are summarized in the following theorem:

Theorem 9.2.5.3 A matrix $A \in \mathbb{C}^{m \times m}$ is diagonalizable if and only if it has m linearly independent eigenvectors.

Here are some classes of matrices that are diagonalizable:

- Diagonal matrices.

If A is diagonal, then choosing $X = I$ and $A = D$ yields $X^{-1}AX = D$.

- Hermitian matrices.

If A is Hermitian, then the spectral decomposition theorem tells us that there exists unitary matrix Q and diagonal matrix D such that $A = QDQ^H$. Choosing $X = Q$ yields $X^{-1}AX = D$.

- Triangular matrices with distinct diagonal elements.

If U is upper triangular and has distinct diagonal elements, then by [Homework 9.2.3.4](#) we know we can find an eigenvector associated with each diagonal element and by design those eigenvectors are linearly independent. Obviously, this can be extended to lower triangular matrices as well.

Homework 9.2.5.2 Let $A \in \mathbb{C}^{m \times m}$ have distinct eigenvalues.

ALWAYS/SOMETIMES/NEVER: A is diagonalizable.

Answer. ALWAYS

Now prove it!

Solution. Let $A = QUQ^H$ be the Schur decomposition of matrix A . Since U is upper triangular, and has the same eigenvalues as A , it has distinct entries along its diagonal. Hence, by our earlier observations, there exists a nonsingular matrix X such that $X^{-1}UX = D$, a diagonal matrix. Now,

$$X^{-1}Q^H A Q X = X^{-1} U X = D$$

and hence $Y = QX$ is the nonsingular matrix that diagonalizes A .



YouTube: <https://www.youtube.com/watch?v=PMtZNl8CHzM>

9.2.6 Jordan Canonical Form



YouTube: <https://www.youtube.com/watch?v=amD2FOSXfls>

Homework 9.2.6.1 Compute the eigenvalues of $k \times k$ matrix

$$J_k(\mu) = \begin{pmatrix} \mu & 1 & 0 & \cdots & 0 & 0 \\ 0 & \mu & 1 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & \mu & 1 \\ 0 & 0 & 0 & \cdots & 0 & \mu \end{pmatrix} \quad (9.2.1)$$

where $k > 1$. For each eigenvalue compute a basis for the subspace of its eigenvectors (including the zero vector to make it a subspace).

Hint.

- How many linearly independent columns does $\lambda I - J_k(\mu)$ have?
- What does this say about the dimension of the null space $\mathcal{N}(\lambda I - J_k(\mu))$?
- You should be able to find eigenvectors by examination.

Solution. Since the matrix is upper triangular and all entries on its diagonal equal μ . Now,

$$\mu I - J_k(\mu) = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

has $m-1$ linearly independent columns and hence its nullspace is one dimensional: $\dim(\mathcal{N}(\mu I - J_k(\mu))) = 1$. So, we are looking for one vector in the basis of $\mathcal{N}(\mu I - J_k(\mu))$. By examination, $J_k(\mu)e_0 = \mu e_0$ and hence e_0 is an eigenvector associated with the only eigenvalue μ .



YouTube: <https://www.youtube.com/watch?v=QEunPSiFZF0>

The matrix in (9.2.1) is known as a Jordan block.

The point of the last exercise is to show that if A has an eigenvalue of algebraic multiplicity k , then it does not necessarily have k linearly independent eigenvectors. That, in turn, means there are matrices that do not have a full set of eigenvectors. We conclude that there are matrices that are not diagonalizable. We call such matrices defective.

Definition 9.2.6.1 Defective matrix. A matrix $A \in \mathbb{C}^{m \times m}$ that does not have m linearly independent eigenvectors is said to be defective. \diamond

Corollary 9.2.6.2 Matrix $A \in \mathbb{C}^{m \times m}$ is diagonalizable if and only if it is not defective.

Proof. This is an immediate consequence of Theorem 9.2.5.3. \blacksquare

Definition 9.2.6.3 Geometric multiplicity. Let $\lambda \in \Lambda(A)$. Then the geometric multiplicity of λ is defined to be the dimension of $\mathcal{E}_\lambda(A)$ defined by

$$\mathcal{E}_\lambda(A) = \{x \in \mathbb{C}^m \mid Ax = \lambda x\}.$$

In other words, the geometric multiplicity of λ equals the number of linearly independent eigenvectors that are associated with λ . \diamond

Homework 9.2.6.2 Let $A \in \mathbb{C}^{m \times m}$ have the form

$$A = \left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right)$$

where A_{00} and A_{11} are square. Show that

- If (λ, x) is an eigenpair of A_{00} then $(\lambda, \begin{pmatrix} x \\ 0 \end{pmatrix})$ is an eigenpair of A .
- If (μ, y) is an eigenpair of A_{11} then $(\mu, \begin{pmatrix} 0 \\ y \end{pmatrix})$ is an eigenpair of A .
- If $(\lambda, \begin{pmatrix} x \\ y \end{pmatrix})$ is an eigenpair of A then (λ, x) is an eigenpair of A_{00} and (λ, y) is an eigenpair of A_{11} .
- $\Lambda(A) = \Lambda(A_{00}) \cup \Lambda(A_{11})$.

Solution. Let $A \in \mathbb{C}^{m \times m}$ have the form

$$A = \left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right)$$

where A_{00} and A_{11} are square. Show that

- If (λ, x) is an eigenpair of A_{00} then $(\lambda, \begin{pmatrix} x \\ 0 \end{pmatrix})$ is an eigenpair of A .
- $$\left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right) \begin{pmatrix} x \\ 0 \end{pmatrix} = \begin{pmatrix} A_{00}x \\ 0 \end{pmatrix} = \begin{pmatrix} \lambda x \\ 0 \end{pmatrix} = \lambda \begin{pmatrix} x \\ 0 \end{pmatrix}.$$
- If (μ, y) is an eigenpair of A_{11} then $(\mu, \begin{pmatrix} 0 \\ y \end{pmatrix})$ is an eigenpair of A .
- $$\left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right) \begin{pmatrix} 0 \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ A_{11}y \end{pmatrix} = \begin{pmatrix} 0 \\ \mu y \end{pmatrix} = \mu \begin{pmatrix} 0 \\ y \end{pmatrix}.$$
- $\left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right) \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$ implies that

$$\left(\frac{A_{00}x}{A_{11}y} \right) = \left(\frac{\lambda x}{\lambda y} \right),$$

and hence $A_{00}x = \lambda x$ and $A_{11}y = \lambda y$.

- $\Lambda(A) = \Lambda(A_{00}) \cup \Lambda(A_{11})$.

This follows from the first three parts of this problem.

This last homework naturally extends to

$$A = \left(\begin{array}{c|c|c|c} A_{00} & 0 & \cdots & 0 \\ \hline 0 & A_{11} & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & A_{kk} \end{array} \right)$$

The following is a classic result in linear algebra theory that characterizes the relationship between of a matrix and its eigenvectors:



YouTube: <https://www.youtube.com/watch?v=RYg4xLKehDQ>

Theorem 9.2.6.4 Jordan Canonical Form Theorem. Let the eigenvalues of $A \in \mathbb{C}^{m \times m}$ be given by $\lambda_0, \lambda_1, \dots, \lambda_{k-1}$, where an eigenvalue is listed as many times as its geometric multiplicity. There exists a nonsingular matrix X such that

$$X^{-1}AX = \left(\begin{array}{c|c|c|c} J_{m_0}(\lambda_0) & 0 & \cdots & 0 \\ \hline 0 & J_{m_1}(\lambda_1) & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & J_{m_{k-1}}(\lambda_{k-1}) \end{array} \right).$$

For our discussion, the sizes of the Jordan blocks $J_{m_i}(\lambda_i)$ are not particularly important. Indeed, this decomposition, known as the Jordan Canonical Form of matrix A , is not particularly interesting in practice. It is extremely sensitive to perturbation: even the smallest random change to a matrix will make it diagonalizable. As a result, there is no practical mathematical software library or tool that computes it. For this reason, we don't give its proof and don't discuss it further.

9.3 The Power Method and related approaches

9.3.1 The Power Method



YouTube: <https://www.youtube.com/watch?v=gbhHORlNxNM>

The Power Method is a simple method that under mild conditions yields a vector corresponding to the eigenvalue that is largest in magnitude.

Throughout this section we will assume that a given matrix $A \in \mathbb{C}^{m \times m}$ is diagonalizable. Thus, there exists a nonsingular matrix X and diagonal matrix Λ such that $A = X\Lambda X^{-1}$. From the last section, we know that the columns of X equal eigenvectors of A and the elements on the diagonal of Λ equal the eigenvalues:

$$X = \left(\begin{array}{c|c|c|c} x_0 & x_0 & \cdots & x_{m-1} \end{array} \right) \quad \text{and} \quad \Lambda = \left(\begin{array}{c|c|c|c} \lambda_0 & 0 & \cdots & 0 \\ \hline 0 & \lambda_1 & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \lambda_{m-1} \end{array} \right)$$

so that

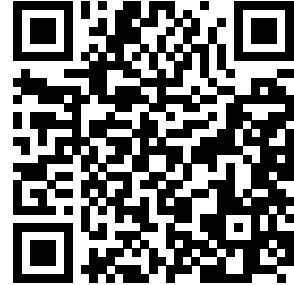
$$Ax_i = \lambda_i x_i \quad \text{for } i = 0, \dots, m-1.$$

For most of this section we will assume that

$$|\lambda_0| > |\lambda_1| \geq \cdots \geq |\lambda_{m-1}|.$$

In particular, λ_0 is the eigenvalue with maximal absolute value.

9.3.1.1 First attempt



YouTube: <https://www.youtube.com/watch?v=sX9pxaH7Wvs>

Let $v^{(0)} \in \mathbb{C}^{m \times m}$ be an ``initial guess''. Our (first attempt at the) Power Method iterates as follows:

```

    Pick  $v^{(0)}$ 
    for  $k = 0, \dots$ 
         $v^{(k+1)} = Av^{(k)}$ 
    endfor
  
```

Clearly $v^{(k)} = A^k v^{(0)}$.

Let

$$v^{(0)} = \psi_0 x_0 + \psi_1 x_1 + \cdots + \psi_{m-1} x_{m-1}.$$

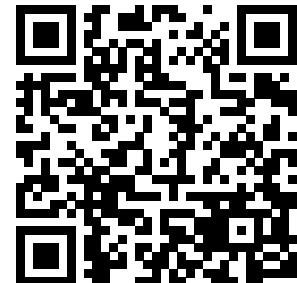
Then

$$\begin{aligned}
 v^{(1)} = Av^{(0)} &= A(\psi_0 x_0 + \psi_1 x_1 + \cdots + \psi_{m-1} x_{m-1}) \\
 &= \psi_0 Ax_0 + \psi_1 Ax_1 + \cdots + \psi_{m-1} Ax_{m-1} \\
 &= \psi_0 \lambda_0 x_0 + \psi_1 \lambda_0 x_1 + \cdots + \psi_{m-1} \lambda_{m-1} x_{m-1}, \\
 v^{(2)} = Av^{(1)} &= A(\psi_0 \lambda_0 x_0 + \psi_1 \lambda_0 x_1 + \cdots + \psi_{m-1} \lambda_{m-1} x_{m-1}) \\
 &= \psi_0 \lambda_0 Ax_0 + \psi_1 \lambda_0 Ax_1 + \cdots + \psi_{m-1} \lambda_{m-1} Ax_{m-1} \\
 &= \psi_0 \lambda_0^2 x_0 + \psi_1 \lambda_0^2 x_1 + \cdots + \psi_{m-1} \lambda_{m-1}^2 x_{m-1}, \\
 &\vdots \\
 v^{(k)} = Av^{(k-1)} &= \psi_0 \lambda_0^k x_0 + \psi_1 \lambda_0^k x_1 + \cdots + \psi_{m-1} \lambda_{m-1}^k x_{m-1}.
 \end{aligned}$$

Now, as long as $\psi_0 \neq 0$ clearly $\psi_0 \lambda_0^k x_0$ will eventually dominate since

$$|\lambda_i| / |\lambda_0| < 1.$$

This means that $v^{(k)}$ will start pointing in the direction of x_0 . In other words, it will start pointing in the direction of an eigenvector corresponding to λ_0 . The problem is that it will become infinitely long if $|\lambda_0| > 1$ or infinitesimally short if $|\lambda_0| < 1$. All is good if $|\lambda_0| = 1$.



YouTube: <https://www.youtube.com/watch?v=LTON9qw8B0Y>

An alternative way of looking at this is to exploit the fact that the eigenvectors, x_i , equal the columns of X . Then

$$y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} = X^{-1}v^{(0)}$$

and

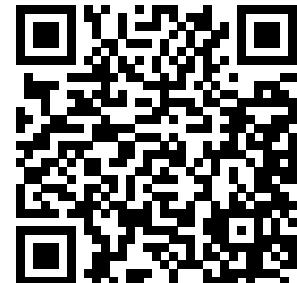
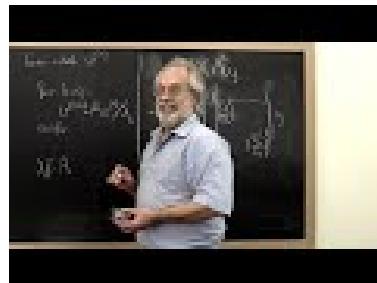
$$\begin{aligned} v^{(0)} &= A^0 v^{(0)} = Xy \\ v^{(1)} &= Av^{(0)} = AXy = X\Lambda y \\ v^{(2)} &= Av^{(1)} = AX\Lambda y = X\Lambda^2 y \\ &\vdots \\ v^{(k)} &= Av^{(k-1)} = AX\Lambda^{k-1}y = X\Lambda^k y \end{aligned}$$

Thus

$$\begin{aligned} v^{(k)} &= \begin{pmatrix} x_0 & x_1 & \cdots & x_{m-1} \end{pmatrix} \begin{pmatrix} \lambda_0^k & 0 & \cdots & 0 \\ 0 & \lambda_1^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{m-1}^k \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} \\ &= \psi_0 \lambda_0^k x_0 + \psi_1 \lambda_1^k x_1 + \cdots + \psi_{m-1} \lambda_{m-1}^k x_{m-1}. \end{aligned}$$

Notice how looking at $v^{(k)}$ in the right basis (the eigenvectors) simplified the explanation.

9.3.1.2 Second attempt



YouTube: https://www.youtube.com/watch?v=MGTGo_TGpTM

Given an initial $v^{(0)} \in \mathbb{C}^m$, a second attempt at the Power Method iterates as follows:

```

Pick  $v^{(0)}$ 
for  $k = 0, \dots$ 
     $v^{(k+1)} = Av^{(k)}/\lambda_0$ 
endfor

```

It is not hard to see that then

$$\begin{aligned}
v^{(k)} &= Av^{(k-1)}/\lambda_0 = A^k v^{(0)}/\lambda_0^k \\
&= \psi_0 \left(\frac{\lambda_0}{\lambda_0}\right)^k x_0 + \psi_1 \left(\frac{\lambda_1}{\lambda_0}\right)^k x_1 + \cdots + \psi_{m-1} \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k x_{m-1} \\
&= \psi_0 x_0 + \psi_1 \left(\frac{\lambda_1}{\lambda_0}\right)^k x_1 + \cdots + \psi_{m-1} \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k x_{m-1}.
\end{aligned}$$

Clearly $\lim_{k \rightarrow \infty} v^{(k)} = \psi_0 x_0$, as long as $\psi_0 \neq 0$, since $\left|\frac{\lambda_k}{\lambda_0}\right| < 1$ for $k > 0$.

Another way of stating this is to notice that

$$A^k = \underbrace{(AA \cdots A)}_{\text{times}} = \underbrace{(X\Lambda X^{-1})(X\Lambda X^{-1}) \cdots (X\Lambda X^{-1})}_{\Lambda^k} = X\Lambda^k X^{-1}.$$

so that

$$\begin{aligned}
v^{(k)} &= A^k v^{(0)}/\lambda_0^k \\
&= A^k X y / \lambda_0^k \\
&= X \Lambda^k X^{-1} X y / \lambda_0^k \\
&= X \Lambda^k y / \lambda_0^k \\
&= X \left(\Lambda^k / \lambda_0^k\right) y = X \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{pmatrix} y.
\end{aligned}$$

Now, since $\left| \frac{\lambda_k}{\lambda_0} \right| < 1$ for $k > 1$ we can argue that

$$\begin{aligned}
 & \lim_{k \rightarrow \infty} v^{(k)} \\
 &= \\
 & \lim_{k \rightarrow \infty} X \left(\begin{array}{c|cc|cc|c} 1 & 0 & \cdots & 0 \\ 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{array} \right) y \\
 &= \\
 & X \left(\begin{array}{c|cc|cc|c} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{array} \right) y \\
 &= \\
 & X\psi_0 e_0 \\
 &= \\
 & \psi_0 X e_0 = \psi_0 x_0.
 \end{aligned}$$

Thus, as long as $\psi_0 \neq 0$ (which means $v^{(0)}$ must have a component in the direction of x_0) this method will eventually yield a vector in the direction of x_0 . However, this time the problem is that we don't know λ_0 when we start.

9.3.1.3 A practical Power Method

The following algorithm, known as the Power Method, avoids the problem of $v^{(k)}$ growing or shrinking in length without requiring λ_0 to be known, by scaling it to be of unit length at each step:

```

Pick  $v^{(0)}$  of unit length
for  $k = 0, \dots$ 
   $v^{(k+1)} = Av^{(k)}$ 
   $v^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|$ 
endfor

```

The idea is that we are only interested in the direction of the eigenvector, and hence it is convenient to rescale the vector to have unit length at each step.

9.3.1.4 The Rayleigh quotient

A question is how to extract an approximation of λ_0 given an approximation of x_0 . The following insights provide the answer:

Definition 9.3.1.1 Rayleigh quotient. If $A \in \mathbb{C}^{m \times m}$ and $x \neq 0 \in \mathbb{C}^m$ then

$$\frac{x^H A x}{x^H x}$$

is known as the **Rayleigh quotient**. ◊

Homework 9.3.1.1 Let x be an eigenvector of A .

ALWAYS/SOMETIMES/NEVER: $\lambda = x^H A x / (x^H x)$ is the associated eigenvalue of A .

Answer. ALWAYS

Now prove it!

Solution. Let x be an eigenvector of A and λ the associated eigenvalue. Then $Ax = \lambda x$. Multiplying on the left by x^H yields $x^H A x = \lambda x^H x$ which, since $x \neq 0$ means that $\lambda = x^H A x / (x^H x)$.

If x is an approximation of the eigenvector x_0 associated with λ_0 , then its Rayleigh quotient is an approximation to λ_0 .

9.3.2 The Power Method: Convergence



YouTube: <https://www.youtube.com/watch?v=P-U4dfwHMwM>

Before we make the algorithm practical, let us examine how fast the iteration converges. This requires a few definitions regarding rates of convergence.

Definition 9.3.2.1 Convergence of a sequence of scalars. Let $\alpha_0, \alpha_1, \alpha_2, \dots \in \mathbb{R}$ be an infinite sequence of scalars. Then α_k is said to converge to α if

$$\lim_{k \rightarrow \infty} |\alpha_k - \alpha| = 0.$$

◊

Definition 9.3.2.2 Convergence of a sequence of vectors. Let $x_0, x_1, x_2, \dots \in \mathbb{C}^m$ be an infinite sequence of vectors. Then x_k is said to converge to x if for any norm $\|\cdot\|$

$$\lim_{k \rightarrow \infty} \|x_k - x\| = 0.$$

◊

Because of the equivalence of norms, discussed in Subsection 1.2.6, if a sequence of vectors converges in one norm, then it converges in all norms.

Definition 9.3.2.3 Rate of convergence. Let $\alpha_0, \alpha_1, \alpha_2, \dots \in \mathbb{R}$ be an infinite sequence of scalars that converges to α . Then

- α_k is said to converge linearly to α if for sufficiently large k

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|$$

for some constant $C < 1$. In other words, if

$$\lim_{k \rightarrow \infty} \frac{|\alpha_{k+1} - \alpha|}{|\alpha_k - \alpha|} = C < 1.$$

- α_k is said to converge superlinearly to α if for sufficiently large k

$$|\alpha_{k+1} - \alpha| \leq C_k |\alpha_k - \alpha|$$

with $C_k \rightarrow 0$. In other words, if

$$\lim_{k \rightarrow \infty} \frac{|\alpha_{k+1} - \alpha|}{|\alpha_k - \alpha|} = 0.$$

- α_k is said to converge quadratically to α if for sufficiently large k

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|^2$$

for some constant C . In other words, if

$$\lim_{k \rightarrow \infty} \frac{|\alpha_{k+1} - \alpha|}{|\alpha_k - \alpha|^2} = C.$$

- α_k is said to converge cubically to α if for large enough k

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|^3$$

for some constant C . In other words, if

$$\lim_{k \rightarrow \infty} \frac{|\alpha_{k+1} - \alpha|}{|\alpha_k - \alpha|^3} = C.$$

◊

Linear convergence can be slow. Let's say that for $k \geq K$ we observe that

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|.$$

Then, clearly, $|\alpha_{k+n} - \alpha| \leq C^n |\alpha_k - \alpha|$. If $C = 0.99$, progress may be very, very slow. If

$|\alpha_k - \alpha| = 1$, then

$$\begin{aligned} |\alpha_{k+1} - \alpha| &\leq 0.99000 \\ |\alpha_{k+2} - \alpha| &\leq 0.98010 \\ |\alpha_{k+3} - \alpha| &\leq 0.97030 \\ |\alpha_{k+4} - \alpha| &\leq 0.96060 \\ |\alpha_{k+5} - \alpha| &\leq 0.95099 \\ |\alpha_{k+6} - \alpha| &\leq 0.94148 \\ |\alpha_{k+7} - \alpha| &\leq 0.93206 \\ |\alpha_{k+8} - \alpha| &\leq 0.92274 \\ |\alpha_{k+9} - \alpha| &\leq 0.91351 \end{aligned}$$

Quadratic convergence is fast. Now

$$\begin{aligned} |\alpha_{k+1} - \alpha| &\leq C|\alpha_k - \alpha|^2 \\ |\alpha_{k+2} - \alpha| &\leq C|\alpha_{k+1} - \alpha|^2 \leq C(C|\alpha_k - \alpha|^2)^2 = C^3|\alpha_k - \alpha|^4 \\ |\alpha_{k+3} - \alpha| &\leq C|\alpha_{k+2} - \alpha|^2 \leq C(C^3|\alpha_k - \alpha|^4)^2 = C^7|\alpha_k - \alpha|^8 \\ &\vdots \\ |\alpha_{k+n} - \alpha| &\leq C^{2^n-1}|\alpha_k - \alpha|^{2^n} \end{aligned}$$

Even if $C = 0.99$ and $|\alpha_k - \alpha| = 1$, then

$$\begin{aligned} |\alpha_{k+1} - \alpha| &\leq 0.99000 \\ |\alpha_{k+2} - \alpha| &\leq 0.970299 \\ |\alpha_{k+3} - \alpha| &\leq 0.932065 \\ |\alpha_{k+4} - \alpha| &\leq 0.860058 \\ |\alpha_{k+5} - \alpha| &\leq 0.732303 \\ |\alpha_{k+6} - \alpha| &\leq 0.530905 \\ |\alpha_{k+7} - \alpha| &\leq 0.279042 \\ |\alpha_{k+8} - \alpha| &\leq 0.077085 \\ |\alpha_{k+9} - \alpha| &\leq 0.005882 \\ |\alpha_{k+10} - \alpha| &\leq 0.000034 \end{aligned}$$

If we consider α the correct result then, eventually, the number of correct digits roughly doubles in each iteration. This can be explained as follows: If $|\alpha_k - \alpha| < 1$, then the number of correct decimal digits is given by

$$-\log_{10} |\alpha_k - \alpha|.$$

Since \log_{10} is a monotonically increasing function,

$$\begin{aligned} \log_{10} |\alpha_{k+1} - \alpha| &\leq \\ \log_{10} C|\alpha_k - \alpha|^2 &= \\ \log_{10}(C) + 2\log_{10} |\alpha_k - \alpha| &\leq \\ 2\log_{10} |\alpha_k - \alpha| & \end{aligned}$$

and hence

$$\underbrace{-\log_{10} |\alpha_{k+1} - \alpha|}_{\substack{\text{number of correct} \\ \text{digits in } \alpha_{k+1}}} \geq 2 \left(\underbrace{-\log_{10} |\alpha_k - \alpha|}_{\substack{\text{number of correct} \\ \text{digits in } \alpha_k}} \right).$$

Cubic convergence is dizzyingly fast: Eventually the number of correct digits triples from one iteration to the next.

For our analysis for the convergence of the Power Method, we define a convenient norm.

Homework 9.3.2.1 Let $X \in \mathbb{C}^{m \times m}$ be nonsingular. Define $\|\cdot\|_{X^{-1}} : \mathbb{C}^m \rightarrow \mathbb{R}$ by $\|y\|_{X^{-1}} = \|X^{-1}y\|$ for some given norm $\|\cdot\| : \mathbb{C}^m \rightarrow \mathbb{R}$.

ALWAYS/SOMETIMES/NEVER: $\|\cdot\|_{X^{-1}}$ is a norm.

Solution. We need to show that

- If $y \neq 0$ then $\|y\|_{X^{-1}} > 0$:

Let $y \neq 0$ and $z = X^{-1}y$. Then $z \neq 0$ since X is nonsingular. Hence

$$\|y\|_{X^{-1}} = \|X^{-1}y\| = \|z\| > 0.$$

- If $\alpha \in \mathbb{C}$ and $y \in \mathbb{C}^m$ then $\|\alpha y\|_{X^{-1}} = |\alpha| \|y\|_{X^{-1}}$:

$$\|\alpha y\|_{X^{-1}} = \|X^{-1}(\alpha y)\| = \|\alpha X^{-1}y\| = |\alpha| \|X^{-1}y\| = |\alpha| \|y\|_{X^{-1}}.$$

- If $x, y \in \mathbb{C}^m$ then $\|x + y\|_{X^{-1}} \leq \|x\|_{X^{-1}} + \|y\|_{X^{-1}}$:

$$\begin{aligned} & \|x + y\|_{X^{-1}} \\ &= \|X^{-1}(x + y)\| \\ &= \|X^{-1}x + X^{-1}y\| \\ &\leq \|X^{-1}x\| + \|X^{-1}y\| \\ &= \|x\|_{X^{-1}} + \|y\|_{X^{-1}}. \end{aligned}$$

What do we learn from this exercise? Recall that a vector z can alternatively be written as $X(X^{-1}z)$ so that the vector $\hat{z} = X^{-1}z$ tells you how to represent the vector z in the basis given by the columns of X . What the exercise tells us is that if we measure a vector by applying a known norm in a new basis, then that is also a norm.

With this insight, we can perform our convergence analysis:

$$\begin{aligned}
 & v^{(k)} - \psi_0 x_0 \\
 &= \\
 & A^k v^{(0)} / \lambda_0^k - \psi_0 x_0 \\
 &= \\
 & X \left(\begin{array}{c|ccccc} 1 & 0 & \cdots & 0 \\ \hline 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{array} \right) X^{-1} v^{(0)} - \psi_0 x_0 \\
 &= \\
 & X \left(\begin{array}{c|ccccc} 1 & 0 & \cdots & 0 \\ \hline 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{array} \right) y - \psi_0 x_0 \\
 &= \\
 & X \left(\begin{array}{c|ccccc} 0 & 0 & \cdots & 0 \\ \hline 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{array} \right) y
 \end{aligned}$$

Hence

$$X^{-1}(v^{(k)} - \psi_0 x_0) = \left(\begin{array}{c|ccccc} 0 & 0 & \cdots & 0 \\ \hline 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{array} \right) y$$

and

$$X^{-1}(v^{(k+1)} - \psi_0 x_0) = \left(\begin{array}{c|ccccc} 0 & 0 & \cdots & 0 \\ \hline 0 & \frac{\lambda_1}{\lambda_0} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \frac{\lambda_{m-1}}{\lambda_0} \end{array} \right) X^{-1}(v^{(k)} - \psi_0 x_0).$$

Now, let $\|\cdot\|$ be a p-norm and $\|\cdot\|_{X^{-1}}$ as defined in [Homework 9.3.2.1](#). Then

$$\begin{aligned}
 \|v^{(k+1)} - \psi_0 x_0\|_{X^{-1}} &= \|X^{-1}(v^{(k+1)} - \psi_0 x_0)\| \\
 &= \left\| \left(\begin{array}{c|ccccc} 0 & 0 & \cdots & 0 \\ \hline 0 & \frac{\lambda_1}{\lambda_0} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \frac{\lambda_{m-1}}{\lambda_0} \end{array} \right) X^{-1}(v^{(k)} - \psi_0 x_0) \right\| \\
 &\leq \left| \frac{\lambda_1}{\lambda_0} \right| \|X^{-1}(v^{(k)} - \psi_0 x_0)\| = \left| \frac{\lambda_1}{\lambda_0} \right| \|v^{(k)} - \psi_0 x_0\|_{X^{-1}}.
 \end{aligned}$$

This shows that, in this norm, the convergence of $v^{(k)}$ to $\psi_0 x_0$ is linear: The difference between current approximation, $v^{(k)}$, and the eventual vector in the direction of the desired eigenvector, ψx_0 , is reduced by at least a constant factor in each iteration.

Now, what if

$$|\lambda_0| = \dots = |\lambda_{n-1}| > |\lambda_n| \geq \dots \geq |\lambda_{m-1}|?$$

By extending the above analysis one can easily show that $v^{(k)}$ will converge to a vector in the subspace spanned by the eigenvectors associated with $\lambda_0, \dots, \lambda_{n-1}$.

An important special case is when $n = 2$: if A is real-valued then λ_0 may be complex-valued in which case its conjugate, $\bar{\lambda}_0$, is also an eigenvalue and hence has the same magnitude as λ_0 . We deduce that $v^{(k)}$ will always be in the space spanned by the eigenvectors corresponding to λ_0 and $\bar{\lambda}_0$.

9.3.3 The Inverse Power Method



YouTube: <https://www.youtube.com/watch?v=yrlYmNdYBCs>

Homework 9.3.3.1 Let $A \in \mathbb{C}^{m \times m}$ be nonsingular, and (λ, x) an eigenpair of A

Which of the follow is an eigenpair of A^{-1} :

- (λ, x) .
- $(\lambda, A^{-1}x)$.
- $(1/\lambda, A^{-1}x)$.
- $(1/\lambda, x)$.

Answer. $(1/\lambda, x)$.

Now justify your answer.

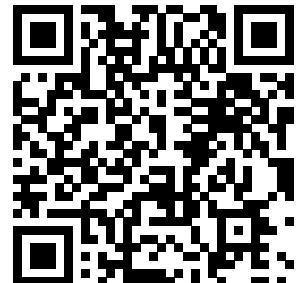
Solution. Since $Ax = \lambda x$ and A is nonsingular, we know that A^{-1} exists and $\lambda \neq 0$. Hence

$$\frac{1}{\lambda}x = A^{-1}x$$

which can be rewritten as

$$A^{-1}x = \frac{1}{\lambda}x.$$

We conclude that $(1/\lambda, x)$ is an eigenpair of A^{-1} .



YouTube: <https://www.youtube.com/watch?v=pKPMuiCNC2s>

The Power Method homes in on an eigenvector associated with the largest (in magnitude) eigenvalue. The Inverse Power Method homes in on an eigenvector associated with the smallest eigenvalue (in magnitude).

Once again, we assume that a given matrix $A \in \mathbb{C}^{m \times m}$ is diagonalizable so that there exist matrix X and diagonal matrix Λ such that $A = X\Lambda X^{-1}$. We further assume that $\Lambda = \text{diag}(\lambda_0, \dots, \lambda_{m-1})$ and

$$|\lambda_0| \geq |\lambda_1| \geq \dots \geq |\lambda_{m-2}| > |\lambda_{m-1}| > 0.$$

Notice that this means that A is nonsingular.

Clearly, if

$$|\lambda_0| \geq |\lambda_1| \geq \dots \geq |\lambda_{m-2}| > |\lambda_{m-1}| > 0,$$

then

$$\left| \frac{1}{\lambda_{m-1}} \right| > \left| \frac{1}{\lambda_{m-2}} \right| \geq \left| \frac{1}{\lambda_{m-3}} \right| \geq \dots \geq \left| \frac{1}{\lambda_0} \right|.$$

Thus, an eigenvector associated with the smallest (in magnitude) eigenvalue of A is an eigenvector associated with the largest (in magnitude) eigenvalue of A^{-1} .



YouTube: <https://www.youtube.com/watch?v=D6KF28ycRB0>

This suggest the following naive iteration (which mirrors the second attempt for the Power Method in Subsubsection 9.3.1.2, but iterating with A^{-1}):

```

for  $k = 0, \dots$ 
     $v^{(k+1)} = A^{-1}v^{(k)}$ 
     $v^{(k+1)} = \lambda_{m-1}v^{(k+1)}$ 
endfor

```

From the analysis of the convergence of in Subsection 9.3.2 for the Power Method algorithm, we conclude that now

$$\|v^{(k+1)} - \psi_{m-1}x_{m-1}\|_{X^{-1}} \leq \left| \frac{\lambda_{m-1}}{\lambda_{m-2}} \right| \|v^{(k)} - \psi_{m-1}x_{m-1}\|_{X^{-1}}.$$

A more practical Inverse Power Method algorithm is given by

```

Pick  $v^{(0)}$  of unit length
for  $k = 0, \dots$ 
     $v^{(k+1)} = A^{-1}v^{(k)}$ 
     $v^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|$ 
endfor

```

We would probably want to factor $PA = LU$ (LU factorization with partial pivoting) once and solve $L(Uv^{(k+1)}) = Pv^{(k)}$ rather than multiplying with A^{-1} .

9.3.4 The Rayleigh Quotient Iteration



YouTube: <https://www.youtube.com/watch?v=700JcvYxbxM>

A basic idea that allows one to accelerate the convergence of the inverse iteration is captured by the following exercise:

Homework 9.3.4.1 Let $A \in \mathbb{C}^{m \times m}$, $\rho \in \mathbb{C}$, and (λ, x) an eigenpair of A .

Which of the follow is an eigenpair of the **shifted** matrix $A - \rho I$:

- (λ, x) .
- $(\lambda, A^{-1}x)$.
- $(\lambda - \rho, x)$.
- $(1/(\lambda - \rho), x)$.

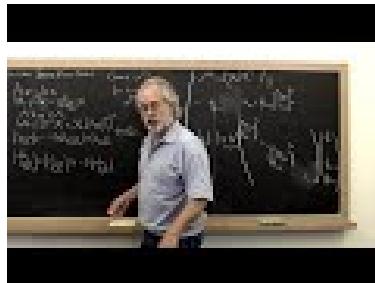
Answer. $(\lambda - \rho, x)$.

Now justify your answer.

Solution. Let $Ax = \lambda x$. Then

$$(A - \rho I)x = Ax - \rho x = \lambda x - \rho x = (\lambda - \rho)x.$$

We conclude that $(\lambda - \rho, x)$ is an eigenpair of $A - \rho I$.



YouTube: <https://www.youtube.com/watch?v=btFWxkXkXZ8>

The matrix $A - \rho I$ is referred to as the matrix A that has been "shifted" by ρ . What the next lemma captures is that shifting A by ρ shifts the spectrum of A by ρ :

Lemma 9.3.4.1 *Let $A \in \mathbb{C}^{m \times m}$, $A = X\Lambda X^{-1}$ and $\rho \in \mathbb{C}$. Then $A - \rho I = X(\Lambda - \rho I)X^{-1}$.*

Homework 9.3.4.2 Prove Lemma 9.3.4.1.

Solution.

$$A - \rho I = X\Lambda X^{-1} - \rho XX^{-1} = X(\Lambda - \rho I)X^{-1}.$$

This suggests the following (naive) iteration: Pick a value ρ close to λ_{m-1} . Iterate

```

Pick  $v^{(0)}$  of unit length
for  $k = 0, \dots$ 
     $v^{(k+1)} = (A - \rho I)^{-1}v^{(k)}$ 
     $v^{(k+1)} = (\lambda_{m-1} - \rho)v^{(k+1)}$ 
endfor

```

Of course one would solve $(A - \rho I)v^{(k+1)} = v^{(k)}$ rather than computing and applying the inverse of A .

If we index the eigenvalues so that

$$|\lambda_{m-1} - \rho| < |\lambda_{m-2} - \rho| \leq \dots \leq |\lambda_0 - \rho|$$

then

$$\|v^{(k+1)} - \psi_{m-1}x_{m-1}\|_{X^{-1}} \leq \left| \frac{\lambda_{m-1} - \rho}{\lambda_{m-2} - \rho} \right| \|v^{(k)} - \psi_{m-1}x_{m-1}\|_{X^{-1}}.$$

The closer to λ_{m-1} the shift ρ is chosen, the more favorable the ratio (constant) that dictates the linear convergence of this modified Inverse Power Method.



YouTube: <https://www.youtube.com/watch?v=fCDYbunugKk>

A more practical algorithm is given by

```

    Pick  $v^{(0)}$  of unit length
    for  $k = 0, \dots$ 
         $v^{(k+1)} = (A - \rho I)^{-1}v^{(k)}$ 
         $v^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|$ 
    endfor

```

where instead of multiplying by the inverse one would want to solve the linear system $(A - \rho I)v^{(k+1)} = v^{(k)}$ instead.

The question now becomes how to chose ρ so that it is a good guess for λ_{m-1} . Often an application inherently supplies a reasonable approximation for the smallest eigenvalue or an eigenvalue of particular interest. Alternatively, we know that eventually $v^{(k)}$ becomes a good approximation for x_{m-1} and therefore the Rayleigh quotient gives us a way to find a good approximation for λ_{m-1} . This suggests the (naive) Rayleigh-quotient iteration:

```

    Pick  $v^{(0)}$  of unit length
    for  $k = 0, \dots$ 
         $\rho_k = v^{(k)H}Av^{(k)} / (v^{(k)H}v^{(k)})$ 
         $v^{(k+1)} = (A - \rho_k I)^{-1}v^{(k)}$ 
         $v^{(k+1)} = (\lambda_{m-1} - \rho_k)v^{(k+1)}$ 
    endfor

```

Here λ_{m-1} is the eigenvalue to which the method eventually converges.

$$\|v^{(k+1)} - \psi_{m-1}x_{m-1}\|_{X^{-1}} \leq \left| \frac{\lambda_{m-1} - \rho_k}{\lambda_{m-2} - \rho_k} \right| \|v^{(k)} - \psi_{m-1}x_{m-1}\|_{X^{-1}}$$

with

$$\lim_{k \rightarrow \infty} (\lambda_{m-1} - \rho_k) = 0$$

which means superlinear convergence is observed. In fact, it can be shown that once k is large enough

$$\|v^{(k+1)} - \psi_{m-1}x_{m-1}\|_{X^{-1}} \leq C\|v^{(k)} - \psi_{m-1}x_{m-1}\|_{X^{-1}}^2,$$

thus achieving quadratic convergence. Roughly speaking this means that every iteration doubles the number of correct digits in the current approximation. To prove this, one shows that $|\lambda_{m-1} - \rho_k| \leq K\|v^{(k)} - \psi_{m-1}x_{m-1}\|_{X^{-1}}$ for some constant K . Details go beyond this discussion.

Better yet, it can be shown that if A is Hermitian, then (once k is large enough)

$$\|v^{(k+1)} - \psi_{m-1}x_{m-1}\| \leq C\|v^{(k)} - \psi_{m-1}x_{m-1}\|^3$$

for some constant C and hence the naive Rayleigh Quotient Iteration achieves cubic convergence for Hermitian matrices. Here our norm $\|\cdot\|_{X^{-1}}$ becomes any p-norm since the Spectral Decomposition Theorem tells us that for Hermitian matrices X can be taken to equal a unitary matrix. Roughly speaking this means that every iteration triples the number of correct digits in the current approximation. This is mind-boggling fast convergence!

