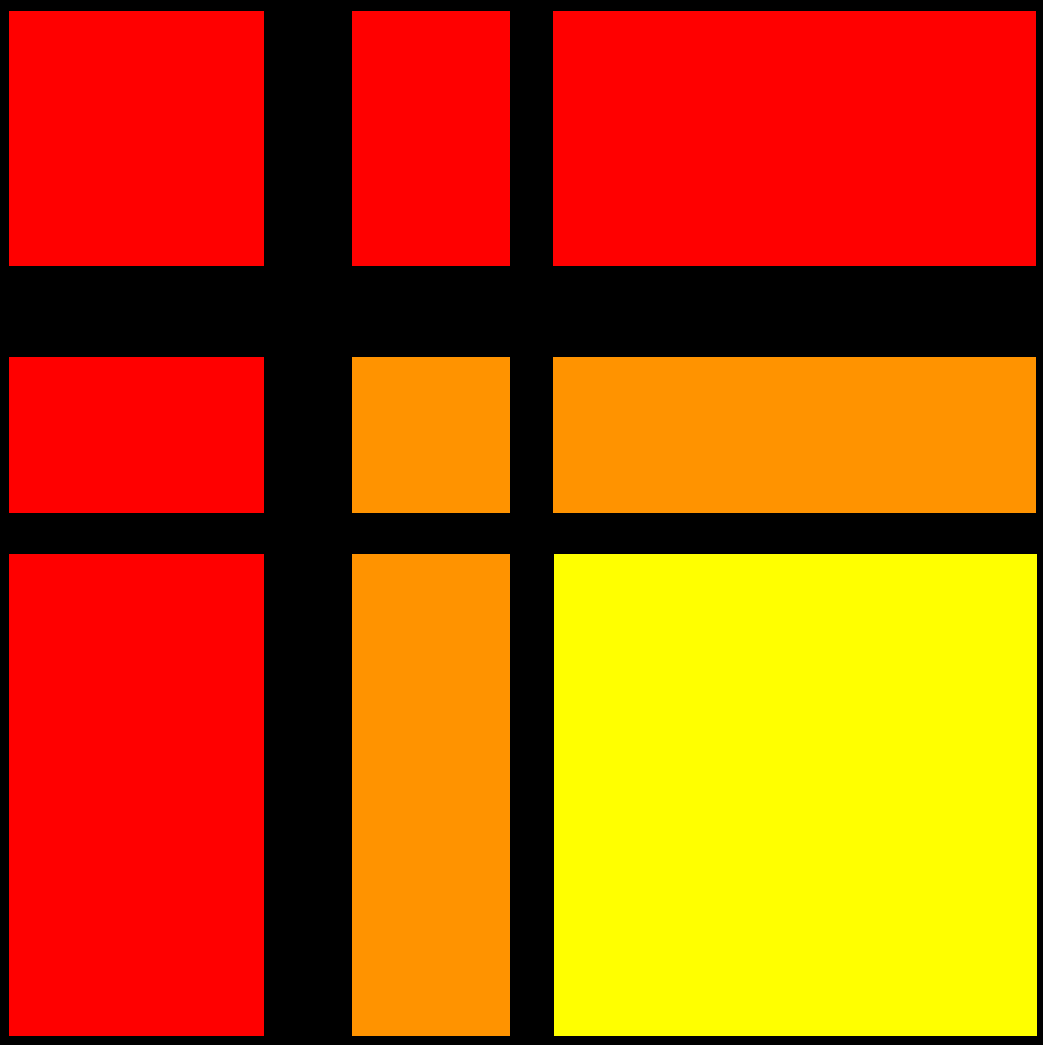
Advanced Linear Algebra
Foundations to Frontiers



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 21, 2020

Edition: Draft Edition 2019–2020

Website: ulaff.net

Acknowledgements

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

Preface

"Advanced Linear Algebra: Foundations to Frontiers" (ALAFF) is an alternative to a traditional text for a graduate course on numerical linear algebra. It intertwines text, videos, exercises, and programming activities in consumable chunks in an effort to keep the learner engaged.

We have used these materials in different settings. It is the primary resource for our course at UT-Austin titled "Numerical Analysis: Linear Algebra" offered through the departments of Computer Science, Mathematics, Statistics and Data Sciences, and Mechanical Engineering, as well as the Computational Science, Engineering, and Mathematics graduate program. This course is also offered as "Advanced Linear Algebra for Computing" through the UT-Austin Masters in Computer Science Online program. Finally, it is the basis for the Massive Open Onine Course (MOOC) titled "Advanced Linear Algebra: Foundations to Frontiers" on the edX platform. It is our hope that others will repurpose ALAFF for other learning settings, either in its entirety or in part.

So as not to overwhelm learners, we have taken the traditional topics of a numerical linear algebra course and organized these into three parts. Orthogonality, Solving Linear Systems, and the Algebraic Eigenvalue Problem.

- Part I: Orthogonality explores orthogonality (which includes a treatment of norms, orthogonal spaces, the Singular Value Decomposition (SVD), and solving linear least squares problems). We start with with these topics since they are prerequisite knowledge for other courses that students often pursue in parallel with (or even before) advanced linear algebra.
- Part II: Solving Linear Systems focuses on so-called direct and iterative methods while also introducing the notion of numerical stability, which quantifies and qualifies how error that is introduced in the original statement of the problem and/or roundoff that occurs in computer arithmetic impacts the correctness of a computation.
- Part III: The Algebraic Eigenvalue Problem focuses on the theory and practice of computing the eigenvalues and eigenvectors of a matrix. This is closely related to the diagonazing a matrix. Practical algorithms for solving the eigenvalue problem are extended so they can be used to compute the SVD. This part, and the course, ends with a discussion of how to achieve high performance on modern computers when performing matrix computations.

While this represents only a selection of advanced topics in linear algebra, we believe that this course leaves you equipped to pursue further related subjects.

ALAFF is part of a collection of learning artifacts that we have developed over the years.

- Linear Algebra: Foundations to Frontiers (LAFF) [26] [27] a full semester undergraduate introduction to linear algebra. For those whose linear algebra fluency is a bit rusty, this is a good resource for brushing up.
- LAFF-On Programming for Correctness [28] [29] is a six-week course that shares our techniques for systematic discovery of families of algorithms for matrix operations from which the best (e.g., highest performing) can be chosen in a context.

- LAFF-On Programming for High Performance [40] (((Unresolved xref, reference "biblio-pfhp-edX"; check spelling or use "provisional" attribute))) is a four-week course in which matrix-matrix multiplication is used to illustrate fundamental techniques for achieving high performance on modern CPUs. In Week 12 of ALAFF, we give you a flavor of how high performance can be achieved for matrix computations.

There is a MOOC on edX associated with each of these materials. Together, they form a loosely-coupled learning experience.

You should use the pretest we have created, "Advanced Linear Algebra: Are You Ready?", [39] to self-assess whether you are ready for ALAFF. It consists of about a dozen questions. When taking it, realize that it is not about whether you can answer those questions. Rather, you should look carefully look at the solutions to the questions, which discuss how some of the more concrete exercises translate to more abstract insights. How the topic of the question fits into ALAFF is discussed as is where to go to review related knowledge.

Robert van de Geijn
Maggie Myers
Austin, 2020

Contents

Week 0

Getting Started

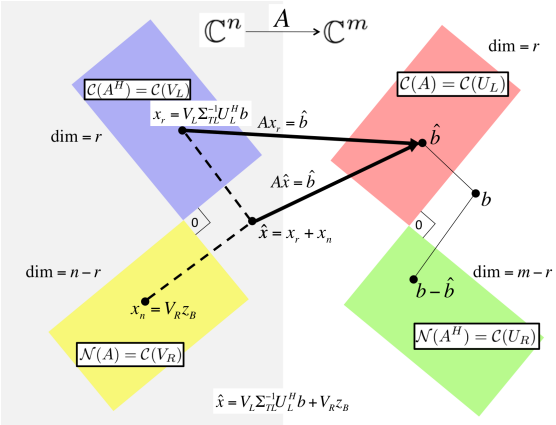
0.1 Opening Remarks

0.1.1 Welcome

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

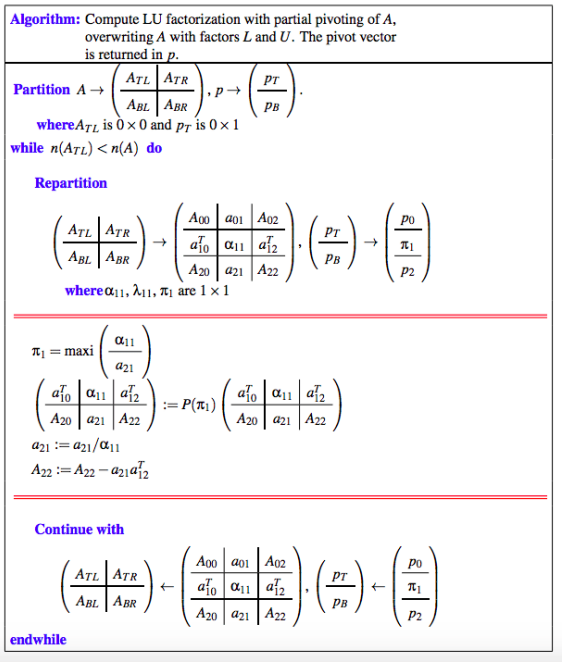
Linear algebra is one of the fundamental tools for computational and data scientists. In Advanced Linear Algebra: Foundations to Frontiers (ALAFF), you build your knowledge, understanding, and skills in linear algebra, practical algorithms for matrix computations, and how floating-point arithmetic, as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

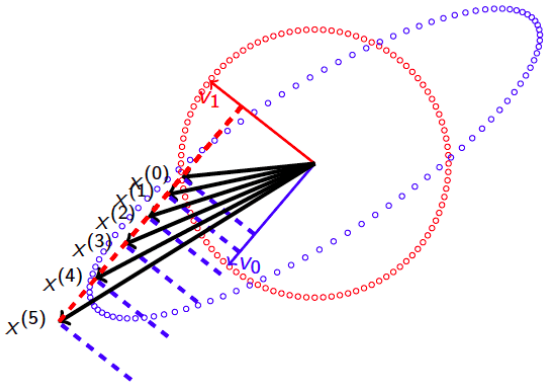
The materials are organized into Weeks that correspond to a chunk of information that is covered in a typical on-campus week. These weeks are arranged into three parts:



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.





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:"

0.1.3 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 materals 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.

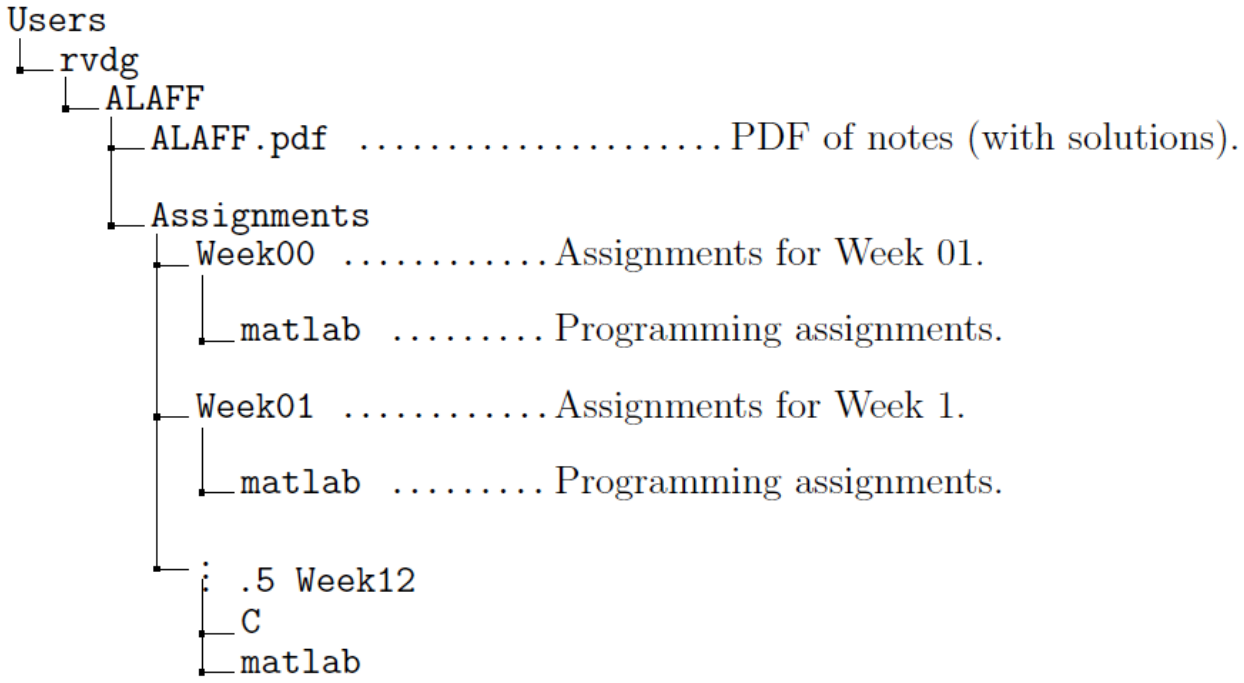


Figure 0.2.2.1 Directory structure for your ALAFF materials. In this example, we cloned the repository in Robert's home directory, rvdg.

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.

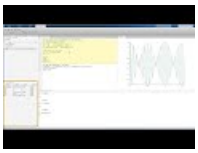
You need relatively little familiarity with MATLAB in order to learn what we want you to learn in this course. So, you could just skip these tutorials altogether, and come back to them if you find you want to know more about MATLAB and its programming language (M-script).

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

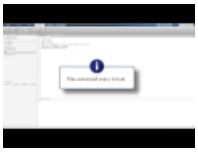
MATLAB Variables

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 encounted 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

  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 you mac. Warning: you may need "root" privileges to do so.

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

  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 --prefix=$HOME/libflame
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 Mathermatics, 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 Analysisat 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 PrincetonUniversity 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 $U\widehat{x} = b$ should leave us with a vector $\widehat{x}$ such

that $x = \widehat{x}$.

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

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

`b = U * x;`	Compute right-hand side b from known solution x.
`xhat = U \ b;`	Solve $U\widehat{x} = b$.
`xhat - x`	Report the difference between $\widehat{x}$ and x.

What do we notice?

Next, check how close $U\widehat{x}$ 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_so

Some things we observe:

- $\widehat{x} - x$ 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 $\widehat{x} - x$ is around 10^{-15}.
- The residual $b - U\widehat{x}$ 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 $\widehat{x} - x$ 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 $U\widehat{x} = b$
Report the Euclidean length of the difference between $\widehat{x}$ and x.

What do we notice?
Next, check how close $U\widehat{x}$ 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_so
Some things we observe:

- $norm(\widehat{x} - x)$, the Euclidean length of $\widehat{x} - x$, is huge. Matlab computed the wrong answer!
- However, the computed $\widehat{x}$ solves a problem that corresponds to a slightly different right-hand side. Thus, $\widehat{x}$ 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 xhat 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? (min(abs(diag(U))) returns it!)
- If U were singular, how many solutions to $U\widehat{x} = b$ would there be? How can we characterize them?
- What is the relationship between $\widehat{x} - x$ 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? (min(abs(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 $U\widehat{x} = b$ 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 $\widehat{x} - x$ 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

1.1.3 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!

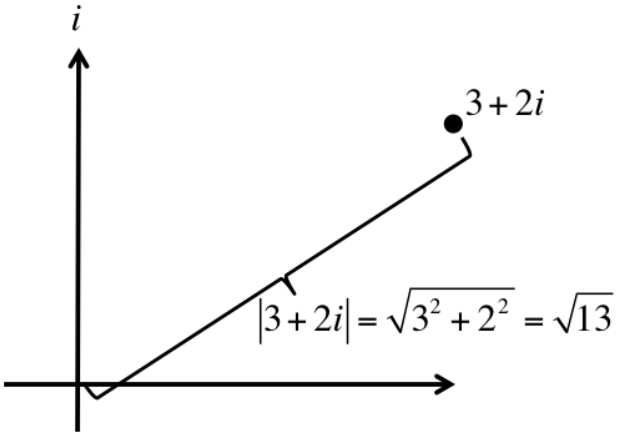
In this course, we mostly allow scalars, vectors, and matrices to be complex-valued. This means we will use terms like "conjugate" and "Hermitian" quite liberally. You will think this is a big deal, but actually, if you just focus on the real case, you will notice that the complex case is just a natural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

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

Recall that $|\cdot| : \mathbb{C} \to \mathbb{R}$ 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

$$\begin{array}{l} |\alpha| \\ \quad = \\ \sqrt{\alpha_r^2 + \alpha_c^2} \\ \quad = \\ \sqrt{\alpha_r^2 - \alpha_c\alpha_r i + \alpha_r\alpha_c i + \alpha_c^2} \\ \quad = \\ \sqrt{(\alpha_r - \alpha_c i)(\alpha_r + \alpha_c i)} \\ \quad = \\ \sqrt{\overline{\alpha}\alpha} \quad , \end{array}$$

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

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

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:

   $\alpha\beta$
   $=$ $<$ substitute $>$
   $(\alpha_r + \alpha_c i)(\beta_r + \beta_c i)$
   $=$ $<$ multiply out $>$
   $\alpha_r\beta_r + \alpha_r\beta_c i + \alpha_c\beta_r i - \alpha_c\beta_c$
   $=$ $<$ commutativity of real multiplication $>$
   $\beta_r\alpha_r + \beta_r\alpha_c i + \beta_c\alpha_r i - \beta_c\alpha_c$
   $=$ $<$ factor $>$
   $(\beta_r + \beta_c i)(\alpha_r + \alpha_c i)$
   $=$ $<$ substitute $>$
   $\beta\alpha$.

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

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. ALWAYS

Now prove it!

Solution 1.

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

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:

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

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

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

At that point, you recognize that

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

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:

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

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:

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

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

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

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

Homework 1.2.1.4 Let $\alpha \in \mathbb{C}$.
ALWAYS/SOMETIMES/NEVER: $\overline{\alpha}\alpha \in \mathbb{R}$

Answer. ALWAYS.
Now prove it!

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

$$
\begin{array}{l}
\overline{\alpha}\alpha \\
\quad = \quad < \text{ instantiate } > \\
\overline{(\alpha_r + \alpha_c i)}(\alpha_r + \alpha_c i) \\
\quad = \quad < \text{ conjugate } > \\
(\alpha_r - \alpha_c i)(\alpha_r + \alpha_c i) \\
\quad = \quad < \text{ multiply out } > \\
\alpha_r^2 + \alpha_c^2,
\end{array}
$$

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.

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

Homework 1.2.1.6 Let $\alpha \in \mathbb{C}$.
ALWAYS/SOMETIMES/NEVER: $|\overline{\alpha}| = |\alpha|$.

Answer. ALWAYS
Now prove it!

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

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

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 $\nu : \mathbb{C}^m \to \mathbb{R}$. 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).

$\Diamond$

Homework 1.2.2.1 TRUE/FALSE: If $\nu : \mathbb{C}^m \to \mathbb{R}$ 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

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

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=bxDDpUZEqBs

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

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

◊

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

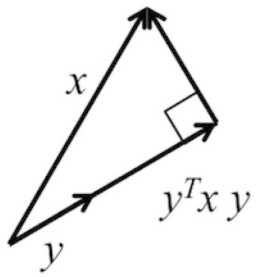
$$x = \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_m \end{pmatrix}$$

then the row vector

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

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

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

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 $\widehat{x} = x/\|x\|_2$ and $\widehat{y} = y/\|y\|_2$. This leaves us to prove that $|\widehat{x}^H \widehat{y}| \leq 1$ since $\|\widehat{x}\|_2 = \|\widehat{y}\|_2 = 1$.

Pick

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

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

$$\begin{array}{l} \alpha \widehat{x}^H \widehat{y} \\ \quad = \quad < \beta = \overline{\beta} \text{ if } \beta \text{ is real } > \\ \overline{\alpha \widehat{x}^H \widehat{y}} \\ \quad = \quad < \text{ property of complex conjugation } > \\ \overline{\alpha} \widehat{y}^H \widehat{x} \end{array}.$$

Now,

$$\begin{array}{l} 0 \\ \quad \leq \quad < \|\cdot\|_2 \text{ is nonnegative definite } > \\ \|\widehat{x} - \alpha \widehat{y}\|_2^2 \\ \quad = \quad < \|z\|_2^2 = z^H z > \\ (\widehat{x} - \alpha \widehat{y})^H (\widehat{x} - \alpha \widehat{y}) \\ \quad = \quad < \text{ multiplying out } > \\ \widehat{x}^H \widehat{x} - \overline{\alpha} \widehat{y}^H \widehat{x} - \alpha \widehat{x}^H \widehat{y} + \overline{\alpha} \alpha \widehat{y}^H \widehat{y} \\ \quad = \quad < \text{ above assumptions and observations } > \\ 1 - 2\alpha \widehat{x}^H \widehat{y} + |\alpha|^2 \\ \quad = \quad < \alpha \widehat{x}^H \widehat{y} = |\widehat{x}^H \widehat{y}|; |\alpha| = 1 > \\ 2 - 2|\widehat{x}^H \widehat{y}|. \end{array}$$

Thus $|\widehat{x}^H \widehat{y}| \leq 1$ 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):

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

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

  $$\begin{array}{ll} & \|x + y\|_2^2 \\ = & < \|z\|_2^2 = z^H z > \\ & (x + y)^H (x + y) \\ = & < \text{ distribute } > \\ & x^H x + y^H x + x^H y + y^H y \\ = & < \overline{\beta} + \beta = 2\text{Real}(\beta) > \\ & x^H x + 2\text{Real}(x^H y) + y^H y \\ \leq & < \text{ algebra } > \\ & x^H x + 2|\text{Real}(x^H y)| + y^H y \\ \leq & < \text{ algebra } > \\ & x^H x + 2|x^H y| + y^H y \\ \leq & < \text{ algebra; Cauchy-Schwarz } > \\ & \|x\|_2^2 + 2\|x\|_2\|y\|_2 + \|y\|_2^2 \\ = & < \text{ algebra } > \\ & (\|x\|_2 + \|y\|_2)^2. \end{array}$$

  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:
$$x = \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{M-1} \end{array} \right).$$

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

Answer. ALWAYS

Now prove it!

Solution.

$$
\begin{array}{l}
\|x\|_2^2 \\
\quad = \quad < \text{ partition vector } > \\
\left\| \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{M-1} \end{array} \right) \right\|_2^2 \\
\quad = \quad < \text{ equivalent definition } > \\
\left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{M-1} \end{array} \right)^H \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{M-1} \end{array} \right) \\
\quad = \quad < \text{ dot product of partitioned vectors } > \\
x_0^H x_0 + x_1^H x_1 + \cdots + x_{M-1}^H x_{M-1} \\
\quad = \quad < \text{ equivalent definition } > \\
\|x_0\|_2^2 + \|x_1\|_2^2 + \cdots + \|x_{M-1}\|_2^2 \\
\quad \geq \quad < \text{ algebra } > \\
\|x_i\|_2^2
\end{array}
$$

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, $\|\cdot\|_1 : \mathbb{C}^m \to \mathbb{R}$, is defined for $x \in \mathbb{C}^m$ by

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

$\Diamond$

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

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

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

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

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

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

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, $\|\cdot\|_\infty : \mathbb{C}^m \to \mathbb{R}$, is defined for $x \in \mathbb{C}^m$ by

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

$\Diamond$

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):

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

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

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

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 $\|\cdot\|_p : \mathbb{C}^m \to \mathbb{R}$ 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:

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

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

1.2.5 Unit ball

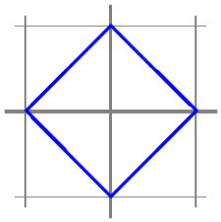
YouTube: https://www.youtube.com/watch?v=aJgrpp7uscw

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:

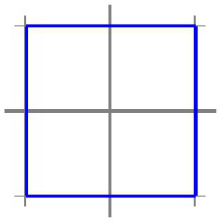
Definition 1.2.5.1 Unit ball. Given norm $\|\cdot\| : \mathbb{C}^m \to \mathbb{R}$, the unit ball with respect to $\|\cdot\|$ is the set $\{x \mid \|x\| = 1\}$ (the set of all vectors with norm equal to one). We will use $\|x\| = 1$ as shorthand for $\{x \mid \|x\| = 1\}$. ◊

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:

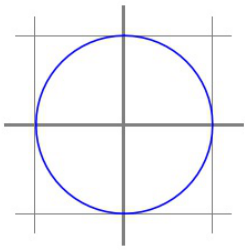
(a) $\|x\|_2 = 1$. (1)



(b) $\|x\|_1 = 1$. (2)

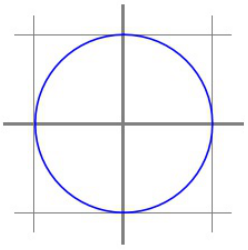


(c) $\|x\|_\infty = 1$. (3)

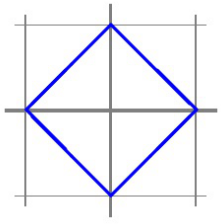


Solution.

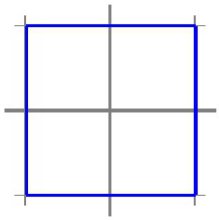
(a) $\|x\|_2 = 1$. (3)



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



(c) $\|x\|_\infty = 1$. (2)



YouTube: https://www.youtube.com/watch?v=Ov77sE90P58

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 $\|\cdot\| : \mathbb{C}^m \to \mathbb{R}$ and $|||\cdot||| : \mathbb{C}^m \to \mathbb{R}$ 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 arbitary vector. W.l.o.g. assume that $x \neq 0$. Then

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

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

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

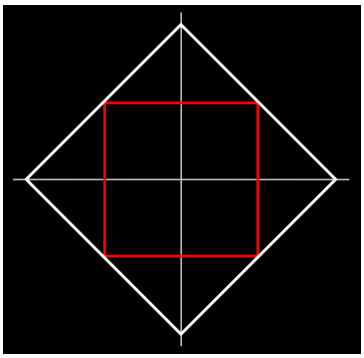
We conclude that

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

where $\sigma = 1/\rho$.

Example 1.2.6.2

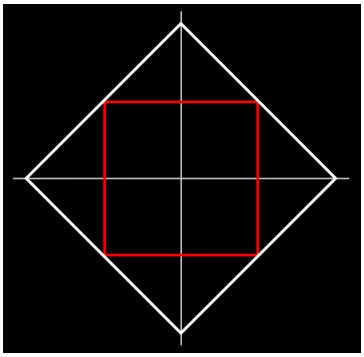
- Let $x \in \mathbb{R}^2$. Use the picture



  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 $x \in \mathbb{R}^2$:

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

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

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

  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 $x \in \mathbb{R}^2$. □

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

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

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

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

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

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

Consider

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

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

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

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

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=xlkiZEbYh38

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 $L : \mathbb{C}^n \to \mathbb{C}^m$. 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 $F : \mathbb{C}^n \to \mathbb{C}^m$ 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 $L : \mathbb{C}^n \to \mathbb{C}^m$ be a linear transformation, $v_0, v_1, \cdots, v_{k-1} \in \mathbb{C}^n$, and $x \in \mathbb{C}^k$. Then*

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

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,

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

◊

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 $\mathbb{C}^n$:

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

Hence, if L is a linear transformation,

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

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:

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

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

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

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:

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

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

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

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:

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

and

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

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:

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

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

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

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

   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).

$\Diamond$

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

$$\begin{array}{ll} \nu(0) & \\ \quad = & < 0 \cdot A = 0 > \\ \nu(0 \cdot A) & \\ \quad = & < \| \cdot \|_\nu \text{ is homogeneous } > \\ 0\nu(A) & \\ \quad = & < \text{ algebra } > \\ 0 & \end{array}$$

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

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

◊

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:

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

Show that

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

Solution.

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

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

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

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:

$$
A = \left(\begin{array}{c} \widetilde{a}_0^T \\ \hline \vdots \\ \hline \widetilde{a}_{m-1}^T \end{array} \right).
$$

Show that

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

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

Solution.

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

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

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

then its **transpose** is defined by

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

◊

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

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

then its **Hermitian transpose** is defined by

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

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

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.

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

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

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

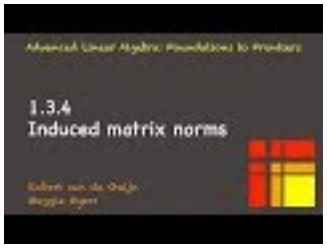
Solution.

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

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, $L : \mathbb{C}^n \to \mathbb{C}^m$, 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 $\|\cdot\|_\mu : \mathbb{C}^m \to \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \to \mathbb{R}$ be vector norms. Define $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m\times n} \to \mathbb{R}$ 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:

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

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 $\|\cdot\|_\mu : \mathbb{C}^m \to \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \to \mathbb{R}$ be vector norms. Define $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m\times n} \to \mathbb{R}$ 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 $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m\times n} \to \mathbb{R}$ *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 jth column, a_j, that is nonzero. Let e_j equal the column of I (the identity matrix) indexed with j. Then

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

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

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

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

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

■

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.

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

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

$\|\cdot\|_p : \mathbb{C}^{m\times n} \to \mathbb{R}$ 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 $\|\cdot\|_2 : \mathbb{C}^{m\times n} \to \mathbb{R}$ by

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

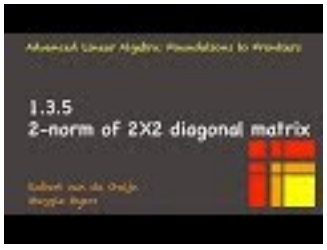
◊

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 $f : D \to R$ 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 $\delta_0, \ldots, \delta_{m-1}$. Show that

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

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

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

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

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

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 $\frac{\|yx^Hz\|_2}{\|z\|_2} = \|y\|_2\|x\|_2$.

Answer. ALWAYS
Now prove it!

Solution. W.l.o.g. assume that $x \neq 0$.
We know by the Cauchy-Schwarz inequality that $|x^Hz| \leq \|x\|_2\|z\|_2$. Hence

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

But also

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

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.

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

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

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

Hint. Proving $\|A\|_2 = \max_{\|x\|_2=\|y\|_2=1} |y^HAx|$ requires you to invoke the Cauchy-Schwarz inequality from

Theorem 1.2.3.3.

Solution.

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

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

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

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

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

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

$$
\begin{array}{ll}
\|A^H\|_2 & \\
\quad = & < \text{ first part of homework } > \\
\max_{\|x\|_2=\|y\|_2=1} |y^H A^H x| & \\
\quad = & < |\overline{\alpha}| = |\alpha| > \\
\max_{\|x\|_2=\|y\|_2=1} |x^H Ay| & \\
\quad = & < \text{ first part of homework } > \\
\|A\|_2. &
\end{array}
$$

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

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

  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 $\|\cdots\|_2$ 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:

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

With this, we can then show that

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

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

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

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

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

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

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

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 $A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$. 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 $\max_{0 \le j < n} \|a_j\|_1 = \|a_J\|_1$. Then

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

Also,

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

Hence

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

which implies that

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

Homework 1.3.6.2 Let $A \in \mathbb{C}^{m \times n}$ and partition $A = \left(\begin{array}{c} \widetilde{a}_0^T \\ \hline \widetilde{a}_1^T \\ \hline \vdots \\ \hline \widetilde{a}_{m-1}^T \end{array} \right)$.

ALWAYS/SOMETIMES/NEVER:

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

Notice that in this exercise $\widetilde{a}_i$ is really $(\widetilde{a}_i^T)^T$ since $\widetilde{a}_i^T$ is the label for the ith row of matrix A.

Hint. Prove it for the real valued case first.

Answer. ALWAYS

Solution. Partition $A = \left(\begin{array}{c} \hline \widetilde{a}_0^T \\ \hline \vdots \\ \hline \widetilde{a}_{m-1}^T \\ \hline \end{array} \right)$. Then

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

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

We also want to show that $\| A \|_\infty \geq \max_{0 \leq i < m} \| \widetilde{a}_i \|_1$. Let k be such that $\max_{0 \leq i < m} \| \widetilde{a}_i \|_1 = \| \widetilde{a}_k \|_1$ and pick $y = \left(\begin{array}{c} \psi_0 \\ \vdots \\ \psi_{n-1} \end{array} \right)$ so that $\widetilde{a}_k^T y = | \alpha_{k,0} | + | \alpha_{k,1} | + \cdots + | \alpha_{k,n-1} | = \| \widetilde{a}_k \|_1$. (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

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

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 $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ and $|||\cdot||| : \mathbb{C}^{m\times n} \to \mathbb{R}$ 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 arbitary matrix. W.l.o.g. assume that $A \neq 0$ (the zero matrix). Then

$$\begin{array}{l} |||A||| \\ \quad = \quad < \text{ algebra } > \\ \frac{|||A|||}{\|A\|}\|A\| \\ \quad \leq \quad < \text{ definition of suppremum } > \\ \left(\sup_{Z\neq 0} \frac{|||Z|||}{\|Z\|}\right)\|A\| \\ \quad = \quad < \text{ homogeneity } > \\ \left(\sup_{Z\neq 0} |||\frac{Z}{\|Z\|}|||\right)\|A\| \\ \quad = \quad < \text{ change of variables } B = Z/\|Z\| > \\ \left(\sup_{\|B\|=1} |||B|||\right)\|A\| \\ \quad = \quad < \text{ the set } \|B\| = 1 \text{ is compact } > \\ \left(\max_{\|B\|=1} |||B|||\right)\|A\| \end{array}$$

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 $\Sigma = \text{diag}(\sigma_0, \ldots, \sigma_{\min(m,n)})$ with $\sigma_0 \geq \sigma_1 \geq \ldots \geq \sigma_{\min(m,n)} \geq 0$. Then

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

and

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

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.

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

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

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

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

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

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

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

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

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

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

Equality is attained for $A = \left(\begin{array}{c|c|c|c} 1 & 1 & \cdots & 1 \end{array} \right)$.

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

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

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, $\|\cdots\| : \mathbb{C}^{m\times n} \to \mathbb{R}$, there are vector norms $\|\cdot\|_\mu : \mathbb{C}^m \to \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \to \mathbb{R}$ 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 $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ is said to be subordinate to vector norms $\|\cdot\|_\mu : \mathbb{C}^m \to \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \to \mathbb{R}$ 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. ◊

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, $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m\times n} \to \mathbb{R}$, 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 $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ 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. ◊

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

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

◊

Theorem 1.3.8.6 *Let $\|\cdot\| : \mathbb{C}^n \to \mathbb{R}$ 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 $\|\cdot\| : \mathbb{C}^n \to \mathbb{R}$ be a vector norm defined for all n and let $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ 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.

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

■

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.

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

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

Solution.

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

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 $\widetilde{a}_i^T$ leads to some possible confusion. In this particular situation, it is best to think of $\widetilde{a}_i$ as a vector that, when transposed, becomes the row of A indexed with i. In this case, $\widetilde{a}_i^T = \overline{\widetilde{a}_i}^H$ and $(\widetilde{a}_i^T)^H = \overline{\widetilde{a}_i}$ (where, recall, $\overline{x}$ 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

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

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

Answer. ALWAYS

Now prove it!

Solution. Partition A by rows:

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

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

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

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:

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

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

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

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. ◊

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 $\widehat{x}$ for which
$$\|A\| = \max_{\|x\|=1} \|Ax\| = \|A\widehat{x}\|,$$
namely the x for which the maximum is attained. This is the direction of maximal magnification. Pick $\widehat{b} = A\widehat{x}$.
- There is an $\widehat{\delta b}$ for which
$$\|A^{-1}\| = \max_{\|x\|\neq 0} \frac{\|A^{-1}x\|}{\|x\|} = \frac{\|A^{-1}\widehat{\delta b}\|}{\|\widehat{\delta b}\|},$$
again, the x for which the maximum is attained.

It is when solving the perturbed system

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

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.

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

Remark 1.4.1.3 This last exercise shows that there will always be choices for b and δb 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=-5l9Ov5RXYo

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

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

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

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

The point of this exercise is as follows:

- If you compare $\alpha = -14.24123$
  $\hat{\alpha} = -14.24723$ 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 canditates 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:

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

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
$$\|A\|_\infty = \max_{0 \le i < m} \|\widetilde{a}_i\|_1,$$
where $\widetilde{a}_i = (\widetilde{a}_i^T)^T$. 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 $0 \le j < m$ such that $\|A\|_\infty = \|Ax\|_\infty$. This x satisfies $\|x\|_\infty = 1$.