A practical Rayleigh Quotient Iteration is given by

```

 $v^{(0)} = v^{(0)} / \|v^{(0)}\|_2$ 
for  $k = 0, \dots$ 
 $\rho_k = v^{(k)H} A v^{(k)}$  (Now  $\|v^{(k)}\|_2 = 1$ )
 $v^{(k+1)} = (A - \rho_k I)^{-1} v^{(k)}$ 
 $v^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|$ 
endfor

```

Remark 9.3.4.2 A concern with the (Shifted) Inverse Power Method and Rayleigh Quotient Iteration is that the matrix with which one solves is likely nearly singular. It turns out that this actually helps: the error that is amplified most is in the direction of the eigenvector associated with the smallest eigenvalue (after shifting, if appropriate).

9.3.5 Discussion

To summarize this section:

- The Power Method finds the eigenvector associated with the largest eigenvalue (in magnitude). It requires a matrix-vector multiplication for each iteration, thus costing approximately $2m^2$ flops per iteration if A is a dense $m \times m$ matrix. The convergence is linear.
- The Inverse Power Method finds the eigenvector associated with the smallest eigenvalue (in magnitude). It requires the solution of a linear system for each iteration. By performing an LU factorization with partial pivoting, the investment of an initial $O(m^3)$ expense then reduces the cost per iteration to approximately $2m^2$ flops. if A is a dense $m \times m$ matrix. The convergence is linear.
- The Rayleigh Quotient Iteration finds an eigenvector, but with which eigenvalue it is associated is not clear from the start. It requires the solution of a linear system for each iteration. If computed via an LU factorization with partial pivoting, the cost per iteration is $O(m^3)$ per iteration, if A is a dense $m \times m$ matrix. The convergence is quadratic if A is not Hermitian, and cubic if it is.

The cost of these methods is greatly reduced if the matrix is sparse, in which case each iteration may require as little as $O(m)$ per iteration.

9.4 Enrichments

9.4.1 How to compute the eigenvalues of a 2×2 matrix

We have noted that finding the eigenvalues of a 2×2 matrix requires the solution to the characteristic polynomial. In particular, if a 2×2 matrix A is real-valued and

$$A = \begin{pmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{pmatrix}$$

then

$$\det(\lambda I - A) = (\lambda - \alpha_{00})(\lambda - \alpha_{11}) - \alpha_{10}\alpha_{01} = \lambda^2 \underbrace{-(\alpha_{00} + \alpha_{11})}_{\beta} \lambda + \underbrace{(\alpha_{00}\alpha_{11} - \alpha_{10}\alpha_{01})}_{\gamma}.$$

It is then tempting to use the quadratic formula to find the roots: $\lambda_0 = \frac{-\beta + \sqrt{\beta^2 - 4\gamma}}{2}$ and $\lambda_1 = \frac{-\beta - \sqrt{\beta^2 - 4\gamma}}{2}$. However, as discussed in [Subsection C.0.2](#), one of these formulae may cause catastrophic cancellation, if γ is small. When is γ small? When $\alpha_{00}\alpha_{11} - \alpha_{10}\alpha_{01}$ is small. In other words, when the determinant of A is small or, equivalently, when A has a small eigenvalue.

In the next week, we will discuss the QR algorithm for computing the Spectral Decomposition of a Hermitian matrix. We do not discuss the QR algorithm for computing the Schur Decomposition of a $m \times m$ non-Hermitian matrix, which uses the eigenvalues of

$$\begin{pmatrix} \alpha_{m-2,m-2} & \alpha_{m-2,m-1} \\ \alpha_{m-1,m-1} & \alpha_{m-1,m-1} \end{pmatrix}$$

to "shift" the matrix. (What this means will become clear next week.) This happened to come up in Robert's dissertation work. Making the "rookie mistake" of not avoiding catastrophic cancellation when computing the roots of a quadratic polynomial cost him three weeks of his life (debugging his code), since the algorithm that resulted did not converge correctly... Don't repeat his mistakes!

9.5 Wrap Up

9.5.1 Additional homework

Homework 9.5.1.1 Let $\|\cdot\|$ be matrix norm induced by a vector norm $\|\cdot\|$. Prove that for any $A \in \mathbb{C}^{m \times m}$, the spectral radius, $\rho(A)$ satisfies $\rho(A) \leq \|A\|$.

Some results in linear algebra depend on there existing a consistent matrix norm $\|\cdot\|$ such that $\|A\| < 1$. The following exercise implies that one can alternatively show that the spectral radius is bounded by one: $\rho(A) < 1$.

Homework 9.5.1.2 Given a matrix $A \in \mathbb{C}^{m \times m}$ and $\epsilon > 0$, there exists a consistent matrix norm $\|\cdot\|$ such that $\|A\| \leq \rho(A) + \epsilon$.

9.5.2 Summary

Definition 9.5.2.1 Eigenvalue, eigenvector, and eigenpair. Let $A \in \mathbb{C}^{m \times m}$. Then $\lambda \in \mathbb{C}$ and nonzero $x \in \mathbb{C}^m$ are said to be an eigenvalue and corresponding eigenvector if $Ax = \lambda x$. The tuple (λ, x) is said to be an eigenpair. \diamond

For $A \in \mathbb{C}^{m \times m}$, the following are equivalent statements:

- A is nonsingular.
- A has linearly independent columns.
- There does not exist $x \neq 0$ such that $Ax = 0$.
- $\mathcal{N}(A) = \{0\}$. (The null space of A is trivial.)
- $\dim(\mathcal{N}(A)) = 0$.
- $\det(A) \neq 0$.

For $A \in \mathbb{C}^{m \times m}$, the following are equivalent statements:

- λ is an eigenvalue of A
- $(\lambda I - A)$ is singular.
- $(\lambda I - A)$ has linearly dependent columns.
- There exists $x \neq 0$ such that $(\lambda I - A)x = 0$.
- The null space of $\lambda I - A$ is nontrivial.
- $\dim(\mathcal{N}(\lambda I - A)) > 0$.
- $\det(\lambda I - A) = 0$.

Definition 9.5.2.2 Spectrum of a matrix. The set of all eigenvalues of A is denoted by $\Lambda(A)$ and is called the spectrum of A . \diamond

Definition 9.5.2.3 Spectral radius. The spectral radius of A , $\rho(A)$, equals the magnitude of the largest eigenvalue, in magnitude:

$$\rho(A) = \max_{\lambda \in \Lambda(A)} |\lambda|.$$

\diamond

Theorem 9.5.2.4 Gershgorin Disk Theorem. Let $A \in \mathbb{C}^{m \times m}$,

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,m-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,m-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,m-1} \end{pmatrix},$$

$$\rho_i(A) = \sum_{j \neq i} |\alpha_{i,j}|,$$

and

$$R_i(A) = \{x \text{ s.t. } |x - \alpha_{i,i}| \leq \rho_i\}.$$

In other words, $\rho_i(A)$ equals the sum of the absolute values of the off diagonal elements of A in row i and $R_i(A)$ equals the set of all points in the complex plane that are within a distance ρ_i of diagonal element $\alpha_{i,i}$. Then

$$\Lambda(A) \subset \cup_i R_i(A).$$

In other words, every eigenvalue lies in one of the disks of radius $\rho_i(A)$ around diagonal element $\alpha_{i,i}$.

Corollary 9.5.2.5 Let A and $R_i(A)$ be as defined in [Theorem 9.5.2.4](#). Let K and K^C be disjoint subsets of $\{0, \dots, m-1\}$ such that $K \cup K^C = \{0, \dots, m-1\}$. In other words, let K be a subset of $\{0, \dots, m-1\}$ and K^C its complement. If

$$(\cup_{k \in K} R_k(A)) \cap (\cup_{j \in K^C} R_j(A)) = \emptyset$$

then $\cup_{k \in K} R_k(A)$ contains exactly $|K|$ eigenvalues of A (multiplicity counted). In other words, if $\cup_{k \in K} R_k(A)$ does not intersect with any of the other disks, then it contains as many eigenvalues of A (multiplicity counted) as there are elements of K .

Some useful facts for $A \in \mathbb{C}^{m \times m}$:

- $0 \in \Lambda(A)$ if and only A is singular.
- The eigenvectors corresponding to distinct eigenvectors are linearly independent.
- Let A be nonsingular. Then (λ, x) is an eigenpair of A if and only if $(1/\lambda, x)$ is an eigenpair of A^{-1} .
- (λ, x) is an eigenpair of A if and only if $(\lambda - \rho, x)$ is an eigenpair of $A - \rho I$.

Some useful facts for Hermitian $A \in \mathbb{C}^{m \times m}$:

- All eigenvalues are real-valued.
- A is HPD if and only if all its eigenvalues are positive.
- If (λ, x) and (μ, y) are both eigenpairs of Hermitian A , then x and y are orthogonal.

Definition 9.5.2.6 The determinant of

$$A = \begin{pmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{pmatrix}$$

is given by

$$\det(A) = \alpha_{00}\alpha_{11} - \alpha_{10}\alpha_{01}.$$

◇

The characteristic polynomial of

$$A = \begin{pmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{pmatrix}$$

is given by

$$\det(\lambda - IA) = (\lambda - \alpha_{00})(\lambda - \alpha_{11}) - \alpha_{10}\alpha_{01}.$$

This is a second degree polynomial in λ and has two roots (multiplicity counted). The eigenvalues of A equal the roots of this characteristic polynomial.

The characteristic polynomial of $A \in \mathbb{C}^{m \times m}$ is given by

$$\det(\lambda - IA).$$

This is a polynomial in λ of degree m and has m roots (multiplicity counted). The eigenvalues of A equal the roots of this characteristic polynomial. Hence, A has m eigenvalues (multiplicity counted).

Definition 9.5.2.7 Algebraic multiplicity of an eigenvalue. Let $A \in \mathbb{C}^{m \times m}$ and $p_m(\lambda)$ its characteristic polynomial. Then the (algebraic) multiplicity of eigenvalue λ_i equals the multiplicity of the corresponding root of the polynomial. \diamond

If

$$p_m(\chi) = \alpha_0 + \alpha_1\chi + \cdots + \alpha_{m-1}\chi^{m-1} + \chi^m$$

and

$$A = \begin{pmatrix} -\alpha_{n-1} & -\alpha_{n-2} & -\alpha_{n-3} & \cdots & -\alpha_1 & -\alpha_0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

then

$$p_m(\lambda) = \det(\lambda I - A).$$

Corollary 9.5.2.8 If $A \in \mathbb{R}^{m \times m}$ is real-valued then some or all of its eigenvalues may be complex-valued. If eigenvalue λ is complex-valued, then its conjugate, $\bar{\lambda}$, is also an eigenvalue. Indeed, the complex eigenvalues of a real-valued matrix come in complex pairs.

Corollary 9.5.2.9 If $A \in \mathbb{R}^{m \times m}$ is real-valued and m is odd, then at least one of the eigenvalues of A is real-valued.

Let λ be an eigenvalue of $A \in \mathbb{C}^{m \times m}$ and

$$\mathcal{E}_\lambda(A) = \{x \in \mathbb{C}^m | Ax = \lambda x\}.$$

bet the set of all eigenvectors of A associated with λ plus the zero vector (which is not considered an eigenvector). This set is a subspace.

The elements on the diagonal of a diagonal matrix are its eigenvalues. The elements on the diagonal of a triangular matrix are its eigenvalues.

Definition 9.5.2.10 Given a nonsingular matrix Y , the transformation $Y^{-1}AY$ is called a similarity transformation (applied to matrix A). \diamond

Let $A, B, Y \in \mathbb{C}^{m \times m}$, where Y is nonsingular, $B = Y^{-1}AY$, and (λ, x) an eigenpair of A . Then $(\lambda, Y^{-1}x)$ is an eigenpair of B .

Theorem 9.5.2.11 Let $A, Y, B \in \mathbb{C}^{m \times m}$, assume Y is nonsingular, and let $B = Y^{-1}AY$. Then $\Lambda(A) = \Lambda(B)$.

Definition 9.5.2.12 Given a nonsingular matrix Q the transformation $Q^H A Q$ is called a unitary similarity transformation (applied to matrix A). \diamond

Theorem 9.5.2.13 Schur Decomposition Theorem. Let $A \in \mathbb{C}^{m \times m}$. Then there exist a unitary matrix Q and upper triangular matrix U such that $A = QUQ^H$. This decomposition is called the Schur decomposition of matrix A .

Theorem 9.5.2.14 Spectral Decomposition Theorem. Let $A \in \mathbb{C}^{m \times m}$ be Hermitian. Then there exist a unitary matrix Q and diagonal matrix $D \in \mathbb{R}^{m \times m}$ such that $A = QDQ^H$. This decomposition is called the spectral decomposition of matrix A .

Theorem 9.5.2.15 Let $A \in \mathbb{C}^{m \times m}$ be of form $A = \begin{pmatrix} A_{TL} & A_{TR} \\ 0 & A_{BR} \end{pmatrix}$, where A_{TL} and A_{BR} are square submatrices. Then $\Lambda(A) = \Lambda(A_{TL}) \cup \Lambda(A_{BR})$.

Definition 9.5.2.16 Diagonalizable matrix. A matrix $A \in \mathbb{C}^{m \times m}$ is said to be diagonalizable if and only if there exists a nonsingular matrix X and diagonal matrix D such that $X^{-1}AX = D$. \diamond

Theorem 9.5.2.17 A matrix $A \in \mathbb{C}^{m \times m}$ is diagonalizable if and only if it has m linearly independent eigenvectors.

If $A \in \mathbb{C}^{m \times m}$ has distinct eigenvalues, then it is diagonalizable.

Definition 9.5.2.18 Defective matrix. A matrix $A \in \mathbb{C}^{m \times m}$ that does not have m linearly independent eigenvectors is said to be defective. \diamond

Corollary 9.5.2.19 Matrix $A \in \mathbb{C}^{m \times m}$ is diagonalizable if and only if it is not defective.

Definition 9.5.2.20 Geometric multiplicity. Let $\lambda \in \Lambda(A)$. Then the geometric multiplicity of λ is defined to be the dimension of $\mathcal{E}_\lambda(A)$ defined by

$$\mathcal{E}_\lambda(A) = \{x \in \mathbb{C}^m | Ax = \lambda x\}.$$

In other words, the geometric multiplicity of λ equals the number of linearly independent eigenvectors that are associated with λ . \diamond

Definition 9.5.2.21 Jordan Block. Define the $k \times k$ Jordan block with eigenvalue μ as

$$J_k(\mu) = \begin{pmatrix} \mu & 1 & 0 & \cdots & 0 & 0 \\ 0 & \mu & 1 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & \mu & 1 \\ 0 & 0 & 0 & \cdots & 0 & \mu \end{pmatrix}$$

\diamond

Theorem 9.5.2.22 Jordan Canonical Form Theorem. Let the eigenvalues of $A \in \mathbb{C}^{m \times m}$ be given by $\lambda_0, \lambda_1, \dots, \lambda_{k-1}$, where an eigenvalue is listed as many times as its geometric multiplicity. There exists a nonsingular matrix X such that

$$X^{-1}AX = \left(\begin{array}{c|c|c|c} J_{m_0}(\lambda_0) & 0 & \cdots & 0 \\ \hline 0 & J_{m_1}(\lambda_1) & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & J_{m_{k-1}}(\lambda_{k-1}) \end{array} \right).$$

A practical Power Method for finding the eigenvector associated with the largest eigenvalue (in magnitude):

```

Pick  $v^{(0)}$  of unit length
for  $k = 0, \dots$ 
     $v^{(k+1)} = Av^{(k)}$ 
     $v^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|$ 
endfor

```

Definition 9.5.2.23 Rayleigh quotient. If $A \in \mathbb{C}^{m \times m}$ and $x \neq 0 \in \mathbb{C}^m$ then

$$\frac{x^H Ax}{x^H x}$$

is known as the **Rayleigh quotient**. ◊

If x is an eigenvector of A , then

$$\frac{x^H Ax}{x^H x}$$

is the associated eigenvalue.

Definition 9.5.2.24 Convergence of a sequence of scalars. Let $\alpha_0, \alpha_1, \alpha_2, \dots \in \mathbb{R}$ be an infinite sequence of scalars. Then α_k is said to converge to α if

$$\lim_{k \rightarrow \infty} |\alpha_k - \alpha| = 0.$$

◊

Definition 9.5.2.25 Convergence of a sequence of vectors. Let $x_0, x_1, x_2, \dots \in \mathbb{C}^m$ be an infinite sequence of vectors. Then x_k is said to converge to x if for any norm $\|\cdot\|$

$$\lim_{k \rightarrow \infty} \|x_k - x\| = 0.$$

◊

Definition 9.5.2.26 Rate of convergence. Let $\alpha_0, \alpha_1, \alpha_2, \dots \in \mathbb{R}$ be an infinite sequence of scalars that converges to α . Then

- α_k is said to converge linearly to α if for sufficiently large k

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|$$

for some constant $C < 1$.

- α_k is said to converge superlinearly to α if for sufficiently large k

$$|\alpha_{k+1} - \alpha| \leq C_k |\alpha_k - \alpha|$$

with $C_k \rightarrow 0$.

- α_k is said to converge quadratically to α if for sufficiently large k

$$|\alpha_{k+1} - \alpha| \leq C |\alpha_k - \alpha|^2$$

for some constant C .

- α_k is said to converge superquadratically to α if for sufficiently large k

$$|\alpha_{k+1} - \alpha| \leq C_k |\alpha_k - \alpha|^2$$

with $C_k \rightarrow 0$.

- α_k is said to converge cubically to α if for large enough k

$$|\alpha_{k+1} - \alpha| \leq C |\alpha_k - \alpha|^3$$

for some constant C .

◊

The convergence of the Power Method is linear.

A practical Inverse Power Method for finding the eigenvector associated with the smallest eigenvalue (in magnitude):

```

Pick  $v^{(0)}$  of unit length
for  $k = 0, \dots$ 
     $v^{(k+1)} = A^{-1}v^{(k)}$ 
     $v^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|$ 
endfor

```

The convergence of the Inverse Power Method is linear.

A practical Rayleigh quation iteration for finding the eigenvector associated with the smallest eigenvalue (in magnitude):

```

Pick  $v^{(0)}$  of unit length
for  $k = 0, \dots$ 
     $\rho_k = v^{(k)H} A v^{(k)}$ 
     $v^{(k+1)} = (A - \rho_k I)^{-1} v^{(k)}$ 
     $v^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|$ 
endfor

```

The convergence of the Rayleigh Quotient Iteration is quadratic (eventually, the number of correct digits doubles in each iteration). If A is Hermitian, the convergence is cubic (eventually, the number of correct digits triples in each iteration).

Week 10

Practical Solution of the Hermitian Eigenvalue Problem

10.1 Opening

10.1.1 Subspace iteration with a Hermitian matrix



YouTube: <https://www.youtube.com/watch?v=kwJ6HMSLv1U>

The idea behind subspace iteration is to perform the Power Method with more than one vector in order to converge to (a subspace spanned by) the eigenvectors associated with a set of eigenvalues.

We continue our discussion by restricting ourselves to the case where $A \in \mathbb{C}^{m \times m}$ is Hermitian. Why? Because the eigenvectors associated with distinct eigenvalues of a Hermitian matrix are mutually orthogonal (and can be chosen to be orthonormal), which will simplify our discussion. Here we repeat the Power Method:

```
v0 := random vector  
v0(0) := v0 / ||v0||2           normalize to have length one  
for k := 0, ...  
    v0 := A v0(k)  
    v0(k+1) := v0 / ||v0||2       normalize to have length one  
endfor
```

In previous discussion, we used $v^{(k)}$ for the current approximation to the eigenvector. We

now add the subscript to it, $v_0^{(k)}$, because we will shortly start iterating with multiple vectors.

Homework 10.1.1.1 You may want to start by executing `git pull` to update your directory `Assignments`.

Examine [Assignments/Week10/matlab/PowerMethod.m](#) which implements

```
[ lambda_0, v0 ] = PowerMethod( A, x, maxiters, illustrate, delay )
```

This routine implements the Power Method, starting with a vector x for a maximum number of iterations `maxiters` or until convergence, whichever comes first. To test it, execute the script in [Assignments/Week10/matlab/test_SubspaceIteration.m](#) which uses the Power Method to compute the largest eigenvalue (in magnitude) and corresponding eigenvector for an $m \times m$ Hermitian matrix A with eigenvalues $1, \dots, m$.

Be sure to click on "Figure 1" to see the graph that is created.

Solution. Watch the video regarding this problem on YouTube: <https://youtu.be/8Bgf1tJeMmg>. (embedding a video in a solution seems to cause PreTeXt trouble...)



YouTube: <https://www.youtube.com/watch?v=wmuWfjwtgcI>

Recall that when we analyzed the convergence of the Power Method, we commented on the fact that the method converges to an eigenvector associated with the largest eigenvalue (in magnitude) if two conditions are met:

- $|\lambda_0| > |\lambda_1|$.
- $v_0^{(0)}$ has a component in the direction of the eigenvector, x_0 , associated with λ_0 .

A second initial vector, $v_1^{(0)}$, does not have a component in the direction of x_0 if it is orthogonal to x_0 . So, if we know x_0 , then we can pick a random vector, subtract out the component in the direction of x_0 , and make this our vector $v_1^{(0)}$ with which we should be able to execute the Power Method to find an eigenvector, x_1 , associated with the eigenvalue that has the second largest magnitude, λ_1 . If we then start the Power Method with this new vector (and don't introduce roundoff error in a way that introduces a component in the direction of x_0), then the iteration will home in on a vector associated with λ_1 (provided A is Hermitian, $|\lambda_1| > |\lambda_2|$, and $v_1^{(0)}$ has a component in the direction of x_1 .) This iteration

would look like

```

 $x_0 := x_0 / \|x_0\|_2$  normalize known eigenvector  $x_0$  to have length one
 $v_1 := \text{random vector}$ 
 $v_1 := v_1 - x_0^H v_1 x_0$  make sure the vector is orthogonal to  $x_0$ 
 $v_1^{(0)} := v_1 / \|v_1\|_2$  normalize to have length one
for  $k := 0, \dots$ 
     $v_1 := A v_1^{(k)}$ 
     $v_1^{(k+1)} := v_1 / \|v_1\|_2$  normalize to have length one
endfor

```

Homework 10.1.1.2 Copy [Assignments/Week10/matLab/PowerMethod.m](#) into `PowerMethodLambda1.m`. Modify it by adding an input parameter x_0 , which is an eigenvector associated with λ_0 (the eigenvalue with largest magnitude).

```
[ lambda_1, v1 ] = PowerMethodLambda1( A, x, x0, maxiters, illustrate, delay )
```

The new function should subtract this vector from the initial random vector as in the above algorithm.

Modify the appropriate line in [Assignments/Week10/matlab/test_SubspaceIteration.m](#), changing (0) to (1), and use it to examine the convergence of the method.

What do you observe?

Solution.

- [Assignments/Week10/answers/PowerMethodLambda1.m](#)

Watch the video regarding this problem on YouTube: <https://youtu.be/48HnBJmQhX8>. (embedding a video in a solution seems to cause PreTeXt trouble...)



YouTube: <https://www.youtube.com/watch?v=0vBFQ84jTMw>

Because we should be concerned about the introduction of a component in the direction of x_0 due to roundoff error, we may want to reorthogonalize with respect to x_0 in each

iteration:

$x_0 := x_0 / \ x_0\ _2$	normalize known eigenvector x_0 to have length one
$v_1 := \text{random vector}$	
$v_1 := v_1 - x_0^H v_1 x_0$	make sure the vector is orthogonal to x_0
$v_1^{(0)} := v_1 / \ v_1\ _2$	normalize to have length one
for $k := 0, \dots$	
$v_1 := A v_1^{(k)}$	
$v_1 := v_1 - x_0^H v_1 x_0$	make sure the vector is orthogonal to x_0
$v_1^{(k+1)} := v_1 / \ v_1\ _2$	normalize to have length one
endfor	

Homework 10.1.1.3 Copy PowerMethodLambda1.m into PowerMethodLambda1Reorth.m and modify it to reorthogonalize with respect to x_0 :

```
[ lambda_1, v1 ] = PowerMethodLambda1Reorth( A, x, v0, maxiters, illustrate, delay );
```

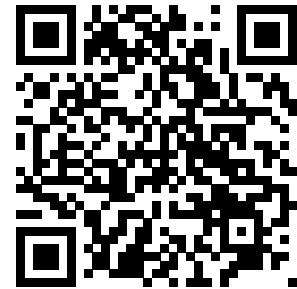
Modify the appropriate line in [Assignments/Week10/matlab/test_SubspaceIteration.m](#), changing (0) to (1), and use it to examine the convergence of the method.

What do you observe?

Solution.

- [Assignments/Week10/answers/PowerMethodLambda1Reorth.m](#)

Watch the video regarding this problem on YouTube: <https://youtu.be/YmZc2oq02kA>. (embedding a video in a solution seems to cause PreTeXt trouble...)



YouTube: <https://www.youtube.com/watch?v=751FAyKch1s>

We now observe that the steps that normalize x_0 to have unit length and then subtract out the component of v_1 in the direction of x_0 , normalizing the result, are exactly those performed by the Gram-Schmidt process. More generally, it is equivalent to computing the QR factorization of the matrix $\begin{pmatrix} x_0 & v_1 \end{pmatrix}$. This suggests the algorithm

$v_1 := \text{random vector}$	
$((x_0 \mid v_1^{(0)}), R) := \text{QR}((x_0 \mid v_1))$	
for $k := 0, \dots$	
$((x_0 \mid v_1^{(k+1)}), R) := \text{QR}((x_0 \mid A v_1^{(k)}))$	
endfor	

Obviously, this redundantly normalizes x_0 . It puts us on the path of a practical algorithm for computing the eigenvectors associated with λ_0 and λ_1 .

The problem is that we typically don't know x_0 up front. Rather than first using the power method to compute it, we can instead iterate with two random vectors, where the first converges to a vector associated with λ_0 and the second to one associated with λ_1 :

```

 $v_0 :=$  random vector
 $v_1 :=$  random vector
 $(\begin{pmatrix} v_0^{(0)} & v_1^{(0)} \end{pmatrix}, R) := \text{QR}(\begin{pmatrix} v_0 & v_1 \end{pmatrix})$ 
for  $k := 0, \dots$ 
     $(\begin{pmatrix} v_0^{(k+1)} & v_1^{(k+1)} \end{pmatrix}, R) := \text{QR}(A \begin{pmatrix} v_0^{(k)} & v_1^{(k)} \end{pmatrix})$ 
endfor
```

We observe:

- If $|\lambda_0| > |\lambda_1|$, the vectors $v_0^{(k)}$ will converge linearly to a vector in the direction of x_0 at a rate dictated by the ratio $|\lambda_1|/|\lambda_0|$.
- If $|\lambda_0| > |\lambda_1| > |\lambda_2|$, the vectors $v_1^{(k)}$ will converge linearly to a vector in the direction of x_1 at a rate dictated by the ratio $|\lambda_2|/|\lambda_1|$.
- If $|\lambda_0| \geq |\lambda_1| > |\lambda_2|$ then $\text{Span}(\{v_0^{(k)}, v_1^{(k)}\})$ will eventually start approximating the subspace $\text{Span}(\{x_0, x_1\})$.



YouTube: <https://www.youtube.com/watch?v=lhBEjMmWLia>

What we have described is a special case of **subspace iteration**. The associated eigenvalue can be approximated via the Rayleigh quotient:

$$\lambda_0 \approx \lambda_0^{(k)} = v_0^{(k)H} A v_0^{(k)} \text{ and } \lambda_1 \approx \lambda_1^{(k)} = v_1^{(k)H} A v_1^{(k)}$$

Alternatively,

$$A^{(k)} = \left(\begin{pmatrix} v_0^{(k)} & v_1^{(k)} \end{pmatrix}^H A \begin{pmatrix} v_0^{(k)} & v_1^{(k)} \end{pmatrix} \right) \text{ converges to } \left(\begin{array}{c|c} \lambda_0 & 0 \\ \hline 0 & \lambda_1 \end{array} \right)$$

if A is Hermitian, $|\lambda_1| > |\lambda_2|$, and $v^{(0)}$ and $v^{(1)}$ have components in the directions of x_0 and x_1 , respectively.

The natural extention of these observations is to iterate with n vectors:

```

 $\hat{V} :=$  random  $m \times n$  matrix
 $(\hat{V}^{(0)}, R) := \text{QR}(\hat{V})$ 
 $A^{(0)} = V^{(0)H} A V^{(0)}$ 
for  $k := 0, \dots$ 
     $(\hat{V}^{(k+1)}, R) := \text{QR}(A \hat{V}^{(k)})$ 
     $A^{(k+1)} = \hat{V}^{(k+1)H} A \hat{V}^{(k+1)}$ 
endfor

```

By extending the reasoning given so far in this unit, if

- A is Hermitian,
- $|\lambda_0| > |\lambda_1| > \dots > |\lambda_{n-1}| > |\lambda_n|$, and
- each v_j has a component in the direction of x_j , an eigenvector associated with λ_j ,

then each $v_j^{(j)}$ will converge to a vector in the direction x_j . The rate with which the component in the direction of x_p , $0 \leq p < n$, is removed from $v_j^{(k)}$, $n \leq j < m$, is dictated by the ratio $|\lambda_p|/|\lambda_j|$.

If some of the eigenvalues have equal magnitude, then the corresponding columns of $\hat{V}^{(k)}$ will eventually form a basis for the subspace spanned by the eigenvectors associated with those eigenvalues.

Homework 10.1.1.4 Copy `PowerMethodLambda1Reorth.m` into `SubspaceIteration.m` and modify it to work with an $m \times n$ matrix V :

```
[ Lambda, V ] = SubspaceIteration( A, V, maxiters, illustrate, delay );
```

Modify the appropriate line in [Assignments/Week10/matlab/test_SubspaceIteration.m](#), changing (0) to (1), and use it to examine the convergence of the method.

What do you observe?

Solution.

- [Assignments/Week10/answers/SubspaceIteration.m](#)

Watch the video regarding this problem on YouTube: <https://youtu.be/Er7jGYS0HbE>. (embedding a video in a solution seems to cause PreTeXt trouble...)

10.1.2 Overview

- 10.1 Opening
 - 10.1.1 Subspace iteration with a Hermitian matrix
 - 10.1.2 Overview
 - 10.1.3 What you will learn

- 10.2 From Power Method to a simple QR algorithm
 - 10.2.1 A simple QR algorithm
 - 10.2.2 A simple shifted QR algorithm
 - 10.2.3 Deflating the problem
 - 10.2.4 Cost of a simple QR algorithm
- 10.3 A Practical Hermitian QR Algorithm
 - 10.3.1 Reducing the cost of the QR algorithm
 - 10.3.2 Reduction to tridiagonal form
 - 10.3.3 Givens' rotations
 - 10.3.4 Simple tridiagonal QR algorithm
 - 10.3.5 The implicit Q theorem
 - 10.3.6 The Francis implicit QR Step
 - 10.3.7 A complete algorithm
- 10.4 Enrichments
 - 10.4.1 QR algorithm among the most important algorithms of the 20th century
 - 10.4.2 Who was John Francis
 - 10.4.3 Casting the reduction to tridiagonal form in terms of matrix-matrix multiplication
 - 10.4.4 Optimizing the tridiagonal QR algorithm
 - 10.4.5 The Method of Multiple Relatively Robust Representations (MRRR)
- 10.5 Wrap Up
 - 10.5.1 Additional homework
 - 10.5.2 Summary

10.1.3 What you will learn

This week, you explore practical methods for finding all eigenvalues and eigenvectors of a Hermitian matrix, building on the insights regarding the Power Method that you discovered last week.

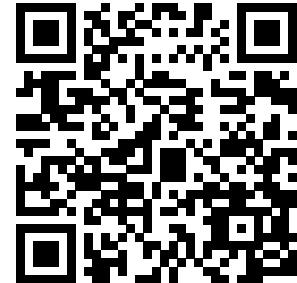
Upon completion of this week, you should be able to

- Formulate and analyze subspace iteration methods.
- Expose the relationship between subspace iteration and simple QR algorithms.

- Accelerate the convergence of QR algorithms by shifting the spectrum of the matrix.
- Lower the cost of QR algorithms by first reducing a Hermitian matrix to tridiagonal form.
- Cast all computation for computing the eigenvalues and eigenvectors of a Hermitian matrix in terms of unitary similarity transformations, yielding the Francis Implicit QR Step.
- Exploit a block diagonal structure of a matrix to deflate the Hermitian eigenvalue problem into smaller subproblems.
- Combine all these insights into a practical algorithm.

10.2 From Power Method to a simple QR algorithm

10.2.1 A simple QR algorithm



YouTube: https://www.youtube.com/watch?v=_vlE7aJGoNE

We now morph the subspace iteration discussed in the last unit into a simple incarnation of an algorithm known as the **QR algorithm**. We will relate this algorithm to performing subspace iteration with an $m \times m$ (square) matrix so that the method finds all eigenvectors simultaneously (under mild conditions). Rather than starting with a random matrix V , we now start with the identity matrix. This yields the algorithm on the left in Figure 10.2.1.1. We contrast it with the algorithm on the right.

```

 $\hat{V} := I$ 
for  $k := 0, \dots$ 
   $(\hat{V}, \hat{R}) := \text{QR}(A\hat{V})$ 
   $\hat{A} = \hat{V}^H A \hat{V}$ 
endfor

```

```

 $V = I$ 
for  $k := 0, \dots$ 
   $(Q, R) := \text{QR}(A)$ 
   $A = RQ$ 
   $V = VQ$ 
endfor

```

Figure 10.2.1.1 Left: Subspace iteration started with $\hat{V} = I$. Right: Simple QR algorithm.

The magic lies in the fact that the matrices computed by the QR algorithm are identical to those computed by the subspace iteration: Upon completion $\hat{V} = V$ and the matrix \hat{A} on

the left equals the (updated) matrix A on the right. To be able to prove this, we annotate the algorithm so we can reason about the contents of the matrices for each iteration.

$\widehat{A}^{(0)} := A$ $\widehat{V}^{(0)} := I$ $\widehat{R}^{(0)} := I$ for $k := 0, \dots$ $(\widehat{V}^{(k+1)}, \widehat{R}^{(k+1)}) := \text{QR}(\widehat{A}\widehat{V}^{(k)})$ $\widehat{A}^{(k+1)} := \widehat{V}^{(k+1)} {}^H \widehat{A}\widehat{V}^{(k+1)}$ endfor	$A^{(0)} := A$ $V^{(0)} := I$ $R^{(0)} := I$ for $k := 0, \dots$ $(Q^{(k+1)}, R^{(k+1)}) := \text{QR}(A^{(k)})$ $A^{(k+1)} := R^{(k+1)} Q^{(k+1)}$ $V^{(k+1)} := V^{(k)} Q^{(k+1)}$ endfor
--	---

Let's start by showing how the QR algorithm applies unitary equivalence transformations to the matrices $A^{(k)}$.

Homework 10.2.1.1 Show that for the algorithm on the right $A^{(k+1)} = Q^{(k+1)} {}^H A^{(k)} Q^{(k+1)}$.

Solution. The algorithm computes the QR factorization of $A^{(k)}$

$$A^{(k)} = Q^{(k+1)} R^{(k+1)}$$

after which

$$A^{(k+1)} := R^{(k+1)} Q^{(k+1)}$$

Hence

$$A^{(k+1)} = R^{(k+1)} Q^{(k+1)} = Q^{(k+1)} {}^H A^{(k)} Q^{(k+1)}.$$

This last homework shows that $A^{(k+1)}$ is derived from $A^{(k)}$ via a unitary similarity transformation and hence has the same eigenvalues as does $A^{(k)}$. This means it also is derived from A via a (sequence of) unitary similarity transformation and hence has the same eigenvalues as does A .

We now prove these algorithms mathematically equivalent.

Homework 10.2.1.2 In the above algorithms, for all k ,

- $\widehat{A}^{(k)} = A^{(k)}$.
- $\widehat{R}^{(k)} = R^{(k)}$.
- $\widehat{V}^{(k)} = V^{(k)}$.

Hint. The QR factorization is unique, provided the diagonal elements of R are taken to be positive.

Solution. We will employ a proof by induction.

- Base case: $k = 0$

This is trivially true:

- $\widehat{A}^{(0)} = A = A^{(0)}$.

- o $\widehat{R}^{(0)} = I = R^{(0)}$.
- o $\widehat{V}^{(0)} = I = V^{(0)}$.
- Inductive step: Assume that $\widehat{A}^{(k)} = A^{(k)}$, $\widehat{R}^{(k)} = R^{(k)}$, and $\widehat{V}^{(k)} = V^{(k)}$. Show that $\widehat{A}^{(k+1)} = A^{(k+1)}$, $\widehat{R}^{(k+1)} = R^{(k+1)}$, and $\widehat{V}^{(k+1)} = V^{(k+1)}$.

From the algorithm on the left, we know that

$$A\widehat{V}^{(k)} = \widehat{V}^{(k+1)}\widehat{R}^{(k+1)}.$$

and

$$\begin{aligned} A^{(k)} &= <(I.H.)> \\ &= <\text{algorithm on left}> \\ &= <\text{algorithm on left}> \\ &= <\text{I.H.}> \\ &= V^{(k)H}\widehat{V}^{(k+1)}\widehat{R}^{(k+1)}. \end{aligned} \tag{10.2.1}$$

But from the algorithm on the right, we know that

$$A^{(k)} = Q^{(k+1)}R^{(k+1)}. \tag{10.2.2}$$

Both (10.2.1) and (10.2.2) are QR factorizations of $A^{(k)}$ and hence, by the uniqueness of the QR factorization,

$$\widehat{R}^{(k+1)} = R^{(k+1)}$$

and

$$Q^{(k+1)} = V^{(k)H}\widehat{V}^{(k+1)}$$

or, equivalently and from the algorithm on the right,

$$\underbrace{V^{(k)}Q^{(k+1)}}_{V^{(k+1)}} = \widehat{V}^{(k+1)}.$$

This shows that

- o $\widehat{R}^{(k+1)} = R^{(k+1)}$ and
- o $\widehat{V}^{(k+1)} = V^{(k+1)}$.

Also,

$$\begin{aligned}
 & \widehat{A}^{(k+1)} \\
 &= \langle \text{algorithm on left} \rangle \\
 &\widehat{V}^{(k+1)H} A \widehat{V}^{(k+1)} \\
 &= \langle \widehat{V}^{(k+1)} = V^{(k+1)} \rangle \\
 &V^{(k+1)H} A V^{(k+1)} \\
 &= \langle \text{algorithm on right} \rangle \\
 &Q^{(k+1)H} V^{(k)H} A V^{(k)} Q^{(k+1)} \\
 &= \langle \text{I.H.} \rangle \\
 &Q^{(k+1)H} \widehat{V}^{(k)H} A \widehat{V}^{(k)} Q^{(k+1)} \\
 &= \langle \text{algorithm on left} \rangle \\
 &Q^{(k+1)H} \widehat{A}^{(k)} Q^{(k+1)} \\
 &= \langle \text{I.H.} \rangle \\
 &Q^{(k+1)H} A^{(k)} Q^{(k+1)} \\
 &= \langle \text{last homework} \rangle \\
 &A^{(k+1)}.
 \end{aligned}$$

- By the Principle of Mathematical Induction, the result holds.

Homework 10.2.1.3 In the above algorithms, show that for all k

- $V^{(k)} = Q^{(0)} Q^{(1)} \dots Q^{(k)}$.
- $A^k = V^{(k)} R^{(k)} \dots R^{(1)} R^{(0)}$. (Note: A^k here denotes A raised to the k th power.)

Assume that $Q^{(0)} = I$.

Solution. We will employ a proof by induction.

- Base case: $k = 0$

$$\underbrace{\begin{matrix} A^0 \\ I \end{matrix}}_{\text{I}} = \underbrace{\begin{matrix} V^{(0)} \\ I \end{matrix}}_{\text{I}} A^0 = \underbrace{\begin{matrix} R^{(0)} \\ I \end{matrix}}_{\text{I}}.$$

- Inductive step: Assume that $V^{(k)} = Q^{(0)} \dots Q^{(k)}$ and $A^k = V^{(k)} R^{(k)} \dots R^{(0)}$. Show that $V^{(k+1)} = Q^{(0)} \dots Q^{(k+1)}$ and $A^{k+1} = V^{(k+1)} R^{(k+1)} \dots R^{(0)}$.

$$V^{(k+1)} = V^{(k)} Q^{(k+1)} = Q^{(0)} \dots Q^{(k)} Q^{(k+1)}.$$

by the inductive hypothesis.

Also,

$$\begin{aligned}
 A^{k+1} &= <\text{definition}> \\
 AA^k &= <\text{inductive hypothesis}> \\
 AV^{(k)}R^{(k)} \dots R^{(0)} &= <\text{inductive hypothesis}> \\
 A\widehat{V}^{(k)}R^{(k)} \dots R^{(0)} &= <\text{left algorithm}> \\
 \widehat{V}^{(k+1)}\widehat{R}^{(k+1)}R^{(k)} \dots R^{(0)} &= <V^{(k+1)} = \widehat{V}^{(k+1)}; R^{(k+1)} = \widehat{R}^{(k+1)}> \\
 V^{(k+1)}R^{(k+1)}R^{(k)} \dots R^{(0)}.
 \end{aligned}$$

- By the Principle of Mathematical Induction, the result holds for all k .

This last exercise shows that

$$A^k = \underbrace{Q^{(0)}Q^{(1)} \dots Q^{(k)}}_{\text{unitary } V^{(k)}} \underbrace{R^{(k)} \dots R^{(1)}R^{(0)}}_{\text{upper triangular } \tilde{R}^{(k)}}$$

which exposes a QR factorization of A^k . Partitioning $V^{(k)}$ by columns

$$V^{(k)} = \left(\begin{array}{c|c|c} v_0^{(k)} & \dots & v_{m-1}^{(k)} \end{array} \right)$$

we notice that applying k iterations of the Power Method to vector e_0 yields

$$A^k e_0 = V^{(k)} \tilde{R}^{(k)} e_0 = V^{(k)} \tilde{\rho}_{0,0}^{(k)} e_0 = \tilde{\rho}_{0,0}^{(k)} V^{(k)} e_0 = \tilde{\rho}_{0,0}^{(k)} v_0^{(k)},$$

where $\tilde{\rho}_{0,0}^{(k)}$ is the $(0,0)$ entry in matrix $\tilde{R}^{(k)}$. Thus, the first column of $V^{(k)}$ equals a vector that would result from k iterations of the Power Method. Similarly, the second column of $V^{(k)}$ equals a vector that would result from k iterations of the Power Method, but orthogonal to $v_0^{(k)}$.



YouTube: <https://www.youtube.com/watch?v=t51YqvNWa0Q>

We make some final observations:

- $A^{(k+1)} = Q^{(k)H} A^{(k)} Q^{(k)}$. This means we can think of $A^{(k+1)}$ as the matrix $A^{(k)}$ but viewed in a new basis (namely the basis that consists of the column of $Q^{(k)}$).

- $A^{(k+1)} = (Q^{(0)} \cdots Q^{(k)})^H A Q^{(0)} \cdots Q^{(k)} = V^{(k)H} A V^{(k)}$. This means we can think of $A^{(k+1)}$ as the matrix A but viewed in a new basis (namely the basis that consists of the column of $V^{(k)}$).
- In each step, we compute

$$(Q^{(k+1)}, R^{(k+1)}) = QR(A^{(k)})$$

which we can think of as

$$(Q^{(k+1)}, R^{(k+1)}) = QR(A^{(k)} \times I).$$

This suggests that in each iteration we perform one step of subspace iteration, but with matrix $A^{(k)}$ and $V = I$:

$$(Q^{(k+1)}, R^{(k+1)}) = QR(A^{(k)}V).$$

- The insight is that the QR algorithm is identical to subspace iteration, except that at each step we reorient the problem (express it in a new basis) and we restart it with $V = I$.

Homework 10.2.1.4 Examine [Assignments/Week10/matlab/SubspaceIterationAllVectors.m](#), which implements the subspace iteration in Figure 10.2.1.1 (left). Examine it by executing the script in [Assignments/Week10/matlab/test_simple_QR_algorithm.m](#).

Solution. Discuss what you observe online with others!

Homework 10.2.1.5 Copy [Assignments/Week10/matlab/SubspaceIterationAllVectors.m](#) into [SimpleQRAlg.m](#) and modify it to implement the algorithm in Figure 10.2.1.1 (right) as

```
function [ Ak, V ] = SimpleQRAlg( A, maxits, illustrate, delay )
```

Modify the appropriate line in [Assignments/Week10/matlab/test_simple_QR_algorithms.m](#), changing (0) to (1), and use it to examine the convergence of the method.

What do you observe?

Solution.

- [Assignments/Week10/answers/SimpleQRAlg.m](#)

Discuss what you observe online with others!

10.2.2 A simple shifted QR algorithm



YouTube: <https://www.youtube.com/watch?v=HIxSCrFX1Ls>

The equivalence of the subspace iteration and the QR algorithm tells us a lot about convergence. Under mild conditions ($|\lambda_0| \geq \dots \geq |\lambda_{n-1}| > |\lambda_n| > \dots > |\lambda_{m-1}|$),

- The first n columns of $V^{(k)}$ converge to a basis for the subspace spanned by the eigenvectors associated with $\lambda_0, \dots, \lambda_{n-1}$.
- The last $m - n$ columns of $V^{(k)}$ converge to the subspace orthogonal to the subspace spanned by the eigenvectors associated with $\lambda_0, \dots, \lambda_{n-1}$.
- If A is Hermitian, then the eigenvectors associated with $\lambda_0, \dots, \lambda_{n-1}$, are orthogonal to those associated with $\lambda_n, \dots, \lambda_{m-1}$. Hence, the subspace spanned by the eigenvectors associated with $\lambda_0, \dots, \lambda_{n-1}$ is orthogonal to the subspace spanned by the eigenvectors associated with $\lambda_n, \dots, \lambda_{m-1}$.
- The rate of convergence with which these subspaces become orthogonal to each other is linear with a constant $|\lambda_n|/|\lambda_{n-1}|$.

What if in this situation we focus on $n = m - 1$? Then

- The last column of $V^{(k)}$ converges to point in the direction of the eigenvector associated with λ_{m-1} , the smallest in magnitude.
- The rate of convergence of that vector is linear with a constant $|\lambda_{m-1}|/|\lambda_{m-2}|$.

In other words, the subspace iteration acts upon the last column of $V^{(k)}$ in the same way as would an inverse iteration. This observation suggests that that convergence can be greatly accelerated by shifting the matrix by an estimate of the eigenvalue that is smallest in magnitude.

Homework 10.2.2.1 Copy `SimpleQRAlg.m` into `SimpleShiftedQRAlgConstantShift.m` and modify it to implement an algorithm that executes the QR algorithm with a shifted matrix $A - \rho I$:

```
function [ Ak, V ] = SimpleShiftedQRAlgConstantShift( A, rho, maxits, illustrate, delay )
```

Modify the appropriate line in `Assignments/Week10/matlab/test_simple_QR_algorithms.m`, changing (0) to (1), and use it to examine the convergence of the method.

Try different values for ρ : 0.0, 0.9, 0.99, 1.0, 1.99, 1.5. What do you observe?

Solution.

- [Assignments/Week10/answers/SimpleShiftedQRAlgConstantShift.m](#)

Discuss what you observe online with others!

We could compute the Rayleigh quotient with the last column of $V^{(k)}$ but a moment of reflection tells us that that estimate is already available as the last element on the diagonal of $A^{(k)}$, because the diagonal elements of $A^{(k)}$ converge to the eigenvalues. Thus, we arrive upon a simple *shifted* QR algorithm in [Figure 10.2.2.1](#). This algorithm inherits the cubic convergence of the Rayleigh quotient iteration, for the last column of V .

```

 $V = I$ 
for  $k := 0, \dots$ 
     $(Q, R) := \text{QR}(A - \alpha_{m-1,m-1}I)$ 
     $A = RQ + \alpha_{m-1,m-1}I$ 
     $V = VQ$ 
endfor

```

Figure 10.2.2.1 Simple shifted QR algorithm.

YouTube: <https://www.youtube.com/watch?v=Fhk0e5JFlsU>

To more carefully examine this algorithm, let us annotate it as we did for the simple QR algorithm in the last unit.

```

 $A^{(0)} = A$ 
 $V^{(0)} = I$ 
 $R^{(0)} = I$ 
for  $k := 0, \dots$ 
     $\mu^{(k)} = \alpha_{m-1,m-1}^{(k)}$ 
     $(Q^{(k+1)}, R^{(k+1)}) := \text{QR}(A^{(k)} - \mu^{(k)}I)$ 
     $A^{(k+1)} = R^{(k+1)}Q^{(k+1)} + \mu^{(k)}I$ 
     $V^{(k+1)} = V^{(k)}Q^{(k+1)}$ 
endfor

```

The following exercises clarify some of the finer points.

Homework 10.2.2.2 For the above algorithm, show that

- $A^{(k+1)} = Q^{(k+1)H} A^{(k)} Q^{(k+1)}$.
- $A^{(k+1)} = V^{(k+1)H} A V^{(k+1)}$.

Solution. The algorithm computes the QR factorization of $A^{(k)} - \mu^{(k)}I$

$$A^{(k)} - \mu^{(k)}I = Q^{(k+1)}R^{(k+1)}$$

after which

$$A^{(k+1)} := R^{(k+1)}Q^{(k+1)} + \mu^{(k)}I$$

Hence

$$\begin{aligned}
 & A^{(k+1)} \\
 &= R^{(k+1)} Q^{(k+1)} + \mu^{(k)} I \\
 &= Q^{(k+1) H} (A^{(k)} - \mu^{(k)} I) Q^{(k+1)} + \mu^{(k)} I \\
 &= Q^{(k+1) H} A^{(k)} Q^{(k+1)} - \mu^{(k)} Q^{(k+1) H} Q^{(k+1)} + \mu^{(k)} I \\
 &= Q^{(k+1) H} A^{(k)} Q^{(k+1)}.
 \end{aligned}$$

This last exercise confirms that the eigenvalues of $A^{(k)}$ equal the eigenvalues of A .

Homework 10.2.2.3 For the above algorithm, show that

$$\begin{aligned}
 & (A - \mu_{k-1} I)(A - \mu_{k-2} I) \cdots (A - \mu_1 I)(A - \mu_0 I) \\
 &= \underbrace{Q^{(0)} Q^{(1)} \cdots Q^{(k)}}_{\text{unitary}} \underbrace{R^{(k)} \cdots R^{(1)} R^{(0)}}_{\text{upper triangular}}.
 \end{aligned}$$

Solution. In this problem, we need to assume that $Q^{(0)} = I$. Also, it helps to recognize that $V^{(k)} = Q^{(0)} \cdots Q^{(k)}$, which can be shown via a simple inductive proof.

This requires a proof by induction.

- Base case: $k = 1$.

$$\begin{aligned}
 & A - \mu_0 I \\
 &= \underbrace{< A^{(0)} = A >}_{A^{(0)} - \mu_0 I} \\
 &= \underbrace{< \text{algorithm} >}_{Q^{(1)} R^{(1)}} \\
 &= \underbrace{< Q^{(0)} = R^{(0)} = I >}_{Q^{(0)} Q^{(1)} R^{(1)} R^{(0)}}
 \end{aligned}$$

- Inductive Step: Assume

$$\begin{aligned}
 & (A - \mu_{k-1} I)(A - \mu_{k-2} I) \cdots (A - \mu_1 I)(A - \mu_0 I) \\
 &= \underbrace{Q^{(0)} Q^{(1)} \cdots Q^{(k)}}_{V^{(k)}} \underbrace{R^{(k)} \cdots R^{(1)} R^{(0)}}_{\text{upper triangular}}
 \end{aligned}$$

Show that

$$\begin{aligned}
 & (A - \mu_k I)(A - \mu_{k-1} I) \cdots (A - \mu_1 I)(A - \mu_0 I) \\
 &= \underbrace{Q^{(0)} Q^{(1)} \cdots Q^{(k+1)}}_{V^{(k+1)}} \underbrace{R^{(k+1)} \cdots R^{(1)} R^{(0)}}_{\text{upper triangular}}
 \end{aligned}$$

Notice that

$$\begin{aligned}
 & (A - \mu_k I)(A - \mu_{k-1} I) \cdots (A - \mu_0 I) \\
 & = \quad < \text{last homework} > \\
 & (V^{(k+1)} A^{(k+1)} V^{(k+1)H} - \mu_k I)(A - \mu_{k-1} I) \cdots (A - \mu_0 I) \\
 & = \quad < I = V^{(k+1)} V^{(k+1)H} > \\
 & V^{(k+1)} (A^{(k+1)} - \mu_k I) V^{(k+1)H} (A - \mu_{k-1} I) \cdots (A - \mu_0 I) \\
 & = \quad < \text{I.H.} > \\
 & V^{(k+1)} (A^{(k+1)} - \mu_k I) V^{(k+1)H} V^{(k)H} R^{(k)} \cdots R^{(0)} \\
 & = \quad < V^{(k+1)H} = Q^{(k+1)H} V^{(k)H} > \\
 & V^{(k+1)} (A^{(k+1)} - \mu_k I) Q^{(k+1)H} R^{(k)} \cdots R^{(0)} \\
 & = \quad < \text{algorithm} > \\
 & V^{(k+1)} R^{(k+1)} Q^{(k+1)H} Q^{(k+1)H} R^{(k)} \cdots R^{(0)} \\
 & = \quad < Q^{(k+1)} Q^{(k+1)H} = I > \\
 & V^{(k+1)} R^{(k+1)} R^{(k)} \cdots R^{(0)}
 \end{aligned}$$

- By the Principle of Mathematical Induction, the result holds.

Homework 10.2.2.4 Copy `SimpleShiftedQRAlgConstantShift.m` into `SimpleShiftedQRAlg` and modify it to implement an algorithm that executes the QR algorithm in Figure 10.2.2.1:

```
function [ Ak, V ] = SimpleShiftedQRAlg( A, maxits, illustrate, delay )
```

Modify the appropriate line in `Assignments/Week10/matlab/test_simple_QR_algorithms.m`, changing (0) to (1), and use it to examine the convergence of the method.

What do you observe?

Solution.

- [Assignments/Week10/answers/SimpleShiftedQRAlg.m](https://www.mathworks.com/help/advlinsolv/ug/simple-shifted-QR-algorithm-deflation.html)

Discuss what you observe online with others!

10.2.3 Deflating the problem



YouTube: <https://www.youtube.com/watch?v=rdWhh3lHuhY>

Recall that if

$$A = \left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right), \quad A_{00}x = \lambda x, \text{ and } A_{11}y = \mu y,$$

then

$$\left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right) \left(\begin{array}{c} x \\ 0 \end{array} \right) = \lambda \left(\begin{array}{c} x \\ 0 \end{array} \right) \text{ and } \left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right) \left(\begin{array}{c} 0 \\ y \end{array} \right) = \mu \left(\begin{array}{c} 0 \\ y \end{array} \right).$$

In other words, $\Lambda(A) = \Lambda(A_{00}) \cup \Lambda(A_{11})$ and eigenvectors of A can be easily constructed from eigenvalues of A_{00} and A_{11} .

This insight allows us to **deflate** a matrix when strategically placed zeroes (or, rather, acceptably small entries) appear as part of the QR algorithm. Let us continue to focus on the Hermitian eigenvalue problem.

Homework 10.2.3.1 Let $A \in \mathbb{C}^{m \times m}$ be a Hermitian matrix and $V \in \mathbb{C}^{m \times m}$ be a unitary matrix such that

$$V^H A V = \left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right).$$

If V_{00} and V_{11} are unitary matrices such that $V_{00}^H A_{00} V_{00} = \Lambda_0$ and $V_{11}^H A_{11} V_{11} = \Lambda_1$, are both diagonal, show that

$$\left(V \left(\begin{array}{c|c} V_{00} & 0 \\ \hline 0 & V_{11} \end{array} \right) \right)^H A \left(V \left(\begin{array}{c|c} V_{00} & 0 \\ \hline 0 & V_{11} \end{array} \right) \right) = \left(\begin{array}{c|c} \Lambda_0 & 0 \\ \hline 0 & \Lambda_1 \end{array} \right).$$

Solution.

$$\begin{aligned} & \left(V \left(\begin{array}{c|c} V_{00} & 0 \\ \hline 0 & V_{11} \end{array} \right) \right)^H A \left(V \left(\begin{array}{c|c} V_{00} & 0 \\ \hline 0 & V_{11} \end{array} \right) \right) \\ &= \langle (XY)^H = Y^H X^H \rangle \\ & \left(\begin{array}{c|c} V_{00} & 0 \\ \hline 0 & V_{11} \end{array} \right)^H V^H A V \left(\begin{array}{c|c} V_{00} & 0 \\ \hline 0 & V_{11} \end{array} \right) \\ &= \langle V^H A V = \text{diag}(A_{00}, A_{11}) \rangle \\ & \left(\begin{array}{c|c} V_{00} & 0 \\ \hline 0 & V_{11} \end{array} \right)^H \left(\begin{array}{c|c} A_{00} & 0 \\ \hline 0 & A_{11} \end{array} \right) \left(\begin{array}{c|c} V_{00} & 0 \\ \hline 0 & V_{11} \end{array} \right) \\ &= \langle \text{partitioned matrix-matrix multiplication} \rangle \\ & \left(\begin{array}{c|c} V_{00}^H A_{00} V_{00} & 0 \\ \hline 0 & V_{11}^H A_{11} V_{11} \end{array} \right) \\ &= \langle V_{00}^H A_{00} V_{00} = \Lambda_0; V_{11}^H A_{11} V_{11} = \Lambda_1 \rangle \\ & \left(\begin{array}{c|c} \Lambda_0 & 0 \\ \hline 0 & \Lambda_1 \end{array} \right). \end{aligned}$$

The point of this last exercise is that if at some point the QR algorithm yields a block diagonal matrix, then the algorithm can proceed to find the spectral decompositions of the blocks on the diagonal, updating the matrix, V , in which the eigenvectors are accumulated.

Now, since it is the last column of $V^{(k)}$ that converges fastest to an eigenvector, eventually we expect $A^{(k)}$ computed as part of the QR algorithm to be of the form

$$A^{(k)} = \left(\begin{array}{c|c} A_{00}^{(k)} & f_{01}^{(k)} \\ \hline f_{01}^{(k)T} & \alpha_{m-1,m-1}^{(k)} \end{array} \right),$$

where $f_{01}^{(k)}$ is small. In other words,

$$A^{(k)} \approx \left(\begin{array}{c|c} A_{00}^{(k)} & 0 \\ \hline 0 & \alpha_{m-1,m-1}^{(k)} \end{array} \right).$$

Once $f_{01}^{(k)}$ is small enough, the algorithm can continue with $A_{00}^{(k)}$. The problem is thus **deflated** to a smaller problem.

What criteria should we use to deflate. If the active matrix is $m \times m$, for now we use the criteria

$$\|f_{01}\|_1 \leq \epsilon_{\text{mach}}(|\alpha_{0,0}^{(k)}| + \dots + |\alpha_{m-1,m-1}^{(k)}|).$$

The idea is that if the magnitudes of the off-diagonal elements of the last row are small relative to the eigenvalues, then they can be considered to be zero. The sum of the absolute values of the diagonal elements is an estimate of the sizes of the eigenvalues. We will refine this criteria later.

Homework 10.2.3.2 Copy `SimpleShiftedQRAlg.m` into `SimpleShiftedQRAlgWithDeflation` and modify it to add deflation.

```
function [ Ak, V ] = SimpleShiftedQRAlgWithDeflation( A, maxits, illustrate, delay )
```

Modify the appropriate lines in [Assignments/Week10/matlab/test_simple_QR_algorithm.m](#), changing (0) to (1), and use it to examine the convergence of the method.

Solution.

- [Assignments/Week10/answers/SimpleShiftedQRAlgWithDeflation.m](#)

Discuss what you observe online with others!

Remark 10.2.3.1 It is possible that deflation can happen anywhere in the matrix and one should check for that. However, it is most likely to happen in the last row and column of the active part of the matrix.

10.2.4 Cost of a simple QR algorithm



YouTube: <https://www.youtube.com/watch?v=c1zJG3T0D44>

The QR algorithms that we have discussed incur the following approximate costs per iteration for an $m \times m$ Hermitian matrix.

- $A \rightarrow QR$ (QR factorization): $\frac{4}{3}m^3$ flops.

- $A := RQ$. A naive implementation would take advantage of R being upper triangular, but not of the fact that A will again be Hermitian, at a cost of m^3 flops. If one also takes advantage of the fact that A is Hermitian, that cost is reduced to $\frac{1}{2}m^3$ flops.
- If the eigenvectors are desired (which is usually the case), the update $V := VQ$ requires an addition $2m^3$ flops.

Any other costs, like shifting the matrix, are inconsequential.

Thus, the cost, per iteration, equals approximately $(\frac{4}{3} + \frac{1}{2})m^3 = \frac{11}{6}m^3$ flops if only the eigenvalues are to be computed. If the eigenvectors are also required, then the cost increases by $2m^3$ flops to become $\frac{23}{6}m^3$ flops.

Let us now consider adding deflation. The rule of thumb is that it takes a few iterations per eigenvalue that is found. Let's say K iterations are needed. Every time an eigenvalue is found, the problem deflates, decreasing in size by one. The cost then becomes

$$\sum_{i=m}^1 K \frac{23}{6} i^3 \approx K \frac{23}{6} \int_0^m x^3 dx = K \frac{23}{6} \frac{1}{4} x^4 |_0^m = K \frac{23}{24} m^4.$$

The bottom line is that the computation requires $O(m^4)$ flops. All other factorizations we have encountered so far require at most $O(m^3)$ flops. Generally $O(m^4)$ is considered prohibitively expensive. We need to do better!

10.3 A Practical Hermitian QR Algorithm

10.3.1 Reduction to tridiagonal form



YouTube: <https://www.youtube.com/watch?v=ETQbxnweKok>

In this section, we see that if $A^{(0)}$ is a tridiagonal matrix, then so are all $A^{(k)}$. This reduces the cost of each iteration of the QR algorithm from $O(m^3)$ flops to $O(m)$ flops if only the eigenvalues are computed and $O(m^2)$ flops if the eigenvectors are also desired. Thus, if matrix A is first reduced to a tridiagonal matrix via (unitary) similarity transformations, then the cost of finding its eigenvalues and eigenvectors is reduced from $O(m^4)$ to $O(m^3)$ flops. Fortunately, there is an algorithm for reducing a matrix to tridiagonal form that requires $O(m^3)$ operations.

The basic algorithm for reducing a Hermitian matrix to tridiagonal form, overwriting the original matrix with the result, can be explained as follows. We assume that A is stored only

in the lower triangular part of the matrix and that only the diagonal and subdiagonal of the tridiagonal matrix is computed, overwriting those parts of A . Finally, the Householder vectors used to zero out parts of A can overwrite the entries that they annihilate (set to zero), much like we did when computing the Householder QR factorization.

Recall that in [Subsubsection 3.3.3.3](#), we introduced the function

$$\left[\begin{pmatrix} \rho \\ u_2 \end{pmatrix}, \tau \right] := \text{Housev} \left(\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \right)$$

to compute the vector $u = \begin{pmatrix} 1 \\ u_2 \end{pmatrix}$ that reflects x into $\boxed{\pm} \|x\|_2 e_0$ so that

$$\left(I - \frac{1}{\tau} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \right) \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \boxed{\pm} n \underbrace{\|x\|_2}_{\rho} e_0.$$

We are going to use a variation on this function:

$$[u, \tau] := \text{Housev}(x)$$

implemented by the function

```
function [ u, tau ] = Housev1( x )
```

We also reintroduce the notation $H(x)$ for the transformation $I - \frac{1}{\tau}uu^H$ where u and τ are computed by $\text{Housev1}(x)$.

We now describe an algorithm for reducing a Hermitian matrix to tridiagonal form:

- Partition $A \rightarrow \begin{pmatrix} \alpha_{11} & \star \\ a_{21} & A_{22} \end{pmatrix}$. Here the \star denotes a part of a matrix that is neither stored nor updated.
- Update $[a_{21}, \tau] := \text{Housev1}(a_{21})$. This overwrites the first element of a_{21} with $\boxed{\pm} \|a_{21}\|_2$ and the remainder with all but the first element of the Householder vector u . Implicitly, the elements below the first element equal zero in the updated matrix A .
- Update

$$A_{22} := H(a_{21})A_{22}H(a_{21}).$$

Since A_{22} is Hermitian both before and after the update, only the lower triangular part of the matrix needs to be updated.

- Continue this process with the updated A_{22} .

This approach is illustrated in [Figure 10.3.1.1](#).

$\begin{array}{cccccc} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{array}$	\longrightarrow	$\begin{array}{ccccc} \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{array}$	\longrightarrow	$\begin{array}{c cccc} \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{array}$	
Original matrix	First iteration	Second iteration			
					$\begin{array}{c cc cc} \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ \hline 0 & \times & \times & \times & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{array} \longrightarrow \begin{array}{c cc cc} \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ \hline 0 & \times & \times & \times & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{array}$
					Third iteration

Figure 10.3.1.1 An illustration of the reduction of a Hermitian matrix to tridiagonal form. The \times s denote nonzero elements in the matrix. The gray entries above the diagonal are not actually updated.

The update of A_{22} warrants closer scrutiny:

$$\begin{aligned}
A_{22} &:= H(a_{21})A_{22}H(a_{21}) \\
&= (I - \frac{1}{\tau}u_{21}u_{21}^H)A_{22}(I - \frac{1}{\tau}u_{21}u_{21}^H) \\
&= (A_{22} - \frac{1}{\tau}u_{21}\underbrace{u_{21}^H A_{22}}_{y_{21}^H})(I - \frac{1}{\tau}u_{21}u_{21}^H) \\
&= A_{22} - \frac{1}{\tau}u_{21}y_{21}^H - \frac{1}{\tau}\underbrace{A_{22}u_{21}}_{y_{21}} u_{21}^H + \frac{1}{\tau^2}u_{21}\underbrace{y_{21}^H u_{21}}_{2\beta} u_{21}^H \\
&= A_{22} - \left(\frac{1}{\tau}u_{21}y_{21}^H - \frac{\beta}{\tau^2}u_{21}u_{21}^H\right) - \left(\frac{1}{\tau}y_{21}u_{21}^H - \frac{\beta}{\tau^2}u_{21}u_{21}^H\right) \\
&= A_{22} - u_{21}\underbrace{\frac{1}{\tau}\left(y_{21}^H - \frac{\beta}{\tau}u_{21}^H\right)}_{w_{21}^H} - \underbrace{\frac{1}{\tau}\left(y_{21} - \frac{\beta}{\tau}u_{21}\right)}_{w_{21}} u_{21}^H \\
&= \underbrace{A_{22} - u_{21}w_{21}^H - w_{21}u_{21}^H}_{\text{Hermitian rank-2 update}}
\end{aligned}$$

This formulation has two advantages: it requires fewer computations and it does not generate an intermediate result that is not Hermitian. An algorithm that implements all these insights is given in [Figure 10.3.1.2](#).

$[A, t] := \text{TriRed-unb}(A, t)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right)$
A_{TL} is 0×0 and t_T has 0 elements
while $m(A_{TL}) < m(A) - 2$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right)$
$[a_{21}, \tau_1] := \text{Housev1}(a_{21})$
$u_{21} = a_{21}$ with first element replaced with 1
Update $A_{22} := H(u_{21})A_{22}H(u_{21})$ via the steps
$\begin{cases} y_{21} := A_{22}u_{21} & \text{(Hermitian matrix-vector multiply!)} \\ \beta := u_{21}^H y_{21}/2 \\ w_{21} := (y_{21} - \beta u_{21})/\tau_1 \\ A_{22} := A_{22} - \text{tril}(u_{21}w_{21}^H + w_{21}u_{21}^H) & \text{(Hermitian rank-2 update)} \end{cases}$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right)$
endwhile

Figure 10.3.1.2 Basic algorithm for reduction of a Hermitian matrix to tridiagonal form.

During the first iteration, when updating $(m-1) \times (m-1)$ matrix A_{22} , the bulk of computation is in the computation of $y_{21} := A_{22}u_{21}$, at $2(m-1)^2$ flops, and $A_{22} := A_{22} - (u_{21}w_{21}^H + w_{21}u_{21}^H)$, at $2(m-1)^2$ flops. The total cost for reducing $m \times m$ matrix A to tridiagonal form is therefore approximately

$$\sum_{k=0}^{m-1} 4(m-k-1)^2 \text{ flops.}$$

By substituting $j = m - k - 1$ we find that

$$\sum_{k=0}^{m-1} 4(m-k-1)^2 \text{ flops} = 4 \sum_{j=0}^{m-1} j^2 \text{ flops} \approx 4 \int_0^m x^2 dx = \frac{4}{3}m^3 \text{ flops.}$$

This equals, approximately, the cost of one QR factorization of matrix A .

Homework 10.3.1.1 A more straight forward way of updating A_{22} is given by

$$\begin{aligned} A_{22} &:= (I - \frac{1}{\tau}u_{21}u_{21}^H)A_{22}(I - \frac{1}{\tau}u_{21}u_{21}^H) \\ &= \underbrace{\left(A_{22} - \frac{1}{\tau}u_{21} \underbrace{u_{21}^H A_{22}}_{y_{21}^H} \right)}_{B_{22}} (I - \frac{1}{\tau}u_{21}u_{21}^H) \\ &= B_{22} - \frac{1}{\tau} \underbrace{B_{22}u_{21}}_{x_{21}} u_{21}^H. \end{aligned}$$

This suggests the steps

- Compute $y_{21} = A_{22}u_{21}$. (Hermitian matrix-vector multiplication).
- Compute $B_{22} = A_{22} - \frac{1}{\tau}u_{21}y_{21}^H$. (Rank-1 update yielding a *nonHermitian* intermediate matrix).
- Compute $x_{21} = B_{22}u_{21}$. (Matrix-vector multiplication).
- Compute $A_{22} = B_{22} - \frac{1}{\tau}x_{21}u_{21}^H$. (Rank-1 update yielding a *Hermitian* final matrix).

Estimate the cost of this alternative approach. What other disadvantage(s) does this approach have?

Solution. During the k th iteration, for $k = 0, 1, \dots, m - 1$ the costs for the various steps are as follows:

- Compute $y_{21} = A_{22}u_{21}$. (Hermitian matrix-vector multiplication). Cost: approximately $2(m - 1)^2$ flops.
- Compute $B_{22} = A_{22} - \frac{1}{\tau}u_{21}y_{21}^H$. (Rank-1 update yielding a *nonHermitian* intermediate matrix). Cost: approximately $2(m - 1)^2$ flops since the intermediate matrix B_{22} is not Hermitian.
- Compute $x_{21} = B_{22}u_{21}$. (Matrix-vector multiplication). Cost: approximately $2(m - 1)^2$ flops.
- Compute $A_{22} = B_{22} - \frac{1}{\tau}x_{21}y_{21}^H$. Only the lower triangular part of A_{22} needs to be computed. Cost: approximately $(m - 1)^2$ flops.

Thus, the total cost per iteration is, approximately

$$7(m - 1)^2 \text{ flops.}$$

The total cost is then, approximately,

$$\sum_{k=0}^{m-1} 7(m - k - 1)^2 \text{ flops} = 7 \sum_{j=0}^{m-1} j^2 \text{ flops} \approx 7 \int_0^m x^2 dx = \frac{7}{3}m^3 \text{ flops.}$$

This almost doubles the cost of the reduction to tridiagonal form.

An additional disadvantage is that a nonsquare intermediate matrix must be stored.

The diagonal elements of a Hermitian matrix are real. Hence the tridiagonal matrix has real values on its diagonal. A post process (that follows the reduction to tridiagonal form) can be used to convert the elements of the subdiagonal to real values as well. The advantage of this is that in the subsequent computation, that computes the eigenvalues of the tridiagonal matrix and accumulates the eigenvectors, only needs to perform real (floating point) arithmetic.

Ponder This 10.3.1.2 Propose a postprocess that converts the off-diagonal elements of a tridiagonal Hermitian matrix to real values. The postprocess must be equivalent to applying a unitary similarity transformation so that eigenvalues are preserved.

You may want to start by looking at

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{1,0} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix},$$

where the diagonal elements are real-valued and the off-diagonal elements are complex-valued. Then move on to

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{1,0} & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{2,1} \\ 0 & \alpha_{2,1} & \alpha_{2,2} \end{pmatrix}.$$

What is the pattern?

Homework 10.3.1.3 You may want to start by executing `git pull` to update your directory `Assignments`.

In directory `Assignments/Week10/matlab/`, you will find the following files:

- `Housev1.m`: An implementation of the function `Housev1`, mentioned in the unit.
- `TriRed.m`: A code skeleton for a function that reduces a Hermitian matrix to a tridiagonal matrix. Only the lower triangular part of the input and output are stored.

`[T, t] = TriRed(A, t)`

returns the diagonal and first subdiagonal of the tridiagonal matrix in `T`, stores the Householder vectors below the first subdiagonal, and returns the scalars τ in vector `t`.

- `TriFromBi.m`: A function that takes the diagonal and first subdiagonal in the input matrix and returns the tridiagonal matrix that they define.

`T = TriFromBi(A)`

- `test_TriRed.m`: A script that tests `TriRed`.

With these resources, you are to complete `TriRed` by implementing the algorithm in [Figure 10.3.1.2](#).

Be sure to look at the hint!

Hint. If `A` holds Hermitian matrix A , storing only the lower triangular part, then Ax is implemented in Matlab as

```
( tril( A ) + tril( A, -1)' ) * x;
```

Updating only the lower triangular part of array `A` with $A := A - B$ is accomplished by

```
A = A - tril( B );
```

Solution.

- [Assignments/Week10/answers/TriRed.m](#).

10.3.2 Givens' rotations



YouTube: <https://www.youtube.com/watch?v=XAvioT6ALAg>

We now introduce another important class of orthogonal matrices known as Givens' rotations. Actually, we have seen these before, in Subsubsection 2.2.5.1, where we simply called them rotations. It is how they are used that makes them Givens' rotations.

Given a vector $x = \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} \in \mathbb{R}^2$, there exists an orthogonal matrix G such that $G^T x = \begin{pmatrix} \pm \|x\|_2 \\ 0 \end{pmatrix}$. The Householder transformation is one example of such a matrix G . An alternative is the Givens' rotation: $G = \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}$ where $\gamma^2 + \sigma^2 = 1$. (Notice that γ and σ can be thought of as the cosine and sine of an angle.) Then

$$\begin{aligned} G^T G &= \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}^T \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix} = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix} \\ &= \begin{pmatrix} \gamma^2 + \sigma^2 & -\gamma\sigma + \gamma\sigma \\ \gamma\sigma - \gamma\sigma & \gamma^2 + \sigma^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \end{aligned}$$

which means that a Givens' rotation is an orthogonal matrix.

Homework 10.3.2.1 Propose formulas for γ and σ such that

$$\begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}^T \underbrace{\begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix}}_x = \begin{pmatrix} \|x\|_2 \\ 0 \end{pmatrix},$$

where $\gamma^2 + \sigma^2 = 1$.

Solution. Take $\gamma = \chi_1/\|x\|_2$ and $\sigma = \chi_2/\|x\|_2$, then $\gamma^2 + \sigma^2 = (\chi_1^2 + \chi_2^2)/\|x\|_2^2 = 1$ and

$$\begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}^T \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} (\chi_1^2 + \chi_2^2)/\|x\|_2 \\ (\chi_1\chi_2 - \chi_1\chi_2)/\|x\|_2 \end{pmatrix} = \begin{pmatrix} \|x\|_2 \\ 0 \end{pmatrix}.$$

Remark 10.3.2.1 We only discuss real-valued Givens' rotations and how they transform real-valued vectors, since the output of our reduction to tridiagonal form, after postprocessing, yields a real-valued tridiagonal symmetric matrix.

Ponder This 10.3.2.2 One could use 2×2 Householder transformations (reflectors) instead of Givens' rotations. Why is it better to use Givens' rotations in this situation.

10.3.3 Simple tridiagonal QR algorithm



YouTube: https://www.youtube.com/watch?v=_IgDCL70PdU

Now, consider the 4×4 tridiagonal matrix

$$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & 0 \\ 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{pmatrix}.$$

From $\begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \end{pmatrix}$, one can compute $\gamma_{1,0}$ and $\sigma_{1,0}$ so that

$$\begin{pmatrix} \gamma_{1,0} & -\sigma_{1,0} \\ \sigma_{1,0} & \gamma_{1,0} \end{pmatrix}^T \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \end{pmatrix} = \begin{pmatrix} \hat{\alpha}_{0,0} \\ 0 \end{pmatrix}.$$

Then

$$\left(\begin{array}{cc|cc} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ 0 & \hat{\alpha}_{1,1} & \hat{\alpha}_{1,2} & 0 \\ \hline 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) = \left(\begin{array}{cc|cc} \gamma_{1,0} & \sigma_{1,0} & 0 & 0 \\ -\sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|cc} \alpha_{0,0} & \alpha_{0,1} & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & 0 \\ \hline 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right).$$

Next, from $\begin{pmatrix} \hat{\alpha}_{1,1} \\ \alpha_{2,1} \end{pmatrix}$, one can compute $\gamma_{2,1}$ and $\sigma_{2,1}$ so that

$$\begin{pmatrix} \gamma_{2,1} & -\sigma_{2,1} \\ \sigma_{2,1} & \gamma_{2,1} \end{pmatrix}^T \begin{pmatrix} \hat{\alpha}_{1,1} \\ \alpha_{2,1} \end{pmatrix} = \begin{pmatrix} \hat{\hat{\alpha}}_{1,1} \\ 0 \end{pmatrix}.$$

Then

$$\left(\begin{array}{c|ccc} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ 0 & \hat{\hat{\alpha}}_{1,1} & \hat{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ \hline 0 & 0 & \hat{\alpha}_{2,2} & \hat{\alpha}_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) = \left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ 0 & \gamma_{2,1} & \sigma_{2,1} & 0 \\ \hline 0 & -\sigma_{2,1} & \gamma_{2,1} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|ccc} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ 0 & \hat{\alpha}_{1,1} & \hat{\alpha}_{1,2} & 0 \\ \hline 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right)$$

Finally, from $\begin{pmatrix} \hat{\alpha}_{2,2} \\ \alpha_{3,2} \end{pmatrix}$, one can compute $\gamma_{3,2}$ and $\sigma_{3,2}$ so that $\begin{pmatrix} \gamma_{3,2} & -\sigma_{3,2} \\ \sigma_{3,2} & \gamma_{3,2} \end{pmatrix}^T \begin{pmatrix} \hat{\alpha}_{2,2} \\ \alpha_{3,2} \end{pmatrix} = \begin{pmatrix} \hat{\alpha}_{2,2} \\ 0 \end{pmatrix}$. Then

$$\left(\begin{array}{cc|cc} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ 0 & \hat{\hat{\alpha}}_{1,1} & \hat{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ \hline 0 & 0 & \hat{\hat{\alpha}}_{2,2} & \hat{\hat{\alpha}}_{2,3} \\ 0 & 0 & 0 & \hat{\alpha}_{3,3} \end{array} \right) = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 1 & 0 & \gamma_{3,2} & \sigma_{3,2} \\ 0 & 1 & -\sigma_{3,2} & \gamma_{3,2} \end{array} \right) \left(\begin{array}{cc|cc} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ 0 & \hat{\hat{\alpha}}_{1,1} & \hat{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ \hline 0 & 0 & \hat{\hat{\alpha}}_{2,2} & \hat{\hat{\alpha}}_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right)$$

The matrix Q is the orthogonal matrix that results from multiplying the different Givens' rotations together:

$$Q = \left(\begin{array}{cc|cc} \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & \gamma_{2,1} & -\sigma_{2,1} & 0 \\ \hline 0 & \sigma_{2,1} & \gamma_{2,1} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & \gamma_{3,2} & -\sigma_{3,2} \\ 0 & 0 & \sigma_{3,2} & \gamma_{3,2} \end{array} \right). \quad (10.3.1)$$

However, it needs not be explicitly formed, as we exploit next.