  More precisely: $\|A\|_\infty = \|\widetilde{a}_k^T\|_1$ for some k. For simplicity, assume A is real valued. Then
$$\begin{array}{rcl} \|A\|_\infty & = & |\alpha_{k,0}| + \cdots + |\alpha_{k,m-1}| \\ & = & \alpha_{k,0}\chi_0 + \cdots + \alpha_{k,m-1}\chi_{m-1}, \end{array}$$
  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 $\|A\|_\infty = \|\widetilde{a}_k\|_1 = \|Ax\|_\infty$.

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

$$\underbrace{\begin{pmatrix} \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \upsilon_{m-2,m-2} & \upsilon_{m-2,m-1} \\ 0 & \cdots & 0 & \upsilon_{m-1,m-1} \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} \vdots \\ \zeta_{m-2} \\ \zeta_{m-1} \end{pmatrix}}_{z} = \underbrace{\begin{pmatrix} \vdots \\ \psi_{m-2} \\ \psi_{m-1} \end{pmatrix}}_{y}.$$

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,

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

  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 provides 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 $x = \begin{pmatrix} \frac{x_0}{x_1} \\ \hline \vdots \\ \hline x_{N-1} \end{pmatrix}$ 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

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

Prove that

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

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

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

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:
$$\overline{x} = \begin{pmatrix} \overline{\chi}_0 \\ \vdots \\ \overline{\chi}_{m-1} \end{pmatrix}.$$
- Transpose of vector:
$$x^T = \begin{pmatrix} \chi_0 & \cdots & \chi_{m-1} \end{pmatrix}$$
- Hermitian transpose (conjugate transpose) of vector:
$$x^H = \overline{x}^T = \overline{x^T} = \begin{pmatrix} \overline{\chi}_0 & \cdots & \overline{\chi}_{m-1} \end{pmatrix}.$$
- Dot product (inner product): $x^H y = \overline{x}^T y = \overline{x^T} y = \overline{\chi}_0\psi_0 + \cdots + \overline{\chi}_{m-1}\psi_{m-1} = \sum_{i=0}^{m-1} \overline{\chi}_i\psi_i$.

Definition 1.6.2.1 Vector norm. Let $\|\cdot\| : \mathbb{C}^m \to \mathbb{R}$. 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): $\|x\|_2 = \sqrt{x^H x} = \sqrt{|\chi_0|^2 + \cdots + |\chi_{m-1}|^2} = \sqrt{\overline{\chi}_0\chi_0 + \cdots + \overline{\chi}_{m-1}\chi_{m-1}}$ $= \sqrt{\sum_{i=0}^{m-1} |\chi_i|^2}$.
- 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: $\|x\|_\infty = \max(|\chi_0|, \ldots, \chi_{m-1}|) = \max_{i=0}^{m-1} |\chi_i| = \lim_{p\to\infty} \|x\|_p$.
- Unit ball: Set of all vectors with norm equal to one. Notation: $\|x\| = 1$.

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

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

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

Definition 1.6.2.3 Linear transformations and matrices. Let $L : \mathbb{C}^n \to \mathbb{C}^m$. 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,

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

$\Diamond$

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

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

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

Partition C, A, and B by rows and columns

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

and

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

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

1. By columns:

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

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

2. By rows:

   $$\begin{pmatrix} \widetilde{c}_0^T \\ \hline \vdots \\ \hline \widetilde{c}_{m-1}^T \end{pmatrix} := \begin{pmatrix} \widetilde{a}_0^T \\ \hline \vdots \\ \hline \widetilde{a}_{m-1}^T \end{pmatrix} B = \begin{pmatrix} \widetilde{a}_0^T B \\ \hline \vdots \\ \hline \widetilde{a}_{m-1}^T B \end{pmatrix}.$$

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

3. As the sum of outer products:

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

   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),

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

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

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

Definition 1.6.2.5 Matrix norm. Let $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$. 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

$$A = \begin{pmatrix} \alpha_{0,0} & \cdots \alpha_{0,n-1} \\ \vdots & \vdots \\ \alpha_{m-1,0} & \cdots \alpha_{m-1,n-1} \end{pmatrix} = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array}\right) = \left(\begin{array}{c} \widetilde{a}_0^T \\ \hline \vdots \\ \hline \widetilde{a}_{m-1}^T \end{array}\right).$$

Then

- Conjugate of matrix:
$$\overline{A} = \begin{pmatrix} \overline{\alpha}_{0,0} & \cdots & \overline{\alpha}_{0,n-1} \\ \vdots & \vdots & \\ \overline{\alpha}_{m-1,0} & \cdots & \overline{\alpha}_{m-1,n-1} \end{pmatrix}.$$
- Transpose of matrix:
$$A^T = \begin{pmatrix} \alpha_{0,0} & \cdots & \alpha_{m-1,0} \\ \vdots & \vdots & \\ \alpha_{0,n-1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix}.$$
- Conjugate transpose (Hermitian transpose) of matrix:
$$A^H = \overline{A}^T = \overline{A^T} = \begin{pmatrix} \overline{\alpha}_{0,0} & \cdots & \overline{\alpha}_{m-1,0} \\ \vdots & \vdots & \\ \overline{\alpha}_{0,n-1} & \cdots & \overline{\alpha}_{m-1,n-1} \end{pmatrix}.$$
- Frobenius norm: $\|A\|_F = \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2} = \sqrt{\sum_{j=0}^{n-1} \|a_j\|_2^2} = \sqrt{\sum_{i=0}^{m-1} \|\widetilde{a}_i\|_2^2}$
- 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: $\|A\|_\infty = \max_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{\|x\|_\infty=1} \|Ax\|_\infty = \max_{0\leq i<m} \|\widetilde{a}_i\|_1 = \|A^H\|_1$.

Theorem 1.6.2.6 Equivalence of matrix norms. *Let $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ and $|||\cdot||| : \mathbb{C}^{m\times n} \to \mathbb{R}$ 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 $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ is said to be subordinate to vector norms $\|\cdot\|_\mu : \mathbb{C}^m \to \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \to \mathbb{R}$ 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. ◊

Definition 1.6.2.8 Consistent matrix norm. A matrix norm $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ 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. ◊

Definition 1.6.2.9 Submultiplicative matrix norm. A consistent matrix norm $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ is said to be submultiplicative if it satisfies

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

◊

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. ◊

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 jth column of pixels as a vector of values, b_j, so that the whole picture is represented by columns as

$$B = \left(\begin{array}{c|c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right),$$

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:

$$\begin{array}{ccc} a_0 & = & b_0 \\ a_1 & = & b_{n/k-1} \\ \vdots & & \vdots \\ a_{k-1} & = & b_{(k-1)n/k-1}, \end{array}$$

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 $a_0, \ldots, a_{k-1}$:

$$b_j \approx \chi_{0,j} a_0 + \chi_{1,j} a_1 + \cdots + \chi_{k-1,j} a_{k-1} = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{k-1} \end{array} \right) \left(\begin{array}{c} \chi_{0,j} \\ \hline \vdots \\ \hline \chi_{k-1,j} \end{array} \right).$$

We can write this more concisely by viewing these chosen columns as the columns of matrix A so that

$$b_j \approx A x_j, \quad \text{where} \quad A = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{k-1} \end{array} \right) \text{ and } x_j = \left(\begin{array}{c} \chi_{0,j} \\ \hline \vdots \\ \hline \chi_{k-1,j} \end{array} \right).$$

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

$$B \approx A \underbrace{(A^T A)^{-1} A^T \left(\begin{array}{c|c|c} b_0 & \cdots & b_{n-1} \end{array} \right)}_{\left(\begin{array}{c|c|c} x_0 & \cdots & x_{n-1} \end{array} \right)} = A \underbrace{(A^T A)^{-1} A^T B}_{X} = AX.$$

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?

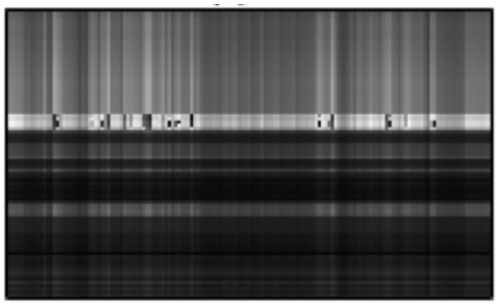
Solution. The matrix AX has rank at most equal to k (it is a rank-k matrix) since each of its columns

can be written as a linear combinations of the columns of A and hence it has at most k linearly independent columns.

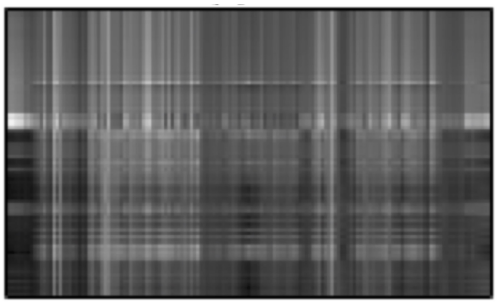
Let's have a look at how effective this approach is for our picture:

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

2.1.3 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

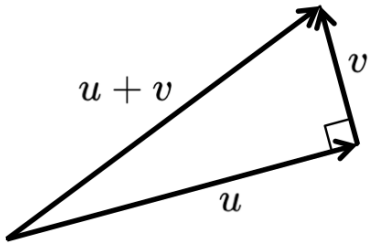
- Determine whether vectors are orthogonal.
- Compute the component of a vector in the direction of another vector.
- Relate sets of orthogonal vectors to orthogonal and unitary matrices.
- Connect unitary matrices to the changing of orthonormal basis.
- Identify transformations that can be represented 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

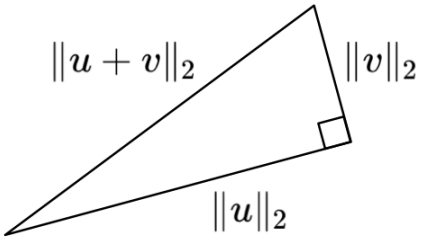
2.2.1 Orthogonal vectors

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

$$\begin{array}{l} u^T u + v^T v = (u + v)^T (u + v) \\ \quad \Leftrightarrow \quad < \text{ multiply out } > \\ u^T u + v^T v = u^T u + u^T v + v^T u + v^T v \\ \quad \Leftrightarrow \quad < u^T v = v^T u \text{ if } u \text{ and } v \text{ are real-valued } > \\ u^T u + v^T v = u^T u + 2u^T v + v^T v \\ \quad \Leftrightarrow \quad < \text{ delete common terms } > \\ 0 = 2u^T v \end{array}$$

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

$$x^H y = \overline{x}^T y = \overline{x^T} y = \overline{\chi}_0 \psi_0 + \overline{\chi}_1 \psi_1 + \cdots + \overline{\chi}_{m-1} \psi_{m-1} = \sum_{i=0}^{m-1} \overline{\chi}_i \psi_i.$$

◊

The notation x^H is short for $\overline{x}^T$, where $\overline{x}$ equals the vector x with all its entries conjugated. So,

$$\begin{array}{l}
x^H y \\
\quad = \quad < \text{ expose the elements of the vectors } > \\
\begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{m-1} \end{pmatrix}^H \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{m-1} \end{pmatrix} \\
\quad = \quad < x^H = \overline{x}^T > \\
\overline{\begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{m-1} \end{pmatrix}}^T \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{m-1} \end{pmatrix} \\
\quad = \quad < \text{ conjugate the elements of } x > \\
\begin{pmatrix} \overline{\chi}_0 \\ \vdots \\ \overline{\chi}_{m-1} \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{m-1} \end{pmatrix} \\
\quad = \quad < \text{ view } x \text{ as a } m \times 1 \text{ matrix and transpose } > \\
\begin{pmatrix} \overline{\chi}_0 \mid \cdots \mid \overline{\chi}_{m-1} \end{pmatrix} \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{mn-1} \end{pmatrix} \\
\quad = \quad < \text{ view } x^H \text{ as a matrix and perform matrix-vector multiplication } > \\
\sum_{i=0}^{m-1} \overline{\chi}_i \psi_i.
\end{array}$$

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.

$$\overline{x^H y} = \overline{\sum_{i=0}^{m-1} \overline{\chi}_i \psi_i} = \sum_{i=0}^{m-1} \overline{\overline{\chi}_i \psi_i} = \sum_{i=0}^{m-1} \overline{\psi}_i \chi_i = y^H x.$$

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 $x^H y = 0$. ◊

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

$$\widehat{b} = \frac{a^T b}{a^T a} a = \frac{a a^T}{a^T a} b$$

equals the component of b in the direction of a and

$$b^\perp = b - \widehat{b} = (I - \frac{a a^T}{a^T a}) b$$

equals the component of b orthogonal to a, since $b = \widehat{b} + b^\perp$ and $\widehat{b}^T b^\perp = 0$. Similarly, if $a, b \in \mathbb{C}^m$ then

$$\widehat{b} = \frac{a^H b}{a^H a} a = \frac{a a^H}{a^H a} b$$

equals the component of b in the direction of a and

$$b^\perp = b - \widehat{b} = (I - \frac{a a^H}{a^H a}) b$$

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

$$\frac{a a^H}{a^H a}$$

while

$$I - \frac{a a^H}{a^H a}$$

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>:

$$\left(\frac{a a^H}{a^H a} \right) \left(\frac{a a^H}{a^H a} \right) = \frac{a a^H}{a^H a}.$$

Interpret what thi s means about a matrix that projects onto a vector.

Answer. ALWAYS.
Now prove it.

Solution.

$$
\begin{array}{l}
\left(\frac{aa^H}{a^Ha}\right)\left(\frac{aa^H}{a^Ha}\right) \\
\quad = \quad < \text{ multiply numerators and denominators } > \\
\frac{aa^Haa^H}{(a^Ha)(a^Ha)} \\
\quad = \quad < \text{ associativity } > \\
\frac{a(a^Ha)a^H}{(a^Ha)(a^Ha)} \\
\quad = \quad < a^Ha \text{ is a scalar and hence commutes to front } > \\
\frac{a^Haaa^H}{(a^Ha)(a^Ha)} \\
\quad = \quad < \text{ scalar division } > \\
\frac{aa^H}{a^Ha}.
\end{array}
$$

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:

$$\left(\frac{aa^H}{a^Ha}\right)\left(I - \frac{aa^H}{a^Ha}\right) = 0$$

(the zero matrix). Interpret what this means.

Answer. ALWAYS.

Now prove it.

Solution.

$$
\begin{array}{l}
\left(\frac{aa^H}{a^Ha}\right)\left(I - \frac{aa^H}{a^Ha}\right) \\
\quad = \quad < \text{ distribute } > \\
\left(\frac{aa^H}{a^Ha}\right) - \left(\frac{aa^H}{a^Ha}\right)\left(\frac{aa^H}{a^Ha}\right) \\
\quad = \quad < \text{ last homework } > \\
\left(\frac{aa^H}{a^Ha}\right) - \left(\frac{aa^H}{a^Ha}\right) \\
\quad = \\
0.
\end{array}
$$

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 $a, b \in \mathbb{C}^n$, $\widehat{b} = \frac{aa^H}{a^Ha}b$, and $b^\perp = b - \widehat{b}$.

ALWAYS/SOMETIMES/NEVER: $\widehat{b}^H b^\perp = 0$.

Answer. ALWAYS.

Now prove it.

Solution.

$$
\begin{array}{ll}
& \widehat{b}^H b^\perp \\
= & < \text{ substitute } \widehat{b} \text{ and } b^\perp > \\
& \left(\frac{aa^H}{a^H a} b\right)^H (b - \widehat{b}) \\
= & < (Ax)^H = x^H A^H; \text{ substitute } b - \widehat{b} > \\
& b^H \left(\frac{aa^H}{a^H a}\right)^H (I - \frac{aa^H}{a^H a}) b \\
= & < (((xy^H)/\alpha)^H = yx^H/\alpha \text{ if } \alpha \text{ is real } > \\
& b^H \frac{aa^H}{a^H a} (I - \frac{aa^H}{a^H a}) b \\
= & < \text{ last homework } > \\
& b^H 0 b \\
= & < 0x = 0; y^H 0 = 0 > \\
& 0.
\end{array}
$$

2.2.3 Orthonormal vectors and matrices

YouTube: https://www.youtube.com/watch?v=GFfvDpj5dzw

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

$$\frac{uu^H}{u^H u} = uu^H.$$

- 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

$$I - \frac{uu^H}{u^H u} = I - uu^H.$$

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.

$$\begin{array}{ll} \left\| \frac{u}{\|u\|_2} \right\|_2 & \\ = & < \text{ homogenuity of norms } > \\ \frac{\|u\|_2}{\|u\|_2} & \\ = & < \text{ algebra } > \\ 1 & \end{array}$$

This last exercise shows that any nonzero vector can be scaled (normalized) to have unit length.

Definition 2.2.3.1 Orthonormal vectors. Let $u_0, u_1, \ldots, u_{n-1} \in \mathbb{C}^m$. 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)

$$\{e_j\}_{i=0}^{m-1} \subset \mathbb{C}^m,$$

where

$$e_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \longleftarrow \text{ entry indexed with } j$$

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 $Q^H Q = I$. ◊

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 $Q = \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array} \right)$.

TRUE/FALSE: Q is an orthonormal matrix if and only if $q_0, q_1, \ldots, q_{n-1}$ are mutually orthonormal.

Answer. TRUE

Now prove it!

Solution. Let $Q \in \mathbb{C}^{m\times n}$ (with $n \leq m$). Partition $Q = \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array}\right)$. Then

$$\begin{array}{rcl} Q^HQ & = & \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array}\right)^H \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array}\right) \\ & = & \left(\begin{array}{c} q_0^H \\ \hline q_1^H \\ \hline \vdots \\ \hline q_{n-1}^H \end{array}\right) \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array}\right) \\ & = & \left(\begin{array}{c|c|c|c} q_0^Hq_0 & q_0^Hq_1 & \cdots & q_0^Hq_{n-1} \\ \hline q_1^Hq_0 & q_1^Hq_1 & \cdots & q_1^Hq_{n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline q_{n-1}^Hq_0 & q_{n-1}^Hq_1 & \cdots & q_{n-1}^Hq_{n-1} \end{array}\right). \end{array}$$

Now consider that $Q^HQ = I$:

$$\left(\begin{array}{c|c|c|c} q_0^Hq_0 & q_0^Hq_1 & \cdots & q_0^Hq_{n-1} \\ \hline q_1^Hq_0 & q_1^Hq_1 & \cdots & q_1^Hq_{n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline q_{n-1}^Hq_0 & q_{n-1}^Hq_1 & \cdots & q_{n-1}^Hq_{n-1} \end{array}\right) = \left(\begin{array}{c|c|c|c} 1 & 0 & \cdots & 0 \\ \hline 0 & 1 & \cdots & 0 \\ \hline \vdots & \vdots & & \vdots \\ \hline 0 & 0 & \cdots & 1 \end{array}\right).$$

Clearly Q is orthonormal if and only if $q_0, q_1, \ldots, q_{n-1}$ are mutually orthonormal.

Homework 2.2.3.3 Let $Q \in \mathbb{C}^{m\times n}$.
ALWAYS/SOMETIMES/NEVER: If $Q^HQ = I$ then $QQ^H = I$.

Answer. SOMETIMES.
Now explain why.

Solution.

- If Q is a square matrix ($m = n$) then $Q^HQ = I$ means $Q^{-1} = Q^H$. But then $QQ^{-1} = I$ and hence $QQ^H = I$.
- If Q is not square, then $Q^HQ = I$ means $m > n$. Hence Q has rank equal to n which in turn means QQ^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), QQ^H cannot equal I.

  More concretely: let $m > 1$ and $n = 1$. Choose $Q = \left(\begin{array}{c} e_0 \end{array}\right)$. Then $Q^HQ = e_0^He_0 = 1 = I$. But

$$QQ^H = e_0e_0^H = \left(\begin{array}{ccc} 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ \vdots & \vdots & \end{array}\right).$$

2.2.4 Unitary matrices

YouTube: https://www.youtube.com/watch?v=izONEmO9uqw

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 $QQ^H = I$.

Answer. SOMETIMES
Now explain it!

Solution. If Q is unitary, then it is an orthonormal matrix and square. Because it is an orthonormal matrix, $Q^HQ = I$. 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 $Q = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}$ is an orthonormal matrix: $Q^TQ = I$. 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 $U^HU = I$ (the identity). ◊

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 $Q^HQ = I$. 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, $(U^H)^HU^H = (UU^H)^H = I$ 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_0U_1, is unitary.

Answer. ALWAYS
Now prove it!

Solution. Obviously, U_0U_1 is a square matrix.
Now,

$$(U_0U_1)^H(U_0U_1) = U_1^H \underbrace{U_0^HU_0}_{I} U_1 = \underbrace{U_1^HU_1}_{I} = I.$$

Hence U_0U_1 is unitary.

Homework 2.2.4.5 Let $U_0, U_1, \ldots, U_{k-1} \in \mathbb{C}^{m\times m}$ all be unitary.
ALWAYS/SOMETIMES/NEVER: Their product, $U_0U_1 \cdots U_{k-1}$, 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

$$
\begin{array}{rcl}
(U_0U_1\cdots U_{k-1})^HU_0U_1\cdots U_{k-1} & = & U_{k-1}^H\cdots U_1^HU_0^HU_0U_1\cdots U_{k-1} \\
& = & U_{k-1}^H\ \cdots\ U_1^H\ \underbrace{U_0^HU_0}_{I}\ U_1\ \cdots\ U_{k-1} \\
& = & U_{k-1}^H\ \cdots\ \underbrace{U_1^H\ I\ U_1}\ \cdots\ U_{k-1} \\
& = & U_{k-1}^H\ \underbrace{\cdots\ I\ \cdots}\ U_{k-1} \\
& = & \underbrace{I}
\end{array}
$$

(When you see a proof that involed $\cdots$, 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.

$$
\begin{array}{l}
\|Ux\|_2^2 \\
\quad = \quad < \text{ alternative definition } > \\
(Ux)^HUx \\
\quad = \quad < (Az)^H = z^HA^H > \\
x^HU^HUx \\
\quad = \quad < U \text{ is unitary } > \\
x^Hx \\
\quad = \quad < \text{ alternative definition } > \\
\|x\|_2^2.
\end{array}
$$

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 $(Ax)^H(Ay) = x^Hy$ for all x, y by considering $\|x-y\|_2^2 = \|A(x-y)\|_2^2$. We then use that to evaluate $e_i^HA^HAe_j$.

Let $x, y \in \mathbb{C}^m$. Then

$$
\begin{array}{l}
\|x-y\|_2^2 = \|A(x-y)\|_2^2 \\
\quad \Leftrightarrow \quad < \text{ alternative definition } > \\
(x-y)^H(x-y) = (A(x-y))^HA(x-y) \\
\quad = \quad < (Bz)^H = z^HB^H > \\
(x-y)^H(x-y) = (x-y)^HA^HA(x-y) \\
\quad \Leftrightarrow \quad < \text{ multiply out } > \\
x^Hx - x^Hy - y^Hx + y^Hy = x^HA^HAx - x^HA^HAy - y^HA^HAx + y^HA^HAy \\
\quad \Leftrightarrow \quad < \text{ alternative definition } ; \overline{x^Hy} = y^Hx > \\
\|x\|_2^2 - (x^Hy + \overline{x^Hy}) + \|y\|_2^2 = \|Ax\|_2^2 - (x^HA^HAy + \overline{x^HA^HAy}) + \|Ay\|_2^2 \\
\quad \Leftrightarrow \quad < \|Ax\|_2 = \|x\|_2 \text{ and } \|Ay\|_2 = \|y\|_2; \alpha + \overline{\alpha} = 2\text{Re}(\alpha) > \\
\text{Re}\left(x^Hy\right) = \text{Re}\left((Ax)^HAy\right)
\end{array}
$$

One can similarly show that $\text{Im}\left(x^Hy\right) = \text{Im}\left((Ax)^HAy\right)$ by considering $A(ix-y)$.

Conclude that $(Ax)^H(Ay) = x^Hy$.

We now use this to show that $A^HA = I$ by using the fact that the standard basis vectors have the property that

$$
e_i^He_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}
$$

and that the i, j entry in A^HA equals $e_i^HA^HAe_j$.

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.

$$
\begin{array}{l}
\|U\|_2 \\
\quad = \quad < \text{ definition } > \\
\max_{\|x\|_2=1} \|Ux\|_2 \\
\quad = \quad < \text{ unitary matrices preserve length } > \\
\max_{\|x\|_2=1} \|x\|_2 \\
\quad = \quad < \text{ algebra } > \\
1
\end{array}
$$

(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.

$$
\begin{array}{l}
\kappa_2 U \\
\quad = \quad < \text{ definition } > \\
\|U\|_2 \|U^{-1}\|_2 \\
\quad = \quad < \text{ both } U \text{ and } U^{-1} \text{ are unitary ; last homework } > \\
1 \times 1 \\
\quad = \quad < \text{ arithmetic } > \\
1
\end{array}
$$

The preservation of length extends to the preservation of norms that have a relation to the 2-norm:d

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.

- $$
\begin{array}{l}
\|U^H A\|_2 \\
\quad = \quad < \text{ definition of 2-norm } > \\
\max_{\|x\|_2=1} \|U^H Ax\|_2 \\
\quad = \quad < U \text{ is unitary and unitary matrices preserve length } > \\
\max_{\|x\|_2=1} \|Ax\|_2 \\
\quad = \quad < \text{ definition of 2-norm } > \\
\|A\|_2.
\end{array}
$$

- $\|AV\|_2$
  $=$ $<$ definition of 2-norm $>$
  $\max_{\|x\|_2=1} \|AVx\|_2$
  $=$ $<$ V^H is unitary and unitary matrices preserve length $>$
  $\max_{\|Vx\|_2=1} \|A(Vx)\|_2$
  $=$ $<$ substitute $y = Vx$ $>$
  $\max_{\|y\|_2=1} \|Ay\|_2$
  $=$ $<$ definition of 2-norm $>$
  $\|A\|_2$.

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

  $\|U^H A\|_F^2$
  $=$ $<$ partition A by columns $>$
  $\|U^H \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right) \|_F^2$
  $=$ $<$ property of matrix-vector multiplication $>$
  $\| \left(\begin{array}{c|c|c} U^H a_0 & \cdots & U^H a_{n-1} \end{array} \right) \|_F^2$
  $=$ $<$ exercise in Chapter 1 $>$
  $\sum_{j=0}^{n-1} \|U^H a_j\|_2^2$
  $=$ $<$ unitary matrices preserve length $>$
  $\sum_{j=0}^{n-1} \|a_j\|_2^2$
  $=$ $<$ exercise in Chapter 1 $>$
  $\|A\|_F^2$.

- 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.

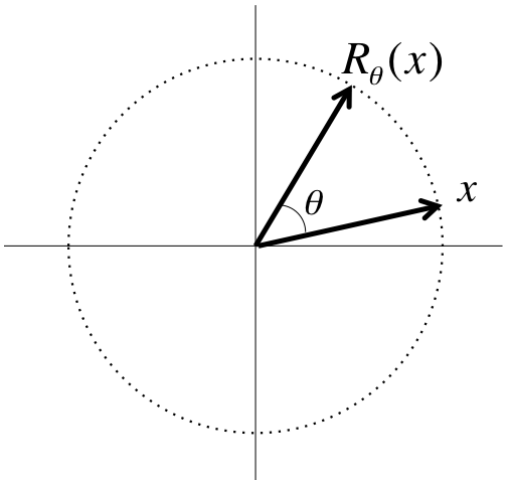
2.2.5 Examples of unitary matrices

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=C0mlDZ28Ohc

A rotation in 2D, $R_\theta : \mathbb{R}^2 \to \mathbb{R}^2$, takes a vector and rotates that vector through the angle θ:

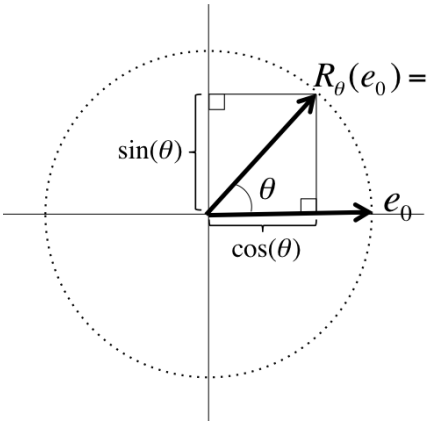


If you think about it,

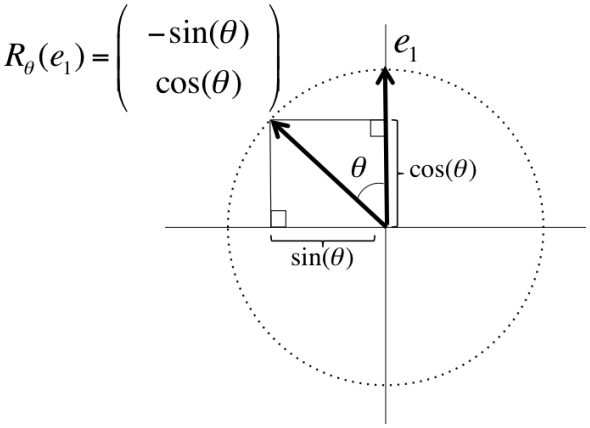
- If you scale a vector first and then rotate it, you get the same result as if you rotate it first and then scale it.
- If you add two vectors first and then rotate, you get the same result as if you rotate them first and then add them.

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 $L : \mathbb{C}^n \to \mathbb{C}^m$ is a linear transformation and A is the matrix that represents it, then the jth column of A, a_j, equals $L(e_j)$. The pictures



and



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,

$$R_\theta(x) = \left(\begin{array}{c|c} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array}\right) \left(\frac{\chi_0}{\chi_1}\right).$$

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.

$$\begin{array}{l}
\left(\begin{array}{c|c} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array}\right)^H \left(\begin{array}{c|c} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array}\right) \\
\quad = \quad < \text{ the matrix is real valued } > \\
\left(\begin{array}{c|c} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array}\right)^T \left(\begin{array}{c|c} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array}\right) \\
\quad = \quad < \text{ transpose } > \\
\left(\begin{array}{cc} \cos(\theta) & \sin(\theta) \\ \hline -\sin(\theta) & \cos(\theta) \end{array}\right) \left(\begin{array}{c|c} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array}\right) \\
\quad = \quad < \text{ multiply } > \\
\left(\begin{array}{c|c} \cos^2(\theta) + \sin^2(\theta) & -\cos(\theta)\sin(\theta) + \sin(\theta)\cos(\theta) \\ \hline -\sin(\theta)\cos(\theta) + \cos(\theta)\sin(\theta) & \sin^2(\theta) + \cos^2(\theta) \end{array}\right) \\
\quad = \quad < \text{ geometry; algebra } > \\
\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\end{array}$$

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

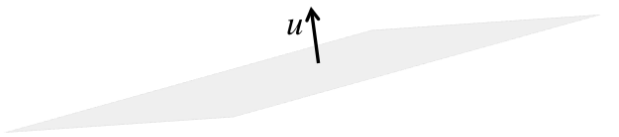
$$\begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix}. = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

from which we conclude that $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$.

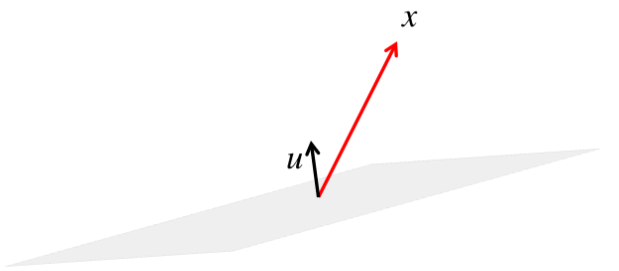
2.2.5.2 Reflections

YouTube: https://www.youtube.com/watch?v=r8S04qqcc-o

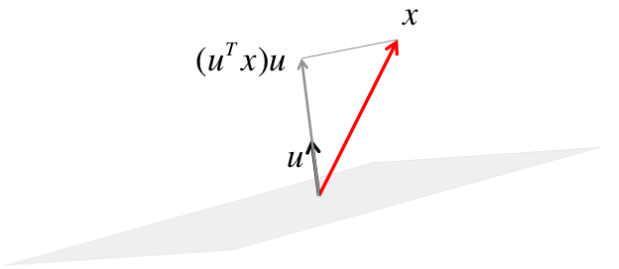
Picture a mirror with its orientation defined by a unit length vector, u, that is orthogonal to it.



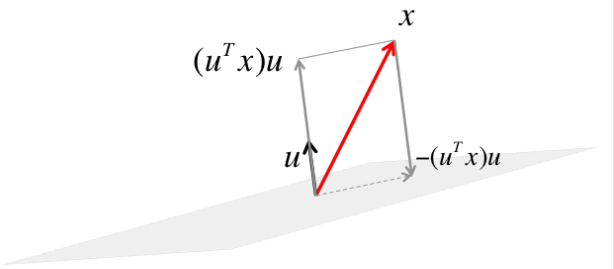
We will consider how a vector, x, is reflected by this mirror.



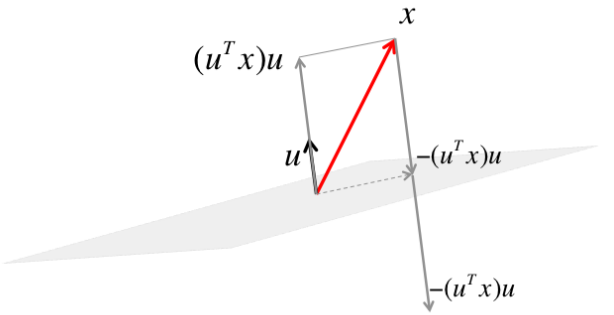
The component of x orthogonal to the mirror equals the component of x in the direction of u, which equals $(u^T x)u$.



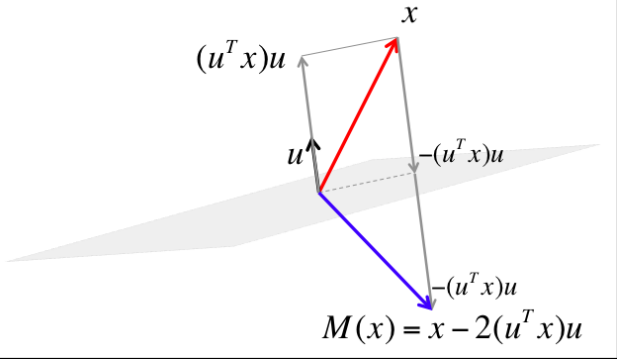
The orthogonal projection of x onto the mirror is then given by the dashed vector, which equals $x - (u^T x)u$.



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 $M : \mathbb{R}^3 \to \mathbb{R}^3$ in the above example is given by $I - 2uu^T$.
Hint. Rearrange $x - 2(u^T x)u$.

Solution. We notice that

$$\begin{array}{l} x - 2(u^T x)u \\ \quad = \quad < \alpha x = x\alpha > \\ x - 2u(u^T x) \\ \quad = \quad < \text{ associativity } > \\ Ix - 2uu^T x \\ \quad = \quad < \text{ distributivity } > \\ (I - 2uu^T)x. \end{array}$$

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 $M : \mathbb{R}^3 \to \mathbb{R}^3$ 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

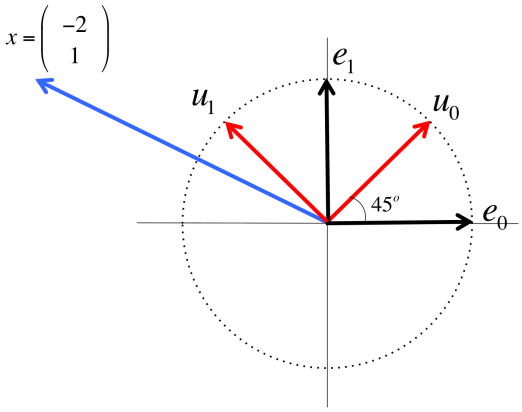
$$\begin{array}{l} (I - 2uu^T)^T(I - 2uu^T) \\ \quad = \quad < (A + B)^T = A^T + B^T > \\ (I^T - (2uu^T)^T)(I - 2uu^T) \\ \quad = \quad < (\alpha AB^T)^T = \alpha BA^T > \\ (I - 2uu^T)(I - 2uu^T) \\ \quad = \quad < \text{ distributivity } > \\ (I - 2uu^T) - (I - 2uu^T)(2uu^T) \\ \quad = \quad < \text{ distributivity } > \\ I - 2uu^T - 2uu^T + 2uu^T 2uu^T \\ \quad = \quad < u^T u = 1 > \\ I - 4uu^T + 4uu^T \\ \quad = \quad < A - A = 0 > \\ I. \end{array}$$

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 $\widehat{x} = \begin{pmatrix} \widehat{\chi}_0 \\ \widehat{\chi}_1 \end{pmatrix}$ such that $\widehat{\chi}_0 u_0 + \widehat{\chi}_1 u_1 = x$.