The next question is how to compute RQ given the QR factorization of the tridiagonal matrix. We notice that

$$\underbrace{\left(\begin{array}{cc|cc} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ 0 & \hat{\hat{\alpha}}_{1,1} & \hat{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ \hline 0 & 0 & \hat{\hat{\alpha}}_{2,2} & \hat{\hat{\alpha}}_{2,3} \\ 0 & 0 & 0 & \hat{\alpha}_{3,3} \end{array} \right)}_{\left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\hat{\alpha}}_{1,1} & \hat{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ \hline 0 & 0 & \hat{\hat{\alpha}}_{2,2} & \hat{\hat{\alpha}}_{2,3} \\ 0 & 0 & 0 & \hat{\alpha}_{3,3} \end{array} \right)} \underbrace{\left(\begin{array}{cc|cc} \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right)}_{\left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{0,1} & \tilde{\alpha}_{0,2} & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\hat{\alpha}}_{1,1} & \tilde{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ \hline 0 & 0 & \hat{\hat{\alpha}}_{2,2} & \hat{\hat{\alpha}}_{2,3} \\ 0 & 0 & 0 & \hat{\alpha}_{3,3} \end{array} \right)} \underbrace{\left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & \gamma_{2,1} & -\sigma_{2,1} & 0 \\ \hline 0 & \sigma_{2,1} & \gamma_{2,1} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right)}_{\left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{0,1} & \tilde{\alpha}_{0,2} & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\hat{\alpha}}_{1,1} & \tilde{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ \hline 0 & 0 & \hat{\hat{\alpha}}_{2,2} & \hat{\hat{\alpha}}_{2,3} \\ 0 & 0 & 0 & \hat{\alpha}_{3,3} \end{array} \right)} \underbrace{\left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & \gamma_{3,2} & -\sigma_{3,2} \\ 0 & 0 & \sigma_{3,2} & \gamma_{3,2} \end{array} \right)}_{\left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\hat{\alpha}}_{0,1} & \tilde{\alpha}_{0,2} & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\hat{\alpha}}_{1,1} & \tilde{\hat{\alpha}}_{1,2} & \tilde{\hat{\alpha}}_{1,3} \\ \hline 0 & 0 & \tilde{\hat{\alpha}}_{2,2} & \tilde{\hat{\alpha}}_{2,3} \\ 0 & 0 & \tilde{\hat{\alpha}}_{3,2} & \tilde{\hat{\alpha}}_{3,3} \end{array} \right)}.$$

A symmetry argument can be used to motivate that $\tilde{\alpha}_{0,2} = \tilde{\alpha}_{1,3} = 0$ (which is why they appear in gray, if you look carefully). This also explains why none of the elements above the first superdiagonal become nonzero.

Remark 10.3.3.1 An important observation is that if A is tridiagonal, then $A \rightarrow QR$ (QR factorization) followed by $A := RQ$ again yields a tridiagonal matrix. In other words, any QR algorithm previously discussed (simple, shifted, with deflation) when started with a tridiagonal matrix will generate a succession of tridiagonal matrices.

10.3.4 The implicit Q theorem



YouTube: <https://www.youtube.com/watch?v=w6clp9UqRRE>

Definition 10.3.4.1 Upper Hessenberg matrix. A matrix is said to be upper Hessenberg if all entries below its first subdiagonal equal zero. \diamond

In other words, an $m \times m$ upper Hessenberg matrix looks like

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \cdots & \alpha_{0,m-1} & \alpha_{0,m-1} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,m-1} & \alpha_{1,m-1} \\ 0 & \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,m-1} & \alpha_{2,m-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & \alpha_{m-2,m-2} & \alpha_{m-2,m-1} \\ 0 & 0 & 0 & \cdots & \alpha_{m-1,m-2} & \alpha_{m-1,m-1} \end{pmatrix}.$$

Obviously, a tridiagonal matrix is a special case of an upper Hessenberg matrix.

The following theorem sets the stage for one of the most remarkable algorithms in numerical linear algebra, which allows us to greatly streamline the implementation of the shifted QR algorithm.

Theorem 10.3.4.2 Implicit Q Theorem. Let $A, B \in \mathbb{C}^{m \times m}$, where B is upper Hessenberg and has only (real) positive elements on its first subdiagonal. Assume there exists a unitary matrix Q such that $Q^H A Q = B$. Then Q and B are uniquely determined by A and the first column of Q .

Proof. Partition

$$Q = \left(\begin{array}{c|c|c|c|c|c} q_0 & q_1 & q_2 & \cdots & q_{m-2} & q_{m-1} \end{array} \right)$$

and

$$B = \left(\begin{array}{c|c|c|c|c|c} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} & \cdots & \beta_{0,m-2} & \beta_{0,m-1} \\ \hline \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & \cdots & \beta_{1,m-2} & \beta_{1,m-1} \\ \hline 0 & \beta_{2,1} & \beta_{2,2} & \cdots & \beta_{2,m-2} & \beta_{2,m-1} \\ \hline 0 & 0 & \beta_{3,2} & \cdots & \beta_{3,m-2} & \beta_{3,m-1} \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline 0 & 0 & 0 & \cdots & \beta_{m-1,m-2} & \beta_{m-1,m-1} \end{array} \right).$$

Notice that $AQ = QB$ and hence

$$\begin{aligned} A \left(\begin{array}{c|c|c|c|c|c} q_0 & q_1 & q_2 & \cdots & q_{m-2} & q_{m-1} \end{array} \right) \\ = \left(\begin{array}{c|c|c|c|c|c} q_0 & q_1 & q_2 & \cdots & q_{m-2} & q_{m-1} \end{array} \right) \\ \left(\begin{array}{c|c|c|c|c|c} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} & \cdots & \beta_{0,m-2} & \beta_{0,m-1} \\ \hline \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & \cdots & \beta_{1,m-2} & \beta_{1,m-1} \\ \hline 0 & \beta_{2,1} & \beta_{2,2} & \cdots & \beta_{2,m-1} & \\ \hline 0 & 0 & \beta_{3,2} & \cdots & \beta_{3,m-2} & \beta_{3,m-1} \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline 0 & 0 & 0 & \cdots & \beta_{m-1,m-2} & \beta_{m-1,m-1} \end{array} \right). \end{aligned}$$

Equating the first column on the left and right, we notice that

$$Aq_0 = \beta_{0,0}q_0 + \beta_{1,0}q_1.$$

Now, q_0 is given and $\|q_0\|_2 = 1$ since Q is unitary. Hence

$$q_0^H A q_0 = \beta_{0,0} q_0^H q_0 + \beta_{1,0} q_0^H q_1 = \beta_{0,0}.$$

Next,

$$\beta_{1,0} q_1 = A q_0 - \beta_{0,0} q_0 = \tilde{q}_1.$$

Since $\|q_1\|_2 = 1$ (it is a column of a unitary matrix) and $\beta_{1,0}$ is assumed to be positive, then we know that

$$\beta_{1,0} = \|\tilde{q}_1\|_2.$$

Finally,

$$q_1 = \tilde{q}_1 / \beta_{1,0}.$$

The point is that the first column of B and second column of Q are prescribed by the first column of Q and the fact that B has positive elements on the first subdiagonal. In this way, it can be successively argued that, one by one, each column of Q and each column of B are prescribed. ■

Homework 10.3.4.1 Give all the details of the above proof.

Solution. Assume that q_1, \dots, q_k and the column indexed with $k-1$ of B have been shown to be uniquely determined under the stated assumptions. We now show that then q_{k+1} and the column indexed by k of B are uniquely determined. (This is the inductive step in the

proof.) Then

$$Aq_k = \beta_{0,k}q_0 + \beta_{1,k}q_1 + \cdots + \beta_{k,k}q_k + \beta_{k+1,k}q_{k+1}.$$

We can determine $\beta_{0,k}$ through $\beta_{k,k}$ by observing that

$$q_j^H A q_k = \beta_{j,k}$$

for $j = 0, \dots, k$. Then

$$\beta_{k+1,k}q_{k+1} = Aq_k - (\beta_{0,k}q_0 + \beta_{1,k}q_1 + \cdots + \beta_{k,k}q_k) = \tilde{q}_{k+1}.$$

Since it is assumed that $\beta_{k+1,k} > 0$, it can be determined as

$$\beta_{k+1,k} = \|\tilde{q}_{k+1}\|_2$$

and then

$$q_{k+1} = \tilde{q}_{k+1}/\beta_{k+1,k}.$$

This way, the columns of Q and B can be determined, one-by-one.

Ponder This 10.3.4.2 Notice the similarity between the above proof and the proof of the existence and uniqueness of the QR factorization!

This can be brought out by observing that

$$\begin{aligned} (q_0 | A) \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & (q_0 | q_1 | q_2 | \cdots | q_{m-2} | q_{m-1}) \end{array} \right) \\ = (q_0 | q_1 | q_2 | \cdots | q_{m-2} | q_{m-1}) \left(\begin{array}{c|cccccc} 1 & \beta_{0,0} & \beta_{0,1} & \beta_{0,2} & \cdots & \beta_{0,m-2} & \beta_{0,m-1} \\ 0 & \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & \cdots & \beta_{1,m-2} & \beta_{1,m-1} \\ 0 & 0 & \beta_{2,1} & \beta_{2,2} & \cdots & \beta_{2,m-1} & \\ 0 & 0 & 0 & \beta_{3,2} & \cdots & \beta_{3,m-2} & \beta_{3,m-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \beta_{m-1,m-2} & \beta_{m-1,m-1} \end{array} \right). \end{aligned}$$

Puzzle through this observation and interpret what it means.



YouTube: <https://www.youtube.com/watch?v=1uDlTfWfH6s>

Remark 10.3.4.3 In our case, A is symmetric tridiagonal, and so is B .

10.3.5 The Francis implicit QR Step



YouTube: https://www.youtube.com/watch?v=RSm_Mqi0aSA

In the last unit, we described how, when $A^{(k)}$ is tridiagonal, the steps

$$\begin{aligned} A^{(k)} &\rightarrow Q^{(k)}R^{(k)} \\ A^{(k+1)} &:= R^{(k)}Q^{(k)} \end{aligned}$$

of an unshifted QR algorithm can be staged as the computation and application of a sequence of Givens' rotations. Obviously, one could explicitly form $A^{(k)} - \mu_k I$, perform these computations with the resulting matrix, and then add $\mu_k I$ to the result to compute

$$\begin{aligned} A^{(k)} - \mu_k I &\rightarrow Q^{(k)}R^{(k)} \\ A^{(k+1)} &:= R^{(k)}Q^{(k)} + \mu_k I. \end{aligned}$$

The Francis QR Step combines these separate steps into a single one, in the process casting all computations in terms of unitary similarity transformations, which ensures numerical stability.

Consider the 4×4 tridiagonal matrix

$$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & 0 \\ 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{pmatrix} - \mu I$$

The first Givens' rotation is computed from $\begin{pmatrix} \alpha_{0,0} - \mu \\ \alpha_{1,0} \end{pmatrix}$, yielding $\gamma_{1,0}$ and $\sigma_{1,0}$ so that

$$\begin{pmatrix} \gamma_{1,0} & -\sigma_{1,0} \\ \sigma_{1,0} & \gamma_{1,0} \end{pmatrix}^T \begin{pmatrix} \alpha_{0,0} - \mu \\ \alpha_{1,0} \end{pmatrix}$$

has a zero second entry. Now, to preserve eigenvalues, any orthogonal matrix that is applied from the left must also have its transpose applied from the right. Let us compute

$$\begin{aligned} &\left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \hat{\alpha}_{1,0} & \hat{\alpha}_{2,0} & 0 \\ \hat{\alpha}_{1,0} & \hat{\alpha}_{1,1} & \hat{\alpha}_{1,2} & 0 \\ \hline \hat{\alpha}_{2,0} & \hat{\alpha}_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) \\ &= \left(\begin{array}{cc|cc} \gamma_{1,0} & \sigma_{1,0} & 0 & 0 \\ -\sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|cc} \alpha_{0,0} & \alpha_{0,1} & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & 0 \\ \hline 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) \left(\begin{array}{cc|cc} \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right). \end{aligned}$$

This is known as "introducing the bulge."

Next, from $\begin{pmatrix} \hat{\alpha}_{1,0} \\ \hat{\alpha}_{2,0} \end{pmatrix}$, one can compute $\gamma_{2,0}$ and $\sigma_{2,0}$ so that

$$\begin{pmatrix} \gamma_{2,0} & -\sigma_{2,0} \\ \sigma_{2,0} & \gamma_{2,0} \end{pmatrix}^T \begin{pmatrix} \hat{\alpha}_{1,0} \\ \hat{\alpha}_{2,0} \end{pmatrix} = \begin{pmatrix} \tilde{\alpha}_{1,0} \\ 0 \end{pmatrix}.$$

Then

$$\begin{aligned} & \left(\begin{array}{c|cc|c} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{1,0} & 0 & 0 \\ \hline \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} \\ 0 & \hat{\alpha}_{2,1} & \hat{\alpha}_{2,2} & \hat{\alpha}_{2,3} \\ \hline 0 & \hat{\alpha}_{3,1} & \hat{\alpha}_{3,2} & \alpha_{3,3} \end{array} \right) \\ &= \left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & \gamma_{2,0} & \sigma_{2,0} & 0 \\ 0 & -\sigma_{2,0} & \gamma_{2,0} & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|ccc} \tilde{\alpha}_{0,0} & \hat{\alpha}_{1,0} & \hat{\alpha}_{2,0} & 0 \\ \hline \hat{\alpha}_{1,0} & \hat{\alpha}_{1,1} & \hat{\alpha}_{1,2} & 0 \\ \hat{\alpha}_{2,0} & \hat{\alpha}_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \hline 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) \left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & \gamma_{2,0} & -\sigma_{2,0} & 0 \\ 0 & \sigma_{2,0} & \gamma_{2,0} & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \end{aligned}$$

again preserves eigenvalues. Finally, from

$$\begin{pmatrix} \hat{\alpha}_{2,1} \\ \hat{\alpha}_{3,1} \end{pmatrix},$$

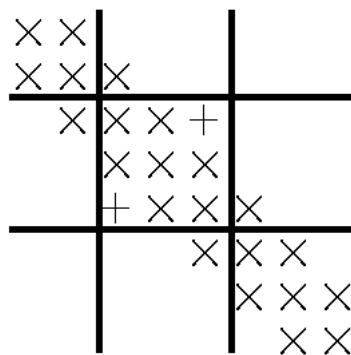
one can compute $\gamma_{3,1}$ and $\sigma_{3,1}$ so that

$$\begin{pmatrix} \gamma_{3,1} & -\sigma_{3,1} \\ \sigma_{3,1} & \gamma_{3,1} \end{pmatrix}^T \begin{pmatrix} \hat{\alpha}_{2,1} \\ \hat{\alpha}_{3,1} \end{pmatrix} = \begin{pmatrix} \tilde{\alpha}_{2,1} \\ 0 \end{pmatrix}.$$

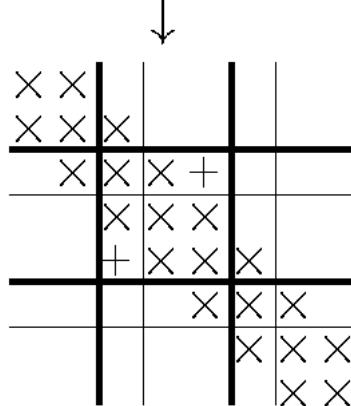
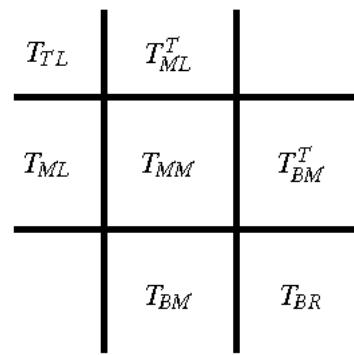
Then

$$\begin{aligned} & \left(\begin{array}{c|cc|c} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{1,0} & 0 & 0 \\ \hline \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \tilde{\alpha}_{2,1} & 0 \\ 0 & \tilde{\alpha}_{2,1} & \tilde{\alpha}_{2,2} & \tilde{\alpha}_{2,3} \\ \hline 0 & 0 & \tilde{\alpha}_{3,2} & \tilde{\alpha}_{3,3} \end{array} \right) \\ &= \left(\begin{array}{c|cc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & \gamma_{3,2} & \sigma_{3,2} \\ 0 & 0 & -\sigma_{3,2} & \gamma_{3,2} \end{array} \right) \left(\begin{array}{c|cc|c} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{1,0} & 0 & 0 \\ \hline \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} \\ 0 & \hat{\alpha}_{2,1} & \hat{\alpha}_{2,2} & \hat{\alpha}_{2,3} \\ \hline 0 & \hat{\alpha}_{3,1} & \hat{\alpha}_{3,2} & \alpha_{3,3} \end{array} \right) \left(\begin{array}{c|cc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & \gamma_{3,1} & -\sigma_{3,1} \\ 0 & 0 & \sigma_{3,1} & \gamma_{3,1} \end{array} \right), \end{aligned}$$

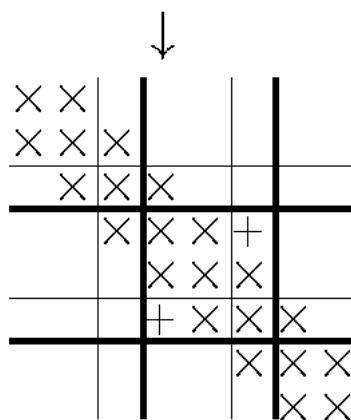
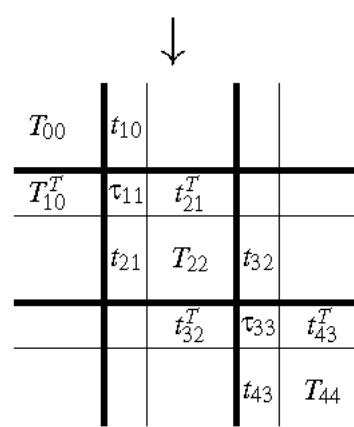
yielding a tridiagonal matrix. The process of transforming the matrix that results from introducing the bulge (the nonzero element $\hat{\alpha}_{2,0}$) back into a tridiagonal matrix is commonly referred to as "chasing the bulge." Moving the bulge one row and column down the matrix is illustrated in [Figure 10.3.5.1](#). The process of determining the first Givens' rotation, introducing the bulge, and chasing the bulge is known as a Francis Implicit QR step. An algorithm for this is given in [Figure 10.3.5.2](#).



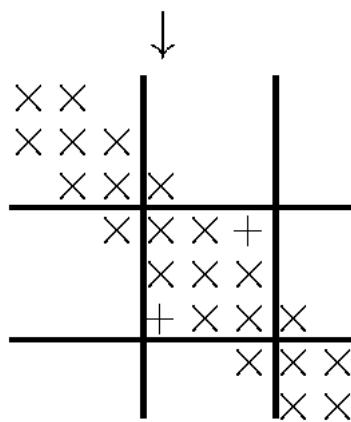
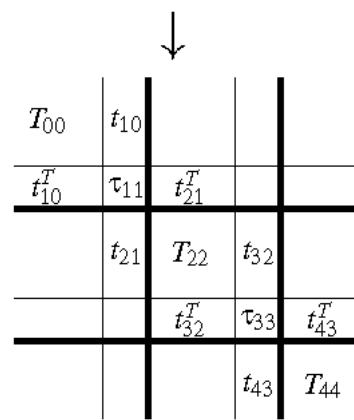
Beginning of iteration



Repartition



Update



End of iteration

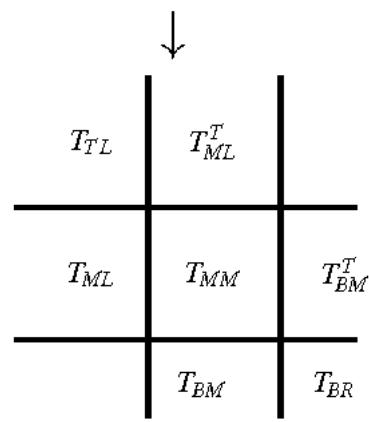


Figure 10.3.5.1 Illustration of how the bulge is chased one row and column forward in the

$T := \text{ChaseBulge}(T)$
$T \rightarrow \left(\begin{array}{c cc} T_{TL} & \star & \star \\ \hline T_{ML} & T_{MM} & \star \\ \hline 0 & T_{BM} & T_{BR} \end{array} \right)$
T_{TL} is 0×0 and T_{MM} is 3×3
while $m(T_{BR}) > 0$
$\left(\begin{array}{c cc} T_{TL} & \star & 0 \\ \hline T_{ML} & T_{MM} & \star \\ \hline 0 & T_{BM} & T_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc cc} T_{00} & \star & 0 & 0 & 0 \\ \hline t_{10}^T & \tau_{11} & \star & 0 & 0 \\ 0 & t_{21} & T_{22} & \star & 0 \\ \hline 0 & 0 & t_{32}^T & \tau_{33} & \star \\ 0 & 0 & 0 & t_{43} & T_{44} \end{array} \right)$
Compute (γ, σ) s.t. $G_{\gamma, \sigma}^T t_{21} = \begin{pmatrix} \tau_{21} \\ 0 \end{pmatrix}$, and assign $t_{21} := \begin{pmatrix} \tau_{21} \\ 0 \end{pmatrix}$
$T_{22} := G_{\gamma, \sigma}^T T_{22} G_{\gamma, \sigma}$
$t_{32}^T := t_{32}^T G_{\gamma, \sigma}$ (not performed during final step)
$\left(\begin{array}{c cc} T_{TL} & \star & 0 \\ \hline T_{ML} & T_{MM} & \star \\ \hline 0 & T_{BM} & T_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc cc} T_{00} & \star & 0 & 0 & 0 \\ \hline t_{10}^T & \tau_{11} & \star & 0 & 0 \\ 0 & t_{21} & T_{22} & \star & 0 \\ \hline 0 & 0 & t_{32}^T & \tau_{33} & \star \\ 0 & 0 & 0 & t_{43} & T_{44} \end{array} \right)$
endwhile

Figure 10.3.5.2 Algorithm for "chasing the bulge" that, given a tridiagonal matrix with an additional nonzero $a_{2,0}$ element, reduces the given matrix back to a tridiagonal matrix.

The described process has the net result of updating $A^{(k+1)} = Q^T A^{(k)} Q^{(k)}$, where Q is the orthogonal matrix that results from multiplying the different Givens' rotations together:

$$Q = \left(\begin{array}{c|cc} \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \hline \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} 1 & 0 & 0 & 0 \\ \hline 0 & \gamma_{2,0} & -\sigma_{2,0} & 0 \\ 0 & \sigma_{2,0} & \gamma_{2,0} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & \gamma_{3,1} & -\sigma_{3,1} \\ 0 & 0 & \sigma_{3,1} & \gamma_{3,1} \end{array} \right).$$

Importantly, the first column of Q , given by

$$\begin{pmatrix} \gamma_{1,0} \\ \sigma_{1,0} \\ 0 \\ 0 \end{pmatrix},$$

is exactly the same first column had Q been computed as in [Subsection 10.3.3 \(10.3.1\)](#). Thus, by the Implicit Q Theorem, the tridiagonal matrix that results from this approach is equal to the tridiagonal matrix that would be computed by applying the QR factorization from that section to $A - \mu I$, $A - \mu I \rightarrow QR$ followed by the formation of $RQ + \mu I$ using the algorithm for computing RQ in [Subsection 10.3.3](#).

Remark 10.3.5.3 In Figure 10.3.5.2, we use a variation of the notation we have encountered when presenting many of our algorithms, n including most recently the reduction to tridiagonal form. The fact is that when implementing the implicitly shifted QR algorithm, it is best to do so by explicitly indexing into the matrix. This tridiagonal matrix is typically stored as just two vectors: one for the diagonal and one for the subdiagonal.

Homework 10.3.5.1 A typical step when "chasing the bulge" one row and column further down the matrix involves the computation

$$\left(\begin{array}{c|cc|c} \alpha_{i-1,i-1} & \times & \times & 0 \\ \widehat{\alpha}_{i,i-1} & \widehat{\alpha}_{i,i} & \times & 0 \\ \hline 0 & \widehat{\alpha}_{i+1,i} & \widehat{\alpha}_{i+1,i+1} & \times \\ \hline 0 & \widehat{\alpha}_{i+2,i} & \widehat{\alpha}_{i+2,i+1} & \alpha_{i+2,i+2} \end{array} \right) = \left(\begin{array}{c|cc|c} 1 & 0 & 0 & 0 \\ 0 & \gamma_i & \sigma_i & 0 \\ 0 & -\sigma_i & \gamma_i & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} \alpha_{i-1,i-1} & \times & \times & 0 \\ \alpha_{i,i-1} & \alpha_{i,i} & \times & 0 \\ \hline \alpha_{i+1,i-1} & \alpha_{i+1,i} & \alpha_{i+1,i+1} & \times \\ \hline 0 & 0 & \alpha_{i+2,i+1} & \alpha_{i+2,i+2} \end{array} \right) \left(\begin{array}{c|cc|c} 1 & 0 & 0 & 0 \\ 0 & \gamma_i & -\sigma_i & 0 \\ 0 & \sigma_i & \gamma_i & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right)$$

Give a strategy (or formula) for computing

$$\left(\begin{array}{c|cc|c} \widehat{\alpha}_{i,i-1} & \widehat{\alpha}_{i,i} & & \\ \hline \widehat{\alpha}_{i+1,i} & \widehat{\alpha}_{i+1,i+1} & & \\ \hline \widehat{\alpha}_{i+2,i} & \widehat{\alpha}_{i+2,i+1} & & \end{array} \right)$$

Solution. Since the subscripts will drive us crazy, let's relabel, add one of the entries above the diagonal, and drop the subscripts on γ and σ :

$$\left(\begin{array}{c|cc|c} \times & \times & \times & 0 \\ \widehat{\epsilon} & \widehat{\kappa} & \widehat{\lambda} & 0 \\ 0 & \widehat{\lambda} & \widehat{\mu} & \times \\ \hline 0 & \widehat{\chi} & \widehat{\psi} & \times \end{array} \right) = \left(\begin{array}{c|cc|c} 1 & 0 & 0 & 0 \\ 0 & \gamma & \sigma & 0 \\ 0 & -\sigma & \gamma & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} \times & \times & \times & 0 \\ \epsilon & \kappa & \lambda & 0 \\ \phi & \lambda & \mu & \times \\ \hline 0 & 0 & \psi & \times \end{array} \right) \left(\begin{array}{c|cc|c} 1 & 0 & 0 & 0 \\ 0 & \gamma & -\sigma & 0 \\ 0 & \sigma & \gamma & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right)$$

With this, the way I would compute the desired results is via the steps

- $\widehat{\epsilon} := \gamma\epsilon + \sigma\phi$
- $\begin{pmatrix} \widehat{\kappa} & \widehat{\lambda} \\ \widehat{\lambda} & \widehat{\mu} \end{pmatrix} := \left[\begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \begin{pmatrix} \kappa & \lambda \\ \lambda & \mu \end{pmatrix} \right] \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}$
- $\widehat{\chi} := \sigma\psi$
- $\widehat{\psi} := \gamma\chi$

Translating this to the update of the actual entries is straight forward.

Ponder This 10.3.5.2 Write a routine that performs one Francis implicit QR step. Use it to write an implicitly shifted QR algorithm.

10.3.6 A complete algorithm



YouTube: <https://www.youtube.com/watch?v=fqiex-FQ-JU>



YouTube: <https://www.youtube.com/watch?v=53XcY9IQDU0>

The last unit shows how one iteration of the QR algorithm can be performed on a tridiagonal matrix by implicitly shifting and then "chasing the bulge." All that is left to complete the algorithm is to note that

- The shift μ_k can be chosen to equal $\alpha_{m-1,m-1}$ (the last element on the diagonal, which tends to converge to the eigenvalue smallest in magnitude). In practice, choosing the shift to be an eigenvalue of the bottom-right 2×2 matrix works better. This is known as the **Wilkinson shift**.
- If $A = QTQ^T$ reduced A to the tridiagonal matrix T before the QR algorithm commenced, then the Givens' rotations encountered as part of the implicitly shifted QR algorithm can be applied from the right to the appropriate columns of Q so that upon completion Q is left overwritten with the eigenvectors of A . Let's analyze this:
 - Reducing the matrix to tridiagonal form requires $O(m^3)$ computations.
 - Forming Q from the Householder vectors requires $O(m^3)$ computations.
 - Applying Givens' rotation to a pairs of columns of Q requires $O(m)$ computation per Givens' rotation. For each Francis implicit QR step $O(n)$ Givens' rotations are computed, making the application of Givens' rotations to Q of cost $O(m^2)$ per iteration of the implicitly shifted QR algorithm. Typically a few (2-3) iterations are needed per eigenvalue that is uncovered (when deflation is incorporated), meaning

that $O(m)$ iterations are needed. Thus, a QR algorithm with a tridiagonal matrix that accumulates eigenvectors requires $O(m^3)$ computation.

Thus, the total cost of computing the eigenvalues and eigenvectors is $O(m^3)$.

- If an element on the subdiagonal becomes zero (or very small), and hence the corresponding element of the superdiagonal, then

$$T = \left(\begin{array}{c|c} T_{00} & 0 \\ \hline 0 & T_{11} \end{array} \right) \quad \begin{array}{c|c} \times & \times \\ \times & \times & \times \\ & \times & \times & \times \\ & & \times & \times & \times \\ \hline & 0 & & & \\ 0 & & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{array}$$

then

- The computation can continue separately with T_{00} and T_{11} .
- One can pick the shift from the bottom-right of T_{00} as one continues finding the eigenvalues of T_{00} , thus accelerating that part of the computation.
- One can pick the shift from the bottom-right of T_{11} as one continues finding the eigenvalues of T_{11} , thus accelerating that part of the computation.
- One must continue to accumulate the eigenvectors by applying the rotations to the appropriate columns of Q .
- Because of the connection between the QR algorithm and the Inverse Power Method, subdiagonal entries near the bottom-right of T are more likely to converge to a zero, so most deflation will happen there.
- A question becomes when an element on the subdiagonal, $\tau_{i+1,i}$ can be considered to be zero. The answer is when $|\tau_{i+1,i}|$ is small relative to $|\tau_i|$ and $|\tau_{i+1,i+1}|$. A typical condition that is used is

$$|\tau_{i+1,i}| \leq \epsilon_{\text{mach}} \sqrt{|\tau_{i,i}| + |\tau_{i+1,i+1}|}.$$

For details, see some of our papers mentioned in the enrichments.

10.4 Enrichments

10.4.1 QR algorithm among the most important algorithms of the 20th century

An article published in SIAM News, a publication of the Society for Industrial and Applied Mathematics, lists the QR algorithm among the ten most important algorithms of the 20th century [10]:

- Barry A. Cipra, [The Best of the 20th Century: Editors Name Top 10 Algorithms](#), SIAM News, Volume 33, Number 4, 2000.

10.4.2 Who was John Francis

Below is a posting by the late Gene Golub in NA Digest Sunday, August 19, 2007 Volume 07 : Issue 34.

From: Gene H Golub \lt golub@stanford.edu \gt

Date: Sun, 19 Aug 2007 13:54:47 -0700 (PDT)

Subject: John Francis, Co-Inventor of QR

Dear Colleagues,

For many years, I have been interested in meeting J G F Francis, one of the co-inventors of the QR algorithm for computing eigenvalues of general matrices. Through a lead provided by the late Erin Brent and with the aid of Google, I finally made contact with him.

John Francis was born in 1934 in London and currently lives in Hove, near Brighton. His residence is about a quarter mile from the sea; he is a widower. In 1954, he worked at the National Research Development Corp (NRDC) and attended some lectures given by Christopher Strachey.

In 1955, '56 he was a student at Cambridge but did not complete a degree. He then went back to NRDC as an assistant to Strachey where he got involved in flutter computations and this led to his work on QR.

After leaving NRDC in 1961, he worked at the Ferranti Corp and then at the University of Sussex. Subsequently, he had positions with various industrial organizations and consultancies. He is now retired. His interests were quite general and included Artificial Intelligence, computer languages, systems engineering. He has not returned to numerical computation.

He was surprised to learn there are many references to his work and that the QR method is considered one of the ten most important algorithms of the 20th century. He was unaware of such developments as TeX and Math Lab. Currently he is working on a degree at the Open University.

John Francis did remarkable work and we are all in his debt. Along with the conjugate gradient method, it provided us with one of the basic tools of numerical analysis.

Gene Golub

10.4.3 Casting the reduction to tridiagonal form in terms of matrix-matrix multiplication

For many algorithms, we have discussed blocked versions that cast more computation in terms of matrix-matrix multiplication, thus achieving portable high performance (when linked to a high-performance implementation of matrix-matrix multiplication). The inconvenient truth is that reduction to tridiagonal form can only be partially cast in terms of matrix-matrix multiplication. This is a severe hindrance to high performance for that first step towards computing all eigenvalues and eigenvector of a Hermitian matrix. Worse, a considerable fraction of the total cost of the computation is in that first step.

For a detailed discussion on the blocked algorithm for reduction to tridiagonal form, we recommend

- [44] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, G. Joseph Elizondo, Families of Algorithms for Reducing a Matrix to Condensed Form, ACM Transactions on Mathematical Software (TOMS) , Vol. 39, No. 1, 2012.

Tridiagonal form is one case of what is more generally referred to as "condensed form."

10.4.4 Optimizing the tridiagonal QR algorithm

As the Givens' rotations are applied to the tridiagonal matrix, they are also applied to a matrix in which eigenvectors are accumulated. While one Implicit Francis QR Step requires $O(n)$ computation for chasing the bulge, this accumulation of the eigenvectors requires $O(n^2)$ computation with $O(n^2)$ data per step. This inherently means the cost of accessing data dominates on current architectures.

In a paper, we showed how accumulating the Givens' rotations for several Francis Steps before applying these to the matrix in which the eigenvectors are being computed allows one to attain high performance similar to that attained by a matrix-matrix multiplication.

- [42] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, Restructuring the Tridiagonal and Bidiagonal QR Algorithms for Performance, ACM Transactions on Mathematical Software (TOMS), Vol. 40, No. 3, 2014.

For computing all eigenvalues and eigenvectors of a dense Hermitian matrix, this approach is competitive with the Method of Relatively Robust Representations (MRRR), which we mention in [Subsection 10.4.5](#)

10.4.5 The Method of Multiple Relatively Robust Representations (MRRR)

The Method of Multiple Relative Robust Representations (MRRR) computes the eigenvalues and eigenvectors of a $m \times m$ tridiagonal matrix in $O(m^2)$ time. It can be argued that

this is within a constant factor of the lower bound for computing these eigenvectors since the eigenvectors constitute $O(m^2)$ data that must be written upon the completion of the computation.

When computing the eigenvalues and eigenvectors of a dense Hermitian matrix, MRRR can replace the implicitly shifted QR algorithm for finding the eigenvalues and eigenvectors of the tridiagonal matrix. The overall steps then become

- Reduce matrix A to tridiagonal form:

$$A \rightarrow Q_A T Q_A^H$$

where T is a tridiagonal real valued matrix. The matrix Q_A is not explicitly formed but instead the Householder vectors that were computed as part of the reduction to tridiagonal form are stored.

- Compute the eigenvalues and eigenvectors of the tridiagonal matrix T :

$$T \rightarrow Q_T D Q_T^T.$$

- "Back transform" the eigenvectors by forming $Q_A Q_T$ (applying the Householder transformations that define Q_A to Q_T).

The details of that method go beyond the scope of this note. We refer the interested reader to

- [12] Inderjit S. Dhillon and Beresford N. Parlett, Multiple Representations to Compute Orthogonal Eigenvectors of Symmetric Tridiagonal Matrices, Lin. Alg. Appl., Vol. 387, 2004.
- [3] Paolo Bientinesi, Inderjit S. Dhillon, Robert A. van de Geijn, A Parallel Eigensolver for Dense Symmetric Matrices Based on Multiple Relatively Robust Representations, SIAM Journal on Scientific Computing, 2005

Remark 10.4.5.1 An important feature of MRRR is that it can be used to find a subset of eigenvectors. This is in contrast to the QR algorithm, which computes all eigenvectors.

10.5 Wrap Up

10.5.1 Additional homework

Homework 10.5.1.1 You may want to do a new "git pull" to update directory `Assignments`

In `Assignments/Week10/matlab` you will find the files

- `Givens_rotation.m`: A function that computes a Givens' rotation from a 2×1 vector x .

- `Francis_Step.m`: A function that performs a Francis Implicit QR Step with a tridiagonal matrix T (stored as the diagonal and subdiagonal of T).
- `Test_Francis_Step.m`: A very rudimentary script that performs a few calls to the function `Francis_Step`. Notice that our criteria for the routine being correct is that the matrix retains the correct eigenvalues.

With this,

1. Investigate the convergence of the $(m, m - 1)$ element of matrix $\mathsf{T1}$.
2. Write a function

```
function  $\mathsf{T} = \text{Spectral\_Decomposition\_Lambda}(\mathsf{T})$ 
```

That returns Λ such that $T = Q\Lambda Q^T$ is the Spectral Decomposition of T . The input matrix T is a tridiagonal matrix where only the lower triangular part of the matrix is stored in the diagonal and first subdiagonal of array T . The diagonal matrix Λ is returned in T . The upper triangular part of the array should not change values. You are encouraged to call the function `Francis_Step` from the function `Spectral_Decomposition_Lambda`. Obviously, you need to incorporate deflation in your implementation. How to handle the final 2×2 matrix is an interesting question... (You may use the matlab function `eig` for this.)

10.5.2 Summary

We have noticed that typos are uncovered relatively quickly once we release the material. Because we "cut and paste" the summary from the materials in this week, we are delaying adding the summary until most of these typos have been identified.

Week 11

Computing the SVD

11.1 Opening

11.1.1 Linking the Singular Value Decomposition to the Spectral Decomposition



YouTube: https://www.youtube.com/watch?v=LaYzn2x_Z8Q

Week 2 introduced us to the Singular Value Decomposition (SVD) of a matrix. For any matrix $A \in \mathbb{C}^{m \times n}$, there exist unitary matrix $U \in \mathbb{C}^{m \times m}$, unitary matrix $V \in \mathbb{C}^{n \times n}$, and $\Sigma \in \mathbb{R}^{m \times n}$ of the form

$$\Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0_{r \times (n-r)} \\ \hline 0_{(m-r) \times r} & 0_{(m-r) \times (n-r)} \end{array} \right), \text{ with } \Sigma_{TL} = \text{diag}(\sigma_0, \dots, \sigma_{r-1}), \quad (11.1.1)$$

and $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{r-1} > 0$

such that $A = U\Sigma V^H$, the SVD of matrix A . We can correspondingly partition $U = (U_L | U_R)$ and $V = (V_L | V_R)$, where U_L and V_L have r columns, in which case

$$A = U_L \Sigma_{TL} V_L^H$$

equals the Reduced Singular Value Decomposition. We did not present practical algorithms for computing this very important result in Week 2, because we did not have the theory and practical insights in place to do so. With our discussion of the QR algorithm in the last week, we can now return to the SVD and present the fundamentals that underlie its computation.

In Week 10, we discovered algorithms for computing the Spectral Decomposition of a Hermitian matrix. The following exercises link the SVD of A to the Spectral Decomposition of $B = A^H A$, providing us with a first hint as to how to practically compute the SVD.

Homework 11.1.1.1 Let $A \in \mathbb{C}^{m \times n}$ and $A = U\Sigma V^H$ its SVD, where Σ has the structure indicated in (11.1.1). Give the Spectral Decomposition of the matrix $A^H A$.

Solution.

$$\begin{aligned} A^H A &= \langle A = U\Sigma V^H \rangle \\ (U\Sigma V^H)^H (U\Sigma V^H) &= \langle (BC)^H = C^H B^H; U^H U = I \rangle \\ V \Sigma^T \Sigma V^H &= \\ V \left(\begin{array}{c|c} \Sigma_{TL}^2 & 0_{r \times (n-r)} \\ \hline 0_{(n-r) \times r} & 0_{(n-r) \times (n-r)} \end{array} \right) V^H. \end{aligned}$$

Homework 11.1.1.2 Let $A \in \mathbb{C}^{m \times n}$ and $A = U\Sigma V^H$ its SVD, where Σ has the structure indicated in (11.1.1). Give the Spectral Decomposition of the matrix AA^H .