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

$$\begin{array}{l} x \\ \quad = \quad < u_0 \text{ and } u_1 \text{ are orthonormal } > \\ \quad \underbrace{(u_0^T x)u_0}_{\substack{\text{component in the} \\ \text{direction of } u_0}} + \underbrace{(u_1^T x)u_1}_{\substack{\text{component in the} \\ \text{direction of } u_1}} \\ \quad = \quad < \text{instantiate } u_0 \text{ and } u_1 > \\ \left(\frac{\sqrt{2}}{2}\begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -2 \\ 1 \end{pmatrix}\right) u_0 + \left(\frac{\sqrt{2}}{2}\begin{pmatrix} -1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -2 \\ 1 \end{pmatrix}\right) u_1 \\ \quad = \quad < \text{evaluate } > \\ -\frac{\sqrt{2}}{2}u_0 + \frac{3\sqrt{2}}{2}u_1. \end{array}$$

- An alternative way to arrive at the same answer that provides more insight. Let $U = \left(\begin{array}{c|c} u_0 & u_1 \end{array} \right)$.

Then

$$
\begin{array}{ll}
x & \\
= & < U \text{ is unitary (or orthogonal since it is real valued)} > \\
UU^T x & \\
= & < \text{instantiate } U > \\
\left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c} u_0^T \\ \hline u_1^T \end{array} \right) x & \\
= & < \text{matrix-vector multiplication} > \\
\left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c} u_0^T x \\ \hline u_1^T x \end{array} \right) & \\
= & < \text{instantiate} > \\
\left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c} \frac{\sqrt{2}}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -2 \\ 1 \end{pmatrix} \\ \hline \frac{\sqrt{2}}{2} \begin{pmatrix} -1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -2 \\ 1 \end{pmatrix} \end{array} \right) & \\
= & < \text{evaluate} > \\
\left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \begin{pmatrix} -\frac{\sqrt{2}}{2} \\ \hline \frac{3\sqrt{2}}{2} \end{pmatrix} & \\
= & < \text{simplify} > \\
\left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\frac{\sqrt{2}}{2} \begin{pmatrix} -1 \\ 3 \end{pmatrix} \right) &
\end{array}
$$

Below we compare side-by-side how to describe a vector x using the standard basis vectors $e_0, \ldots, e_{m-1}$ (on the left) and vectors $u_0, \ldots, u_{m-1}$ (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 $e_0, \ldots, e_{m-1}$:

$$
\begin{array}{ll}
x & \\
= & < x = Ix = II x = II^T x > \\
II^T x & \\
= & < \text{ expose columns of } I > \\
\left(\begin{array}{c|c|c} e_0 & \cdots & e_{m-1} \end{array} \right) \left(\begin{array}{c} \hline e_0^T \\ \hline \vdots \\ \hline e_{m-1}^T \end{array} \right) x & \\
= & < \text{ evaluate } > \\
\left(\begin{array}{c|c|c} e_0 & \cdots & e_{m-1} \end{array} \right) \left(\begin{array}{c} \hline e_0^T x \\ \hline \vdots \\ \hline e_{m-1}^T x \end{array} \right) & \\
= & < e_j^T x = \chi_j > \\
\left(\begin{array}{c|c|c} e_0 & \cdots & e_{m-1} \end{array} \right) \left(\begin{array}{c} \hline \chi_0 \\ \hline \vdots \\ \hline \chi_{m-1} \end{array} \right) & \\
= & < \text{ evaluate } > \\
\chi_0 e_0 + \chi_1 e_1 + \cdots + \chi_{m-1} e_{m-1}. &
\end{array}
$$

The vector $\widehat{x} = \begin{pmatrix} u_0^T x \\ \vdots \\ u_{m-1}^T x \end{pmatrix}$ describes the vector x in terms of the orthonormal basis $u_0, \ldots, u_{m-1}$:

$$
\begin{array}{ll}
x & \\
= & < x = Ix = UU^H x > \\
UU^H x & \\
= & < \text{ expose columns of } U > \\
\left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right) \left(\begin{array}{c} \hline u_0^H \\ \hline \vdots \\ \hline u_{m-1}^H \end{array} \right) x & \\
= & < \text{ evaluate } > \\
\left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right) \left(\begin{array}{c} \hline u_0^H x \\ \hline \vdots \\ \hline u_{m-1}^H x \end{array} \right) & \\
= & < \text{ evaluate } > \\
u_0^H x u_0 + u_1^H x u_1 + \cdots + u_{m-1}^H x u_{m-1}. &
\end{array}
$$

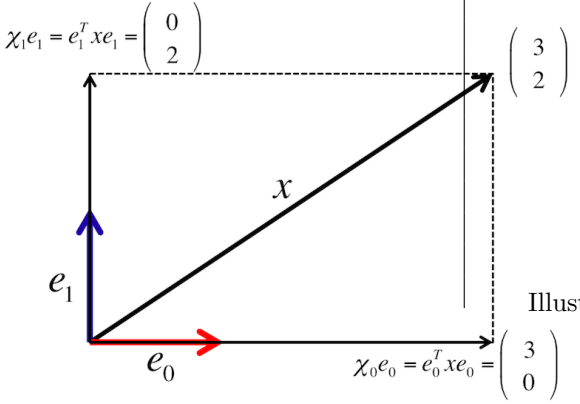


Illustration:

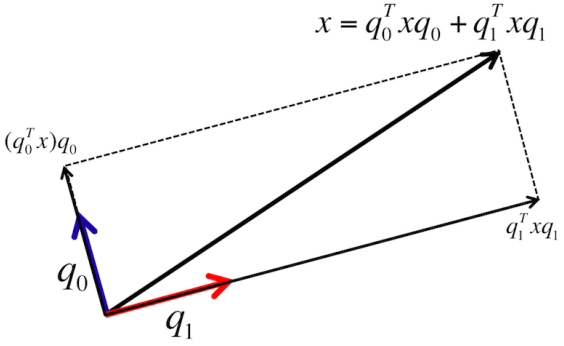


Illustration:

Another way of looking at this is that if $u_0, u_1, \ldots, u_{m-1}$ 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:

$$x = \alpha_0 u_0 + \alpha_1 u_1 + \cdots + \alpha_{m-1} u_{m-1}.$$

Now,

$$
\begin{array}{rcl}
u_i^H x & = & u_i^H (\alpha_0 u_0 + \alpha_1 u_1 + \cdots + \alpha_{i-1} u_{i-1} + \alpha_i u_i + \alpha_{i+1} u_{i+1} + \cdots + \alpha_{m-1} u_{m-1}) \\
& = & \alpha_0 \underbrace{u_i^H u_0}_{0} + \alpha_1 \underbrace{u_i^H u_1}_{0} + \cdots + \alpha_{i-1} \underbrace{u_i^H u_{i-1}}_{0} \\
& & \quad + \alpha_i \underbrace{u_i^H u_i}_{1} + \alpha_{i+1} \underbrace{u_i^H u_{i+1}}_{0} + \cdots + \alpha_{m-1} \underbrace{u_i^H u_{m-1}}_{0} \\
& = & \alpha_i.
\end{array}
$$

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

$$\frac{1}{\|y\|} \leq \|A^{-1}\| \frac{1}{\|x\|}. \tag{2.2.1}$$

Subtracting $y = Ax$ from $y + \delta y = A(x + \delta x)$ yields

$$\delta y = A\delta x$$

and hence

$$\|\delta y\| \leq \|A\|\|\delta x\|. \tag{2.2.2}$$

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*

$$\Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \text{ with } \Sigma_{TL} = \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{r-1} \end{pmatrix} \tag{2.3.1}$$

and $\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > 0$. The values $\sigma_0, \ldots, \sigma_{r-1}$ 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 vector "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.,

$$U = \left(\begin{array}{cccc} u_0 & u_1 & \cdots & u_{n-1} \end{array}\right)$$

and the notation we use to hide such explicit indexing, which we call the FLAME notation, e.g.,

$$U = \left(\begin{array}{c|ccc} U_0 & u_1 & U_2 \end{array}\right).$$

The two linked by

$$\left(\begin{array}{c|c|c} \underbrace{\begin{array}{cc} u_0 & u_{k-1} \end{array}}_{U_0} & \underbrace{u_k}_{u_1} & \underbrace{\begin{array}{cc} u_{k+1} & u_{n-1} \end{array}}_{U_2} \end{array}\right).$$

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 $\widetilde{U} \in \mathbb{C}^{m\times m}$ and $\widetilde{V} \in \mathbb{C}^{n\times n}$ such that*

$$A = \widetilde{U} \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array} \right) \widetilde{V}^H, \text{ where } \sigma_1 = \|A\|_2.$$

Proof. In the below proof, it is really imporant 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 $\widetilde{v}_1 \in \mathbb{C}^n$ such that

- $\|\widetilde{v}_1\|_2 = 1$; and
- $\sigma_1 = \|A\widetilde{v}_1\|_2 = \|A\|_2$.

In other words, $\widetilde{v}_1$ is the vector that maximizes $\max_{\|x\|_2=1} \|Ax\|_2$.

Let $\widetilde{u}_1 = A\widetilde{v}_1/\sigma_1$. Then

$$\|\widetilde{u}_1\|_2 = \|A\widetilde{v}_1\|_2/\sigma_1 = \|A\widetilde{v}_1\|_2/\|A\|_2 = \|A\|_2/\|A\|_2 = 1.$$

Choose $\widetilde{U}_2 \in \mathbb{C}^{m\times(m-1)}$ and $\widetilde{V}_2 \in \mathbb{C}^{n\times(n-1)}$ so that

$$\widetilde{U} = \left(\begin{array}{c|c} \widetilde{u}_1 & \widetilde{U}_2 \end{array} \right) \text{ and } \widetilde{V} = \left(\begin{array}{c|c} \widetilde{v}_1 & \widetilde{V}_2 \end{array} \right)$$

are unitary. Then

$$\begin{array}{l}
\widetilde{U}^H A \widetilde{V} \\
\quad = \quad < \text{ instantiate } > \\
\left(\begin{array}{c|c} \widetilde{u}_1 & \widetilde{U}_2 \end{array} \right)^H A \left(\begin{array}{c|c} \widetilde{v}_1 & \widetilde{V}_2 \end{array} \right) \\
\quad = \quad < \text{ multiply out } > \\
\left(\begin{array}{c|c} \widetilde{u}_1^H A\widetilde{v}_1 & \widetilde{u}_1^H A\widetilde{V}_2 \\ \hline \widetilde{U}_2^H A\widetilde{v}_1 & \widetilde{U}_2^H A\widetilde{V}_2 \end{array} \right) \\
\quad = \quad < A\widetilde{v}_1 = \sigma_1\widetilde{u}_1 > \\
\left(\begin{array}{c|c} \sigma_1\widetilde{u}_1^H\widetilde{u}_1 & \widetilde{u}_1^H A\widetilde{V}_2 \\ \hline \sigma_1\widetilde{U}_2^H\widetilde{u}_1 & \widetilde{U}_2^H A\widetilde{V}_2 \end{array} \right) \\
\quad = \quad < \widetilde{u}_1^H\widetilde{u}_1 = 1; \widetilde{U}_2^H\widetilde{u}_1 = 0; \text{ pick } w = \widetilde{V}_2^H A^H\widetilde{u}_1 \text{ and } B = \widetilde{U}_2^H A\widetilde{V}_2 > \\
\left(\begin{array}{c|c} \sigma_1 & w^H \\ \hline 0 & B \end{array} \right),
\end{array}$$

where $w = \widetilde{V}_2^H A^H\widetilde{u}_1$ and $B = \widetilde{U}_2^H A\widetilde{V}_2$.

We will now argue that $w = 0$, the zero vector of appropriate size:

$$
\begin{array}{ll}
\sigma_1^2 & \\
= & < \text{assumption} > \\
\|A\|_2^2 & \\
= & < \text{2-norm is invariant under multiplication by unitary matrix} > \\
\|\widetilde{U}^H A \widetilde{V}\|_2^2 & \\
= & < \text{definition of } \|\cdot\|_2 > \\
\max_{x \neq 0} \frac{\|\widetilde{U}^H A \widetilde{V} x\|_2^2}{\|x\|_2^2} & \\
= & < \text{see above} > \\
\max_{x \neq 0} \frac{\left\| \left(\begin{array}{c|c} \sigma_1 & w^H \\ \hline 0 & B \end{array} \right) x \right\|_2^2}{\|x\|_2^2} & \\
\geq & < x \text{ replaced by specific vector} > \\
\frac{\left\| \left(\begin{array}{c|c} \sigma_1 & w^H \\ \hline 0 & B \end{array} \right) \left(\begin{array}{c} \sigma_1 \\ \hline w \end{array} \right) \right\|_2^2}{\left\| \left(\begin{array}{c} \sigma_1 \\ \hline w \end{array} \right) \right\|_2^2} & \\
= & < \text{multiply out numerator} > \\
\frac{\left\| \left(\begin{array}{c} \sigma_1^2 + w^H w \\ \hline Bw \end{array} \right) \right\|_2^2}{\left\| \left(\begin{array}{c} \sigma_1 \\ \hline w \end{array} \right) \right\|_2^2} & \\
\geq & < \left\| \left(\begin{array}{c} \psi_1 \\ \hline y_2 \end{array} \right) \right\|_2^2 = \|\psi_1\|_2^2 + \|y_2\|_2^2 \geq \|\psi_1\|_2^2; \left\| \left(\begin{array}{c} \sigma_1 \\ \hline w \end{array} \right) \right\|_2^2 = \sigma_1^2 + w^H w > \\
(\sigma_1^2 + w^H w)^2 / (\sigma_1^2 + w^H w) & \\
= & < \text{algebra} > \\
\sigma_1^2 + w^H w. &
\end{array}
$$

Thus $\sigma_1^2 \geq \sigma_1^2 + w^H w$ which means that $w = 0$ (the zero vector) and $\widetilde{U}^H A \widetilde{V} = \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array} \right)$ so that $A = \widetilde{U} \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array} \right) \widetilde{V}^H$. ■

Hopefully you can see where this is going: If one can recursively find that $B = U_B \Sigma_B V_B^H$, then

$$
\begin{array}{rcl}
A & = & \widetilde{U} \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array} \right) \widetilde{V}^H \\
& = & \widetilde{U} \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & U_B \Sigma_B V_B^H \end{array} \right) \widetilde{V}^H \\
& = & \widetilde{U} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & U_B \end{array} \right) \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & \Sigma_B \end{array} \right) \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & V_B^H \end{array} \right) \widetilde{V}^H \\
& = & \underbrace{\widetilde{U} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & U_B \end{array} \right)}_{U} \underbrace{\left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & \Sigma_B \end{array} \right)}_{\Sigma} \underbrace{\left(\widetilde{V} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & V_B \end{array} \right) \right)^H}_{V^H}.
\end{array}
$$

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

$$
\begin{array}{ll}
\|A\|_2 & \\
\quad = & < \text{definition} > \\
\max_{\|x\|_2=1} \|Ax\|_2 & \\
\quad \geq & < \text{pick a specific vector with } 2-\text{norm equal to one} > \\
\left\| A \begin{pmatrix} 0 \\ \hline z \end{pmatrix} \right\|_2 & \\
\quad = & < \text{instantiate } A > \\
\left\| \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array} \right) \begin{pmatrix} 0 \\ \hline z \end{pmatrix} \right\|_2 & \\
\quad = & < \text{partitioned matrix-vector multiplication} > \\
\left\| \begin{pmatrix} 0 \\ \hline Bz \end{pmatrix} \right\|_2 & \\
\quad = & < \left\| \begin{pmatrix} y_0 \\ \hline y_1 \end{pmatrix} \right\|_2^2 = \|y_0\|_2^2 + \|y_1\|_2^2 > \\
\|Bz\|_2 & \\
\quad = & < \text{assumption about } z > \\
\|B\|_2 & \\
\quad > & < \text{assumption} > \\
\sigma_1. &
\end{array}
$$

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 arbritary 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

  $$A = \begin{pmatrix} 0 \end{pmatrix} = \underbrace{I_{m\times m}}_{U} \left(\begin{array}{c|c} & \\ \hline 0 & \end{array} \right) \underbrace{I_{1\times 1}}_{V^H}$$

  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 $U = \left(\begin{array}{c|c} u_1 & U_2 \end{array} \right)$ is unitary. Then

  $$
  \begin{array}{rcl}
  A & = & \begin{pmatrix} a_1 \end{pmatrix} \\
  & = & \begin{pmatrix} u_1 \end{pmatrix} (\|a_1\|_2) \\
  & = & \left(\begin{array}{c|c} u_1 & U_2 \end{array} \right) \left(\begin{array}{c|c} \|a_1\|_2 & \\ \hline 0 & \end{array} \right) \begin{pmatrix} 1 \end{pmatrix}^H \\
  & = & U \Sigma V^H,
  \end{array}
  $$

  where

  - $U = \left(\begin{array}{c|c} u_0 & U_1 \end{array} \right)$,

- $\Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & \\ \hline 0 & \end{array}\right)$ 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

  $$A = I_{m\times m} \left(\begin{array}{c|c} & \\ \hline 0_{m\times(k+1)} & \end{array}\right) I_{(k+1)\times(k+1)}$$

  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 $\widetilde{U} \in \mathbb{C}^{m\times m}$ and $\widetilde{V} \in \mathbb{C}^{(k+1)\times(k+1)}$ such that $A = \widetilde{U} \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & B \end{array}\right) \widetilde{V}$ 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 $\check{\Sigma}_B = \left(\begin{array}{c|c} \check{\Sigma}_{TL} & 0 \\ \hline 0 & 0 \end{array}\right)$, $\check{\Sigma}_{TL} = \text{diag}(\sigma_2, \cdots, \sigma_{r-1})$, and $\sigma_2 \geq \cdots \geq \sigma_{r-1} > 0$.

  Now, let

  $$U = \widetilde{U} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \check{U}_B \end{array}\right), V = \widetilde{V} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \check{V}_B \end{array}\right), \text{ and } \Sigma = \left(\begin{array}{c|c} \sigma_1 & 0 \\ \hline 0 & \check{\Sigma}_B \end{array}\right).$$

  (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 $\Sigma = \text{diag}(\sigma_0, \ldots, \sigma_{n-1})$. 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.

$$
\begin{array}{rcl}
\|\Sigma\|_2^2 & = & \max_{\|x\|_2=1} \|\Sigma x\|_2^2 \\
& = & \max_{\|x\|_2=1} \left\| \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{n-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \right\|_2^2 \\
& = & \max_{\|x\|_2=1} \left\| \begin{pmatrix} \sigma_0\chi_0 \\ \sigma_1\chi_1 \\ \vdots \\ \sigma_{n-1}\chi_{n-1} \end{pmatrix} \right\|_2^2 \\
& = & \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} |\sigma_j\chi_j|^2 \right] \\
& = & \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} \left[|\sigma_j|^2 |\chi_j|^2 \right] \right] \\
& \leq & \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} \left[\max_{i=0}^{n-1} |\sigma_i|^2 |\chi_j|^2 \right] \right] \\
& = & \max_{\|x\|_2=1} \left[\max_{i=0}^{n-1} |\sigma_i|^2 \sum_{j=0}^{n-1} |\chi_j|^2 \right] \\
& = & \left(\max_{i=0}^{n-1} |\sigma_i| \right)^2 \max_{\|x\|_2=1} \|x\|_2^2 \\
& = & \left(\max_{i=0}^{n-1} |\sigma_i| \right)^2 .
\end{array}
$$

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 $B = UAV^H$. 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 $B = UU_A \Sigma_A V_A^H V^H = (UU_A)\Sigma_A (VV_A)^H$ where both UU_A and VV_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

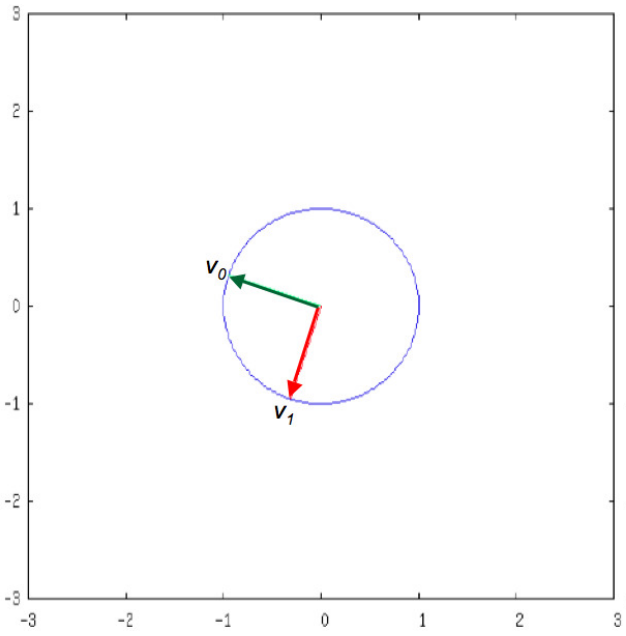
YouTube: https://www.youtube.com/watch?v=XKhCTtX1z6A

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

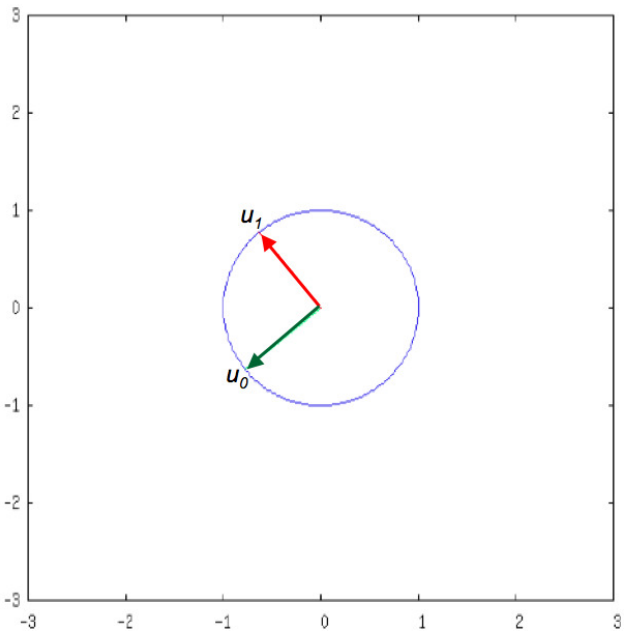
$$A = \left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} \sigma_0 & 0 \\ \hline 0 & \sigma_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T .$$

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:

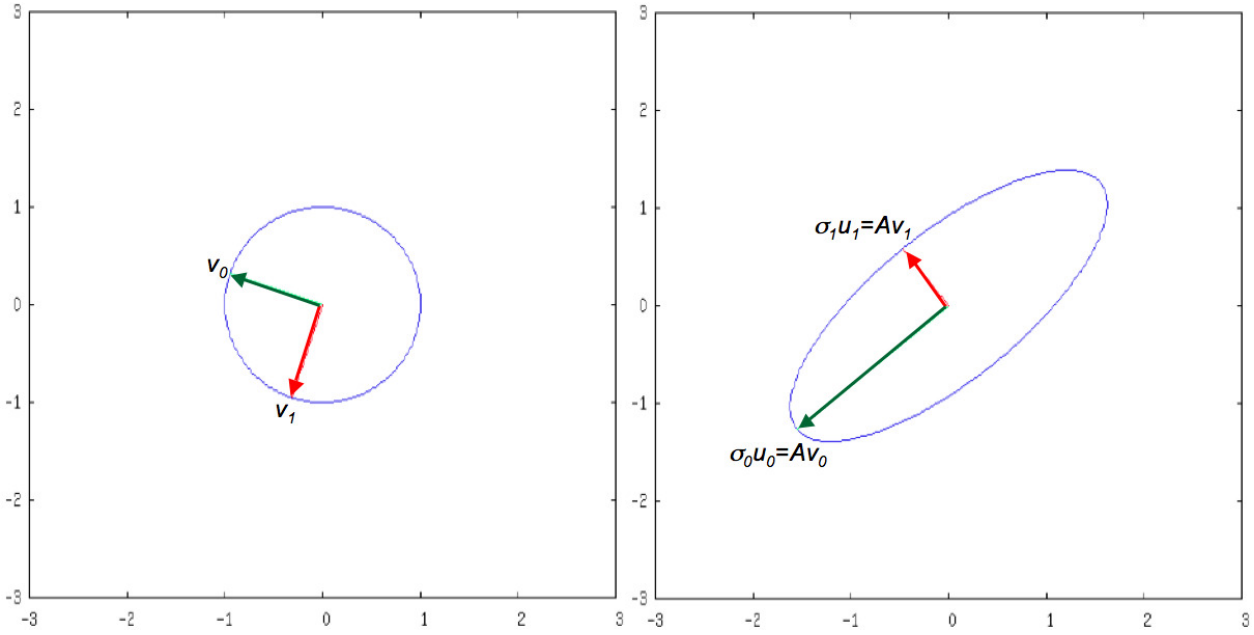
$$\begin{array}{rcl} A & = & \left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} \sigma_0 & 0 \\ \hline 0 & \sigma_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T \\ & = & \left[\left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} \sigma_0 & 0 \\ \hline 0 & \sigma_1 \end{array} \right) \right] \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T \\ & = & \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T . \end{array}$$

Now let us look at how A transforms v_0 and v_1:

$$Av_0 = \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T v_0 = \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \sigma_0 u_0$$

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:

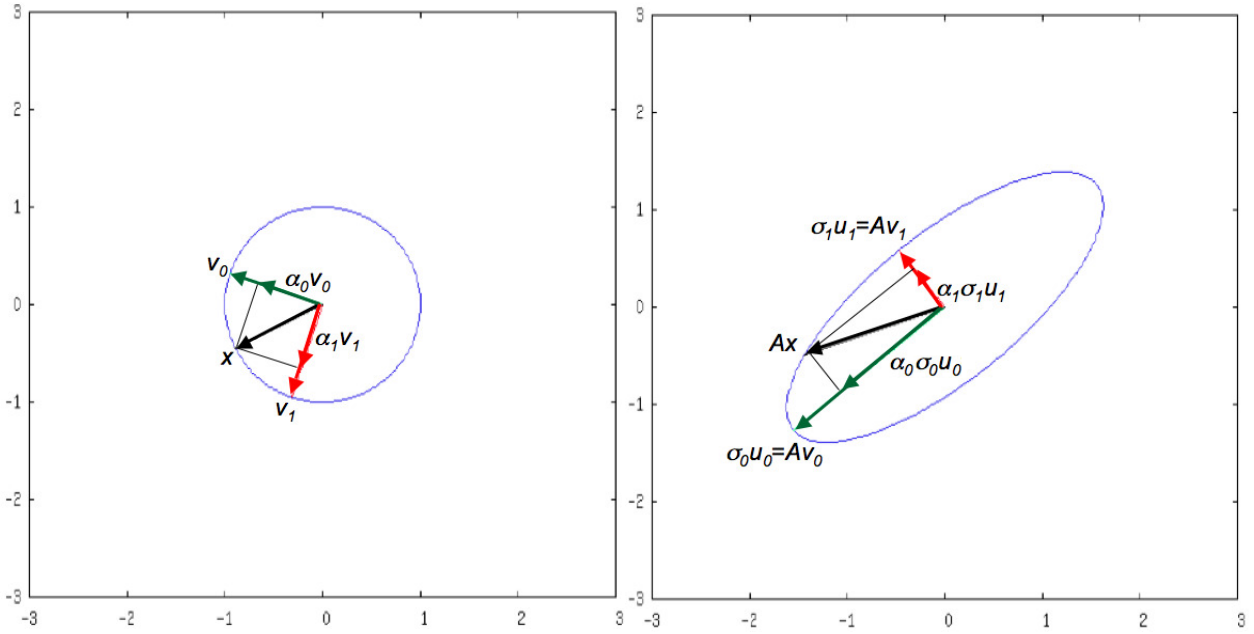


Figure 2.3.2.1 Illustration of how orthonormal vectors v_0 and v_1 are transformed by matrix $A = U\Sigma V$.

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:

$$\begin{array}{rcl} x & = & \underbrace{VV^T}_{I} x = \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T x = \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \left(\begin{array}{c} v_0^T x \\ \hline v_1^T x \end{array} \right) \\ & = & \underbrace{v_0^T x}_{\alpha_0} v_0 + \underbrace{v_1^T x}_{\alpha_1} v_1 = \alpha_0 v_0 + \alpha_0 v_1 = \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \left(\begin{array}{c} \alpha_0 \\ \hline \alpha_1 \end{array} \right). \end{array}$$

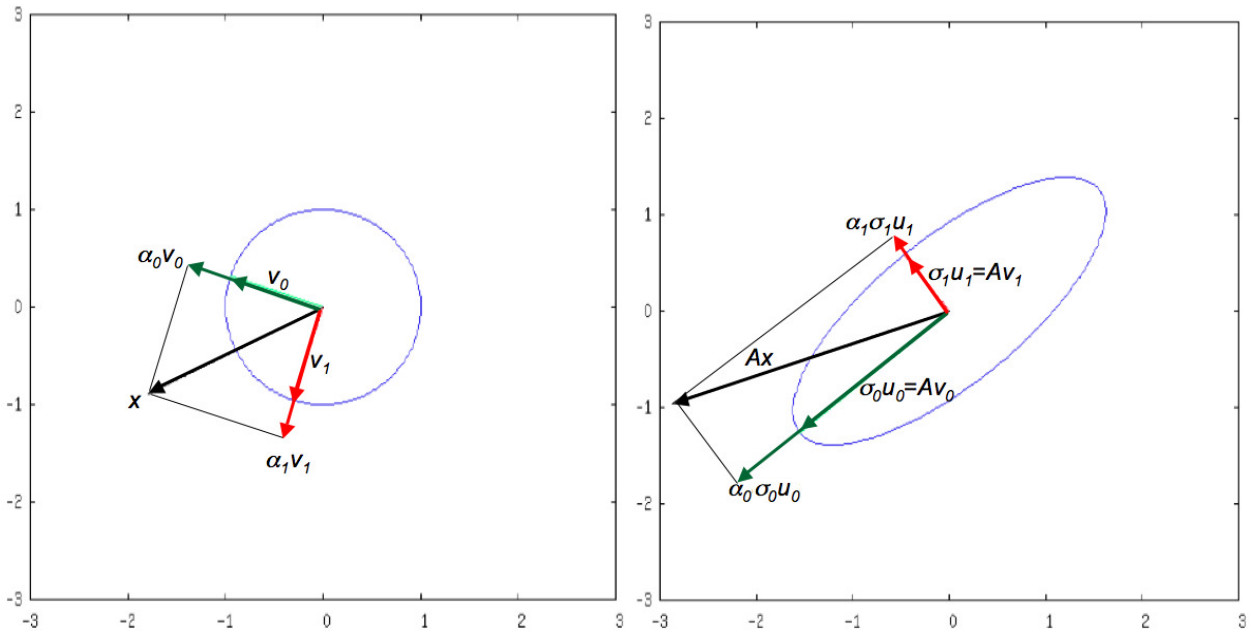
Thus, in the basis formed by v_0 and v_1, its coefficients are α_0 and α_1. Now,

$$\begin{array}{rcl} Ax & = & \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T x \\ & = & \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \left(\begin{array}{c} \alpha_0 \\ \hline \alpha_1 \end{array} \right) \\ & = & \left(\begin{array}{c|c} \sigma_0 u_0 & \sigma_1 u_1 \end{array} \right) \left(\begin{array}{c} \alpha_0 \\ \hline \alpha_1 \end{array} \right) = \alpha_0 \sigma_0 u_0 + \alpha_1 \sigma_1 u_1. \end{array}$$

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

$$A = \underbrace{\left(\begin{array}{c|c} u_0 & u_1 \end{array} \right)}_{U} \underbrace{\left(\begin{array}{c|c} \sigma_0 & 0 \\ \hline 0 & \sigma_1 \end{array} \right)}_{\Sigma} \underbrace{\left(\begin{array}{c|c} v_0 & v_1 \end{array} \right)^T}_{V}.$$

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

$$\begin{aligned}
\underbrace{UU^T}_{I}\, y &= U\, \underbrace{U^T y}_{\widehat{y}} = (u_0^T y)u_0 + (u_1^T y)u_1 \\
&= \left(\begin{array}{c|c} u_0 & u_1 \end{array} \right) \begin{pmatrix} u_0^T y \\ u_1^T y \end{pmatrix} = U \underbrace{\begin{pmatrix} \widehat{\psi}_0 \\ \widehat{\psi}_1 \end{pmatrix}}_{\widehat{y}} \\
\underbrace{VV^T}_{I}\, x &= V\, \underbrace{V^T x}_{\widehat{x}} = (v_0^T x)v_0 + (v_1^T x)v_1 \\
&= \left(\begin{array}{c|c} v_0 & v_1 \end{array} \right) \begin{pmatrix} v_0^T x \\ v_1^T x \end{pmatrix} = V \underbrace{\begin{pmatrix} \widehat{\chi}_0 \\ \widehat{\chi}_1 \end{pmatrix}}_{\widehat{x}}.
\end{aligned}$$

If $y = Ax$ then

$$U\, \underbrace{U^T y}_{\widehat{y}} = \underbrace{U\Sigma V^T x}_{Ax} = U\Sigma\widehat{x}$$

so that

$$\widehat{y} = \Sigma\widehat{x}$$

and

$$\begin{pmatrix} \widehat{\psi}_0 \\ \widehat{\psi}_1 \end{pmatrix} = \begin{pmatrix} \sigma_0\widehat{\chi}_0 \\ \sigma_1\widehat{\chi}_1 \end{pmatrix}.$$

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: $\widehat{y} := \Sigma\widehat{x}$. Also, solving $Ax = y$ for x can be computed by multiplying with the inverse of the diagonal matrix: $\widehat{x} := \Sigma^{-1}\widehat{y}$.

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

$$\underbrace{U^H y}_{\widehat{y}} = \Sigma\, \underbrace{V^H x}_{\widehat{x}}$$

(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

$$\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} \widetilde{u}_1 & \widetilde{U}_2 \end{pmatrix}^H \begin{pmatrix} \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ \widehat{a}_{21} & \widehat{A}_{22} \end{pmatrix} \begin{pmatrix} \widetilde{v}_1 & \widetilde{V}_2 \end{pmatrix} = \begin{pmatrix} \sigma_{11} & 0 \\ 0 & B \end{pmatrix},$$

where $\widehat{A}$ 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 $U = \widetilde{U}U_B$ and $V = \widetilde{V}V_B$.

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 $\widetilde{U}$ and $\widetilde{V}$:

$$\begin{array}{rcl} U & := & U\widetilde{U} \\ V & := & V\widetilde{V}. \end{array}$$

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

$$\begin{array}{rcll} U & = & \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \in \mathbb{C}^{m\times m} & \text{with } U_L \in \mathbb{C}^{m\times k} \\ V & = & \left(\begin{array}{c|c} V_L & V_R \end{array}\right) \in \mathbb{C}^{m\times m} & \text{with } V_L \in \mathbb{C}^{n\times k} \\ A & = & \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) & \text{with } A_{TL} \in \mathbb{C}^{k\times k}, \end{array}$$

where the current contents of A are

$$\begin{array}{rcl} \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) & = & \left(\begin{array}{c|c} U_L & U_R \end{array}\right)^H \left(\begin{array}{c|c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right) \\ & = & \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array}\right). \end{array}$$

This means that in the current step we need to update the contents of A_{BR} with

$$\widetilde{U}^H A \widetilde{V} = \left(\begin{array}{c|c} \sigma_{11} & 0 \\ \hline 0 & \widetilde{B} \end{array}\right)$$

and update

$$\begin{array}{rcl} \left(\begin{array}{c|c} U_L & U_R \end{array}\right) & := & \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \left(\begin{array}{c|c} I_{k\times k} & 0 \\ \hline 0 & \widetilde{U} \end{array}\right) \\ \left(\begin{array}{c|c} V_L & V_R \end{array}\right) & := & \left(\begin{array}{c|c} V_L & V_R \end{array}\right) \left(\begin{array}{c|c} I_{k\times k} & 0 \\ \hline 0 & \widetilde{V} \end{array}\right), \end{array}$$

which simplify to

$$U_{BR} := U_{BR}\widetilde{U} \text{ and } V_{BR} := V_{BR}\widetilde{V}.$$

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 $\Sigma = U^H \widehat{V}$. 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

$$\begin{array}{rcl} \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) & = & \left(\begin{array}{c|c} U_L & U_R \end{array}\right)^H \left(\begin{array}{c|c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right) \\ & = & \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array}\right) \end{array}$$

as the loop-invariant that can be used to prove the correctness of the loop via a proof by induction.

$A = \widehat{A}$

$U := I_{m\times m}; \quad V := I_{n\times n}$

$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right), U \rightarrow \left(\begin{array}{c|c} U_L & U_R \end{array}\right), V \rightarrow \left(\begin{array}{c|c} V_L & V_R \end{array}\right)$ where A_{TL} is $0 \times 0, U_L$ 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 $\widetilde{v}_1$ s.t. $\|\widetilde{v}_1\|_2 = 1$ and $\|A_{BR}\widetilde{v}_1\| = \|A_{BR}\|_2 (= \sigma_{11})$

$\widetilde{u}_1 := A_{BR}\widetilde{v}_1/\sigma_{11}$

pick $\widetilde{V}_2$ and $\widetilde{U}_2$ s.t. $\widetilde{V} = \left(\begin{array}{c|c} \widetilde{v}_1 & \widetilde{V}_2 \end{array}\right)$ and $\widetilde{U} = \left(\begin{array}{c|c} \widetilde{u}_1 & \widetilde{U}_2 \end{array}\right)$ are unitary

$V_R := V_R\widetilde{V}; \quad U_R := U_R\widetilde{U}$

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \rightarrow \left(\begin{array}{c|cc} U_0 & u_1 & U_2 \end{array}\right), \left(\begin{array}{c|c} V_L & V_R \end{array}\right) \rightarrow \left(\begin{array}{c|cc} V_0 & v_1 & V_2 \end{array}\right)$

$\left(\begin{array}{cc} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{cc} \sigma_{11} & 0 \\ 0 & \widetilde{U}_2^H A_{BR} \widetilde{V}_2 \end{array}\right)$

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \leftarrow \left(\begin{array}{cc|c} U_0 & u_1 & U_2 \end{array}\right), \left(\begin{array}{c|c} V_L & V_R \end{array}\right) \leftarrow \left(\begin{array}{cc|c} V_0 & v_1 & V_2 \end{array}\right)$

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} U_L & U_R \end{array}\right)^H \left(\begin{array}{c|c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right) = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array}\right)$

endwhile

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} U_L & U_R \end{array}\right)^H \left(\begin{array}{c|c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right) = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & B \end{array}\right) \wedge \|B\|_2 = 0$

$$\underbrace{\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)}_{A} = \underbrace{\left(\begin{array}{c|c} U_L & U_R \end{array}\right)^H}_{U^H} \underbrace{\left(\begin{array}{c|c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right)}_{\widehat{A}} \underbrace{\left(\begin{array}{c|c} V_L & V_R \end{array}\right)}_{V} = \underbrace{\left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right)}_{\Sigma}$$

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 $\widetilde{v}_1$.
- 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 $A_{22} := \widetilde{U}_2^H A_{BR} \widetilde{V}_2$ 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 implemention 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 marix $V_L \in \mathbb{C}^{n\times r}$, and matrix $\Sigma_{TL} \in \mathbb{R}^{r\times r}$ with $\Sigma_{TL} = \text{diag}(\sigma_0, \ldots, \sigma_{r-1})$ and $\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > 0$ such that $A = U_L \Sigma_{TL} V_L^H$.*

Homework 2.3.4.1 Prove the above corollary.

Solution. Let $A = U\Sigma V^H = \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right)^H$ 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 $\Sigma_{TL} = \text{diag}(\sigma_0, \sigma_1, \cdots, \sigma_{r-1})$ and $\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > 0$. Then

$$
\begin{array}{ll}
A & \\
\quad = & < \text{SVD of } A > \\
U\Sigma V^T & \\
\quad = & < \text{Partitioning} > \\
\left(\begin{array}{c|c} U_L & U_R \end{array}\right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right)^H & \\
\quad = & < \text{partitioned matrix} - \text{matrix multiplication} > \\
U_L \Sigma_{TL} V_L^H. &
\end{array}
$$

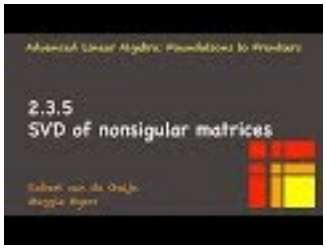
Corollary 2.3.4.2 *Let $A = U_L \Sigma_{TL} V_L^H$ be the Reduced SVD with $U_L = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{r-1} \end{array}\right)$, $V_L = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{r-1} \end{array}\right)$, and $\Sigma_{TL} = \left(\begin{array}{c|c|c} \sigma_0 & & \\ \hline & \ddots & \\ \hline & & \sigma_{r-1} \end{array}\right)$. Then*

$$A = \sigma_0 u_0 v_0^H + \cdots + \sigma_{r-1} u_{r-1} v_{r-1}^H.$$

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:

$$A = \underbrace{\sigma_0 u_0 v_0^H}_{\sigma_0 \left|\ \text{———}\right.} + \underbrace{\sigma_1 u_1 v_1^H}_{\sigma_1 \left|\ \text{———}\right.} + \cdots + \underbrace{\sigma_{r-1} u_{r-1} v_{r-1}^H}_{\sigma_{r-1} \left|\ \text{———}\right.}.$$

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

$$
\begin{array}{l}
\Sigma^{-1} \\
\quad = \quad <> \\
\left(\begin{array}{c|c|c|c} \sigma_0 & 0 & \cdots & 0 \\ \hline 0 & \sigma_1 & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \sigma_{m-1} \end{array}\right)^{-1} \\
\quad = \quad <> \\
\left(\begin{array}{c|c|c|c} 1/\sigma_0 & 0 & \cdots & 0 \\ \hline 0 & 1/\sigma_1 & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & 1/\sigma_{m-1} \end{array}\right).
\end{array}
$$

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

$$
\begin{array}{rcl}
A & = & U\Sigma V^H \\
& = & \left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array}\right) \left(\begin{array}{c|c|c} \sigma_0 & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & \sigma_{m-1} \end{array}\right) \left(\begin{array}{c|c|c} v_0 & \cdots & v_{m-1} \end{array}\right)^H
\end{array}
$$

be its SVD.

The SVD of A^{-1} is given by (indicate all correct answers):

1. $V\Sigma^{-1}U^H$.
2. $\left(\begin{array}{c|c|c} v_0 & \cdots & v_{m-1} \end{array}\right) \left(\begin{array}{c|c|c} 1/\sigma_0 & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & 1/\sigma_{m-1} \end{array}\right) \left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array}\right)^H$
3. $\left(\begin{array}{c|c|c} v_{m-1} & \cdots & v_0 \end{array}\right) \left(\begin{array}{c|c|c} 1/\sigma_{m-1} & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & 1/\sigma_0 \end{array}\right) \left(\begin{array}{c|c|c} u_{m-1} & \cdots & u_0 \end{array}\right)^H$.
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 colums 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.

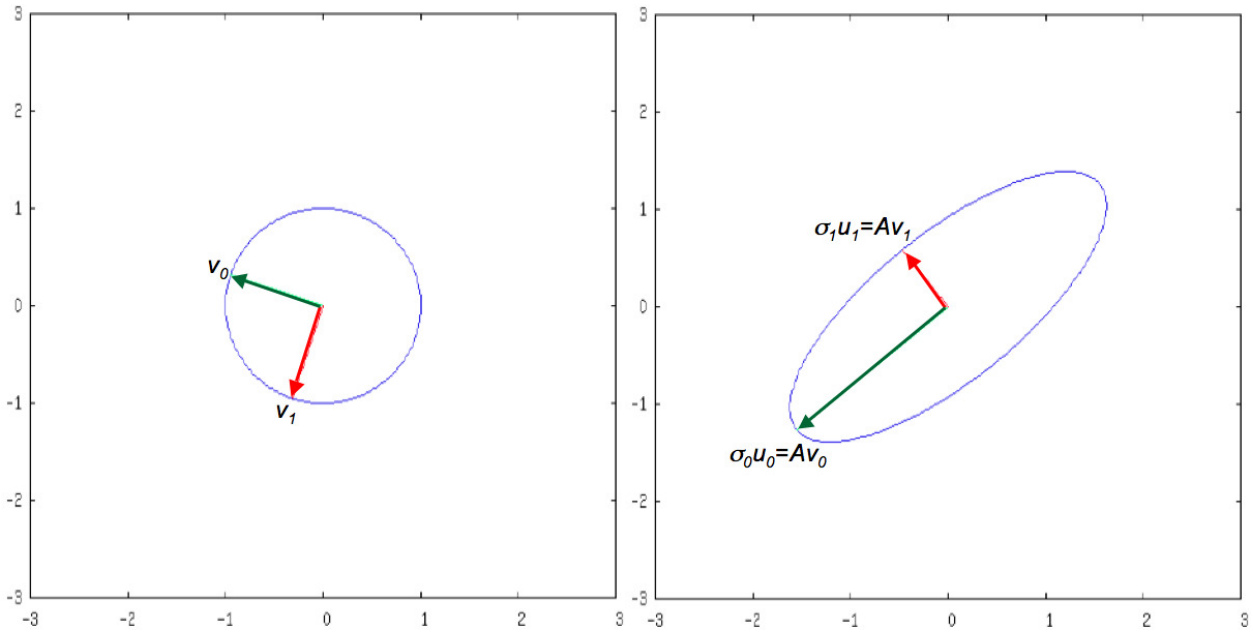
$$
\begin{array}{l}
\|A^{-1}\|_2 \\
\quad = \quad < \text{definition} > \\
\max_{x\neq 0} \frac{\|A^{-1}x\|_2}{\|x\|_2} \\
\quad = \quad < \text{algebra} > \\
\max_{x\neq 0} \frac{1}{\frac{\|x\|_2}{\|A^{-1}x\|_2}} \\
\quad = \quad < \text{algebra} > \\
\frac{1}{\min_{x\neq 0} \frac{\|x\|_2}{\|A^{-1}x\|_2}} \\
\quad = \quad < \text{substitute } z = A^{-1}x > \\
\frac{1}{\min_{Az\neq 0} \frac{\|Az\|_2}{\|z\|_2}} \\
\quad = \quad < A \text{ is nonsingular} > \\
\frac{1}{\min_{z\neq 0} \frac{\|Az\|_2}{\|z\|_2}} \\
\quad = \quad < x = z/\|z\|_2 > \\
\frac{1}{\min_{\|x\|_2=1} \|Ax\|_2}
\end{array}
$$

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

$$
A = \left(\begin{array}{c|c} u_0 & u_1 \end{array}\right) \left(\begin{array}{c|c} \sigma_0 & 0 \\ \hline 0 & \sigma_1 \end{array}\right) \left(\begin{array}{c|c} v_0 & v_1 \end{array}\right)^T
$$

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 $\sigma_0, \cdots, \sigma_{\min(m,n)-1}$ with $\sigma_0 \geq \cdots \geq \sigma_{\min(m,n)-1} \geq 0$. 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:

$$\|A - B\|_2 = \min_{\substack{C \in \mathbb{C}^{m\times n} \\ \text{rank}(C) \le k}} \|A - C\|_2.$$

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

$$Av_j = \sigma_j u_j \text{ for } 0 \le j < \min(m,n),$$

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 $n \le m$. Rewrite $A = U\Sigma V^H$ as $AV = U\Sigma$. Then

$$\begin{array}{l} AV = U\Sigma \quad = \quad < \text{ partition } > \\ A \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right) \\ \qquad = \left(\begin{array}{c|c|c|c|c} u_0 & \cdots & u_{n-1} & u_n & \cdots & u_{m-1} \end{array} \right) \left(\begin{array}{c|c|c} \sigma_0 & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & \sigma_{n-1} \\ \hline 0 & \cdots & 0 \\ \hline \vdots & & \vdots \\ \hline 0 & & 0 \end{array} \right) \\ \quad = \quad < \text{ multiply out } > \\ \left(\begin{array}{c|c|c} Av_0 & \cdots & Av_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c} \sigma_0 u_0 & \cdots & \sigma_{n-1} u_{n-1} \end{array} \right). \end{array}$$

Hence $Av_j = \sigma_j u_j$ for $0 \le j < n$.

Proof of Theorem 2.3.6.1. First, if B is as defined, then $\|A - B\|_2 = \sigma_k$:

$$\begin{array}{l} \|A - B\|_2 \\ \quad = \quad < \text{ multiplication with unitary matrices preserves 2-norm } > \\ \|U^H(A - B)V\|_2 \\ \quad = \quad < \text{ distribute } > \\ \|U^H AV - U^H BV\|_2 \\ \quad = \quad < \text{ use SVD of } A \text{ and partition } > \\ \left\| \Sigma - \left(\begin{array}{c|c} U_L & U_R \end{array} \right)^H B \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \right\|_2 \\ \quad = \quad < \text{ how } B \text{ was chosen } > \\ \left\| \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right) - \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right) \right\|_2 \\ \quad = \quad < \text{ partitioned subtraction } > \\ \left\| \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right) \right\|_2 \\ \quad = \quad <> \\ \|\Sigma_{BR}\|_2 \\ \quad = \quad < \Sigma_{TL} \text{ is } k \times k > \\ \sigma_k \end{array}$$

(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 $j = 0, \ldots, k$.
- Now, let y be any linear combination of $v_0, \ldots, v_k$: $y = \alpha_0 v_0 + \cdots + \alpha_k v_k$. 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
$$\begin{array}{ll} & \|Ay\|_2^2 \\ = & < y = \alpha_0 v_0 + \cdots + \alpha_k v_k > \\ & \|A(\alpha_0 v_0 + \cdots + \alpha_k v_k)\|_2^2 \\ = & < \text{distributivity} > \\ & \|\alpha_0 A v_0 + \cdots + \alpha_k A v_k\|_2^2 \\ = & < A v_j = \sigma_j u_j > \\ & \|\alpha_0 \sigma_0 u_0 + \cdots + \alpha_k \sigma_k u_k\|_2^2 \\ = & < \text{ this works because the } u_j \text{ are orthonormal } > \\ & \|\alpha_0 \sigma_0 u_0\|_2^2 + \cdots + \|\alpha_k \sigma_k u_k\|_2^2 \\ = & < \text{ norms are homogeneous and } \|u_j\|_2 = 1 > \\ & |\alpha_0|^2 \sigma_0^2 + \cdots + |\alpha_k|^2 \sigma_k^2 \\ \geq & < \sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_k \geq 0 > \\ & (|\alpha_0|^2 + \cdots + |\alpha_k|^2)\sigma_k^2 \\ = & < \|y\|_2^2 = |\alpha_0|^2 + \cdots + |\alpha_k|^2 > \\ & \sigma_k^2 \|y\|_2^2. \end{array}$$
so that $\|Ay\|_2 \geq \sigma_k\|y\|_2$. In other words, vectors in the subspace of all linear combinations of $\{v_0, \ldots, v_k\}$ satisfy $\|Ax\|_2 \geq \sigma_k\|x\|_2$. The dimension of this subspace is $k + 1$ (since $\{v_0, \cdots, v_k\}$ 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.

■

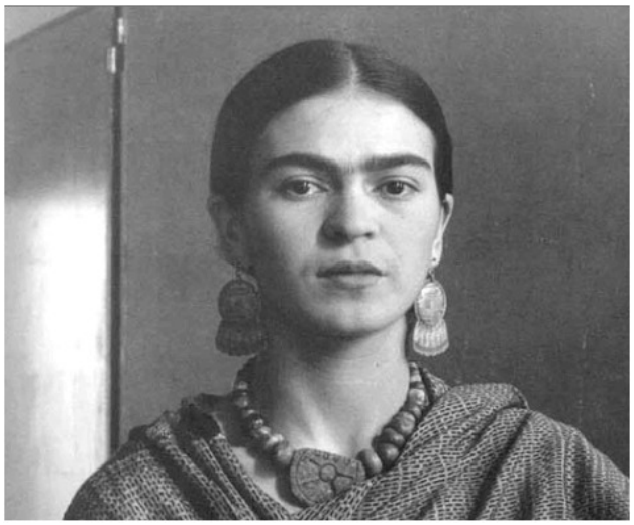
Theorem 2.3.6.1 tells us how to pick the best approximation to a given matrix of a given desired rank. In Section Subsection 2.1.1 we discussed how a low rank matrix can be used to compress data. The SVD thus gives the best such rank-k approximation. Let us revisit this.

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



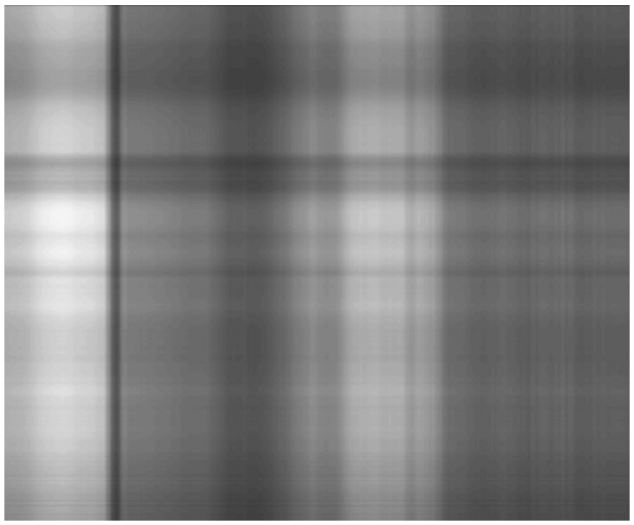
Although the picture is black and white, it was read as if it is a color image, which means a $m \times n \times 3$ array of pixel information is stored. Setting A = IMG(:,:,1) extracts a single matrix of pixel information. (If you start with a color picture, you will want to approximate IMG(:,:,1), IMG(:,:,2), and 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.

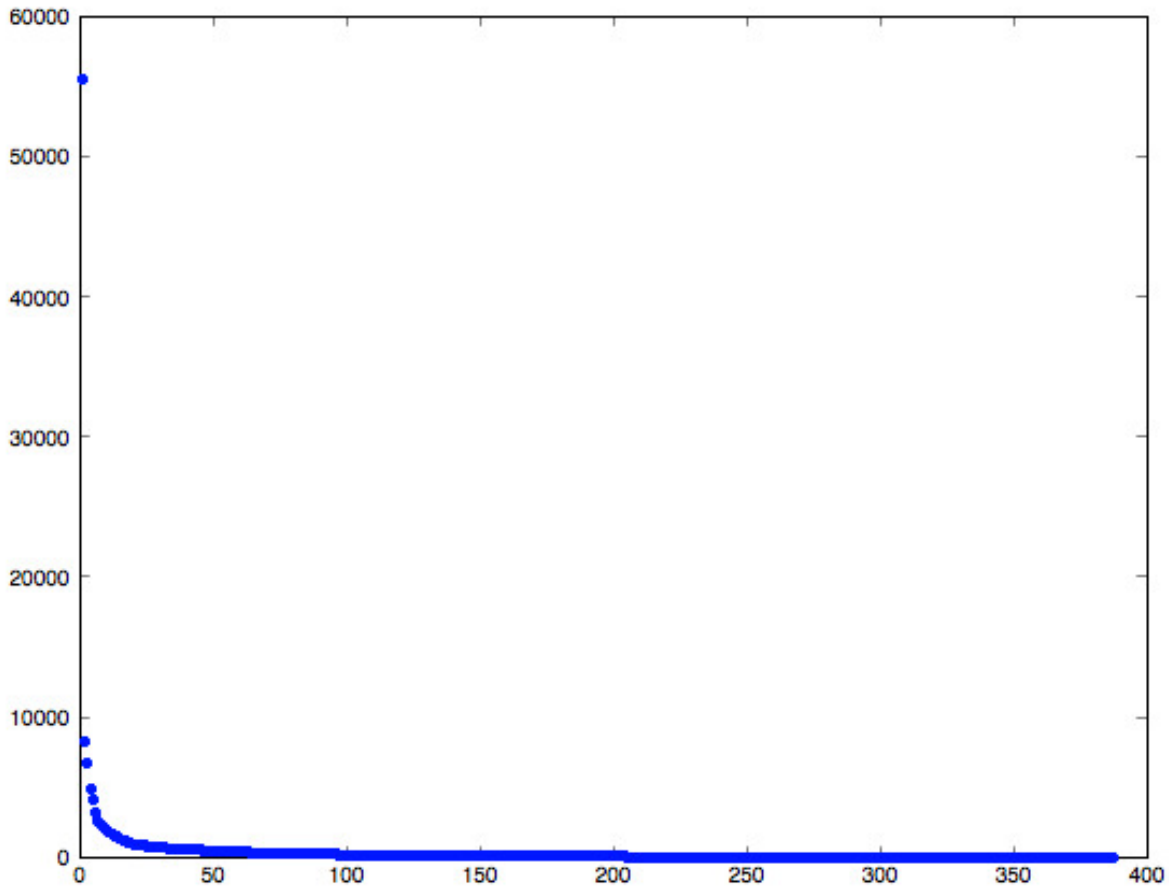
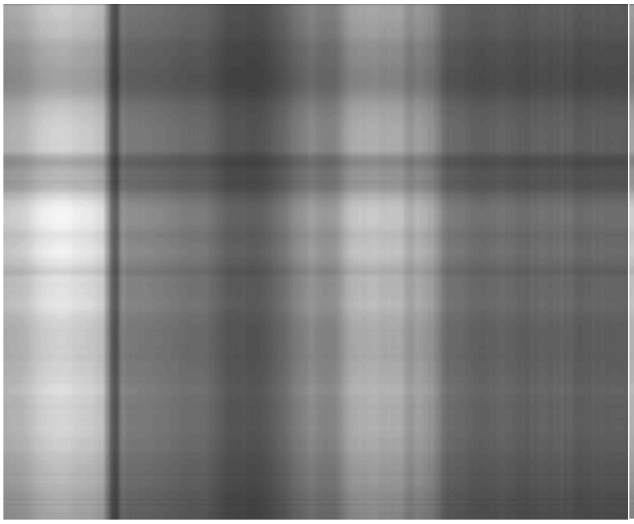
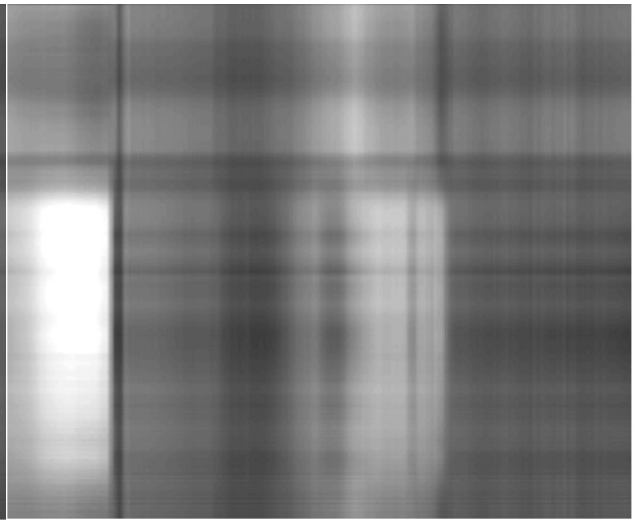


Figure 2.3.6.2 Distribution of singular values for the picture.

k = 1



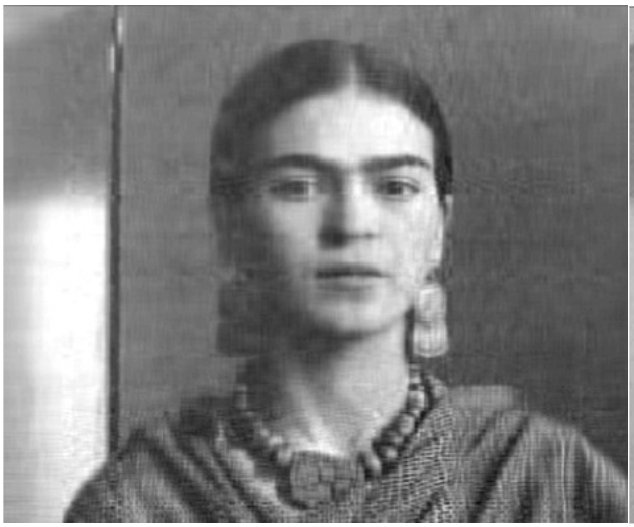
$k = 2$



$k = 5$

$k = 10$

$k = 25$



Original picture

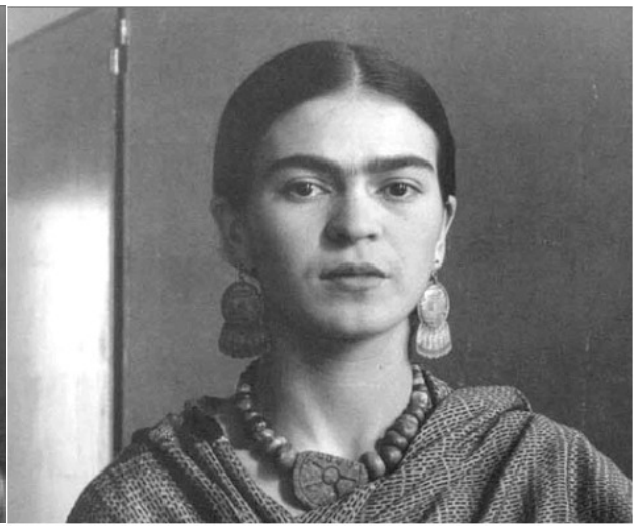


Figure 2.3.6.3 Multiple pictures as generated by the code.

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 $(Ux)^H(Uy) = x^Hy$ 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 $U^HBV = A$.

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 $\left(\begin{array}{c|c} U & 0 \\ \hline 0 & V \end{array}\right)$ 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: $\sum_{0\le i<m} \alpha_{i,j} = 1$. 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 $\|\cdots\|$ is a norm and A is nonsingular, then $\|\cdots\|_{A^{-1}}$ 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

$$\Sigma = \left(\begin{array}{c|c|c|c} \sigma_0 & 0 & \cdots & 0 \\ \hline 0 & \sigma_1 & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \sigma_{m-1} \end{array}\right)$$

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

$$x^H y = \overline{x}^T y = \overline{x^T} y = \overline{\chi}_0\psi_0 + \overline{\chi}_1\psi_1 + \cdots + \overline{\chi}_{m-1}\psi_{m-1} = \sum_{i=0}^{m-1} \overline{\chi}_i\psi_i.$$

- 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

$$\frac{x x^H}{x^H x}.$$

- The component of y orthogonal to x is given by

$$y - \frac{x^H y}{x^H x} x = \left(I - \frac{x x^H}{x^H x} \right) y.$$

  Thus, the matrix that projects a vector onto the space orthogonal to x is given by

$$I - \frac{x x^H}{x^H x}.$$

Given $u, v \in \mathbb{C}^m$ with u of unit length

- The component of v in the direction of u is given by

$$u^H v u = u u^H v.$$

- 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 $u_0, u_1, \ldots, u_{n-1} \in \mathbb{C}^m$. 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 $Q = \left(\begin{array}{c|c|c} q_0 & \cdots & q_{n-1} \end{array} \right)$ is orthonormal iff $\{q_0, \ldots, q_{n-1}$ 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). ◊

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 $\Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right)$ 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 $\sigma_0, \ldots, \sigma_{r-1}$ 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

$$U = \left(\begin{array}{c|c|c} U_L & \vert & U_R \end{array} \right) = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right),$$

$$V = \left(\begin{array}{c|c|c} V_L & \vert & V_R \end{array} \right) = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right),$$

and

$$\Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right), \text{ where } \Sigma_{TL} = \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{r-1} \end{pmatrix} \quad \text{and} \quad \sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > 0.$$

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: $A = U_L \Sigma_{TL} V_L^H$.
- $$A = \underbrace{\sigma_0 u_0 v_0^H}_{\sigma_0 \left| \begin{array}{c} \\ \end{array} \right.} + \underbrace{\sigma_1 u_1 v_1^H}_{\sigma_1 \left| \begin{array}{c} \\ \end{array} \right.} + \cdots + \underbrace{\sigma_{r-1} u_{r-1} v_{r-1}^H}_{\sigma_{r-1} \left| \begin{array}{c} \\ \end{array} \right.}.$$
- 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 is the direction of minimal magnification.
- If $n \leq m$, then $Av_j = \sigma_j u_j$, 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 $\sigma_0, \cdots, \sigma_{\min(m,n)-1}$ with $\sigma_0 \geq \cdots \geq \sigma_{\min(m,n)-1} \geq 0$. 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, $f : \mathbb{R} \to \mathbb{R}$, 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 $\chi_0, \ldots, \chi_{m-1}$:

$$\begin{array}{rcl} f(\chi_0) & = & \phi_0 \\ \vdots & \vdots & \vdots \\ f(\chi_{m-1}) & = & \phi_{m-1}. \end{array}$$

In other words, input to the process are the points

$$(\chi_0, \phi_0), \cdots, (\chi_{m-1}, \phi_{m-1})$$

and we want to determine the polynomal that approximately fits these points. This means that

$$\begin{array}{ccccccccc} \gamma_0 & + & \gamma_1 \chi_0 & + & \cdots & + & \gamma_{n-1} \chi_0^{n-1} & \approx & \phi_0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ \gamma_0 & + & \gamma_1 \chi_{m-1} & + & \cdots & + & \gamma_{n-1} \chi_{m-1}^{n-1} & \approx & \phi_{m-1}. \end{array}$$

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 $\chi_0, \chi_1, \cdots, \chi_{m-1}$ to be equally spaced in the interval $[0, 1]$: for $i = 0, \ldots, m-1$, $\chi_i = ih$,where $h = 1/(m-1)$. Write Matlab code to create the matrix

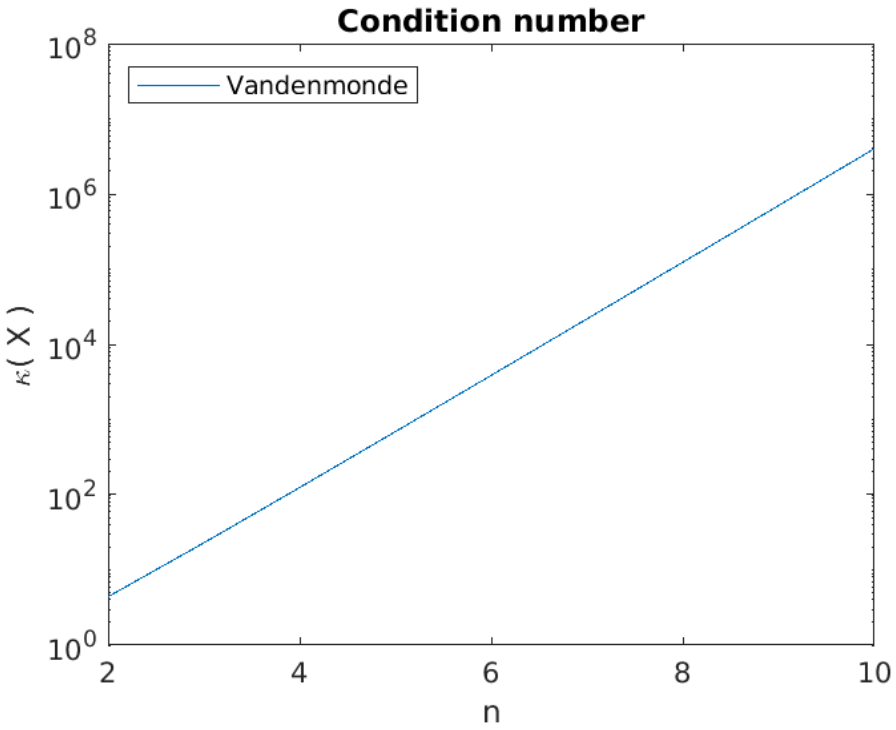
$$X = \begin{pmatrix} 1 & \chi_0 & \cdots & \chi_0^{n-1} \\ 1 & \chi_1 & \cdots & \chi_1^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \chi_{m-1} & \cdots & \chi_{m-1}^{n-1} \end{pmatrix}$$

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).)

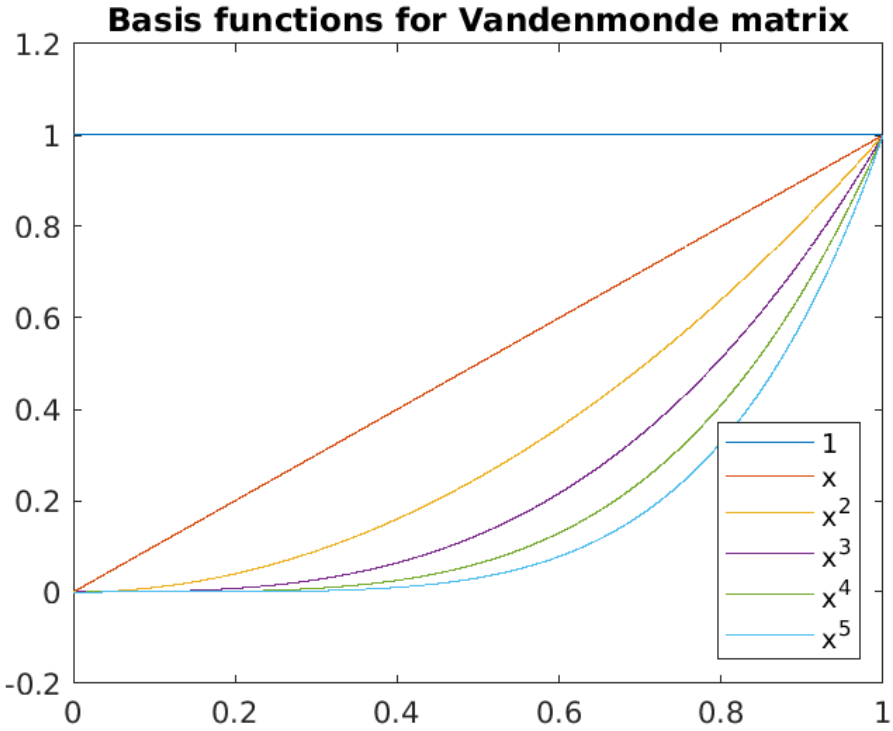
Hint. You may want to use the recurrence $x^{j+1} = xx^j$ and the fact that the $.*$ operator in Matlab performs an element-wise multiplication.

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** $1, x, x^2, \ldots$ 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

$$\begin{array}{rcl} P_0(\chi) & = & 1 \\ P_1(\chi) & = & 2\chi - 1 \\ \vdots & = & \vdots \\ P_{n+1}(\chi) & = & \left((2n+1)(2\chi-1)P_n(\chi) - nP_{n-1}(\chi)\right)/(n+1). \end{array}$$

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

$$(\chi_0, \phi_0), \cdots, (\chi_{m-1}, \phi_{m-1})$$

we can determine the polynomial from

$$\begin{array}{ccccccccc} \gamma_0 P_0(\chi_0) & + & \gamma_1 P_1(\chi_0) & + & \cdots & + & \gamma_{n-1} P_{n-1}(\chi_0) & = & \phi_0 \\ \vdots & & \vdots & & \vdots & & \vdots & \vdots & \vdots \\ \gamma_0 P_0(\chi_{m-1}) & + & \gamma_1 P_1(\chi_{m-1}) & + & \cdots & + & \gamma_{n-1} P_{n-1}(\chi_{m-1}) & = & \phi_{m-1}. \end{array}$$

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 $\chi_0, \ldots, \chi_{n-1}$, 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.