Solution.

$$\begin{aligned} AA^H &= \\ (U\Sigma V^H)(U\Sigma V^H)^H &= \\ U\Sigma \Sigma^T U^H &= \\ U \left(\begin{array}{c|c} \Sigma_{TL}^2 & 0_{r \times (m-r)} \\ \hline 0_{(m-r) \times r} & 0_{(m-r) \times (m-r)} \end{array} \right) U^H. \end{aligned}$$

Last two homeworks expose how to compute the Spectral Decomposition of $A^H A$ or AA^H from the SVD of matrix A . We already discovered practical algorithms for computing the Spectral Decomposition in the last week. What we really want to do is to turn this around: How do we compute the SVD of A from the Spectral Decomposition of $A^H A$ and/or AA^H ?

11.1.2 Overview

- 11.1 Opening
 - 11.1.1 Linking the Singular Value Decomposition to the Spectral Decomposition
 - 11.1.2 Overview
 - 11.1.3 What you will learn
- 11.2 Practical Computation of the Singular Value Decomposition
 - 11.2.1 Computing the SVD from the Spectral Decomposition
 - 11.2.2 A strategy for computing the SVD

- 11.2.3 Reduction to bidiagonal form
- 11.2.4 Implicitly shifted bidiagonal QR algorithm
- 11.3 Jacobi's Method
 - 11.3.1 Jacobi rotation
 - 11.3.2 Jacobi's method for computing the Spectral Decomposition
 - 11.3.3 Jacobi's method for computing the Singular Value Decomposition
- 11.4 Enrichments
 - 11.4.1 Casting the reduction to bidiagonal form in terms of matrix-matrix multiplication
 - 11.4.2 Optimizing the bidiagonal QR algorithm
- 11.5 Wrap Up
 - 11.5.1 Additional homework
 - 11.5.2 Summary

11.1.3 What you will learn

This week, you finally discover practical algorithms for computing the SIngular Value Decomposition.

Upon completion of this week, you should be able to

- Link the (Reduced) SIngular Value Decomposition of A to the Spectral Decomposition of $A^H A$.
- Reduce a matrix to bidiagonal form.
- Transform the implicitly shifted QR algorithm into the implicitly shifted bidiagonal QR algorithm.
- Use Jacobi rotations to propose alternative algorithms, known as Jacobi's Methods, for computing the Spectral Decomposition of a symmetric matrix and Singular Value Decomposition of a general real-valued matrix.

11.2 Practical Computation of the Singular Value Decomposition

11.2.1 Computing the SVD from the Spectral Decomposition



YouTube: <https://www.youtube.com/watch?v=aSvGPY09b48>

Let's see if we can turn the discussion from [Subsection 11.1.1](#) around: Given the Spectral Decomposition of $A^H A$, how can we extract the SVD of A ?

Homework 11.2.1.1 Let $A \in \mathbb{C}^{m \times m}$ be nonsingular and $A^H A = Q D Q^H$, the Spectral Decomposition of $A^H A$. Give a formula for U , V , and Σ so that $A = U \Sigma V^H$ is the SVD of A . (Notice that A is square.)

Solution. Since A is nonsingular, so is $A^H A$ and hence D has positive real values on its diagonal. If we take $V = Q$ and $\Sigma = D^{1/2}$ then

$$A = U \Sigma V^H = U D^{1/2} Q^H.$$

This suggests that we choose

$$U = A V \Sigma^{-1} = A Q D^{-1/2}.$$

We can easily verify that U is unitary:

$$\begin{aligned} U^H U &= \\ &= (A Q D^{-1/2})^H (A Q D^{-1/2}) \\ &= D^{-1/2} Q^H A^H A Q D^{-1/2} \\ &= D^{-1/2} D D^{-1/2} \\ &= I. \end{aligned}$$

The final detail is that the Spectral Decomposition does not require the diagonal elements of D to be ordered from largest to smallest. This can be easily fixed by permuting the columns of Q and, correspondingly, the diagonal elements of D .



YouTube: <https://www.youtube.com/watch?v=sAbXHD4TMSE>

Not all matrices are square and nonsingular. In particular, we are typically interested in the SVD of matrices where $m > n$. Let's examine how to extract the SVD from the Spectral Decomposition of $A^H A$ for such matrices.

Homework 11.2.1.2 Let $A \in \mathbb{C}^{m \times n}$ have full column rank and let $A^H A = Q D Q^H$, the Spectral Decomposition of $A^H A$. Give a formula for the Reduced SVD of A .

Solution. We notice that if A has full column rank, then its Reduced Singular Value Decomposition is given by $A = U_L \Sigma V^H$, where $U_L \in \mathbb{C}^{m \times n}$, $\Sigma \in \mathbb{R}^{n \times n}$, and $V \in \mathbb{C}^{n \times n}$. Importantly, $A^H A$ is nonsingular, and D has positive real values on its diagonal. If we take $V = Q$ and $\Sigma = D^{1/2}$ then

$$A = U_L \Sigma V^H = U_L D^{1/2} Q^H.$$

This suggests that we choose

$$U_L = A V \Sigma^{-1} = A Q D^{-1/2},$$

where, clearly, $\Sigma = D^{1/2}$ is nonsingular. We can easily verify that U_L has orthonormal columns:

$$\begin{aligned} U_L^H U_L &= \\ &= (A Q D^{-1/2})^H (A Q D^{-1/2}) \\ &= D^{-1/2} Q^H A^H A Q D^{-1/2} \\ &= D^{-1/2} D D^{-1/2} \\ &= I. \end{aligned}$$

As before, the final detail is that the Spectral Decomposition does not require the diagonal elements of D to be ordered from largest to smallest. This can be easily fixed by permuting the columns of Q and, correspondingly, the diagonal elements of D .

The last two homeworks gives us a first glimpse at a practical procedure for computing the (Reduced) SVD from the Spectral Decomposition, for the simpler case where A has full column rank.

- Form $B = A^H A$.
- Compute the Spectral Decomposition $B = Q D Q^H$ via, for example, the QR algorithm.

- Permute the columns of Q and diagonal elements of D so that the diagonal elements are ordered from largest to smallest. If P is the permutation matrix such that PDP^T reorders the diagonal of D appropriately, then

$$\begin{aligned}
 A^H A &= \underbrace{Q D Q^H}_{\text{Spectral Decomposition}} \\
 &= \underbrace{Q \underbrace{P^T P}_I D \underbrace{P^T P}_I Q^H}_{\text{insert identities}} \\
 &= \underbrace{(Q P^T) (P D P^T) (P Q^H)}_{\text{associativity}} \\
 &= \underbrace{(Q P^T) (P D P^T) (Q P^T)^H}_{(BC)^H = C^H B^H H}
 \end{aligned}$$

- Let $V = QP^T$, $\Sigma = (PDP^T)^{1/2}$ (which is diagonal), and $U_L = AV\Sigma^{-1}$.

With these insights, we find the Reduced SVD of a matrix with linearly independent columns. If in addition A is square (and hence nonsingular), then $U = U_L$ and $A = U\Sigma V^H$ is its SVD.

Let us now treat the problem in full generality.

Homework 11.2.1.3 Let $A \in \mathbb{C}^{m \times n}$ be of rank r and

$$A^H A = \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right) \left(\begin{array}{c|c} D_{TL} & 0_{r \times (n-r)} \\ \hline 0_{(n-r) \times r} & 0_{(n-r) \times (n-r)} \end{array} \right) \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)^H$$

be the Spectral Decomposition of $A^H A$, where $Q_L \in \mathbb{C}^{n \times r}$ and, for simplicity, we assume the diagonal elements of D_{TL} are ordered from largest to smallest. Give a formula for the Reduced SVD of A .

Solution. The Reduced SVD of A is given by $A = U_L \Sigma_{TL} V_L^H$, where Σ_{TL} is $r \times r$ is diagonal with positive real values along its diagonal, ordered from largest to smallest. If we take $V_L = Q_L$ and $\Sigma_{TL} = D_{TL}^{1/2}$ then

$$A = U_L \Sigma_{TL} V_L^H = U_L D^{1/2} Q_L^H.$$

This suggests that we choose

$$U_L = A V_L \Sigma_{TL}^{-1} = A Q_L D_{TL}^{-1/2}.$$

We can easily verify that U_L has orthonormal columns:

$$\begin{aligned}
 & U_L^H U_L \\
 &= \\
 & (AQ_L D_{TL}^{-1/2})^H (AQ_L D_{TL}^{-1/2}) \\
 &= \\
 & D_{TL}^{-1/2} Q_L^H A^H A Q_L D_{TL}^{-1/2} \\
 &= \\
 & D_{TL}^{-1/2} Q_L^H \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right) \left(\begin{array}{c|c} D_{TL} & 0_{r \times (n-r)} \\ \hline 0_{(n-r) \times r} & 0_{(n-r) \times (n-r)} \end{array} \right) \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)^H Q_L D_{TL}^{-1/2} \\
 &= \\
 & D_{TL}^{-1/2} \left(\begin{array}{c|c} I & 0 \end{array} \right) \left(\begin{array}{c|c} D_{TL} & 0_{r \times (n-r)} \\ \hline 0_{(n-r) \times r} & 0_{(n-r) \times (n-r)} \end{array} \right) \left(\begin{array}{c|c} I & 0 \end{array} \right)^H D_{TL}^{-1/2} \\
 &= \\
 & D_{TL}^{-1/2} D_{TL} D_{TL}^{-1/2} \\
 &= \\
 & I.
 \end{aligned}$$

Although the discussed approaches give us a means by which to compute the (Reduced) SVD that is mathematically sound, the Achilles heel of these is that it hinges on forming $A^H A$. While beyond the scope of this course, the conditioning of computing a Spectral Decomposition of a Hermitian matrix is dictated by the condition number of the matrix, much like solving a linear system is. We recall from [Subsection 4.2.5](#) that we avoid using the Method of Normal Equations to solve the linear least squares problem when a matrix is ill-conditioned. Similarly, we try to avoid computing the SVD from $A^H A$. The problem here is even more acute: it is often the case that A is (nearly) rank deficient (for example, in situations where we desire a low rank approximation of a given matrix) and hence it is frequently the case that the condition number of A is very unfavorable. The question thus becomes, how can we avoid computing $A^H A$ while still benefiting from the insights in this unit?

11.2.2 A strategy for computing the SVD

Remark 11.2.2.1 In this section, we discuss both the QR factorization and the QR algorithm. The QR factorization, discussed in [Week 3](#), is given by $A = QR$. The QR algorithm, which we discussed in [Week 10](#), instead computes the Spectral Decomposition of a Hermitian matrix. It can be modified to compute the Schur Decomposition instead, which we don't discuss in this course. It can also be modified to compute the SVD of a matrix, which we discuss in this, and subsequent, units.



YouTube: <https://www.youtube.com/watch?v=SXP12WwhtJA>

The first observation that leads to a practical algorithm is that matrices for which we wish to compute the SVD are often **tall and skinny**, by which we mean that they have many more rows than they have columns, and it is the Reduced SVD of this matrix that is desired. The methods we will develop for computing the SVD are based on the implicitly shifted QR algorithm that was discussed in [Subsection 10.3.5](#), which requires $O(n^3)$ computation when applied to an $n \times n$ matrix. Importantly, the leading n^3 term has a very large constant relative to, say, the cost of a QR factorization of that same matrix.

Rather than modifying the QR algorithm to work with a tall and skinny matrix, we start by computing its QR factorization, $A = QR$. After this, the SVD of the smaller, $n \times n$ sized, matrix R is computed. The following homework shows how the Reduced SVD of A can be extracted from Q and the SVD of R .

Homework 11.2.2.1 Let $A \in \mathbb{C}^{m \times n}$, with $m \geq n$, and $A = QR$ be its QR factorization where, for simplicity, we assume that $n \times n$ upper triangular matrix R is nonsingular. If $R = \hat{U}\hat{\Sigma}\hat{V}^H$ is the SVD of R , give the Reduced SVD of A .

Solution.

$$\begin{aligned} A &= \\ QR &= \\ Q\hat{U}\hat{\Sigma}\hat{V}^H &= \\ \underbrace{(Q\hat{U})}_{U} \underbrace{\hat{\Sigma}}_{\Sigma} \underbrace{\hat{V}^H}_{V^H}. \end{aligned}$$



YouTube: <https://www.youtube.com/watch?v=T0NYsbdaC78>

While it would be nice if the upper triangular structure of R was helpful in computing its SVD, it is actually the fact that that matrix is square and small (if $n \ll m$) that is significant. For this reason, we now assume that we are interested in finding the SVD of a square matrix A , and ignore the fact that that matrix may be triangular.



YouTube: <https://www.youtube.com/watch?v=sGBD0-PSMN8>

Here are some more observations, details of which will become clear in the next units:

- In [Subsection 10.3.1](#), we saw that an $m \times m$ Hermitian matrix can be reduced to tridiagonal form via a sequence of Householder transformations that are applied from the left and the right to the matrix. This then greatly reduced the cost of the QR algorithm that was used to compute the Spectral Decomposition.

In the next unit, we will see that one can similarly reduce a matrix to **bidiagonal** form, a matrix that has only a nonzero diagonal and super diagonal. In other words, there is a similarity transformation such that

$$A = Q_A B Q_A^H,$$

where B is bidiagonal. Conveniently, B can also be forced to be real-valued.

- The observation now is that $B^T B$ is a real-valued tridiagonal matrix. Thus, if we explicitly form $T = B^T B$, then we can employ the implicitly shifted QR algorithm (or any other tridiagonal eigensolver) to compute its Spectral Decomposition and from that construct the SVD of B , the SVD of the square matrix A , and the Reduced SVD of whatever original $m \times n$ matrix we started with.
- We don't want to explicitly form $B^T B$ because the condition number of B equals the condition number of the original problem (since they are related via unitary transformations).
- In the next units, we will find that we can again employ the Implicit Q Theorem to compute the SVD of B , inspired by the implicitly shifted QR algorithm. The algorithm we develop again casts all updates to B in terms of unitary transformations, yielding a highly accurate algorithm.

Putting these observations together yields a practical methodology for computing the Reduced SVD of a matrix.

11.2.3 Reduction to bidiagonal form



YouTube: <https://www.youtube.com/watch?v=20W5Yi6Q0dY>

Homework 11.2.3.1 Let $B \in \mathbb{R}^{m \times m}$ be a bidiagonal matrix:

$$B = \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & 0 & \cdots & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \beta_{m-2,m-2} & \beta_{m-2,m-1} \\ 0 & 0 & 0 & \cdots & 0 & \beta_{m-1,m-1} \end{pmatrix}.$$

Show that $T = B^T B$ is a tridiagonal symmetric matrix.

Solution.

$$\begin{aligned} B^T B &= \\ &\left(\begin{array}{cccc} \beta_{0,0} & \beta_{0,1} & 0 & \cdots \\ 0 & \beta_{1,1} & \beta_{1,2} & \cdots \\ 0 & 0 & \beta_{2,2} & \cdots \\ \vdots & \ddots & \ddots & \ddots \end{array} \right)^T \left(\begin{array}{cccc} \beta_{0,0} & \beta_{0,1} & 0 & \cdots \\ 0 & \beta_{1,1} & \beta_{1,2} & \cdots \\ 0 & 0 & \beta_{2,2} & \cdots \\ \vdots & \ddots & \ddots & \ddots \end{array} \right) = \\ &\left(\begin{array}{cccc} \beta_{0,0} & 0 & 0 & \cdots \\ \beta_{0,1} & \beta_{1,1} & 0 & \cdots \\ 0 & \beta_{1,2} & \beta_{2,2} & \cdots \\ \vdots & \ddots & \ddots & \ddots \end{array} \right) \left(\begin{array}{cccc} \beta_{0,0} & \beta_{0,1} & 0 & \cdots \\ 0 & \beta_{1,1} & \beta_{1,2} & \cdots \\ 0 & 0 & \beta_{2,2} & \cdots \\ \vdots & \ddots & \ddots & \ddots \end{array} \right) \\ &= \\ &\left(\begin{array}{cccc} \beta_{0,0}^2 & \beta_{0,1}\beta_{0,0} & 0 & \cdots \\ \beta_{0,1}\beta_{0,0} & \beta_{0,1}^2 + \beta_{1,1}^2 & \beta_{1,2}\beta_{1,1} & \cdots \\ 0 & \beta_{1,2}\beta_{1,1} & \beta_{1,2}^2 + \beta_{2,2}^2 & \cdots \\ \vdots & \ddots & \ddots & \ddots \end{array} \right) \end{aligned}$$

Given that we can preprocess our problem by computing its QR factorization, we focus now on the case where $A \in \mathbb{C}^{m \times m}$. The next step is to reduce this matrix to bidiagonal form by multiplying the matrix from the left and right by two sequences of unitary matrices.

Once again, we employ Householder transformations. In Subsubsection 3.3.3.3, we intro-

duced the function

$$\left[\begin{pmatrix} \rho \\ u_2 \end{pmatrix}, \tau \right] := \text{Housev} \left(\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \right),$$

implemented by

```
function [ rho, ...
    u2, tau ] = Housev( chi1, ...
        x2 ),
```

to compute the vector $u = \begin{pmatrix} 1 \\ u_2 \end{pmatrix}$ that reflects x into $\boxed{\pm} \|x\|_2 e_0$ so that

$$\left(I - \frac{1}{\tau} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \right) \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \boxed{\pm} \underbrace{\|x\|_2}_{\rho} e_0.$$

In Subsection 10.3.1, we introduced a variation on this function:

$$[u, \tau] := \text{Housev1}(x)$$

implemented by the function

```
function [ u, tau ] = Housev1( x ).
```

They differ only in how the input and output are passed to and from the function. We also introduce the notation $H(u, \tau)$ for the transformation $I - \frac{1}{\tau}uu^H$.

We now describe an algorithm for reducing a square matrix to bidiagonal form:

- Partition $A \rightarrow \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}$.
- Update $[\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}, \tau_1] := \text{Housev}(\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix})$. This overwrites α_{11} with $\boxed{\pm} \left\| \begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix} \right\|_2$ and a_{21} with u_{21} . Implicitly, a_{21} in the updated matrix equals the zero vector.
- Update $\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} := H(\begin{pmatrix} 1 \\ u_{21} \end{pmatrix}, \tau_1) \begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix}$.

This introduces zeroes below the first entry in the first column, as illustrated by

$$\left(\begin{array}{ccccc} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{array} \right) \longrightarrow \left(\begin{array}{ccccc} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{array} \right)$$

The Householder vector that introduced the zeroes is stored over those zeroes.

Next, we introduce zeroes in the first row of this updated matrix.

- The matrix is still partitioned as $A \rightarrow \begin{pmatrix} \alpha_{11} & a_{12}^T \\ 0 & A_{22} \end{pmatrix}$, where the zeroes have been overwritten with u_{21} .
- We compute $[u_{12}, \rho_1] := \text{Housev1}((a_{12}^T)^T)$. The first element of u_{12} now holds $\boxed{\pm} \| (a_{12}^T)^T \|_2$ and the rest of the elements define the Householder transformation that introduces zeroes in $(a_{12}^T)^T$ below the first element. We store u_{12}^T in a_{12}^T .
- After setting the first entry of u_{12} explicitly to one, we update $A_{22} := A_{22}H(u_{12}, \rho_1)$.

This introduces zeroes to the right of the first entry of a_{12}^T , as illustrated by

$$\left(\begin{array}{ccccc} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{array} \right) \longrightarrow \left(\begin{array}{ccccc} \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{array} \right)$$

The Householder vector that introduced the zeroes is stored over those zeroes.

The algorithm continues this with the updated A_{22} as illustrated in [Figure 11.2.3.1](#).

$\left \begin{array}{ccccc} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{array} \right $	\longrightarrow $\left \begin{array}{ccccc} \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{array} \right $	$\times \quad \quad \times \quad 0 \quad 0 \quad 0$
Original matrix	First iteration	Second iteration
		$\begin{array}{c ccc} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ \hline 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 \end{array}$
		Third iteration
		$\begin{array}{c cc} \times & \times & 0 \\ 0 & \times & \times \\ \hline 0 & 0 & \times \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}$
		Fourth iteration

Figure 11.2.3.1 An illustration of the reduction of a square matrix to bidiagonal form. The \times s denote nonzero elements in the matrix.

Ponder This 11.2.3.2 Fill in the details for the above described algorithm that reduces a square matrix to bidiagonal form. In particular:

- For the update

$$\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} := H\left(\begin{pmatrix} 1 \\ u_{21} \end{pmatrix}, \tau_1\right) \begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix},$$

describe how all the different parts of

$$\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix}$$

are updated. (Hint: look at the QR factorization algorithm in Subsection 3.3.4.)

- For the update $A_{22} := A_{22}H(u_{12}, \rho_1)$, describe explicitly how A_{22} is updated. (Hint: look at Homework 10.3.1.1.)

Next, state the algorithm by completing the skeleton in Figure 11.2.3.2.

Finally, analyze the approximate cost of the algorithm, when started with a $m \times m$ matrix.

$[A, t, r] := \text{BiRed-unb}(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right), r \rightarrow \left(\begin{array}{c} r_T \\ \hline r_B \end{array} \right)$
A_{TL} is 0×0 , t_T, r_T have 0 elements
while $m(A_{TL}) < m(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \end{array} \right), \left(\begin{array}{c} r_T \\ \hline r_B \end{array} \right) \rightarrow \left(\begin{array}{c} r_0 \\ \hline \rho_1 \end{array} \right)$
Update via the steps $\left\{ \quad \right.$
endwhile
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \end{array} \right), \left(\begin{array}{c} r_T \\ \hline r_B \end{array} \right) \leftarrow \left(\begin{array}{c} r_0 \\ \hline \rho_1 \end{array} \right)$
Update via the steps $\left\{ \quad \right.$

Figure 11.2.3.2 Algorithm skeleton for reduction of a square matrix to bidiagonal form.

Ponder This 11.2.3.3 Once you have derived the algorithm in Ponder This 11.2.3.2, implement it.

You may want to start by executing `git pull` to update your directory `Assignments`.

In directory `Assignments/Week11/matlab/`, you will find the following files:

- `Housev.m` and `Housev1.m`: Implementations of the function `Housev` and `Housev1`.
- `BiRed.m`: A code skeleton for a function that reduces a square matrix to bidiagonal form.

`[B, t, r] = BiRed(A, t, r)`

returns the diagonal and first superdiagonal of the bidiagonal matrix in `B`, stores the Householder vectors below the subdiagonal and above the first superdiagonal, and returns the scalars τ and ρ in vectors `t` and `r`.

- `BiFromB.m`: A function that extracts the bidiagonal matrix from matrix B , which also has the Householder vector information in it.

`Bbi = BiFromB(B)`

- `test_BiRed.m`: A script that tests `BiRed`.

These resources give you the tools to implement and test the reduction to bidiagonal form.

11.2.4 Implicitly shifted bidiagonal QR algorithm



YouTube: <https://www.youtube.com/watch?v=V2PaGe52ImQ>

Converting a (tridiagonal) implicitly shifted QR algorithm into a (bidiagonal) implicitly shifted QR algorithm now hinges on some key insights, which we will illustrate with a 4×4 example.

- We start with a bidiagonal matrix $B^{(k)}$

$$B^{(k)} = \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & 0 \\ 0 & 0 & \beta_{2,2} & \beta_{2,3} \\ 0 & 0 & 0 & \beta_{3,3} \end{pmatrix},$$

which is our "current iteration."

- If we explicitly form $T^{(k)} = B^{(k)T}B^{(k)}$, then we would have to form

$$\begin{aligned} T^{(k)} &= \begin{pmatrix} \tau_{0,0} & \tau_{1,0} & 0 & 0 \\ \tau_{1,0} & \tau_{1,1} & \tau_{2,1} & 0 \\ 0 & \tau_{2,1} & \tau_{2,2} & \tau_{3,2} \\ 0 & 0 & \tau_{3,2} & \tau_{3,3} \end{pmatrix} \\ &= \begin{pmatrix} \beta_{0,0}^2 & \beta_{0,1}\beta_{0,0} & 0 & 0 \\ \beta_{0,1}\beta_{0,0} & \beta_{0,1}^2 + \beta_{1,1}^2 & \beta_{1,2}\beta_{1,1} & 0 \\ 0 & \beta_{1,2}\beta_{1,1} & \beta_{1,2}^2 + \beta_{2,2}^2 & \beta_{2,3}\beta_{2,2} \\ 0 & 0 & \beta_{2,3}\beta_{2,2} & \beta_{2,3}^2 + \beta_{3,3}^2 \end{pmatrix} \end{aligned}$$

- The Francis Implicit QR Step would then compute a first Givens' rotation so that

$$\underbrace{\begin{pmatrix} \gamma_0 & -\sigma_0 \\ \sigma_0 & \gamma_0 \end{pmatrix}}_{G_0^T}^T \begin{pmatrix} \tau_{0,0} - \tau_{3,3} \\ \tau_{1,0} \end{pmatrix} = \begin{pmatrix} \times \\ 0 \end{pmatrix}. \quad (11.2.1)$$

- With this Givens' rotation, it would introduce a bulge

$$\begin{aligned} &\left(\begin{array}{cc|cc} \times & \times & \times & 0 \\ \times & \times & \times & 0 \\ \times & \times & \times & \times \\ \hline 0 & 0 & \times & \times \end{array} \right) \\ &= \left(\begin{array}{c|cc} G_0^T & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|cc} \tau_{0,0} & \tau_{1,0} & 0 & 0 \\ \tau_{1,0} & \tau_{1,1} & \tau_{2,1} & 0 \\ \hline 0 & \tau_{2,1} & \tau_{2,2} & \tau_{3,2} \\ 0 & 0 & \tau_{3,2} & \tau_{3,3} \end{array} \right) \left(\begin{array}{c|cc} G_0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right) \end{aligned}$$

- Finally, the bulge would be chased out

$$\begin{aligned} T^{(k+1)} &= \begin{pmatrix} \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix} \\ &= \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & 1 \\ 0 & 0 \end{array} \right) \underbrace{\left[\left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & G_1^T & 0 \\ 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} \times & \times & \times & 0 \\ \times & \times & \times & 0 \\ \times & \times & \times & \times \\ \hline 0 & 0 & \times & \times \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & G_1 & 0 \\ 0 & 0 & 1 \end{array} \right) \right]}_{\left(\begin{array}{cc|cc} \times & \times & 0 & 0 \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{array} \right)} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & 1 \\ 0 & 0 \end{array} \right) G_2. \end{aligned}$$

Obviously, this extends.

Let us examine what would happen if we instead apply these Givens' rotations to $B^{(k)T}B^{(k)}$. Since $T^{(k)} = B^{(k)T}B^{(k)}$, we find that

$$\begin{aligned}
 T^{(k+1)} &= \begin{pmatrix} \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix} \\
 &= \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & 1 \\ \hline 0 & 0 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & G_1^T & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} G_0^T & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left[\begin{pmatrix} \beta_{0,0} & \beta_{0,1} & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & 0 \\ 0 & 0 & \beta_{2,2} & \beta_{2,3} \\ 0 & 0 & 0 & \beta_{3,3} \end{pmatrix}^T \right. \\
 &\quad \times \left. \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & 0 \\ 0 & 0 & \beta_{2,2} & \beta_{2,3} \\ 0 & 0 & 0 & \beta_{3,3} \end{pmatrix} \right] \left(\begin{array}{c|c|c} G_0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & G_1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & G_2 \end{array} \right) \\
 &= \left[\begin{pmatrix} \beta_{0,0} & \beta_{0,1} & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & 0 \\ 0 & 0 & \beta_{2,2} & \beta_{2,3} \\ 0 & 0 & 0 & \beta_{3,3} \end{pmatrix} \left(\begin{array}{c|c|c} G_0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & G_1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & G_2 \end{array} \right) \right]^T \\
 &\quad \times \left[\begin{pmatrix} \beta_{0,0} & \beta_{0,1} & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & 0 \\ 0 & 0 & \beta_{2,2} & \beta_{2,3} \\ 0 & 0 & 0 & \beta_{3,3} \end{pmatrix} \left(\begin{array}{c|c|c} G_0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & G_1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & G_2 \end{array} \right) \right].
 \end{aligned}$$

The observation now is that if we can find two sequences of Givens' rotations such that

$$\begin{aligned}
 B^{(k+1)} &= \begin{pmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{pmatrix} \\
 &= \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & 1 \\ \hline 0 & 0 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & \widehat{G}_1^T & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} \widehat{G}_0^T & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \\
 &\quad \times \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & 0 \\ 0 & 0 & \beta_{2,2} & \beta_{2,3} \\ 0 & 0 & 0 & \beta_{3,3} \end{pmatrix} \\
 &\quad \times \underbrace{\left(\begin{array}{c|c|c} G_0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & \tilde{G}_1 & 0 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & \tilde{G}_2 \end{array} \right)}_Q
 \end{aligned} \tag{11.2.2}$$

then, by the Implicit Q Theorem,

$$\begin{aligned}
 & B^{(k+1)T} B^{(k+1)} \\
 &= \\
 & \left(\begin{array}{cccc} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{array} \right)^T \left(\begin{array}{cccc} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{array} \right) \\
 &= \\
 & \left(\begin{array}{cccc} \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{array} \right) \\
 &= \\
 & T^{(k+1)} \\
 &= \\
 & Q^T T^{(k)} Q.
 \end{aligned}$$

If we iterate in this way, we know that $T^{(k)}$ converge to a diagonal matrix (under mild conditions). This means that the matrices $B^{(k)}$ converge to a diagonal matrix, Σ_B . If we accumulate all Givens' rotations into matrices U_B and V_B , then we end up with the SVD of B :

$$B = U_B \Sigma_B V_B^T,$$

modulo, most likely, a reordering of the diagonal elements of Σ_B and a corresponding reordering of the columns of U_B and V_B .

This leaves us with the question of how to find the two sequences of Givens' rotations mentioned in (11.2.2).

- We know G_0 , which was computed from (11.2.1). Importantly, computing this first Givens' rotation requires only that the elements $\tau_{0,0}$, $\tau_{1,0}$, and $\tau_{m-1,m-1}$ of $T^{(k)}$ to be explicitly formed.
- If we apply it to $B^{(k)}$, we introduce a bulge:

$$\left(\begin{array}{cc|cc} \times & \times & 0 & 0 \\ \textcolor{red}{\times} & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{array} \right) = \left(\begin{array}{cc|cc} \beta_{0,0} & \beta_{0,1} & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & 0 \\ 0 & 0 & \beta_{2,2} & \beta_{2,3} \\ 0 & 0 & 0 & \beta_{3,3} \end{array} \right) \left(\begin{array}{c|cc} G_0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right).$$

- We next compute a Givens' rotation, \hat{G}_0 , that changes the nonzero that was introduced below the diagonal back into a zero.

$$\left(\begin{array}{cccc} \times & \times & \textcolor{red}{\times} & 0 \\ 0 & \times & \times & 0 \\ \hline 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{array} \right) = \left(\begin{array}{c|cc} \hat{G}_0^T & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cccc} \times & \times & 0 & 0 \\ \textcolor{red}{\times} & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{array} \right).$$

- This means we now need to chase the bulge that has appeared above the superdiagonal:

$$\left(\begin{array}{c|cc|c} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & \textcolor{red}{\times} & \times & \times \\ 0 & 0 & 0 & \times \end{array} \right) = \left(\begin{array}{c|cc|c} \times & \times & \textcolor{red}{\times} & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{array} \right) \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ 0 & \widetilde{G}_1 & 0 \\ 0 & 0 & 1 \end{array} \right)$$

- We continue like this until the bulge is chased out the end of the matrix.

The net result is an **implicitly shifted bidiagonal QR algorithm** that is applied directly to the bidiagonal matrix, maintains the bidiagonal form from one iteration to the next, and converges to a diagonal matrix that has the singular values of B on its diagonal. Obviously, deflation can be added to this scheme to further reduce its cost.

11.3 Jacobi's Method

11.3.1 Jacobi rotation



YouTube: <https://www.youtube.com/watch?v=OoMPkg994ZE>

Given a symmetric 2×2 matrix A , with

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix}$$

There exists a rotation,

$$J = \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix},$$

where $\gamma = \cos(\theta)$ and $\sigma = \sin(\theta)$ for some angle θ , such that

$$J^T A J = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix} \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix} = \begin{pmatrix} \lambda_0 & 0 \\ 0 & \lambda_1 \end{pmatrix} = \Lambda.$$

We recognize that

$$A = J \Lambda J^T$$

is the Spectral Decomposition of A . The columns of J are eigenvectors of length one and the diagonal elements of Λ are the eigenvalues.

Ponder This 11.3.1.1 Give a geometric argument that Jacobi rotations exist.

Hint. For this exercise, you need to remember a few things:

- How is a linear transformation, L , translated into the matrix A that represents it, $Ax = L(x)$?
- What do we know about the orthogonality of eigenvectors of a symmetric matrix?
- If A is not already diagonal, how can the eigenvectors be chosen so that they have unit length, first one lies in Quadrant I of the plane, and the other one lies in Quadrant II?
- Draw a picture and deduce what the angle θ must be.

It is important to note that to determine J we do not need to compute θ . We merely need to find one eigenvector of the 2×2 matrix from which we can then compute an eigenvector that is orthogonal. These become the columns of J . So, the strategy is to

- Form the characteristic polynomial

$$\begin{aligned}\det(\lambda I - A) &= (\lambda - \alpha_{0,0})(\lambda - \alpha_{1,1}) - \alpha_{1,0}^2 \\ &= \lambda^2 - (\alpha_{0,0} + \alpha_{1,1})\lambda + (\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}^2),\end{aligned}$$

and solve for its roots, which give us the eigenvalues of A . Remember to us the stable formula for computing the roots of a second degree polynomial, discussed in [Subsection 9.4.1](#).

- Find an eigenvector associated with one of the eigenvalues, scaling it to have unit length and to lie in either Quadrant I or Quadrant II. This means that the eigenvector has the form

$$\begin{pmatrix} \gamma \\ \sigma \end{pmatrix}$$

if it lies in Quadrant I or

$$\begin{pmatrix} -\sigma \\ \gamma \end{pmatrix}$$

if it lies in Quadrant II.

This gives us the γ and σ that define the Jacobi rotation.

Homework 11.3.1.2 With Matlab, use the `eig` function to explore the eigenvalues and eigenvectors of various symmetric matrices:

`[Q, Lambda] = eig(A)`

Try, for example

```
A = [
-1 2
 2 3
]
```

```
A = [
  2 -1
 -1 -2
]
```

How does the matrix Q relate to a Jacobi rotation? How would Q need to be altered for it to be a Jacobi rotation?

Solution.

```
>> A = [
  -1 2
  2 3
]
```

A =

```
-1      2
 2      3
```

```
>> [ Q, Lambda ] = eig( A )
```

Q =

```
-0.9239   0.3827
 0.3827   0.9239
```

Lambda =

```
-1.8284      0
 0      3.8284
```

We notice that the columns of Q need to be swapped for it to become a Jacobi rotation:

$$J = \begin{pmatrix} 0.9239 & 0.3827 \\ -0.3827 & 0.9239 \end{pmatrix} = \begin{pmatrix} 0.3827 & -0.9239 \\ 0.9239 & 0.3827 \end{pmatrix} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = Q \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

```
>> A = [
  2 -1
 -1 -2
]
```

A =

```

2      -1
-1      -2

>> [ Q, Lambda ] = eig( A )

Q =

```

$$\begin{pmatrix} -0.2298 & -0.9732 \\ -0.9732 & 0.2298 \end{pmatrix}$$

```

Lambda =

```

$$\begin{pmatrix} -2.2361 & 0 \\ 0 & 2.2361 \end{pmatrix}$$

We notice that the columns of Q need to be swapped for it to become a Jacobi rotation. If we follow our "recipe", we also need to negate each column:

$$J = \begin{pmatrix} 0.9732 & -0.2298 \\ 0.2298 & 0.9732 \end{pmatrix}.$$

These solutions are not unique. Another way of creating a Jacobi rotation is to, for example, scale the first column so that the diagonal elements have the same sign. Indeed, perhaps that is the easier thing to do:

$$Q = \begin{pmatrix} -0.9239 & 0.3827 \\ 0.3827 & 0.9239 \end{pmatrix} \quad \rightarrow \quad J = \begin{pmatrix} 0.9239 & 0.3827 \\ -0.3827 & 0.9239 \end{pmatrix}.$$

11.3.2 Jacobi's method for computing the Spectral Decomposition



YouTube: <https://www.youtube.com/watch?v=mBn7d9jUjcs>

The oldest algorithm for computing the eigenvalues and eigenvectors of a (symmetric) matrix is due to Jacobi and dates back to 1846.

- [22] C. G. J. Jacobi, Über ein leichtes Verfahren, die in der Theorie der Säkularstörungen vorkommenden Gleichungen numerisch aufzulösen, Crelle's Journal 30, 51-94

(1846).

If we recall correctly (it has been 30 years since we read the paper in German), the paper thanks Jacobi's student Seidel for performing the calculations for a 5×5 matrix, related to the orbits of the planets, by hand...

This is a method that keeps resurfacing, since it parallelizes easily. The operation count tends to be higher (by a constant factor) than that of reduction to tridiagonal form followed by the tridiagonal QR algorithm.

Jacobi's original idea went as follows:

- We start with a symmetric matrix:

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \alpha_{0,3} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \alpha_{3,0} & \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{pmatrix}.$$

- We find the off-diagonal entry with largest magnitude. Let's say it is $\alpha_{3,1}$.
- We compute a Jacobi rotation so that

$$\begin{pmatrix} \gamma_{3,1} & \sigma_{3,1} \\ -\sigma_{3,1} & \gamma_{3,1} \end{pmatrix} \begin{pmatrix} \alpha_{1,1} & \alpha_{1,3} \\ \alpha_{3,1} & \alpha_{3,3} \end{pmatrix} \begin{pmatrix} \gamma_{3,1} & -\sigma_{3,1} \\ \sigma_{3,1} & \gamma_{3,1} \end{pmatrix} = \begin{pmatrix} \times & 0 \\ 0 & \times \end{pmatrix},$$

where the \times s denote nonzero entries.

- We now apply the rotation as a unitary similarity transformation from the left to the rows of A indexed with 1 and 3, and from the right to columns 1 and 3:

$$\begin{pmatrix} \alpha_{0,0} & \times & \alpha_{0,2} & \times \\ \times & \times & \times & 0 \\ \alpha_{2,0} & \times & \alpha_{2,2} & \times \\ \times & 0 & \times & \times \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \gamma_{3,1} & 0 & \sigma_{3,1} \\ 0 & 0 & 1 & 0 \\ 0 & -\sigma_{3,1} & 0 & \gamma_{3,1} \end{pmatrix} \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \alpha_{0,3} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \alpha_{3,0} & \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \gamma_{3,1} & 0 & -\sigma_{3,1} \\ 0 & 0 & 1 & 0 \\ 0 & \sigma_{3,1} & 0 & \gamma_{3,1} \end{pmatrix}.$$

The \times s here denote elements of the matrix that are changed by the application of the Jacobi rotation.

- This process repeats, reducing the off-diagonal element that is largest in magnitude to zero in each iteration.

Notice that each application of the Jacobi rotation is a unitary similarity transformation, and hence preserves the eigenvalues of the matrix. If this method eventually yields a diagonal

matrix, then the eigenvalues can be found on the diagonal of that matrix. We do not give a proof of convergence here.



YouTube: <https://www.youtube.com/watch?v=LC5VKYR7sEM>

Homework 11.3.2.1 Note: In the description of this homework, we index like Matlab does: starting at one. This is in contrast to how we usually index in these notes, starting at zero.

In `Assignments/Week11/matlab`, you will find the following files:

- `Jacobi_rotation.m`: A function that computes a Jacobi rotation from a 2×2 symmetric matrix.
- `Seidel.m`: A script that lets you apply Jacobi's method to a 5×5 matrix, much like Seidel did by hand. Fortunately, you only indicate the off-diagonal element to zero out. Matlab then does the rest.