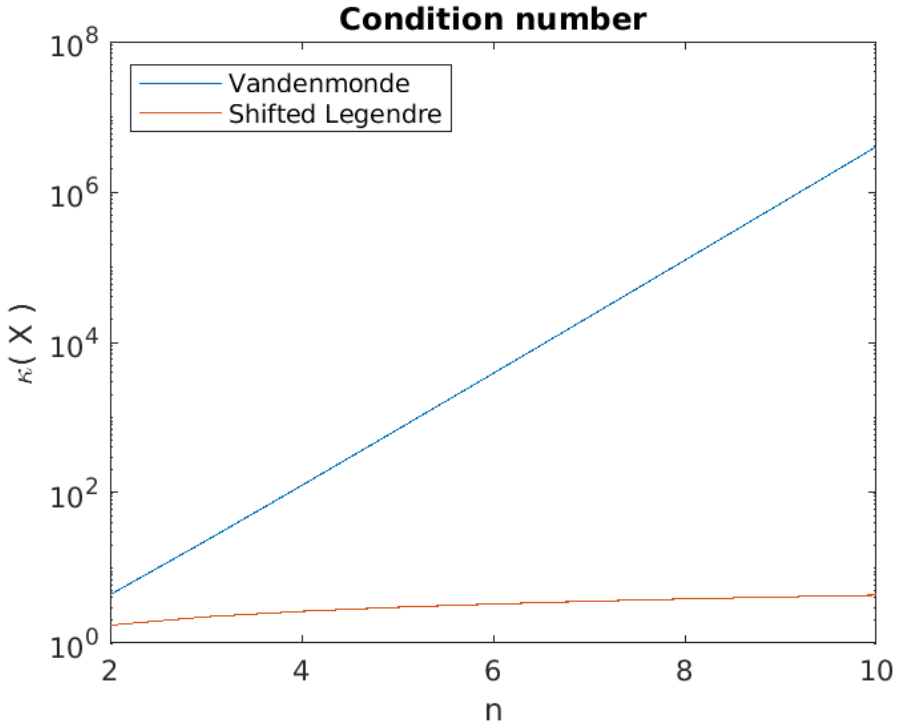
Homework 3.1.1.2 Choose $\chi_0, \chi_1, \cdots, \chi_{m-1}$ to be equally spaced in the interval $[0, 1]$: for $i = 0, \ldots, m-1$, $\chi_i = ih$, where $h = 1/(m-1)$. Write Matlab code to create the matrix

$$X = \begin{pmatrix} 1 & P_1(\chi_0) & \cdots & P_{n-1}(\chi_0) \\ 1 & P_1(\chi_1) & \cdots & P_{n-1}(\chi_1) \\ \vdots & \vdots & & \vdots \\ 1 & P(\chi_{m-1}) & \cdots & P_{n-1}(\chi_{m-1}) \end{pmatrix}$$

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 $X^T X$.

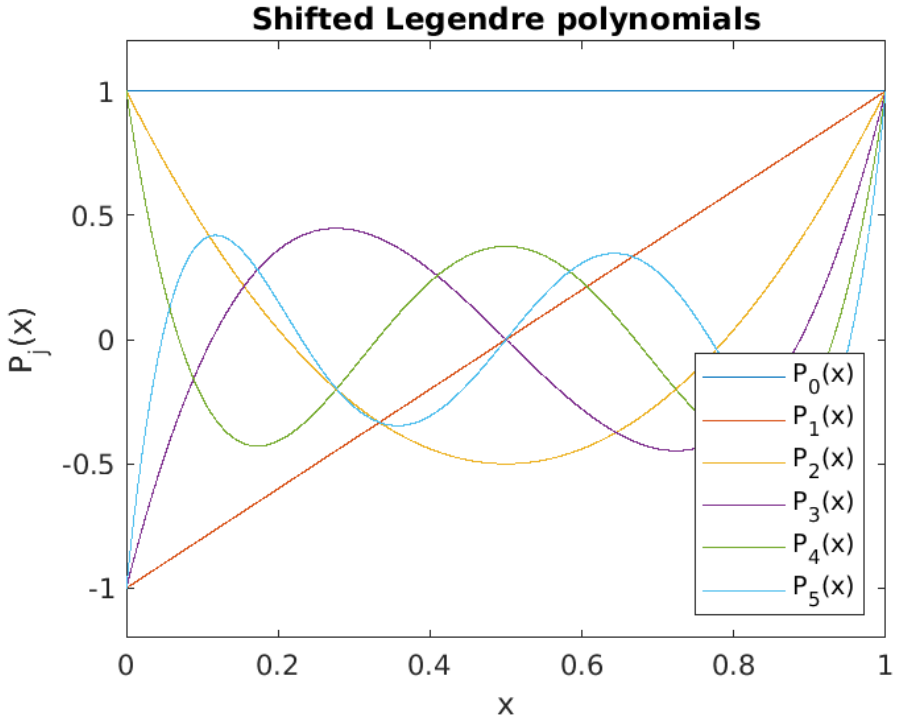
Solution.

- 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 =

        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
```

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

3.1.3 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 $\{a_0, \ldots, a_{n-1}\} \subset \mathbb{C}^m$, the Gram-Schmidt process computes an orthonormal basis $\{q_0, \ldots, q_{n-1}\}$ that spans the same subspace as the original vectors, i.e.

$$\text{Span}(\{a_0, \ldots, a_{n-1}\}) = \text{Span}(\{q_0, \ldots, q_{n-1}\}).$$

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 $\text{Span}(\{a_0, a_1, a_2\}) = \text{Span}(\{q_0, q_1, q_2\})$:
  - $\begin{array}{l} \rho_{0,2} = q_0^H a_2 \\ \rho_{1,2} = q_1^H a_2 \end{array}$ 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
  $$\left(\begin{array}{c|c} a_0 & a_1 \end{array}\right) = \left(\begin{array}{c|c} q_0 & q_1 \end{array}\right) \left(\begin{array}{cc|c} \rho_{0,0} & \rho_{0,1} & \rho_{0,2} \\ 0 & \rho_{1,1} & \rho_{1,2} \\ \hline 0 & 0 & \rho_{2,2} \end{array}\right).$$

- 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

$$Q = \left(\begin{array}{c|c|c|c|c|c|c} q_0 & \cdots & q_{k-1} & q_k & q_{k+1} & \cdots & q_{n-1} \end{array} \right)$$

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, $\text{Span}(\{a_0, \ldots, a_{k-1}, a_k\}) = \text{Span}(\{q_0, \ldots, q_{k-1}, q_k\})$
  Hence
  $$a_k = \rho_{0,k} q_0 + \cdots + \rho_{k-1,k} q_{k-1} + \rho_{k,k} q_k$$
  for some scalars $\rho_{0,k}, \cdots, \rho_{k,k}$.

Let's assume that $q_0, \ldots, q_{k-1}$ 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

$$\begin{array}{rcl} q_k^H a_k & = & q_k^H \left(\rho_{0,k} q_0 + \cdots + \rho_{k-1,k} q_{k-1} + \rho_{k,k} q_k \right) \\ & = & \rho_{0,k} \underbrace{q_k^H q_0}_{0} + \cdots + \rho_{k-1,k} \underbrace{q_k^H q_{k-1}}_{0} + \rho_{k,k} \underbrace{q_k^H q_k}_{1} \end{array}$$

so that

$$\rho_{i,k} = q_i^H a_k,$$

for $i = 0, \ldots, k-1$. 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 $\{a_0, \ldots, a_j\}$ 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

$$\left(\begin{array}{c|ccc} A_0 & a_1 & A_2 \end{array}\right) = \left(\begin{array}{c|ccc} Q_0 & q_1 & Q_2 \end{array}\right) \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array}\right).$$

- Assume that Q_0 and R_{00} have already been computed.

- Since corresponding columns of both sides must be equal, we find that

$$a_1 = Q_0 r_{01} + q_1 \rho_{11}. \tag{3.2.1}$$

  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.

$[Q, R] = \text{CGS-QR}(A)$

$$A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array}\right), Q \rightarrow \left(\begin{array}{c|c} Q_L & Q_R \end{array}\right), R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right)$$

A_L and Q_L has 0 columns and R_{TL} is 0×0

while $n(A_L) < n(A)$

$$\left(\begin{array}{c|c} A_L & A_R \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_0 & a_1 & A_2 \end{array}\right), \left(\begin{array}{c|c} Q_L & Q_R \end{array}\right) \rightarrow \left(\begin{array}{c|cc} Q_0 & q_1 & Q_2 \end{array}\right),$$

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ 0 & 0 & R_{22} \end{array}\right)$$

$r_{01} := Q_0^H a_1$

$a_1^\perp := a_1 - Q_0 r_{01}$

$\rho_{11} := \|a_1^\perp\|_2$

$q_1 := a_1^\perp / \rho_{11}$

$$\left(\begin{array}{c|c} A_L & A_R \end{array}\right) \leftarrow \left(\begin{array}{c|cc} A_0 & a_1 & A_2 \end{array}\right), \left(\begin{array}{c|c} Q_L & Q_R \end{array}\right) \leftarrow \left(\begin{array}{c|cc} Q_0 & q_1 & Q_2 \end{array}\right),$$

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array}\right)$$

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 $A = \left(\begin{array}{c|c} A_0 & a_1 \end{array} \right)$, where A_0 has no columns. Since A has linearly independent columns, $a_1 \neq 0$. Then

$$A = \left(\begin{array}{c} a_1 \end{array} \right) = (q_1)\,(\rho_{11})\,,$$

  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 $A \rightarrow \left(\begin{array}{c|c} A_0 & a_1 \end{array} \right)$.

  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

$$\left(\begin{array}{c|c} \widetilde{Q}_0 & q_1 \end{array} \right) \text{ and } \left(\begin{array}{c|c} \widetilde{R}_{00} & r_{01} \\ \hline 0 & \rho_{11} \end{array} \right)$$

  so that

$$\left(\begin{array}{c|c} A_0 & a_1 \end{array} \right) = \left(\begin{array}{c|c} \widetilde{Q}_0 & q_1 \end{array} \right) \left(\begin{array}{c|c} \widetilde{R}_{00} & r_{01} \\ \hline 0 & \rho_{11} \end{array} \right).$$

  This means that

  - $A_0 = \widetilde{Q}_0 \widetilde{R}_{00}$,
    We choose $\widetilde{Q}_0 = Q_0$ and $\widetilde{R}_{00} = R_{00}$. 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 $j = 0, \ldots, n-1$
  $a_j^\perp := a_j$
  for $k = 0, \ldots, j-1$
    $\rho_{k,j} := q_k^H a_j^\perp$
    $a_j^\perp := a_j^\perp - \rho_{k,j} q_k$
  end
  $\rho_{j,j} := \|a_j^\perp\|_2$
  $q_j := a_j^\perp / \rho_{j,j}$
end

(a) MGS algorithm that computes Q and R from A.

for $j = 0, \ldots, n-1$

  for $k = 0, \ldots, j-1$
    $\rho_{k,j} := a_k^H a_j$
    $a_j := a_j - \rho_{k,j} a_k$
  end
  $\rho_{j,j} := \|a_j\|_2$
  $a_j := a_j / \rho_{j,j}$
end

(b) MGS algorithm that computes Q and R from A, overwriting A with Q.

Homework 3.2.4.1 Assume that $q_0, \ldots, q_{k-1}$ are mutually orthonormal. Let $\rho_{j,k} = q_j^H y$ for $j = 0, \ldots, i-1$. Show that

$$\underbrace{q_i^H y}_{\rho_{i,k}} = q_i^H (y - \rho_{0,k} q_0 - \cdots - \rho_{i-1,k} q_{i-1})$$

for $i = 0, \ldots, k-1$.

Solution.

$$\begin{array}{l} q_i^H (y - \rho_{0,k} q_0 - \cdots - \rho_{i-1,k} q_{i-1}) \\ \quad = \quad < \text{ distribute } > \\ q_i^H y - q_i^H \rho_{0,k} q_0 - \cdots - q_i^H \rho_{i-1,k} q_{i-1} \\ \quad = \quad < \rho_{0,k} \text{ is a scalar } > \\ q_i^H y - \rho_{0,k} \underbrace{q_i^H q_0}_{0} - \cdots - \underbrace{\rho_{i-1,k} q_i^H q_{i-1}}_{0} \end{array}$$

YouTube: https://www.youtube.com/watch?v=0ooNPondq5M

This homework illustrates how, given a vector $y \in \mathbb{C}^m$ and a matrix $Q \in \mathbb{C}^{mxk}$ 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, 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 left successfuly 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 $q_0, \ldots, q_{i-1}$). The algorithm on the right is one variant of the Modified Gram-Schmidt (MGS) algorithm.

$[y^\perp, r] = \text{Proj}\perp\text{Q}_{\text{CGS}}(Q, y)$ (used by CGS)	$[y^\perp, r] = \text{Proj}\perp\text{Q}_{\text{MGS}}(Q, y)$ (used by MGS)
$y^\perp = y$ **for** $i = 0, \ldots, k-1$ $\quad \rho_i := q_i^H y$ $\quad y^\perp := y^\perp - \rho_i q_i$ **endfor**	$y^\perp = y$ **for** $i = 0, \ldots, k-1$ $\quad \rho_i := q_i^H y^\perp$ $\quad y^\perp := y^\perp - \rho_i q_i$ **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)

$$A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array}\right), R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right)$$

A_L has 0 columns and R_{TL} is 0×0

while $n(A_L) < n(A)$

$$\left(\begin{array}{c|c} A_L & A_R \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_0 & a_1 & A_2 \end{array}\right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array}\right)$$

CGS	MGS	MGS (alternative)
$r_{01} := A_0^H a_1$ $a_1 := a_1 - A_0 r_{01}$ $\rho_{11} := \|a_1\|_2$ $a_1 := a_1/\rho_{11}$	$[a_1, r_{01}] = \text{Proj}\perp\text{toQ}_{\text{MGS}}(A_0, a_1)$ $\rho_{11} := \|a_1\|_2$ $a_1 := a_1/\rho_{11}$	$\rho_{11} := \|a_1\|_2$ $a_1 := a_1/\rho_{11}$ $r_{12}^T := a_1^H A_2$ $A_2 := A_2 - a_1 r_{12}^T$

$$\left(\begin{array}{c|c} A_L & A_R \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_0 & a_1 & A_2 \end{array}\right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array}\right)$$

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 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, $\{a_{k+1}, \ldots, a_{n-1}\}$, 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 $j = 0, \ldots, n-1$
  $\rho_{j,j} := \|a_j\|_2$
  $a_j := a_j / \rho_{j,j}$
  for $k = j+1, \ldots, n-1$
    $\rho_{j,k} := a_j^H a_k$
    $a_k := a_k - \rho_{j,k} a_j$
  end
end

(c) MGS algorithm that normalizes the jth 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 $(a_{j+1}, \ldots, a_{n-1})$.

for $j = 0, \ldots, n-1$
  $\rho_{j,j} := \|a_j\|_2$
  $a_j := a_j / \rho_{j,j}$
  for $k = j+1, \ldots, n-1$
    $\rho_{j,k} := a_j^H a_k$
  end
  for $k = j+1, \ldots, n-1$
    $a_k := a_k - \rho_{j,k} a_j$
  end
end

(d) Slight modification of the algorithm in (c) that computes $\rho_{j,k}$ in a separate loop.

for $j = 0, \ldots, n-1$
  $\rho_{j,j} := \|a_j\|_2$
  $a_j := a_j / \rho_{j,j}$
  $\left(\begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right) :=$
    $a_j^H \left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)$
  $\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right) :=$
    $\left(\begin{array}{c|c|c} a_{j+1} - \rho_{j,j+1} a_j & \cdots & a_{n-1} - \rho_{j,n-1} a_j \end{array} \right)$
end

(e) Algorithm in (d) rewritten to expose only the outer loop.

for $j = 0, \ldots, n-1$
  $\rho_{j,j} := \|a_j\|_2$
  $a_j := a_j / \rho_{j,j}$
  $\left(\begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right) :=$
    $a_j^H \left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)$
  $\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right) :=$
    $\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)$
      $- a_j \left(\begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right)$
end

(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 $j = 0, \ldots, n-1$

$\rho_{j,j} := \|a_j\|_2 \qquad (\rho_{11} := \|a_1^\perp\|_2)$

$a_j := a_j / \rho_{j,j} \qquad (a_1 := a_1 / \rho_{11})$

$$\overbrace{\left(\begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right)}^{r_{12}^T} := \underbrace{a_j^H}_{a_1^H} \underbrace{\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)}_{A_2}$$

$$\overbrace{\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)}^{A_2} := \overbrace{\left(\begin{array}{c|c|c} a_{j+1} & \cdots & a_{n-1} \end{array} \right)}^{A_2} - \underbrace{a_j}_{a_1} \underbrace{\left(\begin{array}{c|c|c} \rho_{j,j+1} & \cdots & \rho_{j,n-1} \end{array} \right)}_{r_{12}^T}$$

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 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,

$$\begin{array}{rcl} \left(\begin{array}{c|c} A_L & A_R \end{array} \right) & = & \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right) \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \\ & = & \left(\begin{array}{c|c} Q_L R_{TL} & Q_L R_{TR} + Q_R R_{BR} \end{array} \right). \end{array}$$

Hence

$$A_L = Q_L R_{TL} \quad \text{and} \quad A_R = Q_L R_{TR} + Q_R R_{BR}. \tag{3.2.2}$$

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 $\widehat{A}$ 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.
   - $A_R = A_R - Q_L R_{TR}$.

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 presense 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

$$A = \left(\begin{array}{c|c|c} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{array}\right) = \left(\begin{array}{c|c|c} a_0 & a_1 & a_2 \end{array}\right). \tag{3.2.3}$$

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

First iteration

$\rho_{0,0} = \|a_0\|_2 = \sqrt{1+\varepsilon^2} = \sqrt{1+\varepsilon_{\text{mach}}}$

which is rounded to 1.

$$q_0 = a_0/\rho_{0,0} = \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} /1 = \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix}$$

Second iteration

$\rho_{0,1} = q_0^H a_1 = 1$

$$a_1^\perp = a_1 - \rho_{0,1} q_0 = \begin{pmatrix} 0 \\ -\varepsilon \\ \varepsilon \\ 0 \end{pmatrix}$$

$\rho_{1,1} = \|a_1^\perp\|_2 = \sqrt{2\varepsilon^2} = \sqrt{2}\varepsilon$

$$q_1 = a_1^\perp/\rho_{1,1} = \begin{pmatrix} 0 \\ -\varepsilon \\ \varepsilon \\ 0 \end{pmatrix} /(\sqrt{2}\varepsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}$$

Third iteration

$\rho_{0,2} = q_0^H a_2 = 1$

$\rho_{1,2} = q_1^H a_2 = 0$

$$a_2^\perp = a_2 - \rho_{0,2} q_0 - \rho_{1,2} q_1 = \begin{pmatrix} 0 \\ -\varepsilon \\ 0 \\ \varepsilon \end{pmatrix}$$

$\rho_{2,2} = \|a_2^\perp\|_2 = \sqrt{2\varepsilon^2} = \sqrt{2}\varepsilon$

$$q_2 = a_2^\perp/\rho_{2,2} = \begin{pmatrix} 0 \\ -\varepsilon \\ 0 \\ \varepsilon \end{pmatrix} /(\sqrt{2}\varepsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ 0 \\ \frac{\sqrt{2}}{2} \end{pmatrix}$$

MGS

First iteration

$\rho_{0,0} = \|a_0\|_2 = \sqrt{1+\varepsilon^2} = \sqrt{1+\varepsilon_{\text{mach}}}$

which is rounded to 1.

$$q_0 = a_0/\rho_{0,0} = \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} /1 = \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix}$$

Second iteration

$\rho_{0,1} = q_0^H a_1 = 1$

$$a_1^\perp = a_1 - \rho_{0,1} q_0 = \begin{pmatrix} 0 \\ -\varepsilon \\ \varepsilon \\ 0 \end{pmatrix}$$

$\rho_{1,1} = \|a_1^\perp\|_2 = \sqrt{2\varepsilon^2} = \sqrt{2}\varepsilon$

$$q_1 = a_1^\perp/\rho_{1,1} = \begin{pmatrix} 0 \\ -\varepsilon \\ \varepsilon \\ 0 \end{pmatrix} /(\sqrt{2}\varepsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}$$

Third iteration

$\rho_{0,2} = q_0^H a_2 = 1$

$$a_2^\perp = a_2 - \rho_{0,2} q_0 = \begin{pmatrix} 0 \\ -\varepsilon \\ 0 \\ \varepsilon \end{pmatrix}$$

$\rho_{1,2} = q_1^H a_2^\perp = (\sqrt{2}/2)\varepsilon$

$$a_2^\perp = a_2^\perp - \rho_{1,2} q_1 = \begin{pmatrix} 0 \\ -\varepsilon/2 \\ -\varepsilon/2 \\ \varepsilon \end{pmatrix}$$

$\rho_{2,2} = \|a_2^\perp\|_2 = \sqrt{(6/4)\varepsilon^2} = (\sqrt{6}/2)\varepsilon$

$$q_2 = a_2^\perp/\rho_{2,2} = \begin{pmatrix} 0 \\ -\frac{\varepsilon}{2} \\ -\frac{\varepsilon}{2} \\ \varepsilon \end{pmatrix} /(\frac{\sqrt{6}}{2}\varepsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{6}}{6} \\ -\frac{\sqrt{6}}{6} \\ \frac{\sqrt{6}}{3} \end{pmatrix}$$

Click here to enlarge.

CGS yields the approximate matrix

$$Q \approx \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{array}\right)$$

while MGS yields

$$Q \approx \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & 0 & \frac{\sqrt{6}}{3} \end{array}\right)$$

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:

$$\begin{array}{l} Q^H Q \\ = \\ \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{array}\right)^H \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{array}\right) \\ = \\ \left(\begin{array}{ccc} 1+\epsilon_{\text{mach}} & -\frac{\sqrt{2}}{2}\epsilon & -\frac{\sqrt{2}}{2}\epsilon \\ -\frac{\sqrt{2}}{2}\epsilon & 1 & \frac{1}{2} \\ -\frac{\sqrt{2}}{2}\epsilon & \frac{1}{2} & 1 \end{array}\right). \end{array}$$

  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:

$$\begin{array}{l} Q^H Q \\ = \\ \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & 0 & \frac{\sqrt{6}}{3} \end{array}\right)^H \left(\begin{array}{c|c|c} 1 & 0 & 0 \\ \epsilon & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} \\ 0 & 0 & \frac{\sqrt{6}}{3} \end{array}\right) \\ = \\ \left(\begin{array}{ccc} 1+\epsilon_{\text{mach}} & -\frac{\sqrt{2}}{2}\epsilon & -\frac{\sqrt{6}}{6}\epsilon \\ -\frac{\sqrt{2}}{2}\epsilon & 1 & 0 \\ -\frac{\sqrt{6}}{6}\epsilon & 0 & 1 \end{array}\right). \end{array}$$

  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-eVMSo

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 kth iteration ($0 \le k < n$), 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)

$$
\begin{array}{l}
\sum_{k=0}^{n-1} [2mk + 2mk + 2m + m] \\
\quad = \\
\sum_{k=0}^{n-1} [3m + 4mk] \\
\quad = \\
3mn + 4m \sum_{k=0}^{n-1} k \\
\quad \approx \quad < \sum_{k=0}^{n-1} k = n(n-1)/2 \approx n^2/2 > \\
3mn + 4m\frac{n^2}{2} \\
\quad = \\
3mn + 2mn^2 \\
\quad \approx \quad < 3mn \text{ is of lower order} > \\
2mn^2
\end{array}
$$

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 kth iteration ($0 \le k < n$), 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)

$$
\begin{array}{l}
\sum_{k=0}^{n-1} [2m(n-k-1) + 2m(n-k-1) + 2m + m] \\
\quad = \\
\sum_{k=0}^{n-1} [3m + 4m(n-k-1)] \\
\quad = \\
3mn + 4m \sum_{k=0}^{n-1} (n-k-1) \\
\quad = \quad < \text{Substitute } j = (n-k-1) > \\
3mn + 4m \sum_{j=0}^{n-1} j \\
\quad \approx \quad < \sum_{j=0}^{n-1} j = n(n-1)/2 \approx n^2/2 > \\
3mn + 4m \frac{n^2}{2} \\
\quad = \\
3mn + 2mn^2 \\
\quad \approx \quad < 3mn \text{ is of lower order} > \\
2mn^2
\end{array}
$$

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 $\{q_0, \ldots, q_{j-1}\}$ 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 implementated 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, $\{H_0, \ldots, H_{n-1}\}$, 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.

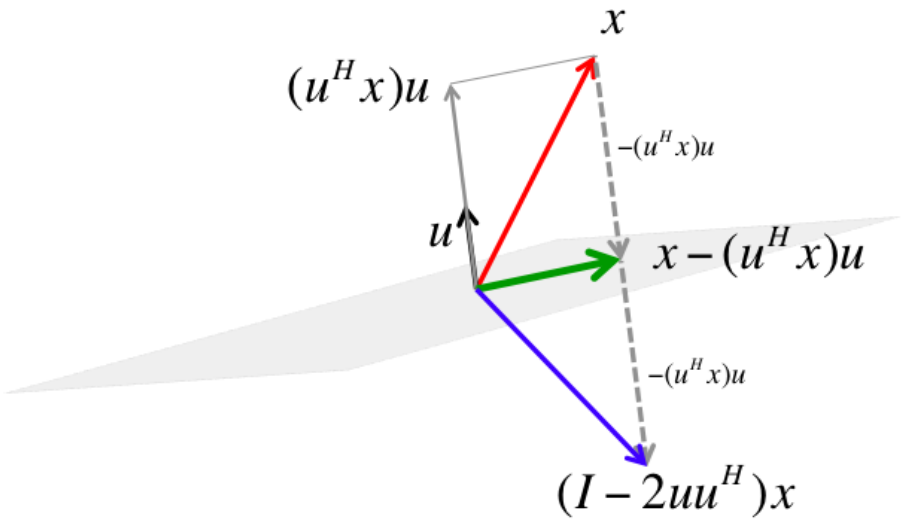
$$Q \begin{pmatrix} R \\ 0 \end{pmatrix} = \begin{pmatrix} Q_L & Q_R \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_L R,$$

The second fundamental insight will be that the desired unitary transformations $\{H_0, \ldots, H_{n-1}\}$ 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 (Resources/Week03/HouseholderTransformation.pptx).

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. ◊

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:

$$\begin{array}{l}(I - 2uu^H)(I - 2uu^H) \\ = \\ I - 2uu^H - 2uu^H + 4u \underbrace{u^H u}_{1} u^H \\ = \\ I - 4uu^H + 4uu^H = I\end{array}$$

- $H = H^H$.

  Solution:

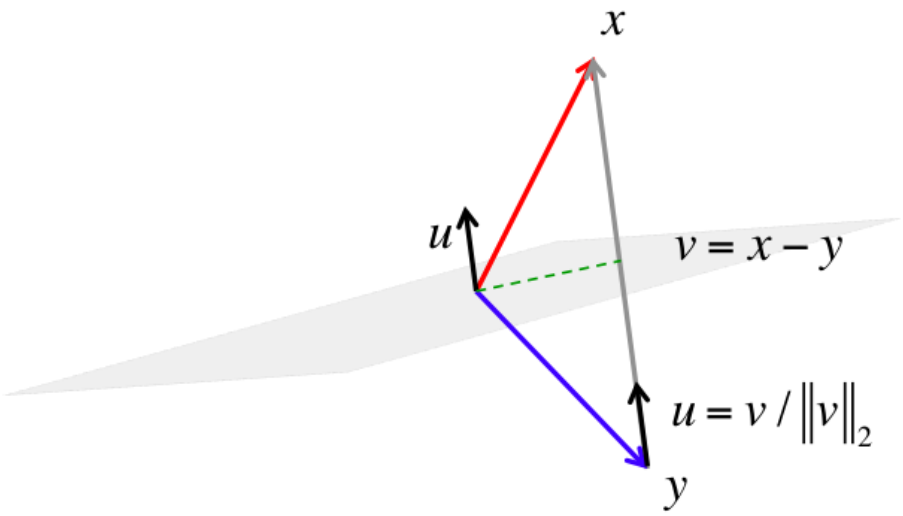
$$\begin{array}{l}(I - 2uu^H)^H \\ = \\ I - 2(u^H)^H u^H \\ = \\ I - 2uu^H\end{array}$$

- $H^H H = I$ (a reflector is unitary).

  Solution:

$$\begin{array}{l}H^H H \\ = \\ HH \\ = \\ I\end{array}$$

YouTube: https://www.youtube.com/watch?v=wmjUHak9yHU



PowerPoint source (Resources/Week03/HouseholderTransformationAsUsed.pptx)

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 $(I - \frac{1}{\tau} v v^T)x = \pm\|x\|_2 e_0$.

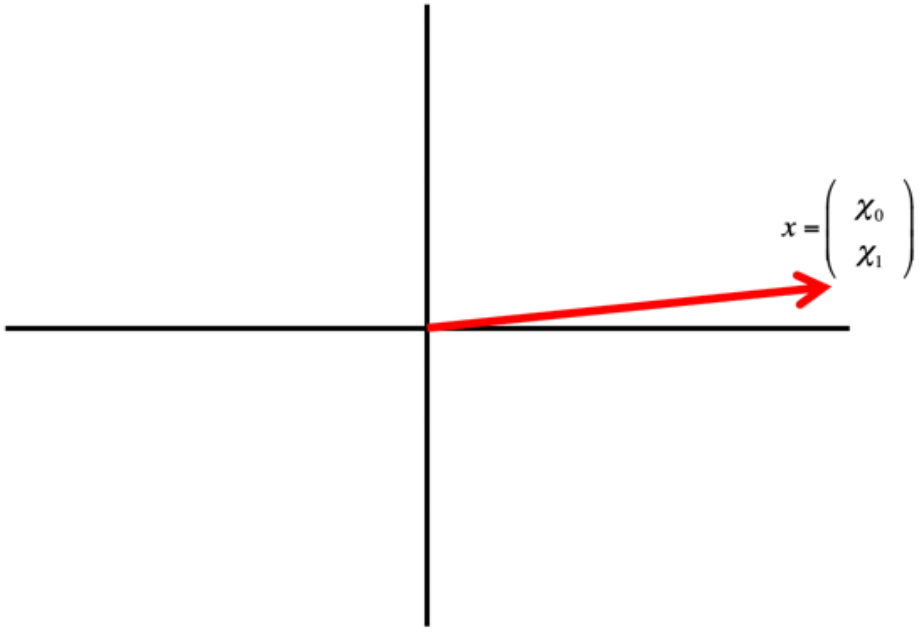
Solution. This is surprisingly messy... It is easier to derive the formula than it is to check it. So, we won't check it! □

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, ν_1, will be $\nu_1 = \chi_1 \mp \|x\|_2$. 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 $v = x + \text{sign}(\chi_1)\|x\|_2 e_0$:

$$v := x + \text{sign}(\chi_1)\|x\|_2 e_0 = \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \text{sign}(\chi_1)\|x\|_2 \\ 0 \end{pmatrix} = \begin{pmatrix} \chi_1 + \text{sign}(\chi_1)\|x\|_2 \\ x_2 \end{pmatrix}.$$

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

$$x = \left(\begin{array}{c|c} \chi_1 \\ \hline x_2 \end{array} \right),$$

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 $u = \left(\begin{array}{c} 1 \\ u_2 \end{array} \right)$ so that

$$\left(I - \frac{1}{\tau} \left(\begin{array}{c} 1 \\ u_2 \end{array} \right) \left(\begin{array}{c} 1 \\ u_2 \end{array} \right)^T \right) \left(\begin{array}{c} \chi_1 \\ x_2 \end{array} \right) = \left(\begin{array}{c} \pm \| x \|_2 \\ 0 \end{array} \right).$$

Notice that y in the previous discussion equals the vector $\left(\begin{array}{c} \pm \| x \|_2 \\ 0 \end{array} \right)$, so the direction of u is given by

$$v = \left(\begin{array}{c} \chi_1 \mp \| x \|_2 \\ x_2 \end{array} \right).$$

We now wish to normalize this vector so its first entry equals "1":

$$u = \frac{v}{\nu_1} = \frac{1}{\chi_1 \mp \| x \|_2} \left(\begin{array}{c} \chi_1 \mp \| x \|_2 \\ x_2 \end{array} \right) = \left(\begin{array}{c} 1 \\ x_2 / \nu_1 \end{array} \right).$$

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

$$x = \left(\begin{array}{c|c} \chi_1 \\ \hline x_2 \end{array} \right),$$

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 $I - \frac{1}{\tau} u u^H$ where u and τ are computed by Housev(x).

Algorithm : $\left[\begin{pmatrix} \rho \\ u_2 \end{pmatrix}, \tau \right] = \text{Housev} \left(\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \right)$	
	$\chi_2 := \|x_2\|_2$
	$\alpha := \left\| \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} \right\|_2 \; (= \|x\|_2)$
$\rho = -\text{sign}(\chi_1)\|x\|_2$	$\rho := -\text{sign}(\chi_1)\alpha$
$\nu_1 = \chi_1 + \text{sign}(\chi_1)\|x\|_2$	$\nu_1 := \chi_1 - \rho$
$u_2 = x_2/\nu_1$	$u_2 := x_2/\nu_1$
	$\chi_2 = \chi_2/\lvert\nu_1\rvert (= \|u_2\|_2)$
$\tau = (1 + u_2^H u_2)/2$	$\tau = (1 + \chi_2^2)/2$

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 )
```

1implements 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 isSv^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.

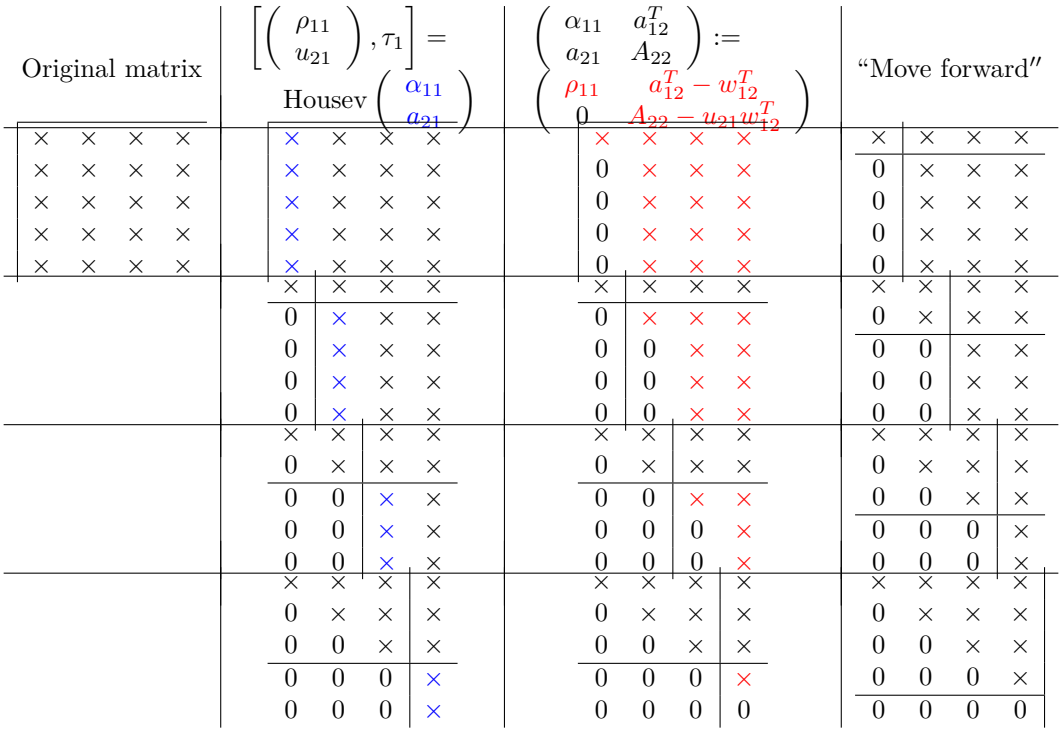


Figure 3.3.4.1 Illustration of Householder QR factorization.

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 trans-

form to A yields

$$\begin{array}{rcl} \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} & := & \left(I - \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix}^H \right) \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} \\ & = & \begin{pmatrix} \rho_{11} & a_{12}^T - w_{12}^T \\ 0 & A_{22} - u_{21} w_{12}^T \end{pmatrix}, \end{array}$$

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

$$\left(\begin{array}{c|c} I & 0 \\ \hline 0 & I - \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix}^H \end{array} \right) = \left(I - \frac{1}{\tau_1} \begin{pmatrix} 0 \\ 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ u_{21} \end{pmatrix}^H \right).$$

Solution.

$$\begin{array}{rcl} \left(\begin{array}{c|c} I & 0 \\ \hline 0 & I - \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix}^H \end{array} \right) & = & I - \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix}^H \end{array} \right) \\ & = & I - \frac{1}{\tau_1} \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & \begin{pmatrix} 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 1 \\ u_{21} \end{pmatrix}^H \end{array} \right) \\ & = & I - \frac{1}{\tau_1} \left(\begin{array}{c|cc} 0 & 0 & 0 \\ \hline 0 & 1 & u_2^H \\ 0 & u_2 & u_2 u_2^H \end{array} \right) \\ & = & \left(I - \frac{1}{\tau_1} \begin{pmatrix} 0 \\ 1 \\ u_{21} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ u_{21} \end{pmatrix}^H \right). \end{array}$$

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

$$\left[\begin{pmatrix} \rho_{11} \\ u_{21} \end{pmatrix}, \tau_1 \right] = \text{Housev} \left(\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix} \right).$$

and update

$$
\begin{array}{rcl}
A & := & \left(\begin{array}{c|c} I & 0 \\ \hline 0 & \left(I - \frac{1}{\tau_1}\begin{pmatrix} 1 \\ u_{21} \end{pmatrix}\begin{pmatrix} 1 \\ u_{21} \end{pmatrix}^H\right) \end{array}\right) \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right) \\
& = & \left(I - \frac{1}{\tau_1}\begin{pmatrix} 0 \\ 1 \\ u_{21} \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ u_{21} \end{pmatrix}^H\right) \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right) \\
& = & \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & a_{12}^T - w_{12}^T \\ 0 & 0 & A_{22} - u_{21}w_{12}^T \end{array}\right),
\end{array}
$$

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

$$
R = \begin{pmatrix} R_{TL} \\ \hline 0 \end{pmatrix} = H_{n-1} \cdots H_1 H_0 \hat{A}
$$

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.

$[A, t] = \text{HQR\_unb\_var1}(A)$

$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$ 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 $n(A_{BR}) > 0$

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array}\right)$

$\left[\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}, \tau_1\right] := \left[\begin{pmatrix} \rho_{11} \\ u_{21} \end{pmatrix}, \tau_1\right] = \text{Housev}\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}$

Update $\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} := \left(I - \frac{1}{\tau_1}\begin{pmatrix} 1 \\ u_{21} \end{pmatrix}\begin{pmatrix} 1 & u_{21}^H \end{pmatrix}\right)\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix}$

via the steps

$w_{12}^T := (a_{12}^T + a_{21}^H A_{22})/\tau_1$

$\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} := \begin{pmatrix} a_{12}^T - w_{12}^T \\ A_{22} - a_{21} w_{12}^T \end{pmatrix}$

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array}\right)$

endwhile

Figure 3.3.4.2 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 kth 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

$$
\begin{aligned}
&\sum_{k=0}^{n-1} 4(m-k)(n-k) \\
&= \\
&4\sum_{j=0}^{n-1}(m-n+j)j \\
&= \\
&4(m-n)\sum_{j=0}^{n-1} j + 4\sum_{j=0}^{n-1} j^2 \\
&= \\
&2(m-n)n(n-1) + 4\sum_{j=0}^{n-1} j^2 \\
&\approx \\
&2(m-n)n^2 + 4\int_0^n x^2 dx \\
&= \\
&2mn^2 - 2n^3 + \tfrac{4}{3}n^3 \\
&= \\
&2mn^2 - \tfrac{2}{3}n^3.
\end{aligned}
$$

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 $[A, t]$ = 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.

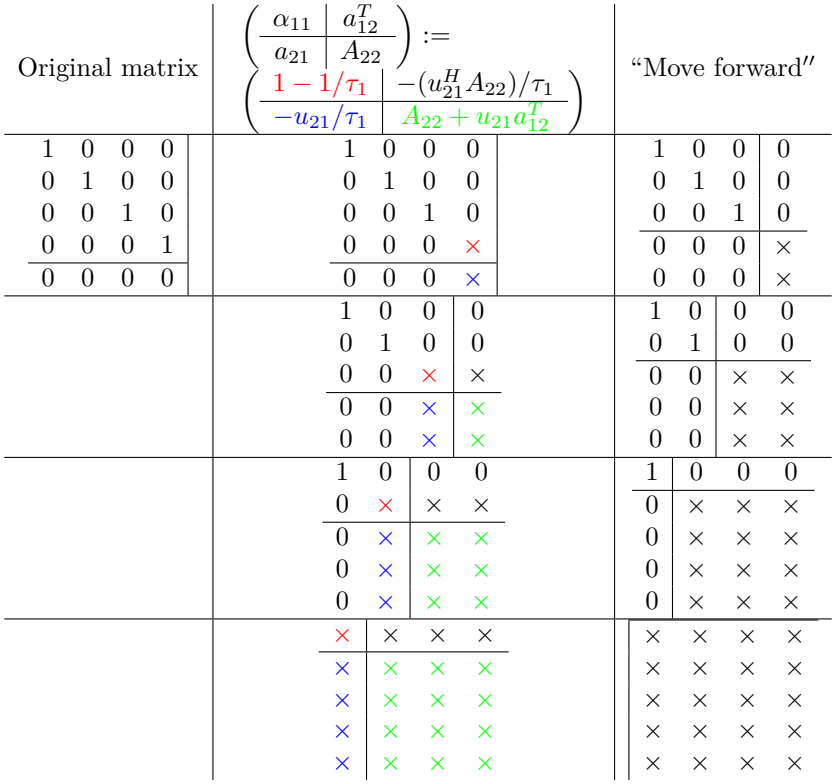


Figure 3.3.5.1 Illustration of the computation of Q.

Notice that to pick out the first n columns we must form

$$Q \left(\begin{array}{c} I_{n\times n} \\ \hline 0 \end{array}\right) = H_0 \cdots H_{n-1} \left(\begin{array}{c} I_{n\times n} \\ \hline 0 \end{array}\right) = H_0 \cdots H_{k-1} \underbrace{H_k \cdots H_{n-1} \left(\begin{array}{c} I_{n\times n} \\ \hline 0 \end{array}\right)}_{B_k}$$

so that $Q = B_0$, where $B_k = H_k \cdots H_{n-1} \left(\begin{array}{c} I_{n\times n} \\ \hline 0 \end{array}\right)$.

Homework 3.3.5.1 ALWAYS/SOMETIMES/NEVER:

$$B_k = H_k \cdots H_{n-1} \left(\begin{array}{c} I_{n\times n} \\ \hline 0 \end{array}\right) = \left(\begin{array}{c|c} I_{k\times k} & 0 \\ \hline 0 & \widetilde{B}_k \end{array}\right).$$

for some $(m-k) \times (n-k)$ matrix $\widetilde{B}_k$.

Answer. ALWAYS

Solution. The proof of this is by induction on k:

- Base case: $k = n$. Then $B_n = \left(\begin{array}{c} I_{n\times n} \\ \hline 0 \end{array}\right)$, which has the desired form.

- Inductive step: Assume the result is true for B_k. We show it is true for B_{k-1}:

$$
\begin{array}{l}
B_{k-1} \\
\quad = \\
H_{k-1} H_k \cdots H_{n-1} \left(\begin{array}{c} I_{n\times n} \\ \hline 0 \end{array} \right) \\
\quad = \\
H_{k-1} B_k \\
\quad = \\
H_{k-1} \left(\begin{array}{c|c} I_{k\times k} & 0 \\ \hline 0 & \widetilde{B}_k \end{array} \right) \\
\quad = \\
\left(\begin{array}{cc} I_{(k-1)\times(k-1)} & 0 \\ 0 & I - \frac{1}{\tau_k} \left(\begin{array}{c} 1 \\ \hline u_k \end{array} \right) \left(\begin{array}{c|c} 1 & u_k^H \end{array} \right) \end{array} \right) \left(\begin{array}{cc|c} I_{(k-1)\times(k-1)} & 0 & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & \widetilde{B}_k \end{array} \right) \\
\quad = \\
\left(\begin{array}{cc} I_{(k-1)\times(k-1)} & 0 \\ 0 & \left(I - \frac{1}{\tau_k} \left(\begin{array}{c} 1 \\ \hline u_k \end{array} \right) \left(\begin{array}{c|c} 1 & u_k^H \end{array} \right) \right) \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{B}_k \end{array} \right) \end{array} \right) \\
\quad = \quad < \text{ choose } y_k^T = u_k^H \widetilde{B}_k / \tau_k > \\
\left(\begin{array}{cc} I_{(k-1)\times(k-1)} & 0 \\ 0 & \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{B}_k \end{array} \right) - \left(\begin{array}{c} 1 \\ \hline u_k \end{array} \right) \left(\begin{array}{c|c} 1/\tau_k & y_k^T \end{array} \right) \end{array} \right) \\
\quad = \\
\left(\begin{array}{cc} I_{(k-1)\times(k-1)} & 0 \\ 0 & \left(\begin{array}{c|c} 1 - 1/\tau_k & -y_k^T \\ \hline -u_k/\tau_k & \widetilde{B}_k - u_k y_k^T \end{array} \right) \end{array} \right) \\
\quad = \\
\left(\begin{array}{cc|c} I_{(k-1)\times(k-1)} & 0 & 0 \\ 0 & 1 - 1/\tau_k & -y_k^T \\ \hline 0 & -u_k/\tau_k & \widetilde{B}_k - u_k y_k^T \end{array} \right) \\
\quad = \\
\left(\begin{array}{cc} I_{(k-1)\times(k-1)} & 0 \\ 0 & \widetilde{B}_{k-1} \end{array} \right).
\end{array}
$$

- By the Principle of Mathematical Induction the result holds for $B_0, \ldots, B_n$.

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 $n(A_{TL}) > 0$

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ \hline t_2 \end{array}\right)$$

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

$$\left(I - \frac{1}{\tau_1}\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right)\left(\begin{array}{c|c} 1 & u_{21}^H \end{array}\right)\right)\left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & A_{22} \end{array}\right)$$

via the steps

$\alpha_{11} := 1 - 1/\tau_1$

$a_{12}^T := -(a_{21}^H A_{22})/\tau_1$

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

$a_{21} := -a_{21}/\tau_1$

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \leftarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ \hline t_2 \end{array}\right)$$

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

$$
\begin{array}{l}
\sum_{k=n-1}^{0} 4(m-k)(n-k) \\
\quad = \qquad < \text{ reverse the order of the summation } > \\
\sum_{k=0}^{n-1} 4(m-k)(n-k) \\
\quad = \\
4\sum_{j=1}^{n}(m-n+j)j \\
\quad = \\
4(m-n)\sum_{j=1}^{n} j + 4\sum_{j=1}^{n} j^2 \\
\quad = \\
2(m-n)n(n+1) + 4\sum_{j=1}^{n} j^2 \\
\quad \approx \\
2(m-n)n^2 + 4\int_0^n x^2 dx \\
\quad = \\
2mn^2 - 2n^3 + \frac{4}{3}n^3 \\
\quad = \\
2mn^2 - \frac{2}{3}n^3.
\end{array}
$$

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 $H_0 y$:

$$
\begin{array}{l}
\left(I - \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \right) \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} \\
\quad = \\
\begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} - \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \underbrace{\begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} / \tau_1}_{\omega_1} \\
\quad = \\
\begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} - \omega_1 \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \\
\quad = \\
\begin{pmatrix} \psi_1 - \omega_1 \\ y_2 - \omega_1 u_2 \end{pmatrix}.
\end{array}
$$

More generally, let us compute $H_k y$:

$$\left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 0 \\ \hline 1 \\ u_2 \end{array} \right) \left(\begin{array}{c} 0 \\ \hline 1 \\ u_2 \end{array} \right)^H \right) \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array} \right) = \left(\begin{array}{c} y_0 \\ \hline \psi_1 - \omega_1 \\ y_2 - \omega_1 u_2 \end{array} \right),$$

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.

$[y] = \text{Apply\_QH}(A, t, y)$

$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right)$

A_{TL} is 0×0 and t_T, y_T have 0 elements

while $n(A_{BR}) < 0$

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

$$\left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array} \right)$$

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

$$\omega_1 := (\psi_1 + a_{21}^H y_2)/\tau_1$$

$$\left(\begin{array}{c} \psi_1 \\ y_2 \end{array} \right) := \left(\begin{array}{c} \psi_1 - \omega_1 \\ y_2 - \omega_1 u_{21} \end{array} \right)$$

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

$$\left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array} \right)$$

endwhile

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

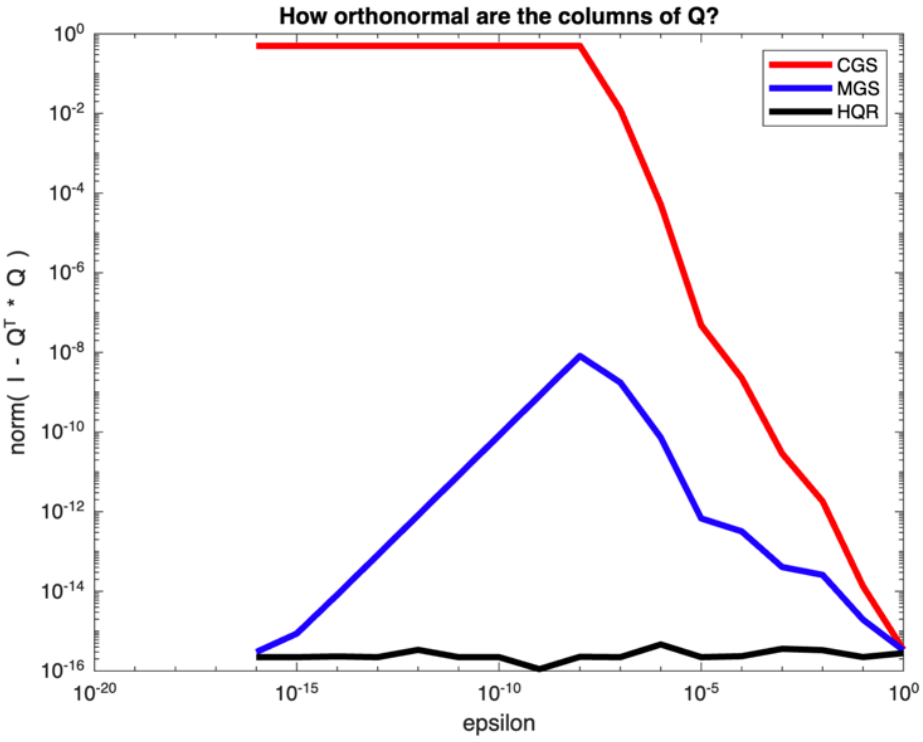
Homework 3.3.7.1 Previous programming assignments have the following routines for computing the QR factorization of a given matrix A:

- Classical Gram-Schmidt (CGS) Homework 3.2.3.1:
  [A_out, R_out] = CGS_QR(A).
- Modified Gram-Schmidt (MGS) Homework 3.2.4.3:
  [A_out, R_out] = MGS_QR(A).
- Householder QR factorization (HQR) Homework 3.3.4.3:
  [A_out, t_out] = HQR(A).
- Form Q from Householder QR factorization Homework 3.3.5.2:
  Q = 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

$$\left(\begin{array}{c|c|c} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{array}\right).$$

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 $H_0, \cdots, H_{k-1}$ are Householder transformations, then

$$H_0 H_1 \cdots H_{k-1} = I - U T^{-1} U^H,$$

where T is an upper triangular matrix. If U stores the Householder vectors that define $H_0, \ldots, H_{k-1}$ (with "1"s explicitly on its diagonal) and t holds the scalars $\tau_0, \ldots, \tau_{k-1}$, then

```
T := FormT( U, t )
```

computes the desired matrix T. Now, applying this UT transformation to a matrix B yields

$$(I - U T^{-1} U^H) B = B - U(T^{-1}(U^H B)),$$

which demontrates 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 multipication $B - UW$, each of which can attain high performance.

In [23] we call the transformation $I - U T^{-1} U^H$ that equals the accumulated Householder transformations the **UT transform** and prove that T can instead by 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 $\tau_0, \ldots, \tau_{k-1}$ (in order). In that paper, we point out similar published results [8] [35] [46] [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:
$$\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} = \left(I - \begin{pmatrix} U_{11} \\ U_{21} \end{pmatrix} T_{11}^{-1} \begin{pmatrix} U_{11} \\ U_{21} \end{pmatrix}^H \right)^H \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} = \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} - \begin{pmatrix} U_{11} \\ U_{21} \end{pmatrix} W_{12} = \begin{pmatrix} A_{12} - U_{11} W_{12} \\ A_{22} - U_{21} W_{12} \end{pmatrix},$$
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 $0 \times 0, t_T$ has 0 rows

while $m(A_{TL}) < m(A)$

choose block size b

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline t_1 \\ \hline t_2 \end{array}\right)$

A_{11} is $b \times b, t_1$ has b rows

$\left[\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, t_1\right] := \text{HQR\_unb\_var1}\left(\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}\right)$

$T_{11} := \text{FormT}\left(\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, t_1\right)$

$W_{12} := T_{11}^{-H}(U_{11}^H A_{12} + U_{21}^H A_{22})$

$\begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} := \begin{pmatrix} A_{12} - U_{11} W_{12} \\ A_{22} - U_{21} W_{12} \end{pmatrix}$

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \leftarrow \left(\begin{array}{c} t_0 \\ t_1 \\ \hline t_2 \end{array}\right)$

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

$$I - \beta v v^H,$$

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 ouput (the **postcondition**) for the algorithm. factorization

- The precondition for the QR factorization is

$$A = \widehat{A}.$$

A contains the original matrix, which we specify by $\widehat{A}$ since A will be overwritten as the algorithm proceeds.

- The postcondition for the QR factorization is

$$A = Q \wedge \widehat{A} = QR \wedge Q^H Q = I. \tag{3.4.1}$$

This specifies that A is to be overwritten by an orthonormal matrix Q and that QR equals the original matrix $\widehat{A}$. 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):

$$\begin{array}{l} \underbrace{\left(\begin{array}{c|c} A_L & A_R \end{array}\right)}_{A} = \underbrace{\left(\begin{array}{c|c} Q_L & Q_R \end{array}\right)}_{Q} \\ \quad \wedge \underbrace{\left(\begin{array}{c|c} \widehat{A}_L & \widehat{A}_R \end{array}\right)}_{\widehat{A}} = \underbrace{\left(\begin{array}{c|c} Q_L & Q_R \end{array}\right)}_{Q} \underbrace{\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right)}_{R} \\ \quad \wedge \underbrace{\left(\begin{array}{c|c} Q_L & Q_R \end{array}\right)^H}_{Q^H} \underbrace{\left(\begin{array}{c|c} Q_L & Q_R \end{array}\right)}_{Q} = \underbrace{\left(\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right)}_{I}. \end{array}$$

(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

$$\begin{array}{l} \left(\begin{array}{c|c} A_L & A_R \end{array}\right) = \left(\begin{array}{c|c} Q_L & Q_R \end{array}\right) \\ \quad \wedge \left(\begin{array}{c|c} \widehat{A}_L & \widehat{A}_R \end{array}\right) = \left(\begin{array}{c|c} Q_L R_{TL} & Q_L R_{TR} + Q_R R_{BR} \end{array}\right) \\ \quad \wedge \left(\begin{array}{c|c} Q_L^H Q_L & Q_L^T Q_R \\ \hline Q_R^H Q_L & Q_R^H Q_R \end{array}\right). = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right). \end{array} \tag{3.4.2}$$

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:

$$\begin{array}{l} \left(\begin{array}{c|c} A_L & A_R \end{array}\right) = \left(\begin{array}{c|c} Q_L & \widehat{A}_R \end{array}\right) \\ \quad \wedge \widehat{A}_L = Q_L R_{TL} \\ \quad \wedge Q_L^H Q_L = I \end{array} \tag{3.4.3}$$

- Loop invariant 2:

$$\begin{array}{l} \left(\begin{array}{c|c} A_L & A_R \end{array} \right) = \left(\begin{array}{c|c} Q_L & \widehat{A}_R - Q_L R_{TR} \end{array} \right) \\ \quad \wedge \left(\begin{array}{c|c} \widehat{A}_L & \widehat{A}_R \end{array} \right) = \left(\begin{array}{c|c} Q_L R_{TL} & Q_L R_{TR} + Q_R R_{BR} \end{array} \right) \\ \quad \wedge Q_L^H Q_L = I \end{array}$$

We note that our knowledge of linear algebra allows us to manipulate this into

$$\begin{array}{l} \left(\begin{array}{c|c} A_L & A_R \end{array} \right) = \left(\begin{array}{c|c} Q_L & \widehat{A}_R - Q_L R_{TR} \end{array} \right) \\ \quad \wedge \widehat{A}_L = Q_L R_{TL} \wedge Q_L^H \widehat{A}_L = R_{TR} \wedge Q_L^H Q_L = I. \end{array} \tag{3.4.4}$$

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 $\left(\begin{array}{c} A \\ \hline B \end{array} \right)$ where A has linearly independent columns. Let

- $A = Q_A R_A$ be the QR factorization of A.
- $\left(\begin{array}{c} R_A \\ \hline B \end{array} \right) = Q_B R_B$ be the QR factorization of $\left(\begin{array}{c} R_A \\ \hline\hline B \end{array} \right)$.
- $\left(\begin{array}{c} A \\ \hline\hline B \end{array} \right) = QR$ be the QR factorization of $\left(\begin{array}{c} A \\ \hline\hline B \end{array} \right)$.

Assume that the diagonal entries of R_A, R_B, and R are all positive. Show that $R = R_B$.

Solution.

$$\left(\begin{array}{c} A \\ \hline\hline B \end{array} \right) = \left(\begin{array}{c|c} Q_A & 0 \\ \hline 0 & I \end{array} \right) \left(\begin{array}{c} R_A \\ \hline B \end{array} \right) = \left(\begin{array}{c|c} Q_A & 0 \\ \hline 0 & I \end{array} \right) Q_B R_B$$

Also, $\left(\begin{array}{c} A \\ \hline B \end{array} \right) = QR$. 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 $\{a_0, \ldots, a_{n-1}\} \subset \mathbb{C}^m$, the Gram-Schmidt process computes an orthonormal basis $\{q_0, \ldots, q_{n-1}\}$ that spans the same subspace as the original vectors, i.e.

$$\text{Span}(\{a_0, \ldots, a_{n-1}\}) = \text{Span}(\{q_0, \ldots, q_{n-1}\}).$$

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 $\text{Span}(\{a_0, a_1, a_2\}) = \text{Span}(\{q_0, q_1, q_2\})$:
  - $\begin{array}{l} \rho_{0,2} = q_0^H a_2 \\ \rho_{1,2} = q_1^H a_2 \end{array}$ 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
  $$\left(\begin{array}{cc|c} a_0 & a_1 & a_2 \end{array}\right) = \left(\begin{array}{cc|c} q_0 & q_1 & q_2 \end{array}\right) \left(\begin{array}{cc|c} \rho_{0,0} & \rho_{0,1} & \rho_{0,2} \\ 0 & \rho_{1,1} & \rho_{1,2} \\ \hline 0 & 0 & \rho_{2,2} \end{array}\right).$$
- 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 $Q \in \mathbb{C}^{m\times n}$:

$[y^\perp, r] = \text{Proj}\perp\text{Q}_{\text{CGS}}(Q, y)$ (used by CGS)	$[y^\perp, r] = \text{Proj}\perp\text{Q}_{\text{MGS}}(Q, y)$ (used by MGS)
$y^\perp = y$ **for** $i = 0, \ldots, k-1$ $\rho_i := q_i^H y$ $y^\perp := y^\perp - \rho_i q_i$ **endfor**	$y^\perp = y$ **for** $i = 0, \ldots, k-1$ $\rho_i := q_i^H y^\perp$ $y^\perp := y^\perp - \rho_i q_i$ **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 $n(A_L) < n(A)$

$$\left(\begin{array}{c|c} A_L & A_R \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_0 & a_1 & A_2 \end{array}\right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array}\right)$$

CGS	MGS	MGS (alternative)
$r_{01} := A_0^H a_1$ $a_1 := a_1 - A_0 r_{01}$ $\rho_{11} := \|a_1\|_2$ $a_1 := a_1/\rho_{11}$	$[a_1, r_{01}] = \text{Proj}\perp\text{toQ}_{\text{MGS}}(A_0, a_1)$ $\rho_{11} := \|a_1\|_2$ $q_1 := a_1/\rho_{11}$	$\rho_{11} := \|a_1\|_2$ $a_1 := a_1/\rho_{11}$ $r_{12}^T := a_1^H A_2$ $A_2 := A_2 - a_1 r_{12}^T$

$$\left(\begin{array}{c|c} A_L & A_R \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_0 & a_1 & A_2 \end{array}\right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} R_{00} & r_{01} & R_{02} \\ 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array}\right)$$

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. ◊

If H is a Householder transformation (reflector), then

- $HH = I$.
- $H = H^H$.
- $H^H H = HH^{=} I$.
- $H^{-1} = H^H = H$.

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 : $\left[\begin{pmatrix}\rho\\ u_2\end{pmatrix}, \tau\right] = \text{Housev}\left(\begin{pmatrix}\chi_1\\ x_2\end{pmatrix}\right)$	
	$\chi_2 := \|x_2\|_2$
	$\alpha := \left\|\begin{pmatrix}\chi_1\\ \chi_2\end{pmatrix}\right\|_2 \; (= \|x\|_2)$
$\rho = -\text{sign}(\chi_1)\|x\|_2$	$\rho := -\text{sign}(\chi_1)\alpha$
$\nu_1 = \chi_1 + \text{sign}(\chi_1)\|x\|_2$	$\nu_1 := \chi_1 - \rho$
$u_2 = x_2/\nu_1$	$u_2 := x_2/\nu_1$
	$\chi_2 = \chi_2/\lvert\nu_1\rvert (= \|u_2\|_2)$
$\tau = (1 + u_2^H u_2)/2$	$\tau = (1 + \chi_2^2)/2$

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 $n(A_{BR}) > 0$

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right) \text{ and } \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ t_2 \end{array}\right)$$

$$\left[\begin{pmatrix}\alpha_{11}\\ a_{21}\end{pmatrix}, \tau_1\right] := \left[\begin{pmatrix}\rho_{11}\\ u_{21}\end{pmatrix}, \tau_1\right] = \text{Housev}\begin{pmatrix}\alpha_{11}\\ a_{21}\end{pmatrix}$$

Update $\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} := \left(I - \frac{1}{\tau_1}\begin{pmatrix}1\\ u_{21}\end{pmatrix}\begin{pmatrix}1 & u_{21}^H\end{pmatrix}\right)\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix}$

via the steps

$$w_{12}^T := (a_{12}^T + a_{21}^H A_{22})/\tau_1$$

$$\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} := \begin{pmatrix} a_{12}^T - w_{12}^T \\ A_{22} - a_{21} w_{12}^T \end{pmatrix}$$

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) \text{ and } \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \leftarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ \hline t_2 \end{array}\right)$$

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 $n(A_{TL}) > 0$

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ \hline t_2 \end{array}\right)$$

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

$$\left(I - \frac{1}{\tau_1}\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right)\left(\begin{array}{c|c} 1 & u_{21}^H \end{array}\right)\right)\left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & A_{22} \end{array}\right)$$

via the steps

$\alpha_{11} := 1 - 1/\tau_1$

$a_{12}^T := -(a_{21}^H A_{22})/\tau_1$

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

$a_{21} := -a_{21}/\tau_1$

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \leftarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ \hline t_2 \end{array}\right)$$

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 $n(A_{BR}) < 0$

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

$$\left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ \hline t_2 \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ \hline y_2 \end{array}\right)$$

Update $\left(\begin{array}{c} \psi_1 \\ \hline y_2 \end{array}\right) := \left(I - \frac{1}{\tau_1}\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right)\left(\begin{array}{c|c} 1 & u_{21}^H \end{array}\right)\right)\left(\begin{array}{c} \psi_1 \\ \hline y_2 \end{array}\right)$

via the steps

$\omega_1 := (\psi_1 + a_{21}^H y_2)/\tau_1$

$$\left(\begin{array}{c} \psi_1 \\ \hline y_2 \end{array}\right) := \left(\begin{array}{c} \psi_1 - \omega_1 \\ \hline y_2 - \omega_1 u_2 \end{array}\right)$$

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

$$\left(\begin{array}{c} t_T \\ \hline t_B \end{array}\right) \leftarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ \hline t_2 \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ \hline y_2 \end{array}\right)$$

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=LPfdOYoQQU0

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

$$\begin{array}{rcl} \chi_0 + \chi_1 \alpha_0 & \approx & \beta_0 \\ \chi_0 + \chi_1 \alpha_1 & \approx & \beta_1 \\ \vdots & \vdots & \vdots \\ \chi_0 + \chi_1 \alpha_{m-1} & \approx & \beta_{m-1}, \end{array}$$

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

4.1.3 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: $x^H y = 0$. ◊

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

$$\begin{array}{ll} & x^H y \\ = & < x \in \mathcal{R}(A) \text{ iff there exists } z \text{ s.t. } x = A^H z > \\ & (A^H z)^H y \\ = & < \text{transposition of product} > \\ & z^H A y \\ = & < y \in \mathcal{N}(A) > \\ & z^H 0 = 0. \end{array}$$

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

$$\begin{array}{ll} & x^H y \\ = & < x \in \mathcal{C}(A) \text{ iff there exists } z \text{ s.t. } x = Az > \\ & (Az)^H y \\ = & < \text{transposition of product} > \\ & z^H A^H y \\ = & < y \in \mathcal{N}(A^H) > \\ & z^H 0 = 0. \end{array}$$

Homework 4.2.1.3 Let $\{s_0, \cdots, s_{r-1}\}$ be a basis for subspace $\mathcal{S} \subset \mathbb{C}^n$ and $\{t_0, \cdots, t_{k-1}\}$ 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 $\{s_0, \cdots, s_{r-1}\}$ are orthogonal to the vectors in $\{t_0, \cdots, t_{k-1}\}$.
3. $s_i^H t_j = 0$ for all $0 \leq i < r$ and $0 \leq j < k$.
4. $\left(\begin{array}{c|c|c} s_0 & \cdots & s_{r-1} \end{array} \right)^H \left(\begin{array}{c|c|c} t_0 & \cdots & t_{k-1} \end{array} \right) = 0$, 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 orthogonoal.

- 3. implies 4.

$$\left(\begin{array}{c|c|c} s_0 & \cdots & s_{r-1} \end{array} \right)^H \left(\begin{array}{c|c|c} t_0 & \cdots & t_{k-1} \end{array} \right) = \begin{pmatrix} s_0^H t_0 & s_0^H t_1 & \cdots \\ s_1^H t_0 & s_1^H t_1 & \cdots \\ \vdots & \vdots & \end{pmatrix}$$

- 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

$$x = \left(\begin{array}{c|c|c} s_0 & \cdots & s_{r-1} \end{array} \right) \begin{pmatrix} \widehat{\chi}_0 \\ \vdots \\ \widehat{\chi}_{r-1} \end{pmatrix} \text{ and } y = \left(\begin{array}{c|c|c} t_0 & \cdots & t_{k-1} \end{array} \right) \begin{pmatrix} \widehat{\psi}_0 \\ \vdots \\ \widehat{\psi}_{k-1} \end{pmatrix}$$

for appropriate choices of $\widehat{x}$ and $\widehat{y}$. But then

$$\begin{array}{rcl} x^H y & = & \left(\begin{array}{c|c|c} s_0 & \cdots & s_{r-1} \end{array} \right) \left(\begin{array}{c} \widehat{\chi}_0 \\ \vdots \\ \widehat{\chi}_{r-1} \end{array} \right) \Bigg)^H \left(\begin{array}{c|c|c} t_0 & \cdots & t_{k-1} \end{array} \right) \left(\begin{array}{c} \widehat{\psi}_0 \\ \vdots \\ \widehat{\psi}_{k-1} \end{array} \right) \\ & = & \left(\begin{array}{c} \widehat{\chi}_0 \\ \vdots \\ \widehat{\chi}_{r-1} \end{array} \right)^H \underbrace{\left(\begin{array}{c|c|c} s_0 & \cdots & s_{r-1} \end{array} \right)^H \left(\begin{array}{c|c|c} t_0 & \cdots & t_{k-1} \end{array} \right)}_{0_{r \times k}} \left(\begin{array}{c} \widehat{\psi}_0 \\ \vdots \\ \widehat{\psi}_{k-1} \end{array} \right) \\ & = & 0 \end{array}$$

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 $\{w_0, \cdots, w_{r-1}\}$ be a basis for $\mathcal{R}(A)$ and $\{w_r, \cdots, w_{n-1}\}$ be a basis for $\mathcal{N}(A)$.

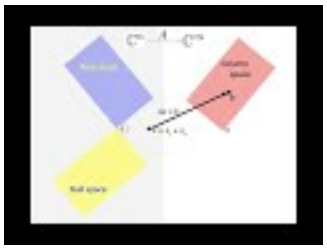
Answer. TRUE

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 $\{w_0, \cdots, w_{r-1}\}$ be a basis for $\mathcal{R}(A)$ and $\{w_r, \cdots, w_{n-1}\}$ be a basis for $\mathcal{N}(A)$. Since we know that these two spaces are orthogonal, we know that $\{w_0, \cdots, w_{r-1}\}$ are orthogonal to $\{w_r, \cdots, w_{n-1}\}$. Hence $\{w_0, \cdots, w_{n-1}\}$ are linearly independent and form a basis for $\mathbb{C}^n$. Thus, there exist coefficients $\{\alpha_0, \cdots, \alpha_{n-1}\}$ such that

$$\begin{array}{l} x = \alpha_0 w_0 + \cdots + \alpha_{n-1} w_{n-1} \\ \quad = \quad < \text{split the summation} > \\ \underbrace{\alpha_0 w_0 + \cdots + \alpha_{r-1} w_{r-1}}_{x_r} + \underbrace{\alpha_r w_r + \cdots + \alpha_{n-1} w_{n-1}}_{x_n} . \end{array}$$



YouTube: https://www.youtube.com/watch?v=ZdlraR_7cMA

Figure 4.2.1.2 captures the insights so far.

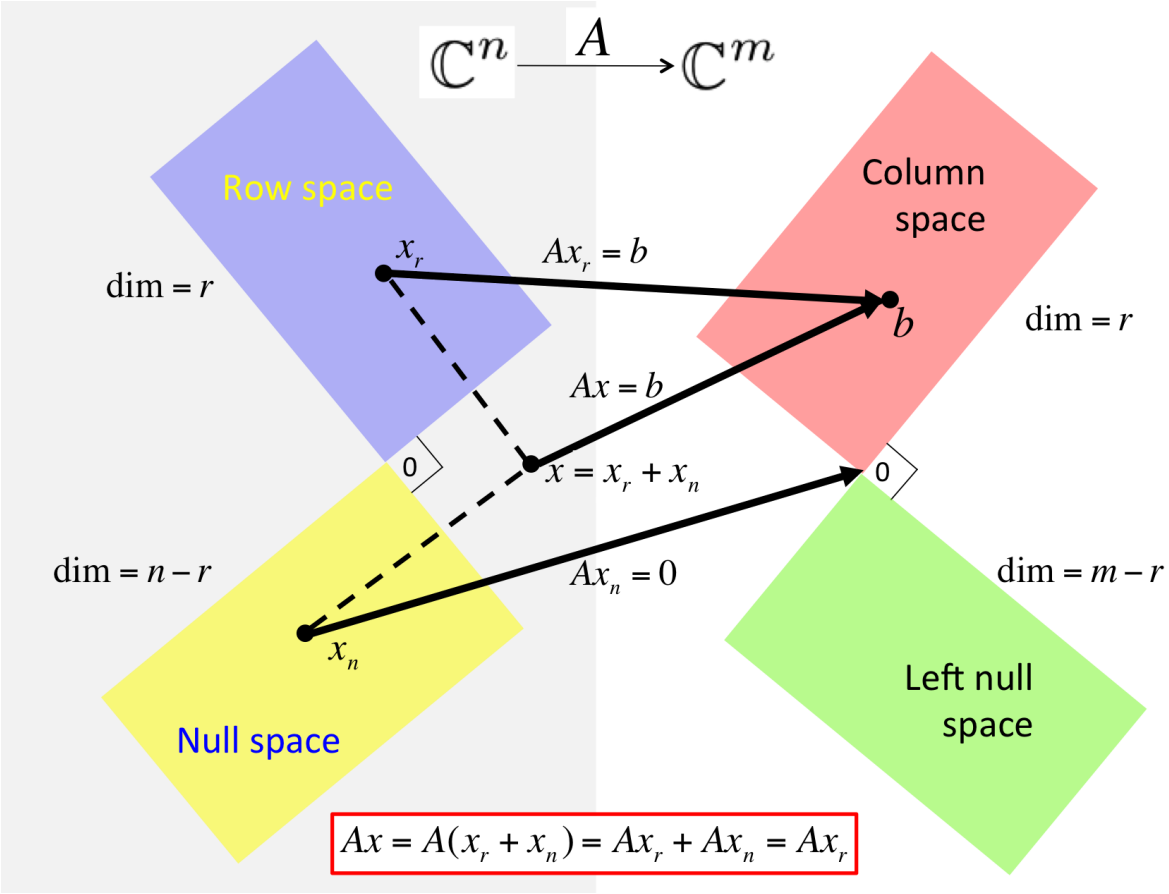


Figure 4.2.1.2 Illustration of the four fundamental spaces and the mapping of a vector $x \in \mathbb{C}^n$ by matrix $A \in \mathbb{C}^{m \times n}$.