Use this to gain insight into how Jacobi's method works. You will notice that finding the off-diagonal element that has largest magnitude is bothersome. You don't need to get it right every time.

Once you have found the diagonal matrix, restart the process. This time, zero out the off-diagonal elements in systematic "sweeps," zeroing the elements in the order.

$$\begin{aligned} (2,1) - (3,1) - (4,1) - (5,1) - \\ (3,2) - (4,2) - (5,2) - \\ (4,3) - (5,3) - \\ (5,4) \end{aligned}$$

and repeating this until convergence. A sweep zeroes every off-diagonal element exactly once (in symmetric pairs).

Solution. Hopefully you noticed that the matrix converges to a diagonal matrix, with the eigenvalues on its diagonal.

The key insight is that applying a Jacobi rotation to zero an element, $\alpha_{i,j}$, reduces the square of the Frobenius norm of the off-diagonal elements of the matrix by $\alpha_{i,j}^2$. In other words, let $\text{off}(A)$ equal the matrix A but with its diagonal elements set to zero. If $J_{i,j}$ zeroes out $\alpha_{i,j}$ (and $\alpha_{j,i}$), then

$$\|\text{off}(J_{i,j}^T A J_{i,j})\|_F^2 = \|\text{off}(A)\|_F^2 - 2\alpha_{i,j}^2.$$

Homework 11.3.2.2 Partition matrix A like

$$\left(\begin{array}{c|c|c|c|c} A_{00} & \textcolor{red}{a_{10}} & A_{20}^T & \textcolor{red}{a_{30}} & A_{40}^T \\ \textcolor{red}{a_{10}^T} & \alpha_{11} & \textcolor{red}{a_{21}^T} & \alpha_{31} & \textcolor{red}{a_{41}^T} \\ \hline A_{20} & \textcolor{red}{a_{21}} & A_{22} & \textcolor{red}{a_{32}} & A_{42}^T \\ \textcolor{red}{a_{30}^T} & \alpha_{31} & \textcolor{red}{a_{32}^T} & \alpha_{33} & \textcolor{red}{a_{43}^T} \\ \hline A_{40} & \textcolor{red}{a_{41}} & A_{42} & \textcolor{red}{a_{43}} & A_{44} \end{array} \right)$$

and let J equal the Jacobi rotation that zeroes the element denoted with α_{31} :

$$\begin{aligned} J^T A J = & \left(\begin{array}{c|c|c|c|c} I & 0 & 0 & 0 & 0 \\ 0 & \gamma_{11} & 0 & \sigma_{31} & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & -\sigma_{31} & 0 & \gamma_{33} & 0 \\ 0 & 0 & 0 & 0 & I \end{array} \right) \left(\begin{array}{c|c|c|c|c} A_{00} & \textcolor{red}{a_{10}} & A_{20}^T & \textcolor{red}{a_{30}} & A_{40}^T \\ \textcolor{red}{a_{10}^T} & \alpha_{11} & \textcolor{red}{a_{21}^T} & \alpha_{31} & \textcolor{red}{a_{41}^T} \\ \hline A_{20} & \textcolor{red}{a_{21}} & A_{22} & \textcolor{red}{a_{32}} & A_{42}^T \\ \textcolor{red}{a_{30}^T} & \alpha_{31} & \textcolor{red}{a_{32}^T} & \alpha_{33} & \textcolor{red}{a_{43}^T} \\ \hline A_{40} & \textcolor{red}{a_{41}} & A_{42} & \textcolor{red}{a_{43}} & A_{44} \end{array} \right) \left(\begin{array}{c|c|c|c|c} I & 0 & 0 & 0 & 0 \\ 0 & \gamma_{11} & 0 & -\sigma_{13} & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & \sigma_{31} & 0 & \gamma_{33} & 0 \\ 0 & 0 & 0 & 0 & I \end{array} \right) \\ = & \left(\begin{array}{c|c|c|c|c} A_{00} & \hat{a}_{10} & A_{20}^T & \hat{a}_{30} & A_{40}^T \\ \hat{a}_{10}^T & \hat{\alpha}_{11} & \hat{a}_{21}^T & 0 & \hat{a}_{41}^T \\ \hline A_{20} & \hat{a}_{21} & A_{22} & \hat{a}_{32} & A_{42}^T \\ \hat{a}_{30}^T & 0 & \hat{a}_{32}^T & \hat{\alpha}_{33} & \hat{a}_{43}^T \\ \hline A_{40} & \hat{a}_{41} & A_{42} & \hat{a}_{43} & A_{44} \end{array} \right) = \hat{A}. \end{aligned}$$

Show that $\|\text{off}(\hat{A})\|_F^2 = \|\text{off}(A)\|_F^2 - 2\alpha_{31}^2$.

Solution. We notice that

$$\begin{aligned} \|\text{off}(A)\|_F^2 = & \|\text{off}(A_{00})\|_F^2 + \|\textcolor{red}{a_{10}}\|_F^2 + \|A_{20}^T\|_F^2 + \|\textcolor{red}{a_{30}}\|_F^2 + \|A_{40}^T\|_F^2 \\ & + \|\textcolor{red}{a_{10}^T}\|_F^2 + \|\textcolor{red}{a_{21}^T}\|_F^2 + \alpha_{31}^2 + \|\textcolor{red}{a_{41}^T}\|_F^2 \\ & + \|A_{20}\|_F^2 + \|\textcolor{red}{a_{21}}\|_F^2 + \|\text{off}(A_{22})\|_F^2 + \|\textcolor{red}{a_{32}}\|_F^2 + \|A_{42}^T\|_F^2 \\ & + \|\textcolor{red}{a_{30}^T}\|_F^2 + \alpha_{31}^2 + \|\textcolor{red}{a_{32}^T}\|_F^2 + \|\textcolor{red}{a_{43}^T}\|_F^2 \\ & + \|A_{40}\|_F^2 + \|\textcolor{red}{a_{41}}\|_F^2 + \|A_{42}\|_F^2 + \|\textcolor{red}{a_{43}}\|_F^2 + \|\text{off}(A_{44})\|_F^2. \end{aligned}$$

and

$$\begin{aligned} \|\text{off}(\hat{A})\|_F^2 = & \|\text{off}(A_{00})\|_F^2 + \|\hat{a}_{10}\|_F^2 + \|A_{20}^T\|_F^2 + \|\hat{a}_{30}\|_F^2 + \|A_{40}^T\|_F^2 \\ & + \|\hat{a}_{10}^T\|_F^2 + \|\hat{a}_{21}^T\|_F^2 + 0 + \|\hat{a}_{41}^T\|_F^2 \\ & + \|A_{20}\|_F^2 + \|\hat{a}_{21}\|_F^2 + \|\text{off}(A_{22})\|_F^2 + \|\hat{a}_{32}\|_F^2 + \|A_{42}^T\|_F^2 \\ & + \|\hat{a}_{30}^T\|_F^2 + 0 + \|\hat{a}_{32}^T\|_F^2 + \|\hat{a}_{43}^T\|_F^2 \\ & + \|A_{40}\|_F^2 + \|\hat{a}_{41}\|_F^2 + \|A_{42}\|_F^2 + \|\hat{a}_{43}\|_F^2 + \|\text{off}(A_{44})\|_F^2. \end{aligned}$$

All submatrices in black show up for $\|\text{off}(A)\|_F^2$ and $\|\text{off}(\hat{A})\|_F^2$. The parts of rows and columns that show up in red and blue is what changes. We argue that the sum of the terms in red are equal for both.

Since a Jacobi rotation is unitary, it preserves the Frobenius norm of the matrix to which it applied. Thus, looking at the rows that are modified by applying J^T from the left, we find

that

$$\begin{aligned}
 & \left\| \begin{array}{c} \|a_{10}^T\|_F^2 \\ + \|a_{30}^T\|_F^2 \end{array} \right. + \left\| \begin{array}{c} \|a_{21}^T\|_F^2 \\ + \|a_{32}^T\|_F^2 \end{array} \right. + \left\| \begin{array}{c} \|a_{41}^T\|_F^2 \\ + \|a_{43}^T\|_F^2 \end{array} \right. = \\
 & \left\| \left(\begin{array}{cc} \gamma_{11} & \sigma_{31} \\ -\sigma_{31} & \gamma_{33} \end{array} \right) \left(\begin{array}{c} a_{10}^T \\ a_{30}^T \end{array} \right) \left(\begin{array}{c} a_{21}^T \\ a_{32}^T \end{array} \right) \left(\begin{array}{c} a_{41}^T \\ a_{43}^T \end{array} \right) \right\|_F^2 = \\
 & \left\| \left(\begin{array}{c} \hat{a}_{10}^T \\ \hat{a}_{30}^T \end{array} \right) \left(\begin{array}{c} \hat{a}_{21}^T \\ \hat{a}_{32}^T \end{array} \right) \left(\begin{array}{c} \hat{a}_{41}^T \\ \hat{a}_{43}^T \end{array} \right) \right\|_F^2 = \\
 & \left\| \begin{array}{c} \|\hat{a}_{10}^T\|_F^2 \\ + \|\hat{a}_{30}^T\|_F^2 \end{array} \right. + \left\| \begin{array}{c} \|\hat{a}_{21}^T\|_F^2 \\ + \|\hat{a}_{32}^T\|_F^2 \end{array} \right. + \left\| \begin{array}{c} \|\hat{a}_{41}^T\|_F^2 \\ + \|\hat{a}_{43}^T\|_F^2 \end{array} \right.
 \end{aligned}$$

Similarly, looking at the columns that are modified by applying J^T from the right, we find that

$$\begin{aligned}
 & \left\| \begin{array}{c} \|a_{10}\|_F^2 \\ + \|a_{30}\|_F^2 \end{array} \right. + \left\| \begin{array}{c} \|a_{21}\|_F^2 \\ + \|a_{32}\|_F^2 \end{array} \right. + \left\| \begin{array}{c} \|a_{41}\|_F^2 \\ + \|a_{43}\|_F^2 \end{array} \right. = \left\| \begin{array}{c} a_{10} \\ a_{30} \\ \hline a_{21} \\ a_{32} \\ \hline a_{41} \\ a_{43} \end{array} \right\|_F^2 \\
 & = \left\| \begin{array}{c} \left(\begin{array}{cc} a_{10} & a_{30} \\ \hline a_{21} & a_{32} \\ \hline a_{41} & a_{43} \end{array} \right) \left(\begin{array}{cc} \gamma_{11} & -\sigma_{31} \\ \sigma_{31} & \gamma_{33} \end{array} \right) \end{array} \right\|_F^2 = \left\| \begin{array}{c} \hat{a}_{10} \\ \hat{a}_{30} \\ \hline \hat{a}_{21} \\ \hat{a}_{32} \\ \hline \hat{a}_{41} \\ \hat{a}_{43} \end{array} \right\|_F^2 \\
 & = + \|\hat{a}_{21}\|_F^2 + \|\hat{a}_{32}\|_F^2 + \|\hat{a}_{41}\|_F^2 + \|\hat{a}_{43}\|_F^2.
 \end{aligned}$$

We conclude that

$$\text{off}(\hat{A}) = \text{off}(A) - 2\alpha_{31}^2.$$

From this exercise, we learn:

- The good news: every time a Jacobi rotation is used to zero an off-diagonal element, $\text{off}(A)$ decreases by twice the square of that element.
- The bad news: a previously introduced zero may become nonzero in the process.

The original algorithm developed by Jacobi searched for the largest (in absolute value) off-diagonal element and zeroed it, repeating this process until all off-diagonal elements were small. The problem with this is that searching for the largest off-diagonal element in an $m \times m$ matrix requires $O(m^2)$ comparisons. Computing and applying one Jacobi rotation as a similarity transformation requires $O(m)$ flops. For large m this is not practical. Instead, it can be shown that zeroing the off-diagonal elements by columns (or rows) also converges to

a diagonal matrix. This is known as the column-cyclic Jacobi algorithm. Zeroing out every pair of off-diagonal elements once is called a sweep. We illustrate this in [Figure 11.3.2.1](#). Typically only a few sweeps (on the order of five) are needed to converge sufficiently.

Sweep 1			
$\begin{matrix} \times & 0 & \times & \times \\ 0 & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{matrix}$	\rightarrow	$\begin{matrix} \times & \times & 0 & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ \times & \times & \times & \times \end{matrix}$	\rightarrow
zero (1, 0)		zero (2, 0)	
$\begin{matrix} \times & \times & \times & \times \\ \times & \times & 0 & \times \\ \times & 0 & \times & \times \\ \times & \times & \times & \times \end{matrix}$	\rightarrow	$\begin{matrix} \times & \times & \times & \times \\ \times & \times & \times & 0 \\ \times & \times & \times & \times \\ \times & 0 & \times & \times \end{matrix}$	\rightarrow
zero (2, 1)		zero (3, 1)	
Sweep 2			
$\begin{matrix} \times & 0 & \times & \times \\ 0 & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{matrix}$	\rightarrow	$\begin{matrix} \times & \times & 0 & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ \times & \times & \times & \times \end{matrix}$	\rightarrow
zero (1, 0)		zero (2, 0)	
$\begin{matrix} \times & \times & \times & \times \\ \times & \times & 0 & \times \\ \times & 0 & \times & \times \\ \times & \times & \times & \times \end{matrix}$	\rightarrow	$\begin{matrix} \times & \times & \times & \times \\ \times & \times & \times & 0 \\ \times & \times & \times & \times \\ \times & 0 & \times & \times \end{matrix}$	\rightarrow
zero (2, 1)		zero (3, 1)	

Figure 11.3.2.1 Column-cyclic Jacobi algorithm.

We conclude by noting that the matrix Q such that $A = Q\Lambda Q^H$ can be computed by accumulating all the Jacobi rotations (applying them to the identity matrix).

11.3.3 Jacobi's method for computing the Singular Value Decomposition



YouTube: <https://www.youtube.com/watch?v=VUktLhUiR7w>

Just like the QR algorithm for computing the Spectral Decomposition was modified to compute the SVD, so can the Jacobi Method for computing the Spectral Decomposition.

The insight is very simple. Let $A \in \mathbb{R}^{m \times n}$ and partition it by columns:

$$A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right).$$

One could form $B = A^T A$ and then compute Jacobi rotations to diagonalize it:

$$\underbrace{\cdots J_{3,1}^T J_{2,1}^T}_{Q^T} B \underbrace{J_{2,1} J_{3,1} \cdots}_{Q} = D.$$

We recall that if we order the columns of Q and diagonal elements of D appropriately, then choosing $V = Q$ and $\Sigma = D^{1/2}$ yields

$$A = U \Sigma V^T = U D^{1/2} Q^T$$

or, equivalently,

$$AQ = U \Sigma = U D^{1/2}.$$

This means that if we apply the Jacobi rotations $J_{2,1}, J_{3,1}, \dots$ from the right to A ,

$$UD^{1/2} = ((AJ_{2,1})J_{3,1})\cdots,$$

then, once B has become (approximately) diagonal, the columns of $\hat{A} = ((AJ_{2,1})J_{3,1})\cdots$ are mutually orthogonal. By scaling them to have length one, setting $\Sigma = \text{diag}(\|\hat{a}_0\|_2, \|\hat{a}_1\|_2, \dots, \|\hat{a}_{n-1}\|_2)$, we find that

$$U = \hat{A} \Sigma^{-1} = AQ(D^{1/2})^{-1}.$$

The only problem is that in forming B , we may introduce unnecessary error since it squares the condition number.

Here is a more practical algorithm. We notice that

$$B = A^T A = \begin{pmatrix} a_0^T a_0 & a_0^T a_1 & \cdots & a_0^T a_{n-1} \\ a_1^T a_0 & a_1^T a_1 & \cdots & a_1^T a_{n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1}^T a_0 & a_{n-1}^T a_1 & \cdots & a_{n-1}^T a_{n-1} \end{pmatrix}.$$

We observe that we don't need to form all of B . When it is time to compute $J_{i,j}$, we need only compute

$$\begin{pmatrix} \beta_{i,i} & \beta_{j,i} \\ \beta_{j,i} & \beta_{j,j} \end{pmatrix} = \begin{pmatrix} a_i^T a_i & a_j^T a_i \\ a_j^T a_i & a_j^T a_j \end{pmatrix},$$

from which $J_{i,j}$ can be computed. By instead applying this Jacobi rotation to B , we observe that

$$J_{i,j}^T B J_{i,j} = J_{i,j}^T A^T A J_{i,j} = (AJ_{i,j})^T (AJ_{i,j})$$

and hence the Jacobi rotation can instead be used to take linear combinations of the i th and j th columns of A :

$$\left(\begin{array}{c|c} a_i & a_j \end{array} \right) := \left(\begin{array}{c|c} a_i & a_j \end{array} \right) \left(\begin{array}{cc} \gamma_{i,j} & -\sigma_{i,j} \\ \sigma_{i,j} & \gamma_{i,j} \end{array} \right).$$

We have thus outlined an algorithm:

- Starting with matrix A , compute a sequence of Jacobi rotations (e.g., corresponding to a column-cyclic Jacobi method) until the off-diagonal elements of $A^T A$ (parts of which are formed as Jacobi rotations are computed) become small. Every time a Jacobi rotation is computed, it updates the appropriate columns of A .
- Accumulate the Jacobi rotations into matrix V , by applying them from the right to an identity matrix:

$$V = ((I \times J_{2,1})J_{3,1}) \cdots$$

- Upon completion,

$$\Sigma = \text{diag}(\|a_0\|_2, \|a_1\|_2, \dots, \|a_{n-1}\|_2)$$

and

$$U = A\Sigma^{-1},$$

meaning that each column of the updated A is divided by its length.

- If necessary, reorder the columns of U and V and the diagonal elements of Σ .

Obviously, there are variations on this theme. Such methods are known as **one-sided Jacobi methods**.

11.4 Enrichments

11.4.1 Principal Component Analysis

The Spectral Decomposition and Singular Value Decomposition are fundamental to a technique in data sciences known as **Principal Component Analysis** (PCA). The following tutorial makes that connection.

- [36] Jonathon Shlens, [A Tutorial on Principal Component Analysis](#), arxiv 1404.1100, 2014.

11.4.2 Casting the reduction to bidiagonal form in terms of matrix-matrix multiplication

As was discussed in [Subsection 10.4.3](#) for the reduction to tridiagonal form, reduction to bidiagonal form can only be partly cast in terms of matrix-matrix multiplication. As for the reduction to tridiagonal form, we recommend

- [44] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, G. Joseph Elizondo, Families of Algorithms for Reducing a Matrix to Condensed Form, ACM Transactions on Mathematical Software (TOMS) , Vol. 39, No. 1, 2012.

Bidiagonal, tridiagonal, and upper Hessenberg form are together referred to as **condensed form**.

11.4.3 Optimizing the bidiagonal QR algorithm

As the Givens' rotations are applied to the bidiagonal matrix, they are also applied to matrices in which the left and right singular vectors are accumulated (matrices U and V). If we start with an $m \times m$ matrix, one step of introducing the bulge and chasing it out the matrix requires $O(m)$ computation. Accumulating the Givens' rotations into U and V requires $O(m^2)$ computation for each such step, with $O(m^2)$ data. As was discussed in Subsection 10.4.4 for the implicitly shifted QR algorithm, this inherently means the cost of accessing data dominates on current architectures.

The paper also mentioned in Subsection 10.4.4, also describes techniques for applying the Givens' rotations for several steps of the Implicitly shifted bidiagonal QR algorithm at the same time, which allows one to attain high performance similar to that attained by a matrix-matrix multiplication.

- [42] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, Restructuring the Tridiagonal and Bidiagonal QR Algorithms for Performance, ACM Transactions on Mathematical Software (TOMS), Vol. 40, No. 3, 2014.

To our knowledge, this yields the fastest implementation for finding the SVD of a bidiagonal matrix.

11.5 Wrap Up

11.5.1 Additional homework

No additional homework yet.

11.5.2 Summary

We have noticed that typos are uncovered relatively quickly once we release the material. Because we "cut and paste" the summary from the materials in this week, we are delaying adding the summary until most of these typos have been identified.

Week 12

Attaining High Performance

12.1 Opening

12.1.1 Simple Implementation of matrix-matrix multiplication

The current coronavirus crisis hit UT-Austin on March 14, a day we spent quickly making videos for Week 11. We have not been back to the office, to create videos for Week 12, since then. We will likely add such videos as time goes on. For now, we hope that the notes suffice.

Remark 12.1.1.1 The exercises in this unit assume that you have installed the BLAS-like Library Instantiation Software (BLIS), as described in [Subsection 0.2.4](#).

Let A , B , and C be $m \times k$, $k \times n$, and $m \times n$ matrices, respectively. We can expose their individual entries as

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,k-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,k-1} \end{pmatrix}, B = \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \\ \beta_{1,0} & \beta_{1,1} & \cdots & \beta_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \beta_{k-1,0} & \beta_{k-1,1} & \cdots & \beta_{k-1,n-1} \end{pmatrix},$$

and

$$C = \begin{pmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \\ \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,n-1} \end{pmatrix}.$$

The computation $C := AB + C$, which adds the result of the matrix-matrix multiplication AB to a matrix C , is defined element-wise as

$$\gamma_{i,j} := \sum_{p=0}^{k-1} \alpha_{i,p} \beta_{p,j} + \gamma_{i,j} \quad (12.1.1)$$

for all $0 \leq i < m$ and $0 \leq j < n$. We add to C because this will make it easier to play with the orderings of the loops when implementing matrix-matrix multiplication. The following

pseudo-code computes $C := AB + C$:

```

for  $i := 0, \dots, m - 1$ 
    for  $j := 0, \dots, n - 1$ 
        for  $p := 0, \dots, k - 1$ 
             $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$ 
        end
    end
end

```

The outer two loops visit each element of C and the inner loop updates $\gamma_{i,j}$ with (12.1.1). We use C programming language macro definitions in order to explicitly index into the matrices, which are passed as one-dimensional arrays in which the matrices are stored in column-major order.

Remark 12.1.1.2 For a more complete discussion of how matrices are mapped to memory, you may want to look at [1.2.1 Mapping matrices to memory](#) in our MOOC titled [LAFF-On Programming for High Performance](#). If the discussion here is a bit too fast, you may want to consult the entire [Section 1.2 Loop orderings](#) of that course.

```

#define alpha( i,j ) A[ (j)*ldA + i ] // map alpha( i,j ) to array A
#define beta( i,j ) B[ (j)*ldB + i ] // map beta( i,j ) to array B
#define gamma( i,j ) C[ (j)*ldC + i ] // map gamma( i,j ) to array C

void MyGemm( int m, int n, int k, double *A, int ldA,
             double *B, int ldB, double *C, int ldC )
{
    for ( int i=0; i<m; i++ )
        for ( int j=0; j<n; j++ )
            for ( int p=0; p<k; p++ )
                gamma( i,j ) += alpha( i,p ) * beta( p,j );
}

```

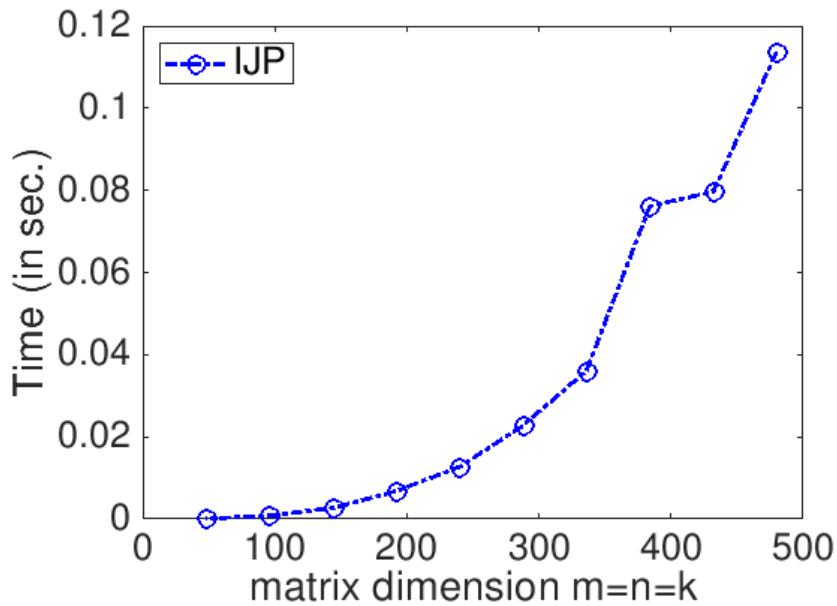
Figure 12.1.1.3 Implementation, in the C programming language, of the IJP ordering for computing matrix-matrix multiplication.

Homework 12.1.1.1 In the file [Assignments/Week12/C/Gemm_IJP.c](#) you will find the simple implementation given in [Figure 12.1.1.3](#) that computes $C := AB + C$. In a terminal window, the directory [Assignments/Week12/C](#), execute

`make IJP`

to compile, link, and execute it. You can view the performance attained on your computer with the Matlab Live Script in [Assignments/Week12/C/data/Plot_IJP.mlx](#) (Alternatively, read and execute [Assignments/Week12/C/data/Plot_IJP_m.m](#).)

On Robert's laptop, [Homework 12.1.1.1](#) yields the graph

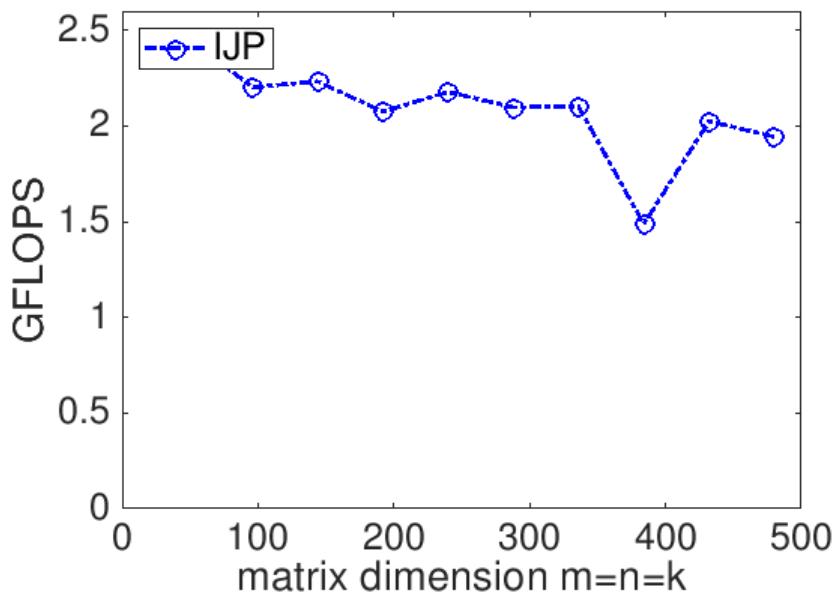


as the curve labeled with IJP. The time, in seconds, required to compute matrix-matrix multiplication as a function of the matrix size is plotted, where $m = n = k$ (each matrix is square). "Irregularities" in the time required to complete can be attributed to a number of factors, including that other processes that are executing on the same processor may be disrupting the computation. One should not be too concerned about those.

The performance of a matrix-matrix multiplication implementation is measured in billions of floating point operations per second (GFLOPS). We know that it takes $2mnk$ flops to compute $C := AB + C$ when C is $m \times n$, A is $m \times k$, and B is $k \times n$. If we measure the time it takes to complete the computation, $T(m, n, k)$, then the rate at which we compute is given by

$$\frac{2mnk}{T(m, n, k)} \times 10^{-9} \text{ GFLOPS.}$$

For our implementation, this yields



Again, don't worry too much about the dips in the curves in this and future graphs. If we controlled the environment in which we performed the experiments (for example, by making sure no other compute-intensive programs are running at the time of the experiments), these would largely disappear.

Remark 12.1.1.4 The `Gemm` in the name of the routine stands for General Matrix-Matrix multiplication. `Gemm` is an acronym that is widely used in scientific computing, with roots in the Basic Linear Algebra Subprograms (BLAS) interface which we will discuss in [Subsection 12.2.5](#).

Homework 12.1.1.2 The IJP ordering is one possible ordering of the loops. How many distinct reorderings of those loops are there?

Answer.

$$3! = 6.$$

Solution.

- There are three choices for the outer-most loop: i , j , or p .
- Once a choice is made for the outer-most loop, there are two choices left for the second loop.
- Once that choice is made, there is only one choice left for the inner-most loop.

Thus, there are $3! = 3 \times 2 \times 1 = 6$ loop orderings.

Homework 12.1.1.3 In directory `Assignments/Week12/C` make copies of [Assignments/Week12/C/Gemm_IJP.c](#) into files with names that reflect the different loop orderings (`Gemm_IPJ.c`, etc.). Next, make the necessary changes to the loops in each file to reflect the ordering encoded in its name. Test the implementations by executing

```
make IPJ
make JIP
...

```

for each of the implementations and view the resulting performance by making the indicated changes to the Live Script in [Assignments/Week12/C/data/Plot_All_Orderings.mlx](#) (Alternatively, use the script in [Assignments/Week12/C/data/Plot_All_Orderings_m.m](#)). If you have implemented them all, you can test them all by executing

```
make All_Orderings
```

Solution.

- [Assignments/Week12/answers/Gemm_IPJ.c](#)
- [Assignments/Week12/answers/Gemm_JIP.c](#)
- [Assignments/Week12/answers/Gemm_JPI.c](#)
- [Assignments/Week12/answers/Gemm_PIJ.c](#)
- [Assignments/Week12/answers/Gemm_PJI.c](#)

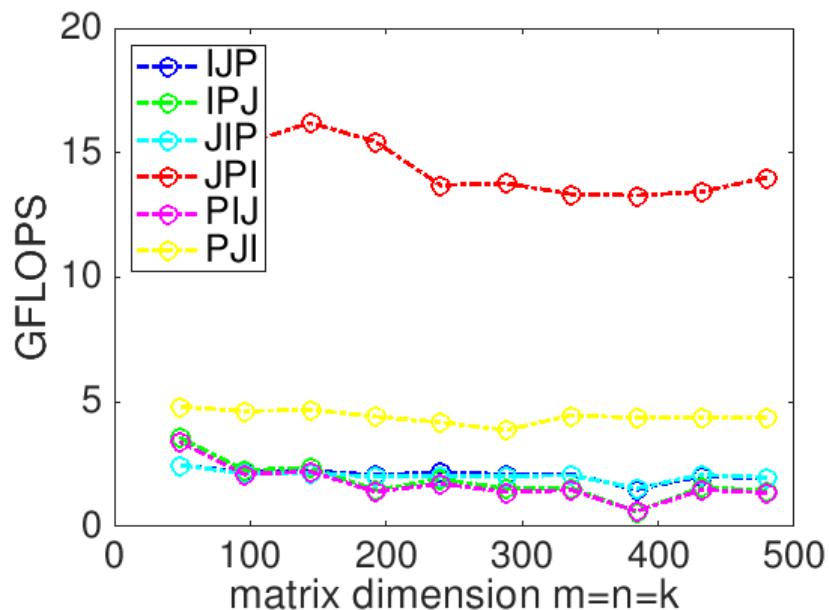


Figure 12.1.1.5 Performance comparison of all different orderings of the loops, on Robert's laptop.

Homework 12.1.1.4 In directory `Assignments/Week12/C/` execute
`make JPI`

and view the results with the Live Script in [Assignments/Week12/C/data/Plot_Openner.mlx](#).

(This may take a little while, since the Makefile now specifies that the largest problem to be executed is $m = n = k = 1500$.)

Next, change that Live Script to also show the performance of the reference implementation provided by the BLAS-like Library Instantiation Software (BLIS): Change

```
% Optionally show the reference implementation performance data
if ( 0 )
```

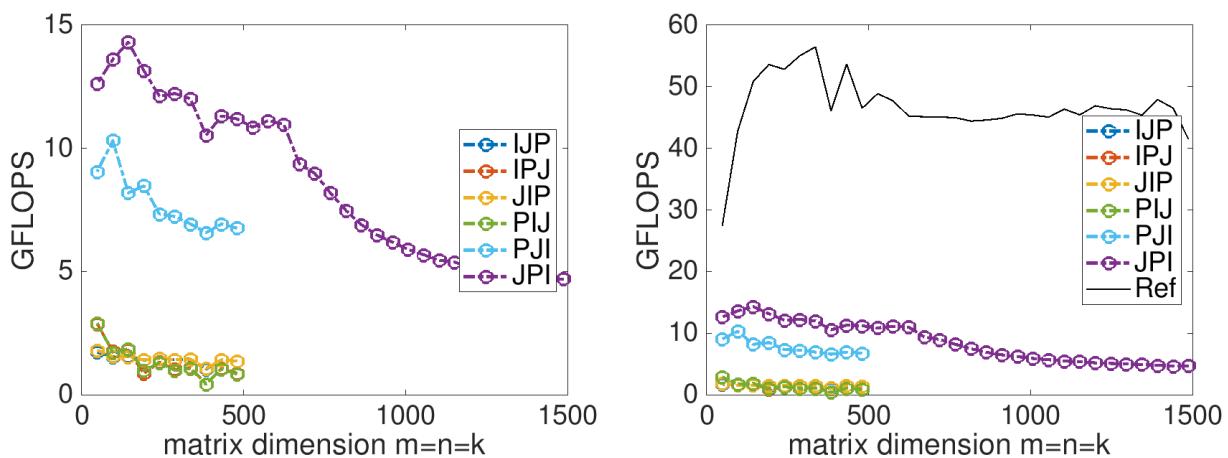
to

```
% Optionally show the reference implementation performance data
if ( 1 )
```

and rerun the Live Script. This adds a plot to the graph for the reference implementation.

What do you observe? Now are you happy with the improvements you made by reordering the loops?

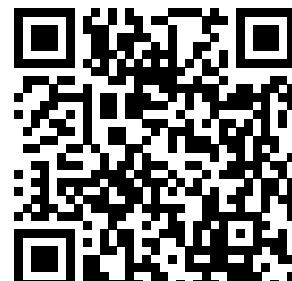
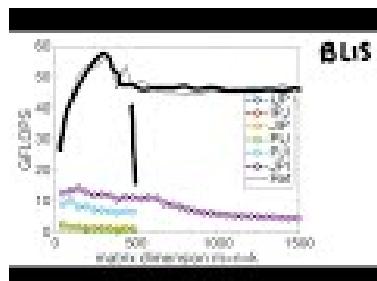
Solution. On Robert's laptop:



Left: Plotting only simple implementations. Right: Adding the performance of the reference implementation provided by BLIS.

Note: the performance in the graph on the left may not exactly match that in the graph earlier in this unit. My laptop does not always attain the same performance. When a processor gets hot, it "clocks down." This means the attainable performance goes down. A laptop is not easy to cool, so one would expect more fluctuation than when using, for example, a desktop or a server.

Here is a video from our course "[LAFF-On Programming for High Performance](#)", which explains what you observe. (It refers to "Week 1" or that course. It is part of the launch for that course.)



YouTube: <https://www.youtube.com/watch?v=eZaq451nuaE>

Remark 12.1.1.6 There are a number of things to take away from the exercises in this unit.

- The ordering of the loops matters. Why? Because it changes the pattern of memory access. Accessing memory contiguously (what is often called "with stride one") improves performance.
- Compilers, which translate high-level code written in a language like C, are not the answer. You would think that they would reorder the loops to improve performance. The widely-used `gcc` compiler doesn't do so very effectively. (Intel's highly-acclaimed `icc` compiler does a much better job, but still does not produce performance that rivals the reference implementation.)
- Careful implementation can greatly improve performance.
- One solution is to learn how to optimize yourself. Some of the fundamentals can be discovered in our MOOC [LAFF-On Programming for High Performance](#). The other solution is to use highly optimized libraries, some of which we will discuss later in this week.

12.1.2 Overview

- 12.1 Opening
 - 12.1.1 Simple Implementation of matrix-matrix multiplication
 - 12.1.2 Overview
 - 12.1.3 What you will learn
- 12.2 Linear Algebra Building Blocks
 - 12.2.1 A simple model of the computer
 - 12.2.2 Opportunities for optimization
 - 12.2.3 Basics of optimizing matrix-matrix multiplication
 - 12.2.4 Optimizing matrix-matrix multiplication, the real story
 - 12.2.5 BLAS and BLIS
- 12.3 Casting Computation in Terms of Matrix-Matrix Multiplication

- 12.3.1 Blocked Cholesky factorization
- 12.3.2 Blocked LU factorization
- 12.3.3 Other high-performance dense linear algebra algorithms
- 12.3.4 Libraries for higher level dense linear algebra functionality
- 12.3.5 Sparse algorithms
- 12.4 Enrichments
 - 12.4.1 Optimizing matrix-matrix multiplication - We've got a MOOC for that!
 - 12.4.2 Deriving blocked algorithms - We've got a MOOC for that too!
 - 12.4.3 Parallel high-performance algorithms
- 12.5 Wrap Up
 - 12.5.1 Additional homework
 - 12.5.2 Summary

12.1.3 What you will learn

This week, you explore how algorithms and architectures interact.

Upon completion of this week, you should be able to

- Amortize the movement of data between memory layers over useful computation to overcome the discrepancy between the speed of floating point computation and the speed with which data can be moved in and out of main memory.
- Block matrix-matrix multiplication for multiple levels of cache.
- Cast matrix factorizations as blocked algorithms that cast most computation in terms of matrix-matrix operations.
- Recognize that the performance of some algorithms is inherently restricted by the memory operations that must be performed.
- Utilize interfaces to high-performance software libraries.
- Find additional resources for further learning.

12.2 Linear Algebra Building Blocks

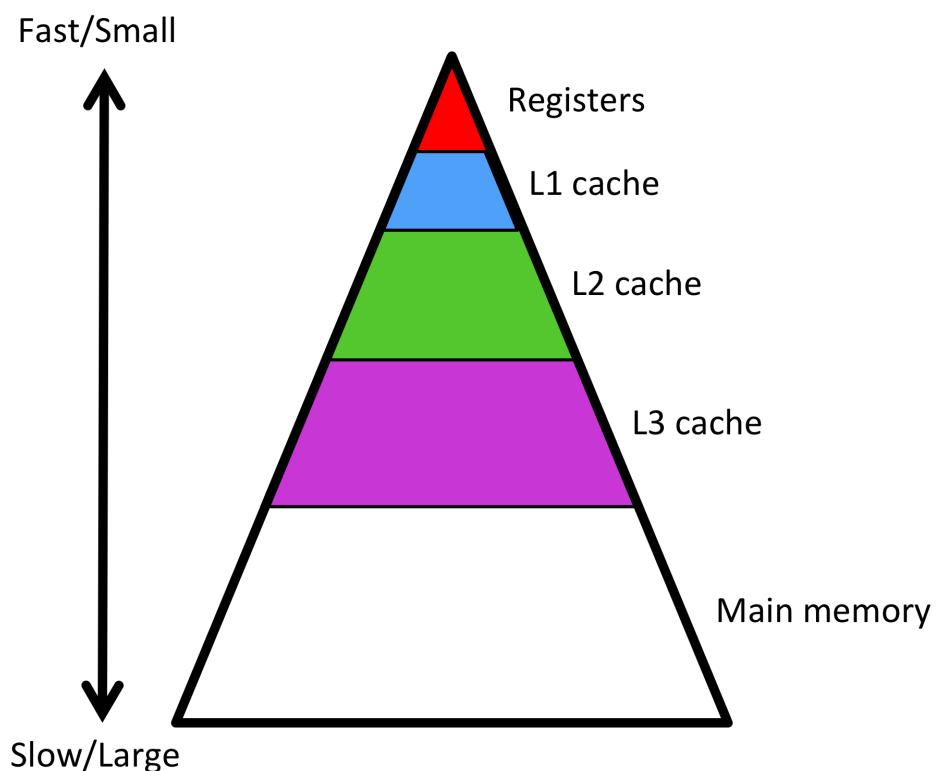
12.2.1 A simple model of the computer

The following is a relevant video from our course "[LAFF-On Programming for High Performance](#)" ([Unit 2.3.1](#)).

The good news about modern processors is that they can perform floating point operations at very high rates. The bad news is that "feeding the beast" is often a bottleneck: In order to compute with data, that data must reside in the registers of the processor and moving data from main memory into a register requires orders of magnitude more time than it does to then perform a floating point computation with that data.