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=oT4KIOxx-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,

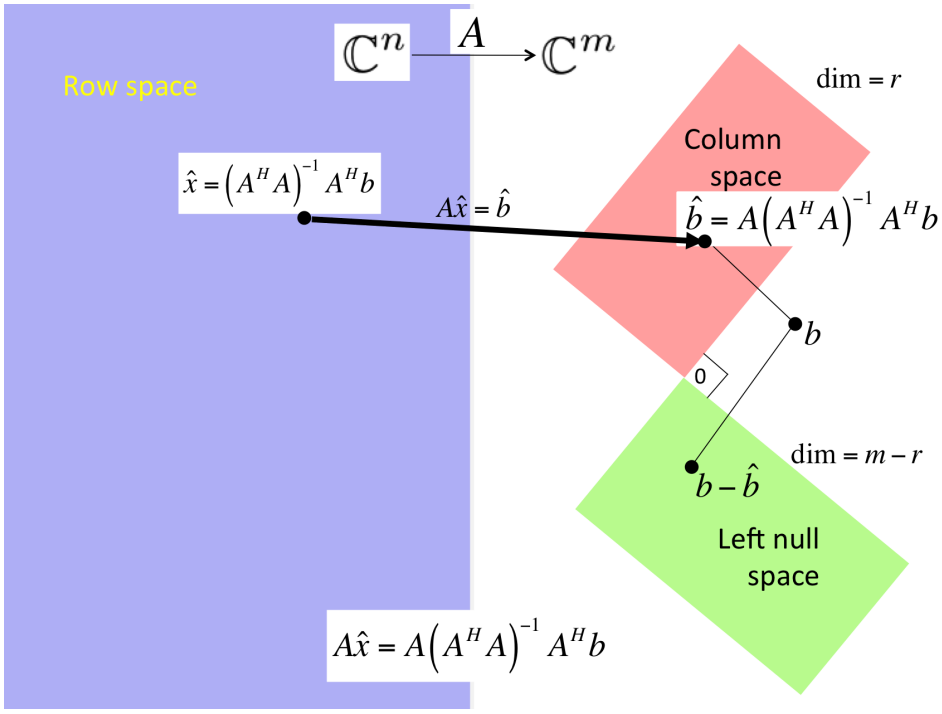
  $$A^H A\hat{x} = A^H b.$$

  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. ◊

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 $A = \begin{pmatrix} e_0 \end{pmatrix}$.

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

$$\begin{array}{l} AA^\dagger \\ \quad = \quad < \text{ instantiate } > \\ \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix} \underbrace{\left(\underbrace{\begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix}^H \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix}}_{1} \right)^{-1}}_{1} \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix}^H \\ \quad = \quad < \text{ simplify } > \\ \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots \end{pmatrix} \\ \quad = \quad < \text{ multiply out } > \\ \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ \vdots & \vdots & \end{pmatrix} \\ \quad = \quad < m > 1 > \\ \neq I. \end{array}$$

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 $x^H Bx > 0$. This follows from the fact that
$$x^H Bx = x^H A^H Ax = (Ax)^H (Ax) = \|Ax\|_2^2.$$
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
$$\|b - A\hat{x}\|_2 = \min_x \|b - Ax\|_2 \tag{4.2.1}$$

and the perturbed problem

$$\|(b+\delta b)-A(\hat{x}+\delta\hat{x})\|_2 = \min_x \|(b+\delta b)-A(x+\delta x)\|_2 \tag{4.2.2}$$

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

$$\hat{b} = A\hat{x} \tag{4.2.3}$$

and the projection of $b+\delta b$ satisfies

$$\hat{b}+\delta\hat{b} = A(\hat{x}+\delta\hat{x}) \tag{4.2.4}$$

where $\delta\hat{b}$ 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

$$\frac{1}{\|\hat{x}\|_2} \leq \frac{\sigma_0}{\cos(\theta)}\frac{1}{\|b\|_2}. \tag{4.2.5}$$

Subtracting (4.2.3) from (4.2.4) yields

$$\delta\hat{b} = A\delta\hat{x}$$

or, equivalently,

$$A\delta\hat{x} = \delta\hat{b}$$

which is solved by

$$\begin{array}{rcl} \delta\hat{x} & = & A^\dagger\delta\hat{b} \\ & = & A^\dagger A(A^HA)^{-1}A^H\delta b \\ & = & (A^HA)^{-1}A^HA(A^HA)^{-1}A^H\delta b \\ & = & A^\dagger\delta b, \end{array}$$

where $A^\dagger = (A^HA)^{-1}A^H$ is the pseudo inverse of A and we recall that $\delta\hat{b} = A(A^HA)^{-1}A^H\delta b$. Hence

$$\|\delta\hat{x}\|_2 \leq \|A^\dagger\|_2\|\delta b\|_2. \tag{4.2.6}$$

Homework 4.2.4.1 Let $A \in \mathbb{C}^{m\times n}$ have linearly independent columns. Show that

$$\|(A^HA)^{-1}A^H\|_2 = 1/\sigma_{n-1},$$

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

$$\begin{array}{l} \|(A^H A)^{-1} A^H\|_2 \\ \quad = \\ \|((U_L \Sigma_{TL} V^H)^H U_L \Sigma_{TL} V^H)^{-1} (U_L \Sigma_{TL} V^H)^H\|_2 \\ \quad = \\ \|(V \Sigma_{TL} U_L^H U_L \Sigma_{TL} V^H)^{-1} V \Sigma_{TL} U_L^H\|_2 \\ \quad = \\ \|(V \Sigma_{TL}^{-1} \Sigma_{TL}^{-1} V^H) V \Sigma_{TL} U_L^H\|_2 \\ \quad = \\ \|V \Sigma_{TL}^{-1} U_L^H\|_2 \\ \quad = \\ \|\Sigma_{TL}^{-1} U_L^H\|_2 \\ \quad = \\ 1/\sigma_{n-1}. \end{array}$$

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

$$\frac{\|\delta\hat{x}\|_2}{\|\hat{x}\|_2} \leq \frac{1}{\cos(\theta)} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta b\|_2}{\|b\|_2}. \tag{4.2.7}$$

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 perpendical 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

$$\kappa_2(A) = \|A\|_2 \|A^\dagger\|_2 = \frac{\sigma_0}{\sigma_{n-1}}.$$

$\Diamond$

It is informative to explicity expose $\cos(\theta) = \|\hat{b}\|_2 / \|b\|_2$ in (4.2.7):

$$\frac{\|\delta\hat{x}\|_2}{\|\hat{x}\|_2} \leq \frac{\|b\|_2}{\|\hat{b}\|_2} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta b\|_2}{\|b\|_2}.$$

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 $\hat{x}$:

$$\frac{\|\delta\hat{x}\|_2}{\|\hat{x}\|_2} \leq \frac{\|b + b_r\|_2}{\|\hat{b}\|_2} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta b\|_2}{\|b + b_r\|_2}.$$

4.2.5 Why using the Method of Normal Equations could be bad

YouTube: https://www.youtube.com/watch?v=W-HnQDsZsOw

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

$$\begin{array}{rcl} \kappa_2(A^H A) & = & \|A^H A\|_2 \|(A^H A)^{-1}\|_2 \\ & = & \|(U\Sigma V^H)^H U\Sigma V^H\|_2 \|((U\Sigma V^H)^H U\Sigma V^H)^{-1}\|_2 \\ & = & \|V\Sigma^2 V^H\|_2 \|V(\Sigma^{-1})^2 V^H\|_2 \\ & = & \|\Sigma^2\|_2 \|(\Sigma^{-1})^2\|_2 \\ & = & \frac{\sigma_0^2}{\sigma_{n-1}^2} = \left(\frac{\sigma_0}{\sigma_{n-1}}\right)^2 = \kappa_2(A)^2. \end{array}$$

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 $A = \begin{pmatrix} U_L \mid U_R \end{pmatrix} \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \begin{pmatrix} V_L \mid V_R \end{pmatrix}^H$ 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 $z = Ax = U_L \Sigma_{TL} V_L^H x = U_L \underbrace{\Sigma_{TL} V_L^H x}_{\widehat{x}} = U_L \widehat{x}$. 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 $z = U_L x = U_L \underbrace{\Sigma_{TL} V_L^H V_L \Sigma_{TL}^{-1}}_{I} x = A \underbrace{V_L \Sigma_{TL}^{-1} x}_{\widehat{x}} = A\widehat{x}$. 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 $A = \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \left(\begin{array}{c|c} V_L & \end{array}\right.$ 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 columns 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.

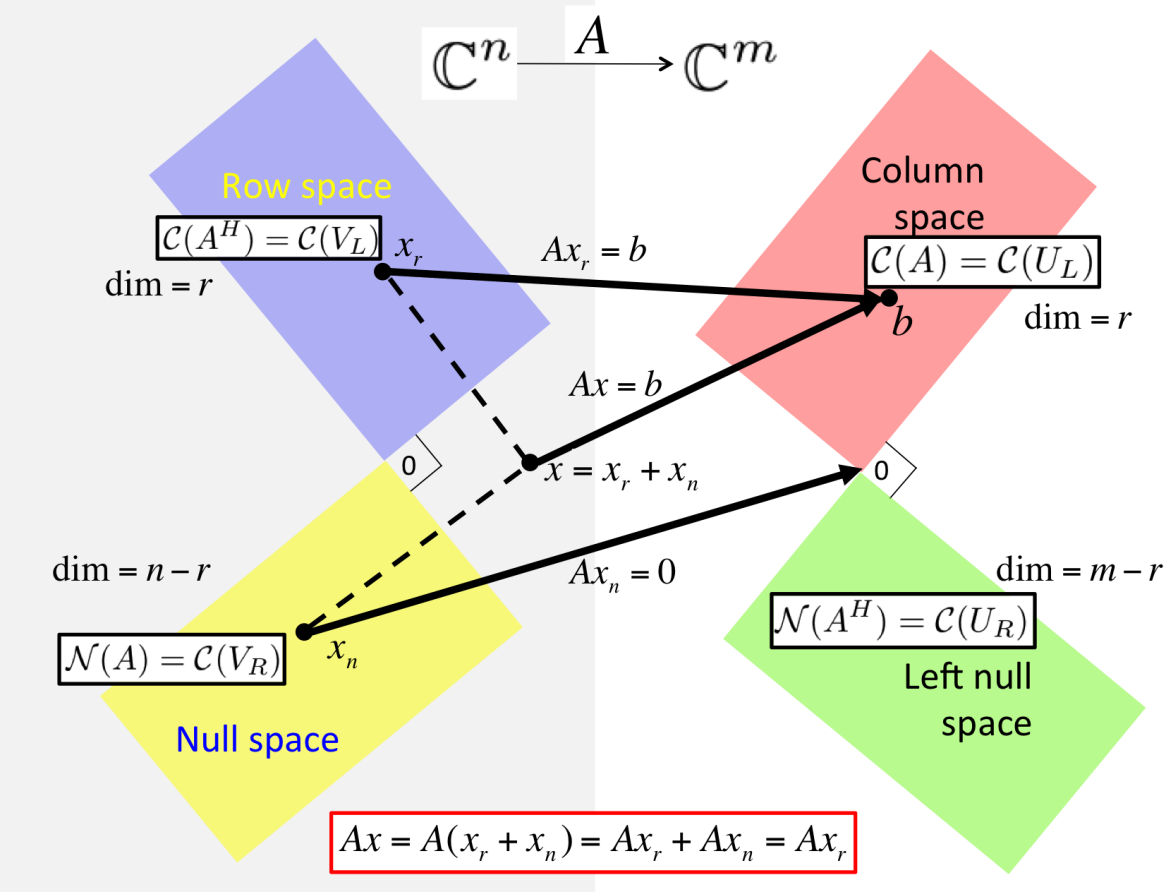
Homework 4.3.1.4 Given $A \in \mathbb{C}^{m\times n}$, let $A = U_L \Sigma_{TL} V_L^H$ equal its Reduced SVD and $A = \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \left(\begin{array}{c|c} V_L & \end{array}\right.$ its 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.

$$
\begin{aligned}
x &= Ix = VV^H x \\
&= \begin{pmatrix} V_L \mid V_R \end{pmatrix} \begin{pmatrix} V_L \mid V_R \end{pmatrix}^H x \\
&= \begin{pmatrix} V_L \mid V_R \end{pmatrix} \begin{pmatrix} V_L^H \\ V_R^H \end{pmatrix} x \\
&= \begin{pmatrix} V_L \mid V_R \end{pmatrix} \begin{pmatrix} V_L^H x \\ V_R^H x \end{pmatrix} \\
&= \underbrace{V_L V_L^H x}_{x_r} + \underbrace{V_R V_R^H x}_{x_n}.
\end{aligned}
$$



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

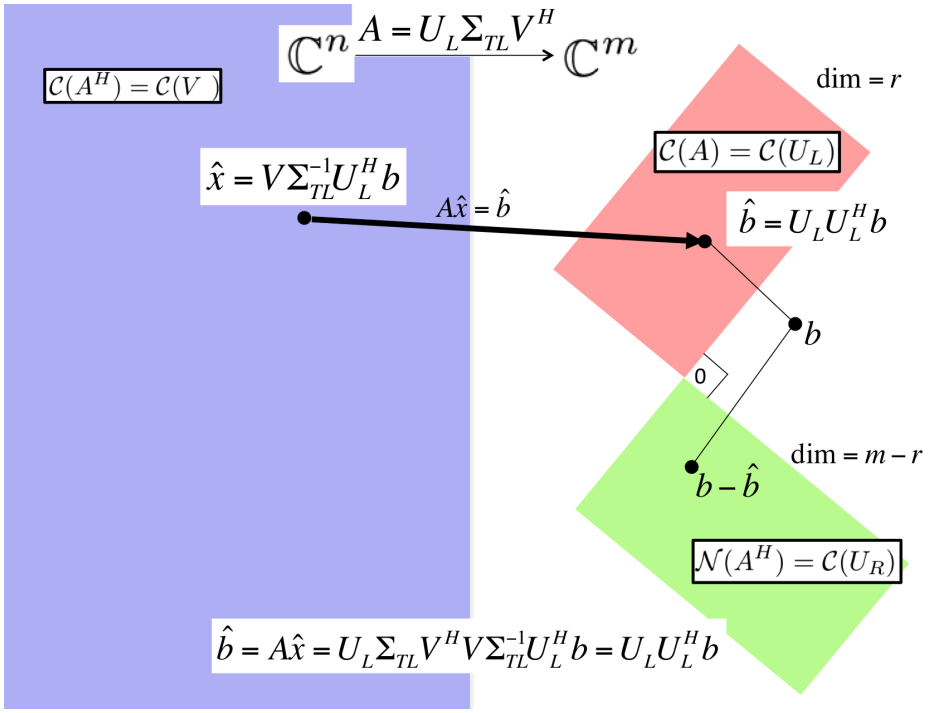
$$\|b - A\hat{x}\|_2 = \min_x \|b - Ax\|_2,$$

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,

$$\begin{array}{l}
\hat{x} \\
\quad = \quad < \text{solution to the normal equations} > \\
(A^H A)^{-1} A^H b \\
\quad = \quad < A = U_L \Sigma_{TL} V^H > \\
\left[(U_L \Sigma_{TL} V^H)^H (U_L \Sigma_{TL} V^H)\right]^{-1} (U_L \Sigma_{TL} V^H)^H b \\
\quad = \quad < (BCD)^H = (D^H C^H B^H) \text{ and } \Sigma_{TL}^H = \Sigma_{TL} > \\
\left[(V \Sigma_{TL} U_L^H)(U_L \Sigma_{TL} V^H)\right]^{-1} (V \Sigma_{TL} U_L^H) b \\
\quad = \quad < U_L^H U_L = I > \\
\left[V \Sigma_{TL} \Sigma_{TL} V^H\right]^{-1} V \Sigma_{TL} U_L^H b \\
\quad = \quad < V^{-1} = V^H \text{ and } (BCD)^{-1} = D^{-1} C^{-1} B^{-1} > \\
V \Sigma_{TL}^{-1} \Sigma_{TL}^{-1} V^H V \Sigma_{TL} U_L^H b \\
\quad = \quad < V^H V = I \text{ and } \Sigma_{TL}^{-1} \Sigma_{TL} = I > \\
V \Sigma_{TL}^{-1} U_L^H b
\end{array}$$



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.

$$
\begin{array}{ll}
\min_{x\in\mathbb{C}^n} \|b - Ax\|_2^2 & \\
\quad = & < \text{ substitute the SVD} A = U\Sigma V^H > \\
\min_{x\in\mathbb{C}^n} \|b - U\Sigma V^H x\|_2^2 & \\
\quad = & < \text{ substitute } I = UU^H \text{ and factor out } U > \\
\min_{x\in\mathbb{C}^n} \|U(U^H b - \Sigma V^H x)\|_2^2 & \\
\quad = & < \text{ multiplication by a unitary matrix preserves two-norm } > \\
\min_{x\in\mathbb{C}^n} \|U^H b - \Sigma V^H x\|_2^2 & \\
\quad = & < \text{ partition, partitioned matrix-matrix multiplication } > \\
\min_{x\in\mathbb{C}^n} \left\| \begin{pmatrix} U_L^H b \\ U_R^H b \end{pmatrix} - \begin{pmatrix} \Sigma_{TL} \\ 0 \end{pmatrix} V^H x \right\|_2^2 & \\
\quad = & < \text{ partitioned matrix-matrix multiplication and addition } > \\
\min_{x\in\mathbb{C}^n} \left\| \begin{pmatrix} U_L^H b - \Sigma_{TL} V^H x \\ U_R^H b \end{pmatrix} \right\|_2^2 & \\
\quad = & < \left\| \begin{pmatrix} v_T \\ v_B \end{pmatrix} \right\|_2^2 = \|v_T\|_2^2 + \|v_B\|_2^2 > \\
\min_{x\in\mathbb{C}^n} \left\|U_L^H b - \Sigma_{TL} V^H x\right\|_2^2 + \left\|U_R^H b\right\|_2^2 &
\end{array}
$$

The x that solves $\Sigma_{TL} V^H x = U_L^H b$ minimizes the expression. That x is given by

$$\hat{x} = V\Sigma_{TL}^{-1} U_L^H b.$$

since Σ_{TL} is a diagonal matrix with only nonzeroes 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:

$$\hat{b} = U_L U_L^H b.$$

(Notice that this is *not* because U_L is unitary, since it isn't. It is because the matrix $U_L U_L^H$ 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

$$U_L \Sigma_{TL} V^H \hat{x} = U_L U_L^H b.$$

Multiplying both sides by U_L^H yields

$$\Sigma_{TL} V^H \hat{x} = U_L^H b.$$

and hence

$$\hat{x} = V\Sigma_{TL}^{-1} U_L^H b.$$

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 $\hat{x} = V\Sigma_{TL}^{-1}U_L^H b$.

  The cost of this is approximately,

  - Form $y_T = U_L^H b$: $2mn$ 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 $\hat{x} = V y_T$: $2n^2$ 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

$$A = \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right)^H$$

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

$$\left(\begin{array}{c|c} U_L & U_R \end{array}\right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right)^H \hat{x} = U_L U_L^H b.$$

Multiplying both sides by U_L^H yields

$$\left(\begin{array}{c|c} I & 0 \end{array}\right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right)^H \hat{x} = U_L^H b$$

or, equivalently,

$$\Sigma_{TL} V_L^H \hat{x} = U_L^H b. \tag{4.3.1}$$

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:

$$\hat{x} = Vz = \left(\begin{array}{c|c} V_L & V_R \end{array}\right) \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) = \underbrace{V_L z_T}_{x_r} + \underbrace{V_R z_B}_{x_n}.$$

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

$$\Sigma_{TL} z_T = U_L^H b.$$

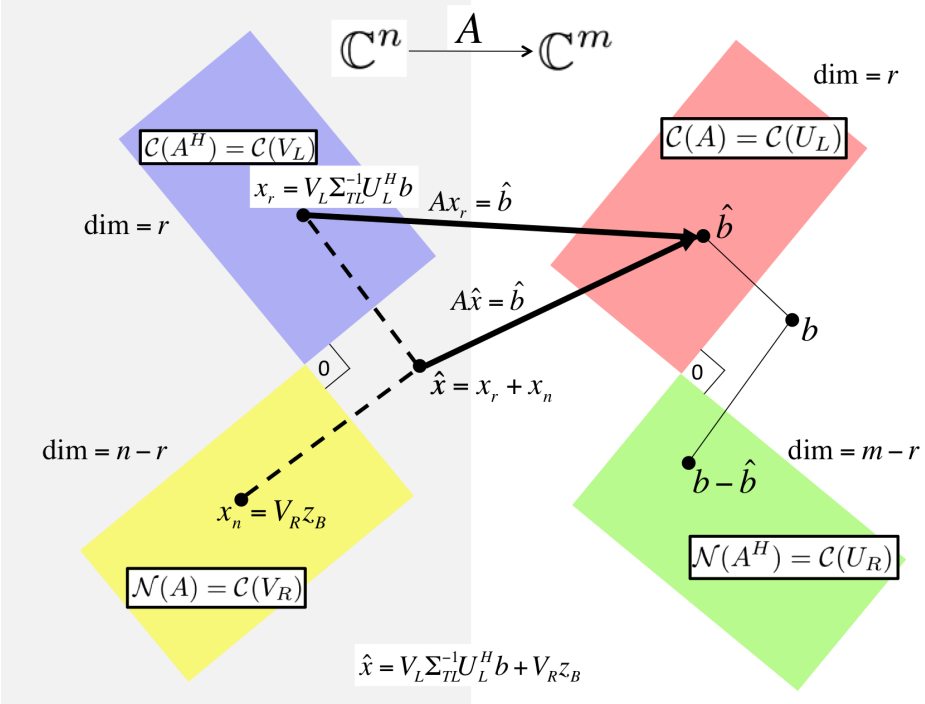
Thus, the solution in the row space is given by

$$x_r = V_L z_T = V_L \Sigma_{TL}^{-1} U_L^H b$$

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 $x^\star$ satisfies

$$\|b - Ax^\star\|_2 = \min_x \|b - Ax\|_2$$

then $\|\hat{x}\|_2 \leq \|x^\star\|_2$.

Solution. The important insight is that

$$x^\star = \underbrace{V_L \Sigma_{TL}^{-1} U_L^H b}_{\hat{x}} + V_R z_B$$

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

$$\|x^\star\|_2^2 = \|\hat{x} + V_R z_B\|_2^2 = \|\hat{x}\|_2^2 + \|V_R z_B\|_2^2 \geq \|\hat{x}\|_2^2$$

and hence $\|\hat{x}\|_2 \leq \|x^\star\|_2$.

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 $Q = \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)$ is unitary: Q_R is an orthonormal basis for the space orthogonal to the space spanned

by Q_L. Now,

$$
\begin{array}{l}
\min_{x\in\mathbb{C}^n} \|b - Ax\|_2^2 \\
\quad = \quad < \text{ substitute } A = Q_L R_{TL} > \\
\min_{x\in\mathbb{C}^n} \|b - Q_L R_{TL} x\|_2^2 \\
\quad = \quad < \text{ two norm is preserved since } Q^H \text{ is unitary } > \\
\min_{x\in\mathbb{C}^n} \|Q^H(b - Q_L R_{TL} x)\|_2^2 \\
\quad = \quad < \text{ partitioning; distributing } > \\
\min_{x\in\mathbb{C}^n} \left\| \begin{pmatrix} Q_L^H \\ Q_R^H \end{pmatrix} b - \begin{pmatrix} Q_L^H \\ Q_R^H \end{pmatrix} Q_L R_{TL} x \right\|_2^2 \\
\quad = \quad < \text{ partitioned matrix-matrix multiplication } > \\
\min_{x\in\mathbb{C}^n} \left\| \begin{pmatrix} Q_L^H b \\ Q_R^H b \end{pmatrix} - \begin{pmatrix} R_{TL} x \\ 0 \end{pmatrix} \right\|_2^2 \\
\quad = \quad < \text{ partitioned matrix addition } > \\
\min_{x\in\mathbb{C}^n} \left\| \begin{pmatrix} Q_L^H b - R_{TL} x \\ Q_R^H b \end{pmatrix} \right\|_2^2 \\
\quad = \quad < \text{ property of the 2-norm: } \left\| \begin{pmatrix} u \\ v \end{pmatrix} \right\|_2^2 = \|u\|_2^2 + \|v\|_2^2 > \\
\min_{x\in\mathbb{C}^n} \left(\left\| Q_L^H b - R_{TL} x \right\|_2^2 + \|Q_R^H b\|_2^2 \right) \\
\quad = \quad < Q_R^H b \text{ is independent of } x > \\
\left(\min_{x\in\mathbb{C}^n} \left\| Q_L^H b - R_{TL} x \right\|_2^2 \right) + \|Q_R^H b\|_2^2 \\
\quad = \quad < \text{ minimized by } \hat{x} \text{ that satisfies } R_{TL}\hat{x} = Q_L^H b > \\
\|Q_R^H b\|_2^2.
\end{array}
$$

Thus, the desired $\hat{x}$ that minimizes the linear least squares problem solves $R_{TL}\hat{x} = Q_L^H y$. 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,

$$
\begin{array}{l}
\hat{x} \\
\quad = \quad < \text{Solution to the Normal Equations } > \\
(A^H A)^{-1} A^H b \\
\quad = \quad < A = QR > \\
\left[(QR)^H (QR)\right]^{-1} (QR)^H b \\
\quad = \quad < (BC)^H = (C^H B^H) > \\
\left[R^H Q^H Q R\right]^{-1} R^H Q^H b \\
\quad = \quad < Q^H Q = I > \\
\left[R^H R\right]^{-1} R^H Q^H b \\
\quad = \quad < (BC)^{-1} = C^{-1} B^{-1} > \\
R^{-1} R^{-H} R^H Q^H b \\
\quad = \quad < R^{-H} R^H = I > \\
R^{-1} Q^H b.
\end{array}
$$

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, $H_0, \ldots, H_{n-1}$, so that

$$\underbrace{H_{n-1} \cdots H_0}_{Q^H} A = \left(\begin{array}{c} R_{TL} \\ \hline 0 \end{array}\right).$$

$[A, t] = \text{HouseQR\_unb\_var1}(A)$ overwrites A with the Householder vectors that define $H_0, \cdots, H_{n-1}$ below the diagonal and R_{TL} in the upper triangular part.

Rather than explicitly computing Q and then computing $\widetilde{y} := Q^H y$, we can instead apply the Householder transformations:

$$\widetilde{y} := H_{n-1} \cdots H_0 y,$$

overwriting y with $\widehat{y}$. After this, the vector y is partitioned as $y = \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$ 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

$$\begin{array}{l} AP(p)^T \\ \quad = \quad < \text{ definition of } P(p) > \\ A \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}^T \\ \quad = \quad < \text{ transpose } > \\ A \begin{pmatrix} e_{\pi_0} \mid \cdots \mid e_{\pi_{n-1}} \end{pmatrix} \\ \quad = \quad < \text{ matrix multiplication by columns } > \\ \begin{pmatrix} Ae_{\pi_0} \mid \cdots \mid Ae_{\pi_{n-1}} \end{pmatrix} \\ \quad = \quad < Be_j = b_j > \\ \begin{pmatrix} a_{\pi_0} \mid \cdots \mid a_{\pi_{n-1}} \end{pmatrix}. \end{array}$$

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*

$$AP(p)^T = Q_L \begin{pmatrix} R_{TL} \mid R_{TR} \end{pmatrix}.$$

Proof. Let p be the permutation vector such that the first r columns of $A^P = AP(p)^T$ are linearly independent. Partition

$$A^P = AP(p)^T = \begin{pmatrix} A_L^P \mid A_R^P \end{pmatrix}$$

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

$$A^P = AP(p)^T = \begin{pmatrix} A_L^P \mid A_R^P \end{pmatrix} = Q_L \begin{pmatrix} R_{TL} \mid R_{TR} \end{pmatrix}.$$

■

Let us examine how this last theorem can help us solve the LLS

$$\text{Find } \hat{x} \in \mathbb{C}^n \text{ such that } \|b - A\hat{x}\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2$$

when $\text{rank}(A) \leq n$:

$$\begin{array}{ll} & \min_{x \in \mathbb{C}^n} \|b - Ax\|_2 \\ = & < P(p)^T P(p) = I > \\ & \min_{x \in \mathbb{C}^n} \|b - AP(p)^T P(p)x\|_2 \\ = & < AP(p)^T = Q_L \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) > \\ & \min_{x \in \mathbb{C}^n} \|b - Q_L \underbrace{\left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) P(p)x}_{w} \|_2 \\ = & < \text{substitute } w = \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) P(p)x > \\ & \min_{w \in \mathbb{C}^r} \|b - Q_L w\|_2 \end{array}$$

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

$$z = \left(\frac{z_T}{z_B} \right)$$

for $P(p)\hat{x}$ we find that

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right) \left(\frac{z_T}{z_B} \right) = Q_L^H b.$$

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, $\widetilde{P}(j) \in \mathbb{C}^{n\times n}$, that swaps the first element of the vector to which it is applied with the element indexed with j:

$$\widetilde{P}(j)x = \begin{pmatrix} e_j^T \\ e_1^T \\ \vdots \\ e_{j-1}^T \\ e_0^T \\ e_{j+1}^T \\ \vdots \\ e_{n-1}^T \end{pmatrix} x = \begin{pmatrix} e_j^T x \\ e_1^T x \\ \vdots \\ e_{j-1}^T x \\ e_0^T x \\ e_{j+1}^T x \\ \vdots \\ e_{n-1}^T x \end{pmatrix} = \begin{pmatrix} \chi_j \\ \chi_1 \\ \vdots \\ \chi_{j-1} \\ \chi_0 \\ \chi_{j+1} \\ \vdots \\ \chi_{n-1} \end{pmatrix}.$$

Another way of stating this is that

$$\widetilde{P}(j) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & I_{(j-1)\times(j-1)} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{(n-j-1)\times(n-j-1)} \end{pmatrix},$$

where $I_{k\times k}$ equals the $k \times k$ identity matrix. When applying $\widetilde{P}(j)$ from the right to a matrix, it swaps the first column and the column indexed with j. Notice that $\widetilde{P}(j)^T = \widetilde{P}(j)$ and $\widetilde{P}(j) = \widetilde{P}(j)^{-1}$.

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 $A := A\widetilde{P}(\pi_1)$, 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 substracts 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.

$[A, R, p] :=$ 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 $n(A_L) < n(A)$

$\left(\begin{array}{c|c} A_L & A_R \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_0 & a_1 & A_2 \end{array}\right)$,

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

$\pi_1 =$ DetermineColumnIndex$(\left(\begin{array}{cc} a_1 & A_2 \end{array}\right))$

$\left(\begin{array}{cc} a_1 & A_2 \end{array}\right) := \left(\begin{array}{cc} a_1 & A_2 \end{array}\right) \widetilde{P}(\pi_1)$

$\rho_{11} := \|a_1\|_2$

$a_1 := a_1 / \rho_{11}$

$r_{12}^T := a_1^T A_2$

$A_2 := A_2 - a_1 r_{12}^T$

$\left(\begin{array}{c|c} A_L & A_R \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_0 & a_1 & A_2 \end{array}\right)$,

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

endwhile

Figure 4.5.1.2 Simple implementation of RRQR via MGS. Incorporating a stopping critera 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

$$a_i = (a_i - a_i^T q q) + a_i^T q q$$

and

$$(a_i - a_i^T q q)^T q = a_i^T q - a_i^T q q^T q = a_i^T q - a_i^T q = 0.$$

This means that

$$\|a_i\|_2^2 = \|(a_i - a_i^T q q) + a_i^T q q\|_2^2 = \|a_i - a_i^T q q\|_2^2 + \|a_i^T q q\|_2^2 = \|a_i - \rho_i q\|_2^2 + \|\rho_i q\|_2^2 = \|b_i\|_2^2 + \rho_i^2$$

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 ith entry in v equals the square of the length of the ith column of A. In other words, the ith 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

$$v \rightarrow \left(\begin{array}{c} \nu_1 \\ \hline v_2 \end{array} \right).$$

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 ith entry of v_2 must be updated by subtracting off the square of the ith entry of r_{12}^T.

Let us introduce the functions v = ComputeWeights(A) and v = 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.

$[A, R, p] :=$ RRQR_MSG(A, R, p)

$v :=$ ComputeWeights(A)

$$A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right), R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right), v \rightarrow \left(\begin{array}{c} v_T \\ \hline v_B \end{array} \right)$$

A_L has 0 columns, R_{TL} is 0×0, p_T has 0 rows, v_T has 0 rows

while $n(A_L) < n(A)$

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|cc} A_0 & a_1 & A_2 \end{array} \right),$$

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|cc} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array} \right),$$

$$\left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ \hline p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ \hline v_B \end{array} \right) \rightarrow \left(\begin{array}{c} v_0 \\ \hline \nu_1 \\ \hline v_2 \end{array} \right)$$

$$[\left(\begin{array}{c} \nu_1 \\ \hline v_2 \end{array} \right), \pi_1] = \text{DeterminePivot}(\left(\begin{array}{c} \nu_1 \\ \hline v_2 \end{array} \right))$$

$$\left(\begin{array}{c|cc} A_0 & a_1 & A_2 \end{array} \right) := \left(\begin{array}{c|cc} A_0 & a_1 & A_2 \end{array} \right) \widetilde{P}(\pi_1)^T$$

$\rho_{11} := \|a_1\|_2$

$a_1 := a_1 / \rho_{11}$

$r_{12}^T := q_1^T A_2$

$A_2 := A_2 - q_1 r_{12}^T$

$v_2 := \text{UpdateWeights}(v_2, r_{12})$

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{cc|c} A_0 & a_1 & A_2 \end{array} \right),$$

$$\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{cc|c} R_{00} & r_{01} & R_{02} \\ r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array} \right),$$

$$\left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ \hline p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ \hline v_B \end{array} \right) \leftarrow \left(\begin{array}{c} v_0 \\ \hline \nu_1 \\ \hline v_2 \end{array} \right)$$

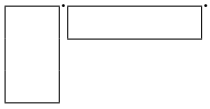
endwhile

Figure 4.5.1.3 RRQR via MGS, with optimization. Incorporating a stopping critera 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

$$A \approx Q_L \begin{pmatrix} R_{TL} & R_{TR} \end{pmatrix} =$$



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.

$[A, t, p] = \text{HQRP\_unb\_var1}(A)$

$v := \text{ComputeWeights}(A)$

$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right), t \rightarrow \begin{pmatrix} t_T \\ \hline t_B \end{pmatrix}, p \rightarrow \begin{pmatrix} p_T \\ \hline p_B \end{pmatrix}, v \rightarrow \begin{pmatrix} v_T \\ \hline v_B \end{pmatrix}$

A_{TL} is 0×0 and t_T has 0 elements

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

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right), \begin{pmatrix} t_T \\ \hline t_B \end{pmatrix} \rightarrow \begin{pmatrix} t_0 \\ \hline \tau_1 \\ t_2 \end{pmatrix},$$

$$\begin{pmatrix} p_T \\ \hline p_B \end{pmatrix} \rightarrow \begin{pmatrix} p_0 \\ \hline \pi_1 \\ p_2 \end{pmatrix}, \begin{pmatrix} v_T \\ \hline v_B \end{pmatrix} \rightarrow \begin{pmatrix} v_0 \\ \hline \nu_1 \\ v_2 \end{pmatrix}$$

$$\left[\begin{pmatrix} \nu_1 \\ v_2 \end{pmatrix}, \pi_1\right] = \text{DeterminePivot}\left(\begin{pmatrix} \nu_1 \\ v_2 \end{pmatrix}\right)$$

$$\left(\begin{array}{cc} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{cc} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{array}\right) P(\pi_1)^T$$

$$\left[\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}, \tau_1\right] := \left[\begin{pmatrix} \rho_{11} \\ u_{21} \end{pmatrix}, \tau_1\right] = \text{Housev}\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}$$

$$w_{12}^T := (a_{12}^T + a_{21}^H A_{22})/\tau_1$$

$$\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} := \begin{pmatrix} a_{12}^T - w_{12}^T \\ A_{22} - a_{21} w_{12}^T \end{pmatrix}$$

$v_2 = \text{UpdateWeight}(v_2, a_{12})$

...