In order to achieve high performance for the kinds of operations that we have encountered, one has to have a very high-level understanding of the memory hierarchy of a modern processor. Modern architectures incorporate multiple (compute) cores in a single processor. In our discussion, we blur this and will talk about the processor as if it has only one core.

It is useful to view the memory hierarchy of a processor as a pyramid.



At the bottom of the pyramid is the computer's main memory. At the top are the processor's registers. In between are progressively larger cache memories: the L1, L2, and L3 caches. (Some processors now have an L4 cache.) To compute, data must be brought into registers, of which there are only a few. Main memory is very large and very slow. The strategy for overcoming the cost (in time) of loading data is to amortise that cost over many

computations while it resides in a faster memory layer. The question, of course, is whether an operation we wish to perform exhibits the opportunity for such amortization.

Ponder This 12.2.1.1 For the processor in your computer, research the number of registers it has, and the sizes of the various caches.

12.2.2 Opportunities for optimization

We now examine the opportunity for the reuse of data for different linear algebra operations that we have encountered. In our discussion, we assume that scalars, the elements of vectors, and the elements of matrices are all stored as floating point numbers and that arithmetic involves floating point computation.

Example 12.2.2.1 Consider the dot product

$$\rho := x^T y + \rho,$$

where ρ is a scalar, and x and y are vectors of size m .

- How many floating point operations (flops) are required to compute this operation?
- If the scalar ρ and the vectors x and y are initially stored in main memory and are written back to main memory, how many reads and writes (memops) are required? (Give a reasonably tight lower bound.)
- What is the ratio of flops to memops?

Solution.

- *How many floating point operations are required to compute this operation?*
The dot product requires m multiplies and m additions, for a total of $2m$ flops.
- *If the scalar ρ and the vectors x and y are initially stored in main memory and (if necessary) are written back to main memory, how many reads and writes (memory operations) are required? (Give a reasonably tight lower bound.)*
 - The scalar ρ is moved into a register and hence only needs to be read once and written once.
 - The m elements of x and m elements of y must be read (but not written).

Hence the number of memops is $2(m + 1) \approx 2m$.

- *What is the ratio of flops to memops?*

$$\frac{2m \text{ flops}}{2(m + 1) \text{ memops}} \approx 1 \frac{\text{flops}}{\text{memops}}.$$

We conclude that the dot product does not exhibit an opportunity for the reuse of most data. \square

Homework 12.2.2.1 Consider the axpy operation

$$y := \alpha x + y,$$

where α is a scalar, and x and y are vectors of size m .

- How many floating point operations (flops) are required to compute this operation?
- If the scalar α and the vectors x and y are initially stored in main memory and are written back to main memory (if necessary), how many reads and writes (memops) are required? (Give a reasonably tight lower bound.)
- What is the ratio of flops to memops?

Solution.

- *How many floating point operations are required to compute this operation?*
The axpy operation requires m multiplies and m additions, for a total of $2m$ flops.
- *If the scalar α and the vectors x and y are initially stored in main memory and (if necessary) are written back to main memory, how many reads and writes (memory operations) are required? (Give a reasonably tight lower bound.)*
 - The scalar α is moved into a register, and hence only needs to be read once. It does not need to be written back to memory.
 - The m elements of x are read, and the m elements of y must be read and written.

Hence the number of memops is $3m + 1 \approx 3m$.

- *What is the ratio of flops to memops?*

$$\frac{2m \text{ flops}}{3m + 1 \text{ memops}} \approx \frac{2}{3} \frac{\text{flops}}{\text{memops}}.$$

We conclude that the axpy operation also does not exhibit an opportunity for the reuse of most data.

The time for performing a floating point operation is orders of magnitude less than that of moving a floating point number from and to memory. Thus, for an individual dot product or axpy operation, essentially all time will be spent in moving the data and the attained performance, in GFLOPS, will be horrible. The important point is that there just isn't much reuse of data when executing these kinds of "vector-vector" operations.

[Example 12.2.2.1](#) and [Homework 12.2.2.1](#) appear to suggest that, for example, when computing a matrix-vector multiplication, one should do so by taking dot products of rows with the vector rather than by taking linear combinations of the columns, which casts the computation in terms of axpy operations. It is more complicated than that: the fact that the algorithm that uses axpys computes with columns of the matrix, which are stored con-

tiguous when column-major order is employed, makes accessing memory cheaper.

Homework 12.2.2.2 Consider a matrix-vector multiplication

$$y := Ax + y,$$

where A is $m \times m$ and x and y are vectors of appropriate size.

- How many floating point operations (flops) are required to compute this operation?
- If the matrix and vectors are initially stored in main memory and are written back to main memory (if necessary), how many reads and writes (memops) are required? (Give a reasonably tight lower bound.)
- What is the ratio of flops to memops?

Solution.

- *How many floating point operations are required to compute this operation?*
 $y := Ax + y$ requires m^2 multiplies and m^2 additions, for a total of $2m^2$ flops.
- *If the matrix and vectors are initially stored in main memory and are written back to main memory, how many reads and writes (memops) are required? (Give a reasonably tight lower bound.)*

To come up with a reasonably tight lower bound, we observe that every element of A must be read (but not written). Thus, a lower bound is m^2 memops. The reading and writing of x and y contribute a lower order term, which we tend to ignore.

- *What is the ratio of flops to memops?*

$$\frac{2m^2 \text{ flops}}{m^2 \text{ memops}} \approx 2 \frac{\text{flops}}{\text{memops}}.$$

While this ratio is better than either the dot product's or the axpy operation's, it still does not look good.

What we notice is that there is a (slightly) better opportunity for reuse of data when computing a matrix-vector multiplication than there is when computing a dot product or axpy operation. Can the lower bound on data movement that is given in the solution be attained? If you bring y into, for example, the L1 cache, then it only needs to be read from main memory once and is kept in a layer of the memory that is fast enough to keep up with the speed of floating point computation for the duration of the matrix-vector multiplication. Thus, it only needs to be read and written once from and to main memory. If we then compute $y := Ax + y$ by taking linear combinations of the columns of A , staged as axpy operations, then at the appropriate moment an element of x with which an axpy is performed can be moved into a register and reused. This approach requires each element of A to be read once, each element of x to be read once, and each element of y to be read and written

(from and to main memory) once. If the vector y is too large for the L1 cache, then it can be partitioned into subvectors that do fit. This would require the vector x to be read into registers multiple times. However, x itself might then be reused from one of the cache memories.

Homework 12.2.2.3 Consider the rank-1 update

$$A := xy^T + A,$$

where A is $m \times m$ and x and y are vectors of appropriate size.

- How many floating point operations (flops) are required to compute this operation?
- If the matrix and vectors are initially stored in main memory and are written back to main memory (if necessary), how many reads and writes (memops) are required? (Give a reasonably tight lower bound.)
- What is the ratio of flops to memops?

Solution.

- *How many floating point operations are required to compute this operation?*
 $A := xy^T + A$ requires m^2 multiplies and m^2 additions, for a total of $2m^2$ flops. (One multiply and one add per element in A .)
- *If the matrix and vectors are initially stored in main memory and are written back to main memory, how many reads and writes (memops) are required? (Give a reasonably tight lower bound.)*

To come up with a reasonably tight lower bound, we observe that every element of A must be read and written. Thus, a lower bound is $2m^2$ memops. The reading of x and y contribute a lower order term. These vectors need not be written, since they don't change.

- *What is the ratio of flops to memops?*

$$\frac{2m^2 \text{ flops}}{2m^2 \text{ memops}} \approx 1 \frac{\text{flops}}{\text{memops}}.$$

What we notice is that data reuse when performing a matrix-vector multiplication with an $m \times m$ matrix is twice as favorable as is data reuse when performing a rank-1 update. Regardless, there isn't much reuse and since memory operations are much slower than floating point operations in modern processors, none of the operations discussed so far in the unit can attain high performance (if the data starts in main memory).

Homework 12.2.2.4 Consider the matrix-matrix multiplication

$$C := AB + C,$$

where A , B , and C are all $m \times m$ matrices.

- How many floating point operations (flops) are required to compute this operation?
- If the matrices are initially stored in main memory and (if necessary) are written back to main memory, how many reads and writes (memops) are required? (Give a simple lower bound.)
- What is the ratio of flops to memops for this lower bound?

Solution.

- *How many floating point operations are required to compute this operation?*

$C := AB + C$ requires m^3 multiplies and m^3 additions, for a total of $2m^3$ flops.

- *If the matrices are initially stored in main memory and (if necessary) are written back to main memory, how many reads and writes (memops) are required? (Give a simple lower bound.)*

To come up with a reasonably tight lower bound, we observe that every element of A , B , and C must be read and every element of C must be written. Thus, a lower bound is $4m^2$ memops.

- *What is the ratio of flops to memops?*

$$\frac{2m^3 \text{ flops}}{4m^2 \text{ memops}} \approx \frac{m}{2} \frac{\text{flops}}{\text{memops}}.$$

The lower bounds that we give in this unit are simple, but useful. There is actually a tight lower bound on the number of reads and writes that must be performed by a matrix-matrix multiplication. Details can be found in

- [50] Tyler Michael Smith, Bradley Lowery, Julien Langou, Robert A. van de Geijn, A Tight I/O Lower Bound for Matrix Multiplication, [arxiv.org:1702.02017v2](https://arxiv.org/abs/1702.02017v2), 2019. (To appear in ACM Transactions on Mathematical Software.)

The bottom line: operations like matrix-matrix multiplication exhibit the opportunity for reuse of data.

12.2.3 Basics of optimizing matrix-matrix multiplication

Let us again consider the computation

$$C := AB + C,$$

where A , B , and C are $m \times m$ matrices. *If m is small enough, then* we can read the three matrices into the L1 cache, perform the operation, and write the updated matrix C back to memory. In this case,

- During the computation, the matrices are in a fast memory (the L1 cache), which can keep up with the speed of floating point computation and

- The cost of moving each floating point number from main memory into the L1 cache is amortized over $m/2$ floating point computations.

If m is large enough, then the cost of moving the data becomes insignificant. (If carefully orchestrated, some of the movement of data can even be overlapped with computation, but that is beyond our discussion.)

We immediately notice there is a tension: m must be small so that all three matrices can fit in the L1 cache. Thus, this only works for relatively small matrices. However, for small matrices, the ratio $m/2$ may not be favorable enough to offset the very slow main memory.

Fortunately, matrix-matrix multiplication can be orchestrated by partitioning the matrices that are involved into submatrices, and computing with these submatrices instead. We recall that if we partition

$$C = \left(\begin{array}{c|c|c|c} C_{0,0} & C_{0,1} & \cdots & C_{0,N-1} \\ \hline C_{1,0} & C_{1,1} & \cdots & C_{1,N-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline C_{M-1,0} & C_{M-1,1} & \cdots & C_{M-1,N-1} \end{array} \right),$$

$$A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,K-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,K-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,K-1} \end{array} \right),$$

and

$$B = \left(\begin{array}{c|c|c|c} B_{0,0} & B_{0,1} & \cdots & B_{0,N-1} \\ \hline B_{1,0} & B_{1,1} & \cdots & B_{1,N-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline B_{K-1,0} & B_{K-1,1} & \cdots & B_{K-1,N-1} \end{array} \right),$$

where $C_{i,j}$ is $m_i \times n_j$, $A_{i,p}$ is $m_i \times k_p$, and $B_{p,j}$ is $k_p \times n_j$, with $\sum_{i=0}^{M-1} m_i = m$, $\sum_{j=0}^{N-1} n_j = n$, and $\sum_{p=0}^{K-1} k_p = k$, then

$$C_{i,j} := \sum_{p=0}^{K-1} A_{i,p} B_{p,j} + C_{i,j}.$$

If we choose each m_i , n_j , and k_p small enough, then the submatrices fit in the L1 cache. This still leaves us with the problem that these sizes must be reasonably small if the ratio of flops to memops is to be sufficient. The answer to that is to block for multiple levels of caches.

12.2.4 Optimizing matrix-matrix multiplication, the real story

High-performance implementations of matrix-matrix multiplication (and related operations) block the computation for multiple levels of caches. This greatly reduces the overhead related to data movement between memory layers. In addition, at some level the implementation pays very careful attention to the use of vector registers and vector instructions so as to

leverage parallelism in the architecture. This allows multiple floating point operations to be performed simultaneous within a single processing core, which is key to high performance on modern processors.

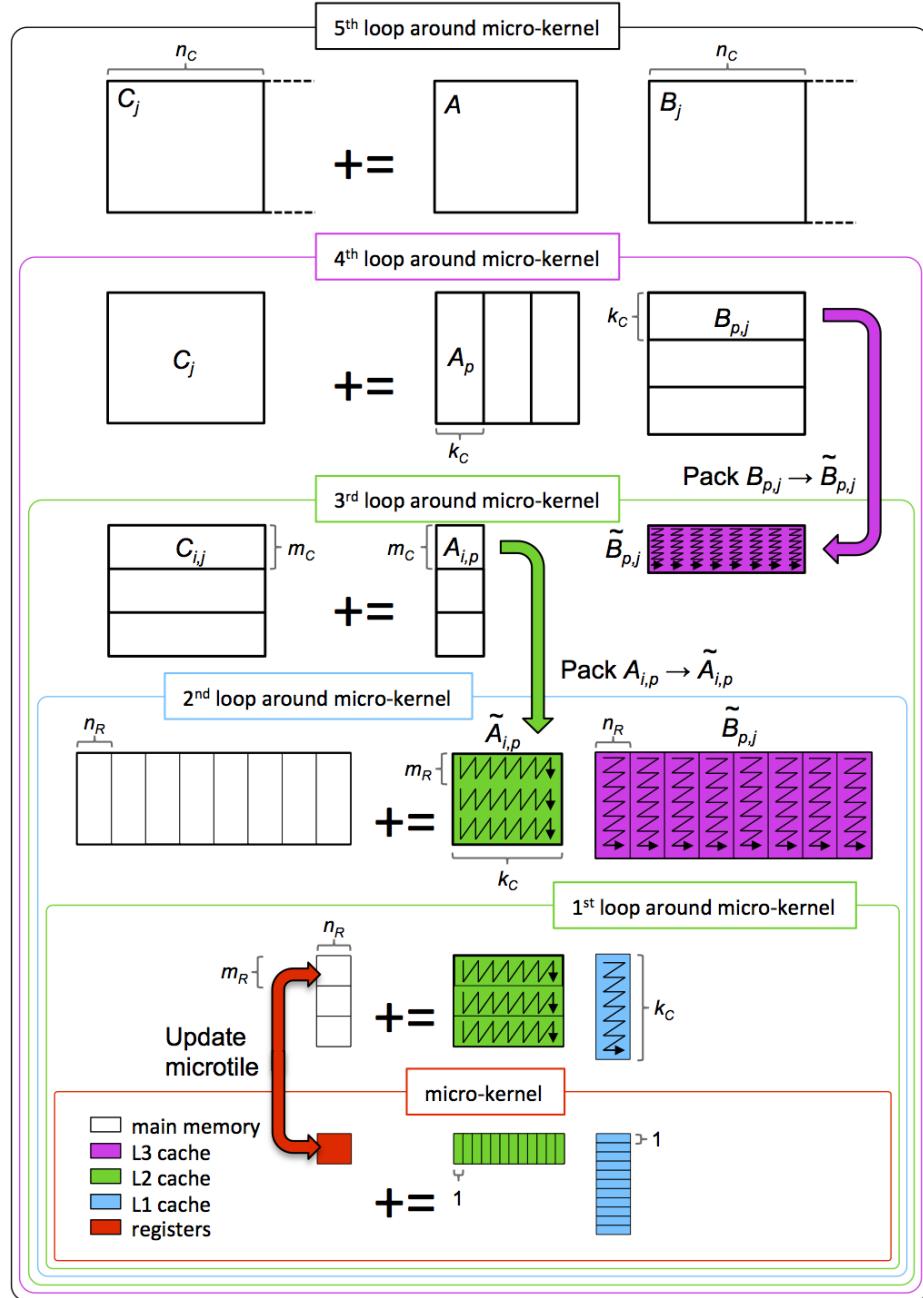
Current high-performance libraries invariably build upon the insights in the paper

- [49] Kazushige Goto and Robert van de Geijn, Anatomy of High-Performance Matrix Multiplication, ACM Transactions on Mathematical Software, Vol. 34, No. 3: Article 12, May 2008.

This paper is generally considered a "must read" paper in high-performance computing. The techniques in that paper were "refactored" (carefully layered so as to make it more maintainable) in BLIS, as described in

- [52] Field G. Van Zee and Robert A. van de Geijn, BLIS: A Framework for Rapidly Instantiating BLAS Functionality, ACM Journal on Mathematical Software, Vol. 41, No. 3, June 2015.

The algorithm described in both these papers can be captured by the picture in [Figure 12.2.4.1](#).



Picture adapted from [51]. Click to enlarge. [PowerPoint source](#) (step-by-step).

Figure 12.2.4.1 Blocking for multiple levels of cache, with packing.

This is not the place to go into great detail. Once again, we point out that the interested learner will want to consider our Massive Open Online Course titled "[LAFF-On Programming for High Performance](#)" [39]. In that MOOC, we illustrate issues in high performance by exploring how high performance matrix-matrix multiplication is implemented in practice.

12.2.5 BLAS and BLIS

To facilitate portable high performance, scientific and machine learning software is often written in terms of the Basic Linear Algebra Subprograms (BLAS) interface [24] [15] [14]. This interface supports widely-used basic linear algebra functionality. The idea is that if optimized implementations of this interface are available for different target computer architectures, then a degree of portable high performance can be achieved by software that is written in terms of this interface. The interface was designed for the Fortran programming language starting in the 1970s. In our discussions, we will also show how to interface to it from the C programming language. As we discuss the BLAS, it may be useful to keep the "Quick Reference Guide" [9] to this interface handy.

In addition, our own BLAS-like Library Instantiation Software (BLIS) [52][47] is not only a framework for the rapid instantiation of high-performing BLAS through the traditional BLAS interface, but also an extended interface that, we believe, is more natural for C programming. We refer to this interface as the **BLIS typed interface** [48], to distinguish it from the traditional BLAS interface and the object based **BLIS Object API**.

A number of open source and vendors high-performance implementations of the BLAS interface are available. For example,

- Our **BLAS-like Library Instantiation Software (BLIS)** is a widely-used open source implementation of the BLAS for modern CPUs. It underlies AMD's **Optimizing CPU Libraries (AOCL)**.
- Arm's **Arm Performance Libraries**.
- Cray's **Cray Scientific and Math Libraries (CSML)**.
- IBM's **Engineering and Scientific Subroutine Library (ESSL)**.
- Intel's **Math Kernels Library (MKL)**.
- NVIDIA's **cuBLAS**.

12.2.5.1 Level-1 BLAS (vector-vector functionality)

The original BLAS [24] interface was proposed in the 1970s, when vector supercomputers like the Cray 1 and Cray 2 reigned supreme. On this class of computers, high performance was achieved if computation could be cast in terms of vector-vector operations with vectors that were stored contiguous in memory (with "stride one"). These are now called "level-1 BLAS" because they perform $O(n)$ computation on $O(n)$ data (when the vectors have size n). The "1" refers to $O(n) = O(n^1)$ computation.

We here list the vector-vector functionality that is of importance in this course, for the case where we compute with double precision real-valued floating point numbers.

- **DOT**: Returns $x^T y$, the dot product of real-valued x and y .

- Traditional BLAS interface:

```
FUNCTION DDOT( N, X, INCX, Y, INCY )
```

- C:

```
double ddot_( int* n, double* x, int* incx, double* y, int* incy );
```

- BLIS typed interface for computing $\rho := \alpha x^T y + \beta \rho$ ([details](#)):

```
void bli_ddotvx( conj_t conjx, conj_t conjy, dim_t n,
                  double* alpha, double* x, inc_t incx, double* y, inc_t incy,
                  double* beta, double* rho );
```

- AXPY: Updates $y := \alpha x + y$, the scaled vector addition of x and y .

- Traditional BLAS interface:

```
SUBROUTINE DAXPY( N, ALPHA, X, INCX, Y, INCY )
```

- C:

```
void daxpy_( int* n, double* alpha, double* x, int* incx,
              double* y, int* incy );
```

- BLIS typed interface ([details](#)):

```
void bli_daxpyf( conj_t conjx, dim_t n,
                  double* alpha, double* x, inc_t incx, double* y, inc_t incy );
```

- IAMAX: Returns the index of the element of x with maximal absolute value (indexing starts at 1).

- Traditional BLAS interface:

```
FUNCTION IDAMAX( N, X, INCX )
```

- C:

```
int idamax_( int* n, double* x, int* incx );
```

- BLIS typed interface ([details](#)):

```
void bli_damaxv( dim_t n, double* x, inc_t incx, dim_t* index );
```

- NRM2: Returns $\|x\|_2$, the 2-norm of real-valued x .

- Traditional BLAS interface:

```
FUNCTION DNRM2( N, X, INCX )
```

- C:

```
double dnrm2_( int* n, double* x, int* incx );
```

- BLIS typed interface ([details](#)):

```
void bli_dnormfv( dim_t n, double* x, inc_t incx, double* norm );
```

Versions of these interfaces for single precision real, single precision complex, and double precision complex can be attained by replacing the appropriate D with S, C, or Z in the call to the Fortran BLAS interface, or d with s, c, or z in the C and BLIS typed interfaces.

12.2.5.2 Level-2 BLAS (matrix-vector functionality)

In addition to providing portable high performance, a second purpose of the BLAS is to improve readability of code. Many of the algorithms we have encountered are cast in terms of matrix-vector operations like matrix-vector multiplication and the rank-1 update. By writing the code in terms of routines that implement such operations, the code more closely resembles the algorithm that is encoded. The level-2 BLAS [15] interface provides matrix-vector functionality. The "level-2" captures that they perform $O(n^2)$ computation (on $O(n^2)$ data), when the matrix invoked is of size $n \times n$.

We here list the the matrix-vector operations that are of importance in this course, for the case where we compute with double precision real-valued floating point numbers.

- GEMV: Updates $y := \alpha \text{op}(A)x + \beta y$, where $\text{op}(A)$ indicates whether A is to be transposed.

- Traditional BLAS interface:

```
SUBROUTINE DGEMV( TRANS, M, N, ALPHA, A, LDA, X, INCX,
                  BETA, Y, INCY )
```

- C:

```
void dgemv_( char* transa, int* m, int* n,
             double* alpha, double* a, int* lda, double* x, int* incx,
             double* beta, double* y, int* incy );
```

- BLIS typed interface for computing $y := \alpha \text{op}_A(A)\text{op}_x(x) + \beta y$ ([details](#)):

```
void bli_dgemv( trans_t transa, conj_t conjx, dim_t m, dim_t n,
                double* alpha, double* A, inc_t rsa, inc_t csa,
                double* x, inc_t incx, double* beta, double* y, inc_t incy );
```

- SYMV: Updates $y := \alpha Ax + \beta y$, where A is symmetric and only the upper or lower triangular part is stored.

- Traditional BLAS interface:

```
SUBROUTINE DSYMV( UPLO, N, ALPHA, A, LDA, X, INCX,
                  BETA, Y, INCY )
```

- C:

```
void dsymv_( char* uplo, int* n,
             double* alpha, double* a, int* lda, double* x, int* incx,
             double* beta, double* y, int* incy );
```

- BLIS typed interface ([details](#)):

```
void bli_dhemv( uplo_t uploa, conj_t conja, conj_t conjx, dim_t n,
                 double* alpha, double* A, inc_t rsa, inc_t csa,
                 double* x, inc_t incx, double* beta, double* y, inc_t incy );
```

- TRSV: Updates $x := \alpha \text{op}(A)^{-1}x$, where $\text{op}(A)$ indicates whether A is to be transposed and A is either (unit) upper or lower triangular.

- Traditional BLAS interface:

```
SUBROUTINE DTRSV( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )
```

- C:

```
void dtrsv_( char* uplo, char* transa, char* diag, int* n,
             double* a, int* lda, double* x, int* incx )
```

- BLIS typed interface ([details](#)):

```
void bli_dtrsv( uplo_t uploa, trans_t transa, diag_t diag, dim_t n,
                 double* alpha, double* A, inc_t rsa, inc_t csa,
                 double* x, inc_t incx );
```

- GER: Updates $A := \alpha xy^T + A$:

- Traditional BLAS interface:

```
SUBROUTINE DGER( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )
```

- C:

```
void dger_( int* m, int* n, double* alpha, double* x, int* incx,
            double* y, int* incy, double* a, int* lda );
```

- BLIS typed interface ([details](#)):

```
void bli_dger( conj_t conjx, conj_t conjy, dim_t m, dim_t n,
               double* alpha, double* x, inc_t incx, double* y, inc_t incy,
               double* A, inc_t rsa, inc_t csa );
```

- SYR: Updates $A := \alpha xx^T + A$, where A is symmetric and stored in only the upper or lower triangular part of array A:

- Traditional BLAS interface:

```
SUBROUTINE DSYR( UPLO, N, ALPHA, X, INCX, A, LDA )
```

- C:

```
void dsyr_( char* uplo, int* n, double* alpha, double* x, int* incx,
            double* a, int* lda );
```

- BLIS typed interface ([details](#)):

```
void bli_dher( uplo_t uploa, conj_t conjx, dim_t n,
               double* alpha, double* x, inc_t incx, double* A, inc_t rsa, inc_t csa );
```

- SYR2: Updates $A := \alpha(xy^T + yx^T) + A$, where A is symmetric and stored in only the upper or lower triangular part of array A:

- Traditional BLAS interface:

```

SUBROUTINE DSYR2( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA )
  ○ C:
    void dsyr2_( char* uplo, int* n, double* alpha, double* x, int* incx,
                 double* y, int* incy, double* a, int* lda );
  ○ BLIS typed interface (details):
    void bli_dher2( uplo_t uploa, conj_t conjx, conj_t conjy, dim_t n,
                    double* alpha, double* x, inc_t incx, double* y, inc_t incy,
                    double* A, inc_t rsa, inc_t csa );

```

12.2.5.3 Level-3 BLAS (matrix-matrix functionality)

To attain high performance, computation has to be cast in terms of operations that reuse data many times (for many floating point operations). Matrix-matrix operations that are special cases of matrix-matrix multiplication fall into this category. The strategy is to cast higher level linear algebra functionality can be implemented so that most computation is in terms of matrix-matrix operations by calling the level-3 BLAS routines [14]. The "level-3" captures that these perform $O(n^3)$ computation (on $O(n^2)$ data), when the matrices involved are of size $n \times n$.

We here list the the matrix-matrix operations that are of importance in this course, for the case where we compute with double precision real-valued floating point numbers.

- GEMM: Updates $C := \alpha \text{op}_A(A) \text{op}_B(B) + \beta C$, where $\text{op}_A(A)$ and $\text{op}_A(A)$ indicate whether A and/or B are to be transposed.

○ Traditional BLAS interface:

```

SUBROUTINE DGEMM( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
                  BETA, C, LDC )

```

○ C:

```

void dgemm_( char* transa, char* transb, int* m, int* n, int* k,
             double* alpha, double* a, int* lda, double* b, int* ldb,
             double* beta, double* c, int* ldc );

```

○ BLIS typed interface for computing $C := \alpha \text{op}_A(A) \text{op}_B(B) + \beta C$ ([details](#)):

```

void bli_dgemm( trans_t transa, trans_t transb,
                 dim_t m, dim_t n, dim_t k,
                 double* alpha, double* A, inc_t rsa, inc_t csa,
                 double* B, inc_t rsB, inc_t csb,
                 double* beta, double* C, inc_t rsc, inc_t csc );

```

- SYMM: Updates $C := \alpha AB + \beta C$ or $C := \alpha BA + \beta C$, where A is symmetric and only the upper or lower triangular part is stored.

○ Traditional BLAS interface:

```
SUBROUTINE DSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
                  BETA, C, LDC )
```

- o C:

```
void dsymm_( char* uplo, char* uplo, int* m, int* n,
            double* alpha, double* a, int* lda, double* b, int* ldb,
            double* beta, double* c, int* ldc );
```

- o BLIS typed interface ([details](#)):

```
void bli_dhemm( side_t sidea, uplo_t uploa, conj_t conja, trans_t transb,
                dim_t m, dim_t n, double* alpha, double* A, inc_t rsa, inc_t csa,
                double* B, inc_t rsb, inc_t csb,
                double* beta, double* C, inc_t rsc, inc_t csc );
```

- TRSM: Updates $B := \alpha \text{op}(A)^{-1}B$ or $B := \alpha B \text{op}(A)^{-1}$, where $\text{op}(A)$ indicates whether A is to be transposed and A is either (unit) upper or lower triangular.

- o Traditional BLAS interface:

```
SUBROUTINE DTRSM( SIDE, UPLO, TRANSA, DIAG, M, N,
                  ALPHA, A, LDA, B, LDB, C, LDC )
```

- o C:

```
void dtrsm_( char* side, char* uplo, char* transa, char* diag,
             int* m, int*, n, double *alpha, double* a, int* lda,
             double* b, int* ldb )
```

- o BLIS typed interface ([details](#)):

```
void bli_dtrsm( side_t sidea, uplo_t uploa, trans_t transa, diag_t diag,
                dim_t m, dim_t n, double* alpha, double* A, inc_t rsa, inc_t csa,
                double* B, inc_t rsb, inc_t csb );
```

- SYRK: Updates $C := \alpha AA^T + \beta C$ or $C := \alpha A^T A + \beta C$, where C is symmetric and stored in only the upper or lower triangular part of array C:

- o Traditional BLAS interface:

```
SUBROUTINE DSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA,
                  BETA, C, LDC )
```

- o C:

```
void dsyrk_( char* uplo, char* trans, int* n, int* k,
             double* alpha, double* A, int* lda, double* beta, double* C, int* ldc );
```

- o BLIS typed interface ([details](#)):

```
void bli_dherk( uplo_t uploc, transt_t transa, dim_t n, dim_t k,
                double* alpha, double* A, inc_t rsa, inc_t csa
                double* beta, double* C, inc_t rsc, inc_t csc );
```

- SYR2K: Updates $C := \alpha(AB^T + BA^T) + \beta C$ or $C := \alpha(A^T B + B^T A) + \beta C$, where C is symmetric and stored in only the upper or lower triangular part of array C :

- Traditional BLAS interface:

```
SUBROUTINE DSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB,
                   BETA, C, LDC )
```

- C :

```
void dsyr2k_( char* uplo, char *trans, int* n, int* k, double* alpha, double* A,
              double* b, int* ldb, double* beta, double* c, int* ldc );
```

- BLIS typed interface ([details](#)):

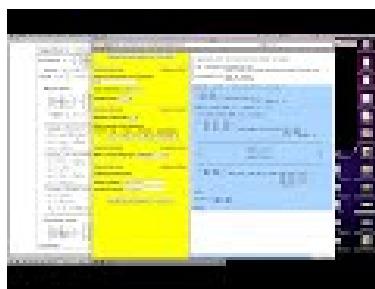
```
void bli_dher2k( uplo_t uploc, trans_t transab, dim_t n, dim_t k,
                  double* alpha, double* A, inc_t rsa, inc_t csa,
                  double* B, inc_t rsc, inc_t csc,
                  double* beta, double* C, inc_t rsc, inc_t csc );
```

These operations are often of direct importance to scientific or machine learning applications. In the next section, we show how higher-level linear algebra operations can be cast in terms of this basic functionality.

12.3 Casting Computation in Terms of Matrix-Matrix Multiplication

12.3.1 Blocked Cholesky factorization

In the following video, we demonstrate how high-performance algorithms can be quickly translated to code using the FLAME abstractions. It is a long video that was recorded in a single sitting and has not been edited. (You need not watch the whole video if you "get the point.") The purpose is to convey the importance of programming in a way that reflects how one naturally derives and explains an algorithm. In the next unit, you will get to try such implementation yourself, for the LU factorization.



YouTube: <https://www.youtube.com/watch?v=PJ6ektH977o>

- The notes to which this video refers can be found at <http://www.cs.utexas.edu/users/flame/Notes/NotesOnChol.pdf>.

- You can find all the implementations that are created during the video in the directory `Assignments/Week12/Chol/`. They have been updated slightly since the video was created in 2011. In particular, the Makefile was changed so that now the BLIS implementation of the BLAS is used rather than OpenBLAS.
- The Spark tool that is used to generate code skeletons can be found at <http://www.cs.utexas.edu/users/flame/Spark/>.
- The following reference may be useful:
 - A Quick Reference Guide to the FLAME API to BLAS functionality can be found at <http://www.cs.utexas.edu/users/flame/pubs/FLAMEC-BLAS-Quickguide.pdf>.
 - [41] Field Van Zee, *libflame: The Complete Reference*, <http://www.lulu.com>, 2009.

12.3.2 Blocked LU factorization

Homework 12.3.2.1 Consider the LU factorization $A = LU$, where A is an $m \times m$ matrix, discussed in [Subsection 5.1.1](#) and subsequent units. The right-looking algorithm for computing it is given by

$A = LU(A)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array} \right)$
$a_{21} := a_{21}/\alpha_{11}$
$A_{22} := A_{22} - a_{21}a_{12}^T$
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{ccc c} A_{00} & a_{01} & A_{02} & \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T & \\ A_{20} & a_{21} & A_{22} & \end{array} \right)$
endwhile

Figure 12.3.2.1 Right-looking LU factorization algorithm.

For simplicity, we do not consider pivoting yet.

- How many floating point operations (flops) are required to compute this operation?
- If the matrix is initially stored in main memory and is written back to main memory, how many reads and writes (memops) are required? (Give a simple lower bound.)
- What is the ratio of flops to memops for this lower bound?

Solution.

- How many floating point operations are required to compute this operation?
- From [Homework 5.2.2.1](#), we know that this right-looking algorithm requires $\frac{2}{3}m^3$ flops.
- If the matrix is initially stored in main memory and is written back to main memory, how many reads and writes (memops) are required? (Give a simple lower bound.)
- We observe that every element of A must be read and written. Thus, a lower bound is $2m^2$ memops.
- What is the ratio of flops to memops?

$$\frac{\frac{2}{3}m^3 \text{ flops}}{2m^2 \text{ memops}} \approx \frac{m}{3} \frac{\text{flops}}{\text{memops}}.$$

The insight is that the ratio of computation to memory operations, $m/3$, does not preclude high performance. However, the algorithm in [Figure 12.3.2.1](#) casts most computation in terms of rank-1 updates and hence will not achieve high performance. The reason is that the performance of each of those individual updates is limited by the unfavorable ratio of floating point operations to memory operations.

Just like it is possible to rearrange the computations for a matrix-matrix multiplication in order to reuse data that is brought into caches, one could carefully rearrange the computations needed to perform an LU factorization. While one can do so for an individual operation, think of the effort that this would involve for every operation in (dense) linear algebra. Not only that, this effort would likely have to be repeated for new architectures as they become available.

Remark 12.3.2.2 The key observation is that *if* we can cast most computation for the LU factorization in terms of matrix-matrix operations (level-3 BLAS functionality) *and* we link an implementation to a high-performance library with BLAS functionality, *then* we can achieve portable high performance for our LU factorization algorithm.

Let us examine how to cast most computation for the LU factorization in terms of matrix-matrix operations. You may want to start by reviewing the discussion of how to derive the (unblocked) algorithm in [Figure 12.3.2.1](#), in [Subsection 5.2.2](#), which we repeat below on the left. The derivation of a so-called **blocked algorithm** is given to its right.

Partition

$$A \rightarrow \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}, L \rightarrow \begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix},$$

$$\text{and } U \rightarrow \begin{pmatrix} v_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix}.$$

Partition

$$A \rightarrow \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, L \rightarrow \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix},$$

$$\text{and } U \rightarrow \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix},$$

where A_{11} , L_{11} , and U_{11} are $b \times b$.

Plug partitioned matrices into $A = LU$.

$$= \underbrace{\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} v_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix}}_{\begin{pmatrix} v_{11} & u_{12}^T \\ v_{11}l_{21} & l_{21}u_{12}^T + L_{22}U_{22} \end{pmatrix}}.$$

Equate the submatrices and manipulate

- $\alpha_{11} := v_{11} = \alpha_{11}$ (No-op).
- $a_{12}^T := u_{12}^T = a_{12}^T$ (No-op).
- $a_{21} := l_{21} = a_{21}/v_{11} = a_{21}/\alpha_{11}$.
- $A_{22} := A_{22} - l_{21}u_{12}^T = A_{22} - a_{21}a_{12}^T$.

The derivation on the left yields the algorithm in [Figure 12.3.2.1](#).

Plug partitioned matrices into $A = LU$.

$$= \underbrace{\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}}_{\begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{pmatrix}}.$$

Equate the submatrices and manipulate

- $A_{11} := L \setminus U_{11}$ (overwrite A_{11} with its LU factorization).
- $A_{12} := U_{12} = L_{11}^{-1}A_{12}$ (triangular solve with multiple right-hand sides).
- $A_{21} := L_{21} = A_{21}U_{11}^{-1}$ (triangular solve with multiple right-hand sides).
- $A_{22} := A_{22} - L_{21}U_{12} = A_{22} - A_{21}A_{12}$ (matrix-matrix multiplication).

The derivation on the right yields the algorithm in [Figure 12.3.2.3](#).

$A = \text{LU-blk}(A, b)$
$A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$
A_{TL} is 0×0
while $n(A_{TL}) < n(A)$
choose block size b
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c cc} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{array} \right)$
A_{11} is (at most) $b \times b$
<hr/>
$A_{11} := L \setminus U_{11} = LU(A_{11})$ LU factorization
$A_{12} := U_{12} = L_{11}^{-1}A_{12}$ TRSM
$A_{21} := A_{21}U_{11}^{-1}$ TRSM
$A_{22} := A_{22} - A_{21}A_{12}$ GEMM
<hr/>
$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c cc} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12}^T \\ A_{20} & A_{21} & A_{22} \end{array} \right)$
endwhile

Figure 12.3.2.3 Blocked right-looking LU factorization algorithm.

Let us comment on each of the operations in [Figure 12.3.2.3](#).

- $A_{11} := L \setminus U_{11} = LU(A_{11})$ indicates that we need to compute the LU factorization of A_{11} , overwriting that matrix with the unit lower triangular matrix L_{11} and upper triangular matrix U_{11} . Since A_{11} is $b \times b$ and we usually take $b \ll m$, not much of the total computation is in the operation, and it can therefore be computed with, for example, an unblocked algorithm. Also, if it is small enough, it will fit in one of the smaller caches and hence the memory overhead of performing the rank-1 updates will not be as dramatic.
- $A_{12} := U_{12} = L_{11}^{-1} A_{12}$ is an instance of solving $LX = B$ with unit lower triangular matrix L for X . This is referred to as a **triangular solve with multiple right-hand sides** (TRSM) since we can partition B and X by columns so that

$$L \begin{pmatrix} x_0 & | & x_1 & | & \cdots \\ \hline Lx_0 & | & Lx_1 & | & \cdots \end{pmatrix} = \begin{pmatrix} b_0 & | & b_1 & | & \cdots \end{pmatrix},$$

and hence for each column of the right-hand side, b_j , we need to solve a triangular system, $Lx_j = b_j$.

- $A_{21} := L_{21} = A_{21}U_{11}^{-1}$ is an instance of solving $XU = B$, where U is upper triangular. We notice that if we partition X and B by rows, then

$$\underbrace{\begin{pmatrix} \tilde{x}_0^T \\ \tilde{x}_1^T \\ \vdots \\ \tilde{x}_n^T \end{pmatrix}}_{\begin{pmatrix} \tilde{x}_0^T U \\ \tilde{x}_1^T U \\ \vdots \\ \tilde{x}_n^T U \end{pmatrix}} U = \begin{pmatrix} \tilde{b}_0^T \\ \tilde{b}_1^T \\ \vdots \\ \tilde{b}_n^T \end{pmatrix}$$

and we recognize that each row, \tilde{x}_i^T , is computed from $\tilde{x}_i^T U = \tilde{b}_i^T$ or, equivalently, by solving $U^T (\tilde{x}_i^T)^T = (\tilde{b}_i^T)^T$. We observe it is also a triangular solve with multiple right-hand sides (TRSM).

- The update $A_{22} := A_{22} - A_{21}A_{12}$ is an instance of $C := \alpha AB + \beta C$, where the k (inner) size is small. This is often referred to as a **rank-k update**.

In the following homework, you will determine that most computation is now cast in terms of the rank-k update (matrix-matrix multiplication).

Homework 12.3.2.2 For the algorithm in [Figure 12.3.2.3](#), analyze the (approximate) number of flops that are performed by the LU factorization of A_{11} and updates of A_{21} and A_{12} , aggregated over all iterations. You may assume that the size of A , m , is an integer multiple of the block size b , so that $m = Kb$. Next, determine the ratio of the flops spent in the

indicated operations to the total flops.

Solution. During the k th iteration, when A_{00} is $(kb) \times (kb)$, we perform the following number of flops in these operations:

- $A_{11} := LU(A_{11})$: approximately $\frac{2}{3}b^3$ flops.
- $A_{12} := L_{11}^{-1}A_{12}$: During the k th iteration, A_{00} is $(kb) \times (kb)$, A_{11} is $b \times b$, and A_{12} is $b \times ((K - k - 1)b)$. (It helps to draw a picture.) Hence, the total computation spent in the operation is approximately $((K - k - 1)b)b^2 = (K - k - 1)b^3$ flops.
- $A_{21} := A_{21}U_{11}^{-1}$: During the k th iteration, A_{00} is $(kb) \times (kb)$, A_{11} is $b \times b$, and A_{21} is $((K - k - 1)b) \times b$. Hence, the total computation spent in the operation is approximately $((K - k - 1)b)b^2 = (K - k - 1)b^3$ flops.

If we sum this over all K iterations, we find that the total equals

$$\begin{aligned} & \sum_{k=0}^{K-1} \left[\frac{2}{3}b^3 + 2(K - k - 1)b^3 \right] \\ &= \left[\frac{2}{3}K + 2 \sum_{k=0}^{K-1} (K - k - 1) \right] b^3 \\ &= \left[\frac{2}{3}K + 2 \sum_{j=0}^{K-1} j \right] b^3 \\ &\approx \left[\frac{2}{3}K + K^2 \right] b^3 = \\ & \frac{2}{3}b^2m + bm^2. \end{aligned}$$

Thus, the ratio of time spent in these operation to the total cost of the LU factorization is

$$\frac{\frac{2}{3}b^2m + bm^2}{\frac{2}{3}m^3} = \left(\frac{b}{m} \right)^2 + \frac{3}{2} \frac{b}{m}.$$

From this last exercise, we learn that if b is fixed and m gets large, essentially all computation is in the update $A_{22} := A_{22} - A_{21}A_{12}$, which we know can attain high performance.

Homework 12.3.2.3 In directory `Assignments/Week12/matlab/`, you will find the following files:

- `LU_right_looking.m`: an implementation of the unblocked right-looking LU factorization.
- `LU_blk_right_looking.m`: A code skeleton for a function that implements a blocked LU factorization.

`[A_out] = LU_blk_right_looking(A, nb_alg)`

performs a blocked LU factorization with block size b equal to `nb_alg`, overwriting `A_out` with L and U .

- `time_LU.m`: A test routine that tests the various implementations.

These resources give you the tools to implement and test the blocked LU factorization.

1. Translate the blocked LU factorization in [Figure 12.3.2.3](#) into code by completing `LU_blk_right_looking.m`.
2. Test your implementation by executing `time_LU.m`.
3. Once you get the right answer, try different problem sizes to see how the different implementations perform. Try $m = 500, 1000, 1500, 2000$.

On the discussion forum, discuss what you think are some of the reasons behind the performance you observe.

Hint. You can extract L_{11} and U_{11} from A_{11} with

```
L11 = tril( A11,-1 ) + eye( size( A11 ) );
A11 = triu( A11 );
```

Don't invert L_{11} and U_{11} . In the command window execute

```
help /
help \
```

to read up on how those operators allow you to solve with a matrix.

Solution.

- [LU_blk_right_looking.m](#).

Notice that your blocked algorithm gets MUCH better performance than does the unblocked algorithm. However, the native LU factorization of Matlab does much better yet. The call `lu(A)` by Matlab links to a high performance implementation of the LAPACK interface, which we will discuss later.

Homework 12.3.2.4 For this exercise you will implement the unblocked and blocked algorithm in C. To do so, you need to install the BLAS-like Library Instantiation Software (BLAS) (see [Subsubsection 0.2.4.1](#)) and the libflame library (see [Subsubsection 0.2.4.2](#)). Even if you have not previously programmed in C, you should be able to follow along.

In `Assignments/Week12/LU/FLAMEC/`, you will find the following files:

- `LU_unb_var5.c` : a skeleton for implementing the unblocked right-looking LU factorization (which we often call "Variant 5").
- `LU_blk_var5.c` : a skeleton for implementing the blocked right-looking LU factorization.
- `REF_LU.c`: A simple triple-nested loop implementation of the algorithm.
- `driver.c`: A "driver" for testing various implementations. This driver creates matrices of different sizes and checks the performance and correctness of the result.

- **Makefile:** A "makefile" that compiles, links, and executes. To learn about Makefiles, you may want to check [Wikipedia](#). (search for "Make" and choose "Make (software").)

Our libflame library allows a simple translation from algorithms that have been typeset with the FLAME notation (like those in [Figure 12.3.2.1](#) and [Figure 12.3.2.3](#)). The BLAS functionality discussed earlier in this week is made available through an interface that hides details like size, stride, etc. A Quick Reference Guide for this interface can be found at <http://www.cs.utexas.edu/users/flame/pubs/FLAMEC-BLAS-Quickguide.pdf>.

With these resources, complete

- `LU_unb_var5.c` and
- `LU_blk_var5.c`.

You can skip the testing of `LU_blk_var5` by changing the appropriate TRUE to FALSE in `driver.c`.

Once you have implemented one or both, you can test by executing `make test` in a terminal session (provided you are in the correct directory). This will compile and link, yielding the executable `driver.x`, and will then execute this driver. The output is redirected to the file `output.m`. Here is a typical line in the output:

```
data_REF( 10, 1:2 ) = [ 2000 2.806644e+00 ];
data_FLAME( 10, 1:2 ) = [ 2000 4.565624e+01 ];
data_unb_var5( 10, 1:3 ) = [ 2000 3.002356e+00 4.440892e-16];
data_blk_var5( 10, 1:3 ) = [ 2000 4.422223e+01 7.730705e-12];
```

- The first line reports the performance of the reference implementation (a simple triple-nested loop implementation of LU factorization without pivoting). The last number is the rate of computation in GFLOPS.
- The second line reports the performance of the LU factorization without pivoting that is part of the libflame library. Again, the last number reports the rate of computation in GFLOPS.
- The last two lines report the rate of performance (middle number) and difference of the result relative to the reference implementation (last number), for your unblocked and blocked implementations. It is important to check that the last number is small. For larger problem sizes, the reference implementation is not executed, and this difference is not relevant.

The fact that the blocked version shows a larger difference than does the unblocked is not significant here. Both have roundoff error in the result (as does the reference implementation) and we cannot tell which is more accurate.

You can cut and past the `output.m` file into matlab to see the performance data presented as a graph.

Ponder This 12.3.2.5 In Subsection 5.5.1, we discuss five unblocked algorithms for computing LU factorization. Can you derive the corresponding blocked algorithms? You can use the materials in Assignments/Week12/LU/FLAMEC/ to implement and test all five.

12.3.3 Other high-performance dense linear algebra algorithms

Throughout this course, we have pointed to papers that discuss the high-performance implementation of various operations. Here we review some of these.

12.3.3.1 High-performance QR factorization

We saw in Week 3 that the algorithm of choice for computing the QR factorization is based on Householder transformations. The reason is that this casts the computation in terms of the application of unitary transformations, which do not amplify error. In Subsection 3.4.1, we discussed how the Householder QR factorization algorithm can be cast in terms of matrix-matrix operations.

An important point to note is that in order to cast the computation in terms of matrix-matrix multiplication, one has to form the "block Householder transformation"

$$I + UT^{-1}U^H.$$

When the original matrix is $m \times n$, this requires $O(mb^2)$ floating point operations to be performed to compute the upper triangular matrix T in each iteration, which adds $O(mnb)$ to the total cost of the QR factorization. This is computation that an unblocked algorithm does not perform. In return, the bulk of the computation is performed much faster, which in the balance benefits performance. Details can be found in, for example,

- [23] Thierry Joffrain, Tze Meng Low, Enrique S. Quintana-Orti, Robert van de Geijn, Field G. Van Zee, Accumulating Householder transformations, revisited, ACM Transactions on Mathematical Software, Vol. 32, No 2, 2006.

Casting the Rank-Revealing QR factorization, discussed in Subsection 4.5.2, in terms of matrix-matrix multiplications is trickier. In order to determine what column should be swapped at each step, the rest of the matrix has to be at least partially updated. One solution to this is to use a randomized algorithm, as discussed in

- [25] Per-Gunnar Martinsson, Gregorio Quintana-Orti, Nathan Heavner, Robert van de Geijn, Householder QR Factorization With Randomization for Column Pivoting (HQRRP), SIAM Journal on Scientific Computing, Vol. 39, Issue 2, 2017.

12.3.3.2 Optimizing reduction to condensed form

In Subsection 10.3.1 and Subsection 11.2.3, we discussed algorithms for reducing a matrix to tridiagonal and bidiagonal form, respectively. These are special cases of the reduction of a matrix to condensed form. The algorithms in those sections cast most of the computation in terms of matrix-vector multiplication and rank-1 or rank-2 updates, which are matrix-vector

operations that do not attain high performance. In enrichments in those chapters, we point to papers that cast some of the computation in terms of matrix-matrix multiplication. Here, we discuss the basic issues.

When computing the LU, Cholesky, or QR factorization, it is possible to factor a panel of columns before applying an accumulation of the transformations that are encountered to the rest of the matrix. It is this that allows the computation to be mostly cast in terms of matrix-matrix operations. When computing the reduction to tridiagonal or bidiagonal form, this is not possible. The reason is that if we have just computed a unitary transformation, this transformation must be applied to the rest of the matrix both from the left and from the right. What this means is that the next transformation to be computed depends on an update that involves the rest of the matrix. This in turn means that inherently a matrix-vector operation (involving the "rest of the matrix") must be performed at every step at a cost of $O(m^2)$ computation per iteration (if the matrix is $m \times m$). The insight is that $O(m^3)$ computation is cast in terms of matrix-vector operations, which is of the same order as the total computation.

While this is bad news, there is still a way to cast about half the computation in terms of matrix-matrix multiplication for the reduction to tridiagonal form. Notice that this means the computation is sped up by at most a factor two, since even if the part that is cast in terms of matrix-matrix multiplication takes no time at all relative to the rest of the computation, this only cuts the time to completion in half.

The reduction to bidiagonal form is trickier yet. It requires the fusing of a matrix-vector multiplication with a rank-1 update in order to reuse data that is already in cache.

Details can be found in, for example,

- [44] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, G. Joseph Elizondo, Families of Algorithms for Reducing a Matrix to Condensed Form, ACM Transactions on Mathematical Software (TOMS) , Vol. 39, No. 1, 2012.

12.3.3.3 Optimizing the implicitly shifted QR algorithm

Optimizing the QR algorithm for computing the Spectral Decomposition or Singular Value Decomposition gets even trickier. Part of the cost is in the reduction to condensed form, which we already have noted exhibits limited opportunity for casting computation in terms of matrix-matrix multiplication. Once the algorithm proceeds to the implicitly shifted QR algorithm, most of the computation is in the accumulation of the eigenvectors or singular vectors. In other words, it is in the application of the Givens' rotations from the right to the columns of a matrix Q in which the eigenvectors are being computed. Let us look at one such application to two columns, q_i and q_j :

$$\left(\begin{array}{c|c} q_i & q_j \end{array} \right) := \left(\begin{array}{c|c} q_i & q_j \end{array} \right) \left(\begin{array}{cc} \gamma & -\sigma \\ \sigma & \gamma \end{array} \right) = \left(\begin{array}{c|c} \gamma q_i + \sigma q_j & -\sigma q_i + \gamma q_j \end{array} \right).$$

The update of each column is a vector-vector operation, requiring $O(m)$ computation with $O(m)$ data (if the vectors are of size m). We have reasoned that for such an operation it is the cost of accessing memory that dominates. In

- [42] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, Restructuring the Tridiagonal and Bidiagonal QR Algorithms for Performance, ACM Transactions on Mathematical Software (TOMS), Vol. 40, No. 3, 2014.

we discuss how the rotations from many Francis Steps can be saved up and applied to Q at the same time. By carefully orchestrating this so that data in cache can be reused, the performance can be improved to rival that attained by a matrix-matrix operation.

12.3.4 Libraries for higher level dense linear algebra functionality

The Linear Algebra Package (LAPACK) is the most widely used interface for higher level linear algebra functionality like LU, Cholesky, and QR factorization and related solvers as well as eigensolvers. LAPACK was developed as an open source linear algebra software library which was then embraced as an implementation and/or interface by scientific software libraries that are supported by vendors.

A number of open source and vendors high-performance implementations of the BLAS interface are available. For example,

- The original open source LAPACK implementation [1] available from netlib at <http://www.netlib.org/lapack/>.
- Our `libflame` library is an open source implementation of LAPACK functionality that leverages a programming style that is illustrated in [Subsection 12.3.1](#) and [Subsection 12.3.2](#). It includes an LAPACK-compatable interface. It underlies AMD's [Optimizing CPU Libraries \(AOCL\)](#).
- Arm's [Arm Performance Libraries](#).
- Cray's [Cray Scientific and Math Libraries \(CSML\)](#).
- IBM's [Engineering and Scientific Subroutine Library \(ESSL\)](#).
- Intel's [Math Kernels Library \(MKL\)](#).
- NVIDIA's [cuBLAS](#).

12.3.5 Sparse algorithms

Iterative methods inherently perform few floating point operations relative to the memory operations that need to be performed. For example, the Conjugate Gradient Method discussed in [Section 8.3](#) typically spends most of its time in a sparse matrix-vector multiplication, where only two floating point operations are performed per nonzero element in the matrix. As a result, attaining high performance with such algorithms is inherently difficult.

A (free) text that gives a nice treatment of high performance computing, including sparse methods, is

- [17] Victor Eijkhout, Introduction to High-Performance Scientific Computing, lulu.com.
<http://pages.tacc.utexas.edu/~eijkhout/istc/istc.html>

12.4 Enrichments

12.4.1 BLIS and beyond

One of the strengths of the approach to implementing matrix-matrix multiplication described in [Subsection 12.2.4](#) is that it can be applied to related operations. A recent talk discusses some of these.

- Robert van de Geijn and Field Van Zee, "The BLIS Framework: Experiments in Portability," SIAM Conference on Parallel Processing for Scientific Computing (PP20). SIAM Activity group on Supercomputing Best Paper Prize talk. 2020. https://www.youtube.com/watch?v=1biep1Rh_08

12.4.2 Optimizing matrix-matrix multiplication - We've got a MOOC for that!

We reiterate that we offer a Massive Open Online Course titled "LAFF-On Programming for High Performance" in which we use matrix-matrix multiplication as an example through which we illustrate how high performance can be achieved on modern CPUs.

- [39] "[LAFF-On Programming for High Performance](#)", a four week Massive Open Online Course offered on the [edX platform](#) (free for auditors).

12.4.3 Deriving blocked algorithms - We've got a MOOC for that too!

Those who delve deeper into how to achieve high performance for matrix-matrix multiplication find out that it is specifically a rank-k update, the case of $C := \alpha AB + \beta C$ where the k (inner) size is small, that achieves high performance. The blocked LU factorization that we discussed in [Subsection 5.5.2](#) takes advantage of this by casting most of its computation in the matrix-matrix multiplication $A_{22} := A_{22} - A_{21}A_{12}$. A question becomes: how do I find blocked algorithms that cast most computation in terms of a rank-k updates?

The FLAME notation that we use in this course has made it possible for us to develop a systematic methodology for discovering (high-performance) algorithms. This was published in

- [4] Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Orti, Robert A. van de Geijn, The science of deriving dense linear algebra algorithms, ACM Transactions on Mathematical Software (TOMS), 2005.

and various other publications that can be found on the FLAME project publication web site <http://www.cs.utexas.edu/~flame/web/FLAMEPublications.html>.

You can learn these techniques, which derive algorithms hand-in-hand with their proofs of correctness, through our Massive Open Online Course

- [28] "[LAFF-On Programming for Correctness](#)", a six week Massive Open Online Course offered on the [edX platform](#) (free for auditors).

12.4.4 Parallel high-performance algorithms

Modern processors achieve high performance by extracting parallelism at the instruction level and by incorporating multiple cores that can collaborate to compute an operation. Beyond that, parallel supercomputers consist of computational nodes, each of which consists of multiple processing cores and local memory, that can communicate through a communication network.

Many of the issues encountered when mapping (dense) linear algebra algorithms to such distributed memory computers can be illustrated by studying matrix-matrix multiplication. A classic paper is

- [40] Robert van de Geijn and Jerrell Watts, SUMMA: Scalable Universal Matrix Multiplication Algorithm, *Concurrency: Practice and Experience*, Volume 9, Number 4, 1997.

The techniques described in that paper are generalized in the more recent paper

- [34] Martin D. Schatz, Robert A. van de Geijn, and Jack Poulson, Parallel Matrix Multiplication: A Systematic Journey, *SIAM Journal on Scientific Computing*, Volume 38, Issue 6, 2016.

12.5 Wrap Up

12.5.1 Additional homework

No additional homework yet.

12.5.2 Summary

We have noticed that typos are uncovered relatively quickly once we release the material. Because we "cut and paste" the summary from the materials in this week, we are delaying adding the summary until most of these typos have been identified.

Appendix A

Are you ready?

We have created a document "[Advanced Linear Algebra: Are You Ready?](#)" that a learner can use to self-assess their readiness for a course on numerical linear algebra.

Appendix B

Notation

B.0.1 Householder notation

Alston Householder introduced the convention of labeling matrices with upper case Roman letters (A, B , etc.), vectors with lower case Roman letters (a, b , etc.), and scalars with lower case Greek letters (α, β , etc.). When exposing columns or rows of a matrix, the columns of that matrix are usually labeled with the corresponding Roman lower case letter, and the individual elements of a matrix or vector are usually labeled with "the corresponding Greek lower case letter," which we can capture with the triplets $\{A, a, \alpha\}$, $\{B, b, \beta\}$, etc.

$$A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) = \left(\begin{array}{cc|cc|c} \alpha_{0,0} & & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & & \vdots & & \vdots \\ \alpha_{m-1,0} & & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{array} \right)$$

and

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{m-1} \end{pmatrix},$$

where α and χ is the lower case Greek letters "alpha" and "chi," respectively. You will also notice that in this course we start indexing at zero. We mostly adopt this convention (exceptions include i, j, p, m, n , and k , which usually denote integer scalars.)

Appendix C

Knowledge from Numerical Analysis

Typically, an undergraduate numerical analysis course is considered a prerequisite for a graduate level course on numerical linear algebra. There are, however, relatively few concepts from such a course that are needed to be successful in this course. In this appendix, we very briefly discuss some of these concepts.

C.0.1 Cost of basic linear algebra operations

C.0.2 Catastrophic cancellation

Recall that if

$$\chi^2 + \beta\chi + \gamma = 0$$

then the quadratic formula gives the largest root of this quadratic equation:

$$\chi = \frac{-\beta + \sqrt{\beta^2 - 4\gamma}}{2}.$$

Example C.0.2.1 We use the quadratic equation in the exact order indicated by the parentheses in

$$\chi = \left[\frac{[-\beta + [\sqrt{[\beta^2] - [4\gamma]}]]}{2} \right],$$

truncating every expression within square brackets to three significant digits, to solve

$$\chi^2 + 25\chi + \gamma = 0$$

$$\begin{aligned} \chi &= \left[\frac{[-25 + [\sqrt{[25^2] - [4]}]]}{2} \right] = \left[\frac{[-25 + [\sqrt{625 - 4}]]}{2} \right] \\ &= \left[\frac{[-25 + [\sqrt{621}]]}{2} \right] = \left[\frac{[-25 + 24.9]}{2} \right] = \left[\frac{-0.1}{2} \right] = -0.05. \end{aligned}$$

Now, if you do this to the full precision of a typical calculator, the answer is instead approximately -0.040064 . The relative error we incurred is, approximately, $0.01/0.04 = 0.25$.

What is going on here? The problem comes from the fact that there is error in the 24.9 that is encountered after the square root is taken. Since that number is close in magnitude, but of opposite sign to the -25 to which it is added, the result of $-25 + 24.9$ is mostly error.

This is known as catastrophic cancellation: adding two nearly equal numbers of opposite sign, at least one of which has some error in it related to roundoff, yields a result with large relative error.

Now, one can use an alternative formula to compute the root:

$$\chi = \frac{-\beta + \sqrt{\beta^2 - 4\gamma}}{2} = \frac{-\beta + \sqrt{\beta^2 - 4\gamma}}{2} \times \frac{-\beta - \sqrt{\beta^2 - 4\gamma}}{-\beta - \sqrt{\beta^2 - 4\gamma}},$$

which yields

$$\chi = \frac{2\gamma}{-\beta - \sqrt{\beta^2 - 4\gamma}}.$$

Carrying out the computations, rounding intermediate results, yields $-.0401$. The relative error is now $0.00004/0.040064 \approx .001$. It avoids catastrophic cancellation because now the two numbers of nearly equal magnitude are added instead. \square

Remark C.0.2.2 The point is: if possible, avoid creating small intermediate results that amplify into a large relative error in the final result.

Notice that in this example it is not inherently the case that a small relative change in the input is amplified into a large relative change in the output (as is the case when solving a linear system with a poorly conditioned matrix). The problem is with the standard formula that was used. Later we will see that this is an example of an unstable algorithm.

Appendix D

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <<http://www.fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE. The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS. This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING. You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY. If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated lo-

cation until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS. You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties — for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS. You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of

the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS. You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS. A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION. Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and

disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION. You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE. The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING. “Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents. To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with... Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

References

- [1] Ed Anderson, Zhaojun Bai, James Demmel, Jack J. Dongarra, Jeremy DuCroz, Ann Greenbaum, Sven Hammarling, Alan E. McKenney, Susan Ostrouchov, and Danny Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, 1992.
- [2] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM Press, 1993. [[PDF](#)]
- [3] Paolo Bientinesi, Inderjit S. Dhillon, Robert A. van de Geijn, *A Parallel Eigensolver for Dense Symmetric Matrices Based on Multiple Relatively Robust Representations*, SIAM Journal on Scientific Computing, 2005
- [4] Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Orti, Robert A. van de Geijn, *The science of deriving dense linear algebra algorithms*, ACM Transactions on Mathematical Software (TOMS), 2005.
- [5] Paolo Bientinesi, Enrique S. Quintana-Orti, Robert A. van de Geijn, *Representing linear algebra algorithms in code: the FLAME application program interfaces*, ACM Transactions on Mathematical Software (TOMS), 2005
- [6] Paolo Bientinesi, Robert A. van de Geijn, *Goal-Oriented and Modular Stability Analysis*, SIAM Journal on Matrix Analysis and Applications , Volume 32 Issue 1, February 2011.
- [7] Paolo Bientinesi, Robert A. van de Geijn, *The Science of Deriving Stability Analyses*, FLAME Working Note #33. Aachen Institute for Computational Engineering Sciences, RWTH Aachen. TR AICES-2008-2. November 2008.
- [8] Christian Bischof and Charles Van Loan, *The WY Representation for Products of Householder Matrices*, SIAM Journal on Scientific and Statistical Computing, Vol. 8, No. 1, 1987.
- [9] *Basic Linear Algebra Subprograms - A Quick Reference Guide*, University of Tennessee, Oak Ridge National Laboratory, Numerical Algorithms Group Ltd.
- [10] Barry A. Cipra, *The Best of the 20th Century: Editors Name Top 10 Algorithms*,

- SIAM News, Volume 33, Number 4, 2000. Available from <https://archive.siam.org/pdf/news/637.pdf>.
- [11] A.K. Cline, C.B. Moler, G.W. Stewart, and J.H. Wilkinson, *An estimate for the condition number of a matrix*, SIAM J. Numer. Anal., 16 (1979).
 - [12] Inderjit S. Dhillon and Beresford N. Parlett, *Multiple Representations to Compute Orthogonal Eigenvectors of Symmetric Tridiagonal Matrices*, Lin. Alg. Appl., Vol. 387, 2004.
 - [13] Jack J. Dongarra, Jeremy DuCroz, Ann Greenbaum, Sven Hammarling, Alan E. McKenney, Susan Ostrouchov, and Danny Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, 1992.
 - [14] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff, *A Set of Level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, Vol. 16, No. 1, pp. 1-17, March 1990.
 - [15] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson, *An Extended Set of {FORTRAN} Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, Vol. 14, No. 1, pp. 1-17, March 1988.
 - [16] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, 1979.
 - [17] Victor Eijkhout, *Introduction to High-Performance Scientific Computing*, lulu.com. <http://pages.tacc.utexas.edu/~eijkhout/istc/istc.html>
 - [18] Leslie V. Foster, *Gaussian elimination with partial pivoting can fail in practice*, SIAM Journal on Matrix Analysis and Applications, 15, 1994.
 - [19] Gene H. Golub and Charles F. Van Loan, *Matrix Computations, Fourth Edition*, Johns Hopkins Press, 2013.
 - [20] Brian C. Gunter, Robert A. van de Geijn, *Parallel out-of-core computation and updating of the QR factorization*, ACM Transactions on Mathematical Software (TOMS), 2005.
 - [21] N. Higham, *A Survey of Condition Number Estimates for Triangular Matrices*, SIAM Review, 1987.
 - [22] C. G. J. Jacobi, *Über ein leichtes Verfahren, die in der Theorie der Säkular-störungen vorkommenden Gleichungen numerisch aufzulösen*, Crelle's Journal 30, 51-94, 1846.
 - [23] Thierry Joffrain, Tze Meng Low, Enrique S. Quintana-Orti, Robert van de Geijn, Field G. Van Zee, *Accumulating Householder transformations, revisited*, ACM Transactions on Mathematical Software, Vol. 32, No 2, 2006.
 - [24] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Basic Linear Algebra Subprograms for Fortran Usage*, ACM Transactions on Mathematical Software, Vol. 5, No. 3, pp. 308-323, Sept. 1979.
 - [25] Per-Gunnar Martinsson, Gregorio Quintana-Orti, Nathan Heavner, Robert van de

- Geijn, *Householder QR Factorization With Randomization for Column Pivoting (HQRRP)*, SIAM Journal on Scientific Computing, Vol. 39, Issue 2, 2017.
- [26] Margaret E. Myers, Pierce M. van de Geijn, and Robert A. van de Geijn, *Linear Algebra: Foundations to Frontiers - Notes to LAFF With*, self-published at [ulaff.net](#), 2014.
 - [27] Margaret E. Myers and Robert A. van de Geijn, *Linear Algebra: Foundations to Frontiers*, [ulaff.net](#), 2014. A Massive Open Online Course offered on [edX](#).
 - [28] Margaret E. Myers and Robert A. van de Geijn, *LAFF-On Programming for Correctness*, self-published at [ulaff.net](#), 2017.
 - [29] Margaret E. Myers and Robert A. van de Geijn, *LAFF-On Programming for Correctness*, A Massive Open Online Course offered on [edX](#).
 - [30] J. Novembre, T. Johnson, K. Bryc, Z. Kutalik, A.R. Boyko, A. Auton, A. Indap, K.S. King, S. Bergmann, M.. Nelson, M. Stephens, C.D. Bustamante, *Genes mirror geography within Europe*, Nature, 2008
 - [31] Devangi N. Parikh, Margaret E. Myers, Richard Vuduc, Robert A. van de Geijn, *A Simple Methodology for Computing Families of Algorithms*, FLAME Working Note #87, The University of Texas at Austin, Department of Computer Science, Technical Report TR-18-06. [arXiv:1808.07832](#).
 - [32] C. Puglisi, *Modification of the Householder method based on the compact WY representation*, SIAM Journal on Scientific Computing, Vol. 13, 1992.
 - [33] Gregorio Quintana-Orti, Xioabai Sun, and Christof H. Bischof, *A BLAS-3 version of the QR factorization with column pivoting*, SIAM Journal on Scientific Computing, 19, 1998.
 - [34] Martin D. Schatz, Robert A. van de Geijn, and Jack Poulson, *Parallel Matrix Multiplication: A Systematic Journey*, SIAM Journal on Scientific Computing, Volume 38, Issue 6, 2016.
 - [35] Robert Schreiber and Charles Van Loan, *A Storage-Efficient WY Representation for Products of Householder Transformations*, SIAM Journal on Scientific and Statistical Computing, Vol. 10, No. 1, 1989.
 - [36] Jonathon Shlens, *A Tutorial on Principal Component Analysis*, [arxiv 1404.1100](#), 2014.
 - [37] G.W. Stewart, *Matrix Algorithms, Volume I: Basic Decompositions*, SIAM Press, 2001.
 - [38] Robert van de Geijn and Kazushige Goto, *BLAS (Basic Linear Algebra Subprograms)*, Encyclopedia of Parallel Computing, Part 2, pp. 157-164, 2011. If you don't have access, you may want to read an [advanced draft](#).
 - [39] Robert van de Geijn, Margaret Myers, and Devangi N. Parikh, *LAFF-On Programming for High Performance*, [ulaff.net](#), 2019.
 - [40] Robert van de Geijn and Jerrell Watts, *SUMMA: Scalable Universal Matrix Multiplication Algorithm*, Concurrency: Practice and Experience, Volume 9, Number 4, 1997.

- [41] Field G. Van Zee, *libflame: The Complete Reference*, <http://www.lulu.com>, 2009. [free PDF]
- [42] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, *Restructuring the Tridiagonal and Bidiagonal QR Algorithms for Performance*, ACM Transactions on Mathematical Software (TOMS), Vol. 40, No. 3, 2014. Available free from <http://www.cs.utexas.edu/~flame/web/FLAMEPublications.html> Journal Publication #33. Click on the title of the paper.
- [43] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, *Restructuring the Tridiagonal and Bidiagonal QR Algorithms for Performance*. ACM Transactions on Mathematical Software (TOMS) , 2014. Available free from <http://www.cs.utexas.edu/~flame/web/FLAMEPublications.html> Journal Publication #33. Click on the title of the paper.
- [44] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, G. Joseph Elizondo, *Families of Algorithms for Reducing a Matrix to Condensed Form*. ACM Transactions on Mathematical Software (TOMS) , Vol, No. 1, 2012. Available free from <http://www.cs.utexas.edu/~flame/web/FLAMEPublications.html> Journal Publication #26. Click on the title of the paper.
- [45] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM Journal on Scientific and Statistical Computing, Vol. 9, No. 1, 1988.
- [46] Stephen J. Wright, *A Collection of Problems for Which {G}aussian Elimination with Partial Pivoting is Unstable*, SIAM Journal on Scientific Computing, Vol. 14, No. 1, 1993.
- [47] *BLAS-like Library Instantiation Software Framework*, GitHub repository.
- [48] *BLIS typed interface*, <https://github.com/flame/blis/blob/master/docs/BLISTypedAPI.md>.
- [49] Kazushige Goto and Robert van de Geijn, *Anatomy of High-Performance Matrix Multiplication*, ACM Transactions on Mathematical Software, Vol. 34, No. 3: Article 12, May 2008.
- [50] Tyler Michael Smith, Bradley Lowery, Julien Langou, Robert A. van de Geijn, *A Tight I/O Lower Bound for Matrix Multiplication*, [arxiv.org:1702.02017v2](https://arxiv.org/abs/1702.02017v2), 2019. (To appear in ACM Transactions on Mathematical Software.)
- [51] Field G. Van Zee and Tyler M. Smith, *Implementing High-performance Complex Matrix Multiplication via the 3M and 4M Methods*, ACM Transactions on Mathematical Software, Vol. 44, No. 1, pp. 7:1-7:36, July 2017.
- [52] Field G. Van Zee and Robert A. van de Geijn, *BLIS: A Framework for Rapidly Instantiating BLAS Functionality*, ACM Journal on Mathematical Software, Vol. 41, No. 3, June 2015. You can access this article for free by visiting the [Science of High-Performance Computing group webpage](#) and clicking on the title of Journal Article 39.

Index

- (Euclidean) length, 84
 I , 43
 $[\cdot]$, 340
 ϵ_{mach} , 172
 $\text{fl}(\cdot)$, 339
 γ_n , 355, 379
 ∞ -norm (vector), 84
 ∞ -norm, vector, 30
 $\kappa(A)$, 76, 88
 $\max_i(\cdot)$, 286
 \overline{A} , 50
 \overline{x} , 95
 $\overline{}$, 19
 θ_j , 353, 379
 $| \cdot |$, 18
 e_j , 100
 p -norm (vector), 84
 p -norm, matrix, 55
 p -norm, vector, 31
1-norm (vector), 84
1-norm, vector, 29
2-norm (vector), 84
2-norm, matrix, 56
2-norm, vector, 25

absolute value, 18, 84
ACM, 341
Alternative Computational Model, 341
axpy, 296

backward stable implementation, 343
- Basic Linear Algebra Subprograms, 305, 588
BLAS, 305, 588
blocked algorithm, 200
catastrophic cancellation, 623
Cauchy-Schwarz inequality, 26
CGS, 158
characteristic polynomial, 471, 473
chasing the bulge, 546
Cholesky decomposition, 255
Cholesky factor, 299
Cholesky factorization, 222, 255
Cholesky factorization theorem, 299, 329
Classical Gram-Schmidt, 158
complex conjugate, 19
complex product, 84
condition number, 76, 88, 224, 250
conjugate, 19, 84
conjugate (of matrix), 87
conjugate (of vector), 84
conjugate of a matrix, 50
conjugate transpose (of matrix), 87
conjugate transpose (of vector), 84
consistent matrix norm, 71, 88
cost of basic linear algebra operations, 623
cubic convergence, 497, 513

defective matrix, 489, 511
deflation, 530
descent methods, 416
determinant, 472

- direction of maximal magnification, 77
distance, 18
dot product, 84, 95
- eigenpair, 464, 507
eigenvalue, 464, 507
eigenvector, 464, 507
- elementary elementary pivot matrix, 241
equivalence style proof, 22
- Euclidean distance, 18
exact descent method, 420
- fill-in, 393
fixed-point equation, 405
- FLAME notation, 118
- floating point numbers, 334
flop, 594, 595, 597
- forward substitution, 260
- Frobenius norm, 47, 87
- Gauss transform, 273
Gaussian elimination, 258
Gaussian elimination with row exchanges, 278
Geometric multiplicity, 489, 511
Givens' rotation, 539
gradient, 417
Gram-Schmidt orthogonalization, 158
- Hermitian, 50
Hermitian Positive Definite, 222, 297
Hermitian positive definite, 297
Hermitian transpose, 26, 49
Hermitian transpose (of matrix), 87
Hermitian transpose (of vector), 84
Hessenberg matrix, 542
homogeneity (of absolute value), 19
homogeneity (of matrix norm), 46, 86
homogeneity (of vector norm), 24, 84
Householder reflector, 179, 208
Householder transformation, 178, 179, 208
HPD, 222, 297
- identity matrix, 43
Implicit Q Theorem, 542
- induced matrix norm, 51, 52
infinity norm, 30
inner product, 84, 95
- Jordan Canonical Form, 488
- Krylov subspace, 445, 458
- left pseudo inverse, 220
left pseudo-inverse, 90
left singular vector, 118, 149
Legendre polynomials, 154
linear convergence, 496, 497, 512
linear least squares, 212
linear transformation, 41
LLS, 212
- LU decomposition, 255, 268, 277, 324
LU factorization, 255, 258, 268, 277, 324
LU factorization - existence, 269, 324
LU factorization algorithm (bordered), 273
LU factorization algorithm (left-looking), 271
LU factorization algorithm (right-looking), 264
LU factorization with complete pivoting, 296
LU factorization with partial pivoting, 286
LU factorization with partial pivoting (right-looking algorithm), 286
LU factorization with pivoting, 278
- machine epsilon, 172, 337, 338, 377
magnitude, 18
matrix, 41, 43
matrix 1-norm, 87
matrix 2-norm, 56, 87
matrix ∞ -norm, 87
matrix p -norm, 55
matrix norm, 46, 86
matrix norm, 2-norm, 56
matrix norm, p -norm, 55
matrix norm, consistent, 71, 88
matrix norm, Frobenius, 47
matrix norm, induced, 51, 52

- matrix norm, submultiplicative, 70, 71, 88
matrix norm, subordinate, 71, 88
matrix p-norm, 87
matrix-vector multiplication, 43
memop, 594, 595, 598
Method of Multiple Relatively Robust Representations (MRRR), 553
Method of Normal Equations, 219
method of normal equations, 214
MRRR, 553

natural ordering, 384
nested dissection, 396
norm, 12
norm, Frobenius, 47
norm, infinity, 30
norm, matrix, 46, 86
norm, vector, 24, 84
normal equations, 214, 219
numerical stability, 331

orthogonal matrix, 102
orthogonal projection, 90
orthogonal vectors, 96
orthonormal matrix, 100
orthonormal vectors, 99
over-relaxation, 409

parent functions, 153
partial pivoting, 279, 286
pivot, 279
pivot element, 279
positive definite, 297
positive definiteness (of absolute value), 19
positive definiteness (of matrix norm), 46, 86
positive definiteness (of vector norm), 24, 84
precondition, 308
principal leading submatrix, 268, 324
pseudo inverse, 220, 223
pseudo-inverse, 90

QR algorithm, 521
QR decomposition, 151
QR Decomposition Theorem, 161, 207
QR factorization, 151
QR factorization with column pivoting, 241
quadratic convergence, 497, 513

Rank Revealing QR, 241
rank-k update, 612
Rayleigh quotient, 496, 512
Rayleigh Quotient Iteration, 503
reflector, 178, 179, 208
residual, 14
right pseudo inverse, 221
right singular vector, 118, 149
rotation, 107
row pivoting, 279
RRQR, 241

Schur decomposition, 479, 480, 511
Schur Decomposition Theorem, 480, 511
SCM, 340
separator, 393
shifted inverse power method, 503
shifted QR algorithm, 526
similarity transformation, 479, 510
Singular Value Decomposition, 89, 92
singular vector, 118, 149
solving triangular systems, 290
SOR, 409
sparse linear system, 382
Spectral decomposition, 479
spectral decomposition, 482, 511
Spectral Decomposition Theorem, 482, 511
spectral radius, 465, 508
spectrum, 465, 508
stability, 331
standard basis vector, 42, 85
Standard Computational Model, 340
submultiplicative matrix norm, 70, 71, 88
subordinate matrix norm, 71, 88
subspace iteration, 514, 518
successive over-relaxation, 409

- superlinear convergence, 497, 513
superquadratic convergence, 513
SVD, 89, 92
symmetric positive definite, 298, 329

tall and skinny, 563
The Francis implicit QR Step, 545
The implicit Q theorem, 542
transpose, 49
transpose (of matrix), 87
transpose (of vector), 84
triangle inequality (for absolute value)), 19
triangle inequality (for matrix norms)), 46, 86
triangle inequality (for vector norms)), 24, 84
triangular solve with multiple right-hand sides, 612
triangular system, 290
TRSM, 612

unit ball, 32, 84
unit roundoff, 337, 338, 377
unit roundoff error, 172
unitary matrix, 102, 148
unitary similarity transformation, 480, 511
upper Hessenberg matrix, 542

Vandermonde matrix, 152
vector 1-norm, 29, 84
vector 2-norm, 25, 84
vector ∞ -norm, 30, 84
vector p -norm, 84
vector p -norm, 31
vector norm, 24, 84
vector norm, 1-norm, 29
vector norm, 2-norm, 25
vector norm, ∞ -norm, 30
vector norm, p -norm, 31

Wilkinson shift, 550

Colophon

This article was authored in, and produced with, PreTeXt.