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, Xioabai 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 too 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 $\widehat{x}$ 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 $\widehat{x}$ 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 states 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 $\mathcal{R}(A) = \mathcal{C}(A^H)$.

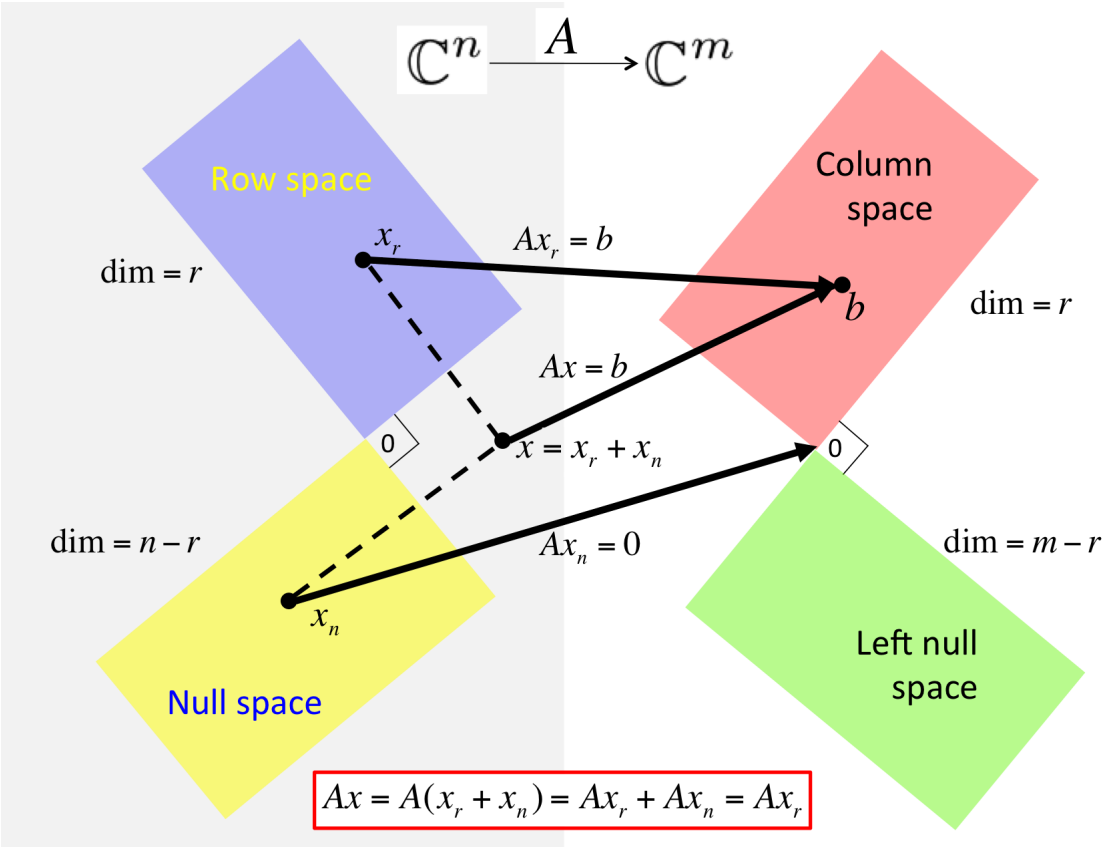
- The left null space, which is equal to the set of all vectors

$$\{x \mid x^H A = 0\}.$$

  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 $x_n \in \mathcal{N}(A)$.

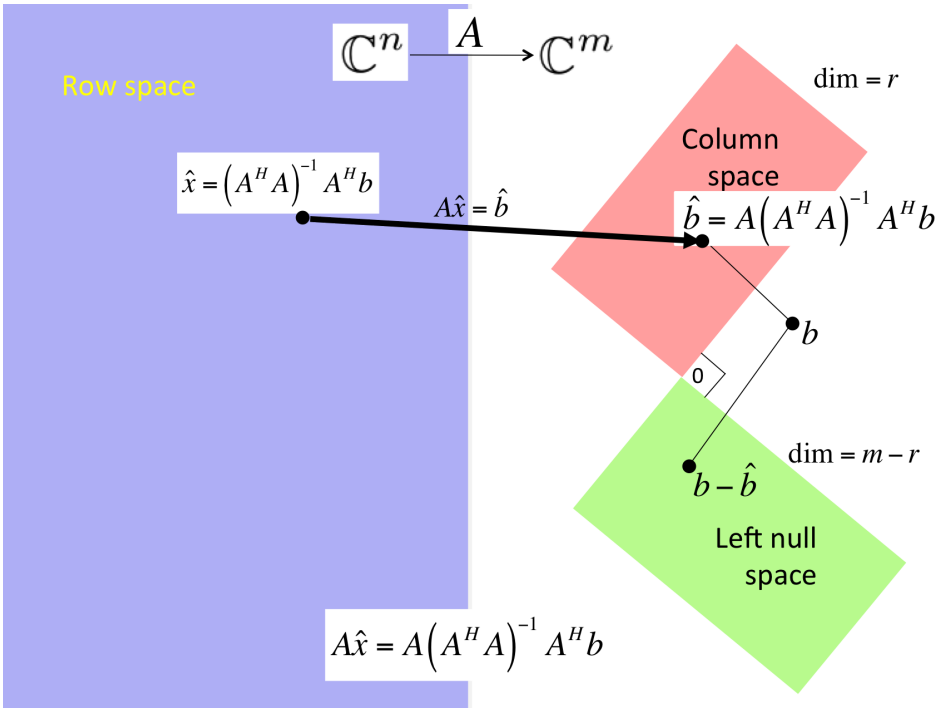
These insights are summarized in the following picture, which also captures the orthogonality of the spaces.



If A has linearly independent columns, then the solution of LLS, $\widehat{x}$, 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 $\widehat{x} = A^\dagger b$.

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

$$\kappa_2(A) = \|A\|_2 \|A^\dagger\|_2 = \frac{\sigma_0}{\sigma_{n-1}}.$$

$\diamondsuit$

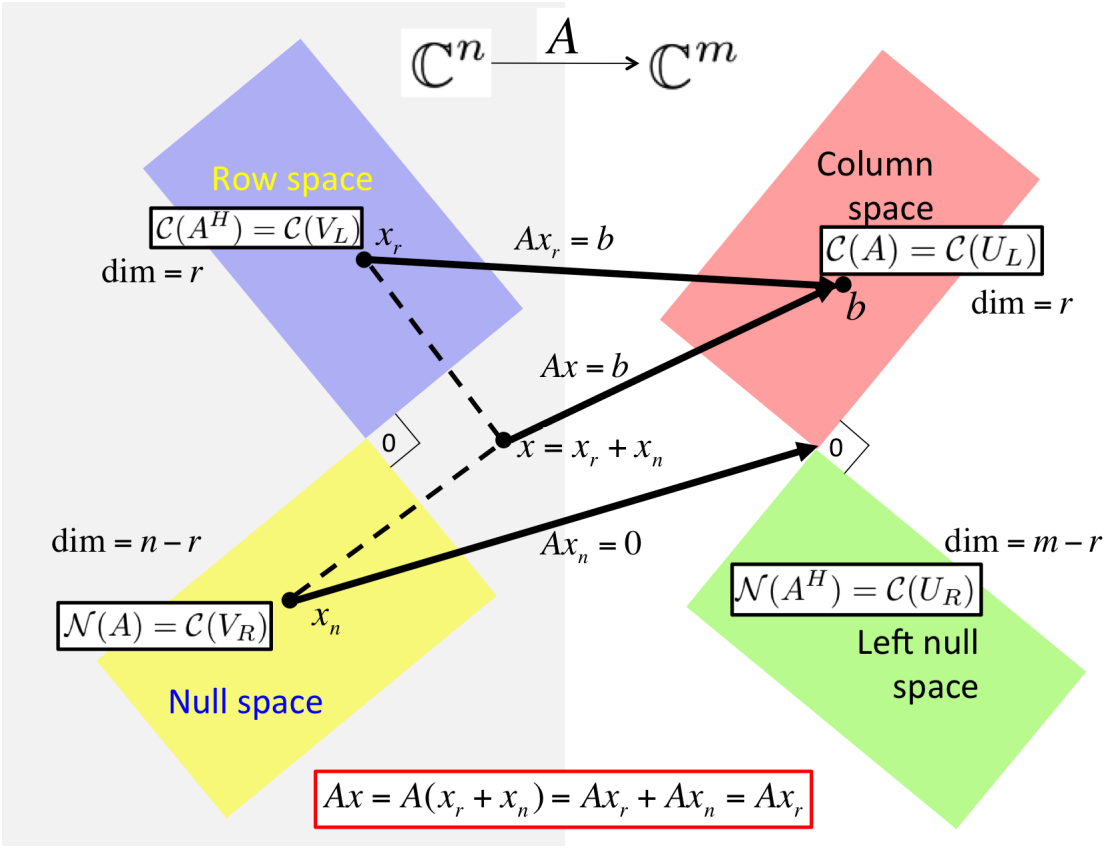
Assuming A has linearly independent columns, let $\hat{b} = A\hat{x}$ where $\hat{b}$ is the projection of b onto the column space of A (in other words, $\hat{x}$ solves the LLS problem), $\cos(\theta) = \|\hat{b}\|_2 / \|b\|_2$, and $\hat{b} + \delta\hat{b} = A(\hat{x} + \delta\hat{x})$, where $\delta\hat{b}$ equals the projection of δb onto the column space of A. Then

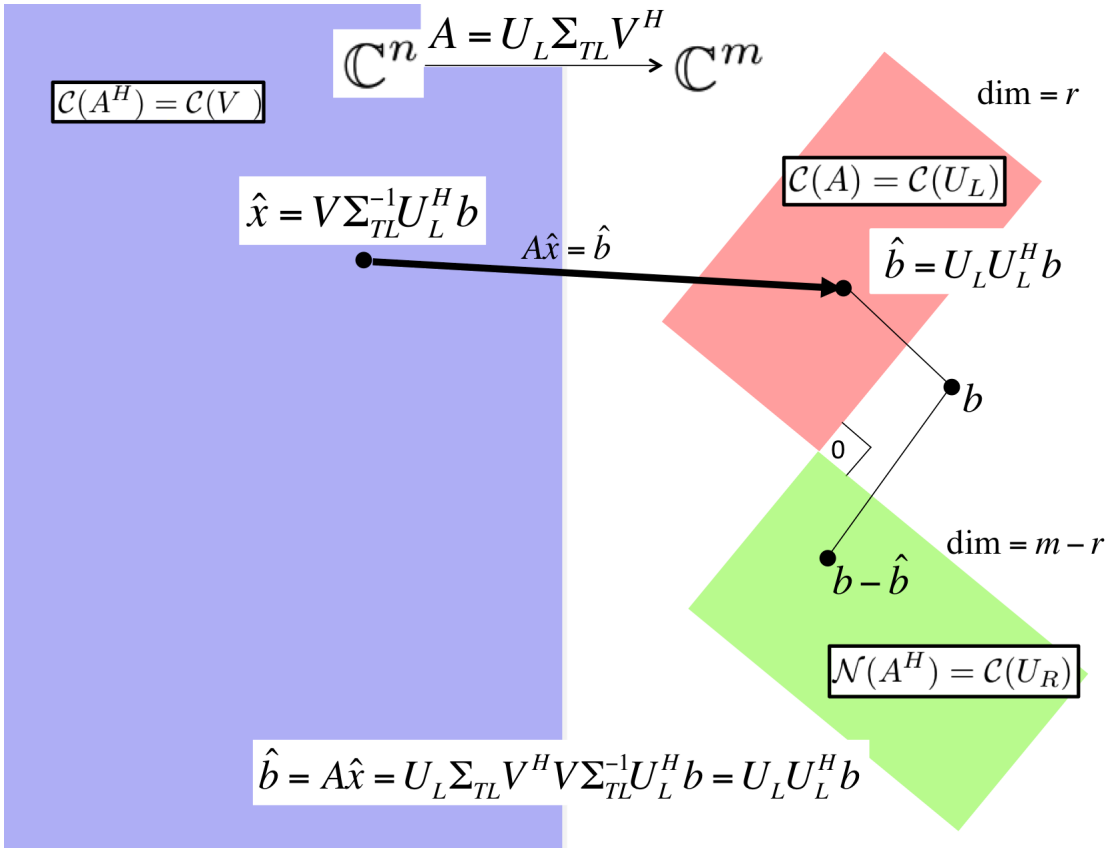
$$\frac{\|\delta\hat{x}\|_2}{\|\hat{x}\|_2} \leq \frac{1}{\cos(\theta)} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta b\|_2}{\|b\|_2}$$

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 $A = \left(\begin{array}{c|c} U_L & U_R \end{array}\right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \left(\begin{array}{c|c} V_L & V_R \end{array}\right)^H$ 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*
- $\mathcal{N}(A^H) = \mathcal{C}(U_R)$.


$\mathbb{C}^n \xrightarrow{A} \mathbb{C}^m$
Row space
$\mathcal{C}(A^H) = \mathcal{C}(V_L)$
dim = r
x_r
$Ax_r = b$
Column space
$\mathcal{C}(A) = \mathcal{C}(U_L)$
dim = r
b
$Ax = b$
0
$x = x_r + x_n$
0
dim = $n - r$
$Ax_n = 0$
dim = $m - r$
$\mathcal{N}(A) = \mathcal{C}(V_R)$
x_n
$\mathcal{N}(A^H) = \mathcal{C}(U_R)$
Left null space
Null space
$Ax = A(x_r + x_n) = Ax_r + Ax_n = Ax_r$



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 $A = \begin{pmatrix} U_L \mid U_R \end{pmatrix} \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array}\right) \begin{pmatrix} V_L \mid V_R \end{pmatrix}^H$ 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

$$\left(\begin{array}{rrr|r} 2 & -1 & 1 & 1 \\ -2 & 2 & 1 & -1 \\ 4 & -4 & 1 & 5 \end{array}\right)$$

to upper triangular form, overwriting the zeroes that are introduced with the multipliers.

Solution.

$$\left(\begin{array}{rrr|r} 2 & -1 & 1 & 1 \\ -1 & 1 & 2 & 0 \\ 2 & -2 & 3 & 3 \end{array}\right)$$

YouTube: https://www.youtube.com/watch?v=Tt0OQikd-nI

$A = \text{LU}(A)$

$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

A_{TL} is 0×0

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

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

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

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

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

endwhile

Figure 5.1.1.1 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

5.1.3 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 $j := 0, \ldots, n-1$
  for $i := j+1, \ldots, n-1$
    $\lambda_{i,j} := \alpha_{i,j} / \alpha_{j,j}$
    $\alpha_{i,j} := 0$
    for $k = j+1, \ldots, n-1$
      $\alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j} \alpha_{j,k}$
    endfor
    $\beta_i := \beta_i - \lambda_{i,j} \beta_j$
  endfor
endfor

} 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 $j := 0, \ldots, n-1$
  for $i := j+1, \ldots, n-1$
    $\lambda_{i,j} := \alpha_{i,j} / \alpha_{j,j}$
    $\alpha_{i,j} := 0$
    for $k = j+1, \ldots, n-1$
      $\alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j} \alpha_{j,k}$
    endfor
  endfor
endfor

} subtract $\lambda_{i,j}$ times row j from row k

update b using multipliers (forward substitution)
for $j := 0, \ldots, n-1$
  for $i := j+1, \ldots, n-1$
    $\beta_i := \beta_i - \lambda_{i,j} \beta_j$
  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 $j := 0, \ldots, n-1$
  for $i := j+1, \ldots, n-1$
    $\lambda_{i,j} := \alpha_{i,j} / \alpha_{j,j}$
    $\alpha_{i,j} := 0$
  endfor
  } compute multipliers
  for $i := j+1, \ldots, n-1$
    for $k = j+1, \ldots, n-1$
      $\alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j} \alpha_{j,k}$
    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 $j := 0, \ldots, n-1$
  for $i := j+1, \ldots, n-1$
    $\alpha_{i,j} := \lambda_{i,j} = \alpha_{i,j} / \alpha_{j,j}$
  endfor
  } compute all multipliers
  for $i := j+1, \ldots, n-1$
    for $k = j+1, \ldots, n-1$
      $\alpha_{i,k} := \alpha_{i,k} - \alpha_{i,j} \alpha_{j,k}$
    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 $j := 0, \ldots, n-1$

$$\begin{pmatrix} \alpha_{j+1,j} \\ \vdots \\ \alpha_{n-1,j} \end{pmatrix} := \begin{pmatrix} \lambda_{j+1,j} \\ \vdots \\ \lambda_{n-1,j} \end{pmatrix} / \alpha_{j,j}$$

$$\begin{pmatrix} \alpha_{j+1,j+1} & \cdots & \alpha_{j+1,n-1} \\ \vdots & & \vdots \\ \alpha_{n-1,j+1} & \cdots & \alpha_{n-1,n-1} \end{pmatrix} :=$$

$$\begin{pmatrix} \alpha_{j+1,j+1} & \cdots & \alpha_{j+1,n-1} \\ \vdots & & \vdots \\ \alpha_{n-1,j+1} & \cdots & \alpha_{n-1,n-1} \end{pmatrix} - \begin{pmatrix} \alpha_{j+1,j} \\ \vdots \\ \alpha_{n-1,j} \end{pmatrix} \begin{pmatrix} \alpha_{j,j+1} & \cdots & \alpha_{j,n-1} \end{pmatrix}$$

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

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

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

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

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

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:
  - $\alpha_{21} := \lambda_{21} = \alpha_{21}/\alpha_{11}$:

$$\left(\begin{array}{ccc} 2 & -1 & 1 \\ -2 & 0 & 1 \\ 2 & 0 & -2 \end{array}\right)$$

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

$$\left(\begin{array}{ccc} 2 & -1 & 1 \\ -2 & -2 & 3 \\ 2 & 2 & -4 \end{array}\right)$$

  - 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:

$$\left(\begin{array}{cc|c} 2 & -1 & 1 \\ -2 & -2 & 3 \\ \hline 2 & -1 & -1 \end{array}\right)$$

- Third iteration:
  - $\alpha_{21} := \lambda_{21} = \alpha_{21}/\alpha_{11}$:

$$\left(\begin{array}{cc|c} 2 & -1 & 1 \\ -2 & -2 & 3 \\ \hline 2 & -1 & -1 \end{array}\right)$$

    (computation with empty vector).
  - $A_{22} := A_{22} - a_{21}a_{12}^T$:

$$\left(\begin{array}{cc|c} 2 & -1 & 1 \\ -2 & -2 & 3 \\ \hline 2 & -1 & -1 \end{array}\right)$$

    (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:

$$A \rightarrow \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}, \quad L \rightarrow \begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix}, \quad \text{and} \quad U \rightarrow \begin{pmatrix} \upsilon_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix}.$$

Then $A = LU$ means that

$$\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} \upsilon_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix} = \begin{pmatrix} \upsilon_{11} & u_{12}^T \\ l_{21}\upsilon_{11} & l_{21}u_{12}^T + L_{22}U_{22} \end{pmatrix}.$$

Hence

$$\begin{array}{ll} \alpha_{11} = \upsilon_{11} & a_{12}^T = u_{12}^T \\ a_{21} = \upsilon_{11} l_{21} & A_{22} = l_{21}u_{12}^T + L_{22}U_{22} \end{array}$$

or, equivalently,

$$\begin{array}{ll} \alpha_{11} = \upsilon_{11} & a_{12}^T = u_{12}^T \\ a_{21} = \upsilon_{11} l_{21} & A_{22} - l_{21}u_{12}^T = L_{22}U_{22}. \end{array}$$

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.

$A =$ 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 5.2.2.1 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:

$$\begin{array}{ll} \sum_{k=0}^{n-1} 2(n-k-1)^2 & \\ \quad = & < \text{ change of variable: } j = n-k-1 > \\ \sum_{j=0}^{n-1} 2j^2 & \\ \quad = & < \text{ algebra } > \\ 2\sum_{j=0}^{n-1} j^2 & \\ \quad \approx & < \sum_{j=0}^{n-1} j^2 \approx \int_0^n x^2 dx = n^3/3 > \\ \frac{2}{3}n^3 & \end{array}$$

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:

$$\begin{array}{ll}
\sum_{k=0}^{n-1} 2(m-k-1)(n-k-1) & \\
= & < \text{ change of variable: } j = n-k-1 > \\
\sum_{j=0}^{n-1} 2(m-(n-j-1)-1)j & \\
= & < \text{ simplify } > \\
\sum_{j=0}^{n-1} 2(m-n+j)j & \\
= & < \text{ algebra } > \\
2(m-n)\sum_{j=0}^{n-1} j + 2\sum_{j=0}^{n-1} j^2 & \\
\approx & < \sum_{j=0}^{n-1} j \approx n^2/2 \text{ and } \sum_{j=0}^{n-1} j^2 \approx n^3/3 > \\
(m-n)n^2 + \frac{2}{3}n^3 & \\
= & < \text{ simplify } > \\
mn^2 - \frac{1}{3}n^3 &
\end{array}$$

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 nonzeroes 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

    $$\underbrace{\left(\begin{array}{c|c} A_{00} & a_{01} \\ \hline a_{10}^T & \alpha_{11} \\ \hline A_{20} & a_{21} \end{array}\right)}_{A} = \underbrace{\left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \\ \hline L_{20} & l_{21} \end{array}\right)}_{L} \underbrace{\left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & \upsilon_{11} \end{array}\right)}_{U}. \tag{5.2.1}$$

If we can show that the different parts of L and U exist, are unique, and $\upsilon_{11} \neq 0$, we are done (since then U is nonsingular). (5.2.1) can be rewritten as

$$\left(\begin{array}{c} A_{00} \\ \hline a_{10}^T \\ A_{20} \end{array}\right) = \left(\begin{array}{c} L_{00} \\ \hline l_{10}^T \\ L_{20} \end{array}\right) U_{00} \text{ and } \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ a_{21} \end{array}\right) = \left(\begin{array}{c} L_{00}u_{01} \\ \hline l_{10}^T u_{01} + \upsilon_{11} \\ L_{20}u_{01} + l_{21}\upsilon_{11} \end{array}\right),$$

or, equivalently,

$$\left\{\begin{array}{rcl} L_{00}u_{01} & = & a_{01} \\ \upsilon_{11} & = & \alpha_{11} - l_{10}^T u_{01} \\ l_{21} & = & (a_{21} - L_{20}u_{01})/\upsilon_{11} \end{array}\right.$$

Now, by the Inductive Hypothesis L_{00}, l_{10}^T, and L_{20} exist and are unique. So the question is whether u_{01}, υ_{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.
- υ_{11} exists, is unique, and is nonzero. Since l_{10}^T and u_{01} exist and are unique, $\upsilon_{11} = \alpha_{11} - l_{10}^T u_{01}$ exists and is unique. It is also nonzero since the principal leading submatrix of A given by

  $$\left(\begin{array}{c|c} A_{00} & a_{01} \\ \hline a_{10}^T & \alpha_{11} \end{array}\right) = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \end{array}\right) \left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & \upsilon_{11} \end{array}\right),$$

  is nonsingular by assumption and therefore υ_{11} must be nonzero.
- l_{21} exists and is unique. Since υ_{11} exists, is unique, and is nonzero,

  $$l_{21} = (a_{21} - L_{20}a_{01})/\upsilon_{11}$$

  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.

$A =$ 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}

$\alpha_{11} := \upsilon_{11} = \alpha_{11} - a_{10}^T a_{01}$

$a_{21} := a_{21} - A_{20}a_{01}$

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

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

endwhile

Figure 5.2.3.5 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,

$$\sum_{k=0}^{n-1} \left(k^2 + 2(m-k-1)k\right) \text{ flops.}$$

Let us now simplify this:

$$\begin{array}{ll} & \sum_{k=0}^{n-1} \left(k^2 + 2(m-k-1)k\right) \\ = & \quad < \text{ algebra } > \\ & \sum_{k=0}^{n-1} k^2 + 2\sum_{k=0}^{n-1}(m-k-1)k \\ = & \quad < \text{ algebra } > \\ & \sum_{k=0}^{n-1} 2(m-1)k - \sum_{k=0}^{n-1} k^2 \\ \approx & \quad < \sum_{j=0}^{n-1} j \approx n^2/2 \text{ and } \sum_{j=0}^{n-1} j^2 \approx n^3/3 > \\ & (m-1)n^2 - \frac{1}{3}n^3 \end{array}$$

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

$$A = \begin{pmatrix} A_{00} & a_{01} \\ a_{10}^T & \alpha_{11} \end{pmatrix}, L = \begin{pmatrix} L_{00} & 0 \\ l_{10}^T & 1 \end{pmatrix}, U = \begin{pmatrix} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{pmatrix}.$$

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 υ_{11} in the current iteration. Then fill in the algorithm:

$A =$ 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 $n(A_{TL}) < n(A)$

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

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

endwhile

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. ◊

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

$$\left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & -\lambda_{21} & 1 & 0 \\ 0 & -\lambda_{31} & 0 & 1 \end{array}\right) \left(\begin{array}{c} \widetilde{a}_0^T \\ \hline \widetilde{a}_1^T \\ \widetilde{a}_2^T \\ \widetilde{a}_3^T \end{array}\right) =$$

Solution.

$$\left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & -\lambda_{21} & 1 & 0 \\ 0 & -\lambda_{31} & 0 & 1 \end{array}\right) \left(\begin{array}{c} \widetilde{a}_0^T \\ \hline \widetilde{a}_1^T \\ \widetilde{a}_2^T \\ \widetilde{a}_3^T \end{array}\right) = \left(\begin{array}{c} \widetilde{a}_0^T \\ \hline \widetilde{a}_1^T \\ \left(\begin{array}{c} \widetilde{a}_2^T \\ \widetilde{a}_3^T \end{array}\right) - \left(\begin{array}{c} \lambda_{21} \\ \lambda_{31} \end{array}\right) \widetilde{a}_1^T \end{array}\right) = \left(\begin{array}{c} \widetilde{a}_0^T \\ \hline \widetilde{a}_1^T \\ \widetilde{a}_2^T - \lambda_{21} \widetilde{a}_1^T \\ \widetilde{a}_3^T - \lambda_{31} \widetilde{a}_1^T \end{array}\right).$$

□

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

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & \widehat{a}_{21} & \widehat{A}_{22} \end{array}\right) := \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right)$$

how should l_{21} be chosen if we want $\widehat{a}_{21}$ to be a zero vector?

Solution.

$$\begin{array}{l} \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right) \\ = \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & -l_{21}\alpha_{11} + a_{21} & -l_{21} a_{12}^T + A_{22} \end{array}\right) \\ = \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} - \alpha_{11} l_{21} & A_{22} - l_{21} a_{12}^T \end{array}\right) \end{array}$$

If $l_{21} = a_{21}/\alpha_{11}$ then $\widetilde{a}_{21} = a_{21} - \alpha_{11} a_{21}/\alpha_{11} = 0$.

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"' $\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \end{array}\right)$ and subtract these from the rest of the rows: $\left(\begin{array}{c|c} a_{21} & A_{22} \end{array}\right)$ 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).

$A =$ 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 $n(A_{TL}) < n(A)$

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

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

$$\left.\begin{array}{l} \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right) \\ \quad := \left(\begin{array}{c|cc} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & 0 \end{array}\right)\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right) \\ \quad = \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array}\right) \end{array}\right\} \begin{array}{l} a_{21} := 0 \\ A_{22} := A_{22} - l_{21}a_{12}^T \end{array}$$

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

endwhile

Figure 5.2.4.3 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.

$$\left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I_{(n-k-1)\times(n-k-1)} \end{array}\right)\left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array}\right)$$
$$= \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21}+l_{21} & I \end{array}\right)$$
$$= \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & I \end{array}\right)$$

Starting with an $m\times m$ matrix A, the algorithm computes a sequence of m Gauss transforms $L_0, \ldots, L_{m-1}$, each of the form

$$L_k = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array}\right), \tag{5.2.2}$$

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

$$\tilde{L}_{k-1} = L_0^{-1}L_1^{-1}\ldots L_{k-1}^{-1} = \left(\begin{array}{c|cc} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ L_{20} & 0 & I \end{array}\right) \quad \text{and} \quad L_k^{-1} = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array}\right),$$

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.

$$\begin{array}{rcl} \tilde{L}_k & = & L_0^{-1}L_1^{-1}\cdots L_{k-1}L_k^{-1} = \tilde{L}_{k-1}L_k^{-1} \\ & = & \left(\begin{array}{c|cc} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ L_{20} & 0 & I \end{array}\right)\left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array}\right) \\ & = & \left(\begin{array}{c|cc} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ L_{20} & l_{21} & I \end{array}\right). \end{array}$$

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 chose 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

$$\left(\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array}\right)\left(\begin{array}{c} \chi_0 \\ \chi_1 \end{array}\right) = \left(\begin{array}{c} 2 \\ 1 \end{array}\right)$$

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

$$\left(\begin{array}{c} \chi_0 \\ \chi_1 \end{array}\right) = \left(\begin{array}{c} 1 \\ 2 \end{array}\right).$$

The point of the exercise: Gaussian elemination 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 to 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

$$\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 - 10^{-k} \end{pmatrix}.$$

which becomes

$$\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

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 $\{\pi_0, \pi_1, \ldots, \pi_{n-1}\}$ is a permutation (rearrangement) of the integers $\{0, 1, \ldots, n-1\}$, we define the permutation matrix $P(p)$ by

$$P(p) = \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}.$$

$\Diamond$

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.

$$
\begin{array}{l}
P(p)x \\
\quad = \quad < \text{ definition } > \\
\begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix} x \\
\quad = \quad < \text{ matrix-vector multiplication by rows } > \\
\begin{pmatrix} e_{\pi_0}^T x \\ \vdots \\ e_{\pi_{n-1}}^T x \end{pmatrix} \\
\quad = \quad < e_j^T x = x_j > \\
\begin{pmatrix} \chi_{\pi_0} \\ \vdots \\ \chi_{\pi_{n-1}} \end{pmatrix}
\end{array}
$$

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

$$
p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{n-1} \end{pmatrix} \text{ and } A = \begin{pmatrix} \widetilde{a}_0^T \\ \vdots \\ \widetilde{a}_{n-1}^T \end{pmatrix}.
$$

Evaluate $P(p)A$.

Solution.

$$
\begin{array}{l}
P(p)A \\
\quad = \quad < \text{ definition } > \\
\begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix} A \\
\quad = \quad < \text{ matrix-matrix multiplication by rows } > \\
\begin{pmatrix} e_{\pi_0}^T A \\ \vdots \\ e_{\pi_{n-1}}^T A \end{pmatrix} \\
\quad = \quad < e_j^T A = \widetilde{a}_j^T > \\
\begin{pmatrix} \widetilde{a}_{\pi_0}^T \\ \vdots \\ \widetilde{a}_{\pi_{n-1}}^T \end{pmatrix}
\end{array}
$$

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.

$$
\begin{array}{l}
AP(p)^T \\
\quad = \quad < \text{ definition } > \\
A \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}^T \\
\quad = \quad < \text{ transpose } P(p) > \\
A \begin{pmatrix} e_{\pi_0} & \cdots & e_{\pi_{n-1}} \end{pmatrix} \\
\quad = \quad < \text{ matrix-matrix multiplication by columns } > \\
\begin{pmatrix} Ae_{\pi_0} & \cdots & Ae_{\pi_{n-1}} \end{pmatrix} \\
\quad = \quad < Ae_j = a_j > \\
\begin{pmatrix} a_{\pi_0} & \cdots & a_{\pi_{n-1}} \end{pmatrix}
\end{array}
$$

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.

$$
\begin{array}{l}
PP(p)^T \\
\quad = \quad < \text{ definition } > \\
\begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix} \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}^T \\
\quad = \quad < \text{ transpose } P(p) > \\
\begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix} \begin{pmatrix} e_{\pi_0} & \cdots & e_{\pi_{n-1}} \end{pmatrix} \\
\quad = \quad < \text{ evaluate } > \\
\begin{pmatrix} e_{\pi_0}^T e_{\pi_0} & e_{\pi_0}^T e_{\pi_1} & \cdots & e_{\pi_0}^T e_{\pi_{n-1}} \\ e_{\pi_1}^T e_{\pi_0} & e_{\pi_1}^T e_{\pi_1} & \cdots & e_{\pi_1}^T e_{\pi_{n-1}} \\ \vdots & \vdots & & \vdots \\ e_{\pi_{n-1}}^T e_{\pi_0} & e_{\pi_{n-1}}^T e_{\pi_1} & \cdots & e_{\pi_{n-1}}^T e_{\pi_{n-1}} \end{pmatrix} \\
\quad = \quad < e_i^T e_j = \cdots > \\
\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}
\end{array}
$$

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 $\pi \in \{0, \ldots, n-1\}$ define the elementary pivot matrix

$$\widetilde{P}(\pi) = \left(\begin{array}{c} \hline e_\pi^T \\ \hline e_1^T \\ \vdots \\ e_{\pi-1}^T \\ \hline e_0^T \\ \hline e_{\pi+1}^T \\ \vdots \\ e_{n-1}^T \end{array}\right)$$

or, equivalently,

$$\widetilde{P}(\pi) = \begin{cases} I_n & \text{if } \pi = 0 \\ \left(\begin{array}{c|c|c|c} 0 & 0 & 1 & 0 \\ \hline 0 & I_{\pi-1} & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & I_{n-\pi-1} \end{array}\right) & \text{otherwise,} \end{cases}$$

where n is the size of the permutation matrix. ◊

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

$$p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{k-1} \end{pmatrix}, \quad \text{where} \quad 1 \leq k \leq n \text{ and } 0 \leq \pi_i < n - i, \tag{5.3.1}$$

then $\widetilde{P}(p)$ will denote the sequence of pivots

$$\widetilde{P}(p) = \begin{pmatrix} I_{k-1} & 0 \\ 0 & \widetilde{P}(\pi_{k-1}) \end{pmatrix} \begin{pmatrix} I_{k-2} & 0 \\ 0 & \widetilde{P}(\pi_{k-2}) \end{pmatrix} \cdots \begin{pmatrix} 1 & 0 \\ 0 & \widetilde{P}(\pi_1) \end{pmatrix} \widetilde{P}(\pi_0).$$

(Here $\widetilde{P}(\cdot)$ 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 $\widetilde{P}(p)A$.

Solution.

$$
\begin{array}{l}
\widetilde{P}(p)A \\
\quad = \quad < \text{ instantiate } > \\
\widetilde{P}(\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}) \begin{pmatrix} 0.0 & 0.1 & 0.2 \\ 1.0 & 1.1 & 1.2 \\ 2.0 & 2.1 & 2.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \\
\quad = \quad < \text{ definition of } \widetilde{P}(\cdot) > \\
\left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{P}(\begin{pmatrix} 1 \\ 1 \end{pmatrix}) \end{array}\right) \widetilde{P}(2) \begin{pmatrix} 0.0 & 0.1 & 0.2 \\ 1.0 & 1.1 & 1.2 \\ 2.0 & 2.1 & 2.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \\
\quad = \quad < \text{ swap first row with row indexed with 2 } > \\
\left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{P}(\begin{pmatrix} 1 \\ 1 \end{pmatrix}) \end{array}\right) \left(\begin{array}{ccc} 2.0 & 2.1 & 2.2 \\ \hline 1.0 & 1.1 & 1.2 \\ 0.0 & 0.1 & 0.2 \\ 3.0 & 3.1 & 3.2 \end{array}\right) \\
\quad = \quad < \text{ partitioned matrix-matrix multiplication } > \\
\left(\begin{array}{c} \begin{pmatrix} 2.0 & 2.1 & 2.2 \end{pmatrix} \\ \hline \widetilde{P}(\begin{pmatrix} 1 \\ 1 \end{pmatrix}) \begin{pmatrix} 1.0 & 1.1 & 1.2 \\ 0.0 & 0.1 & 0.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \end{array}\right) \quad = \quad < \text{ swap current first row with row indexed with 1 relative to that row } > \\
\left(\begin{array}{c} \begin{pmatrix} 2.0 & 2.1 & 2.2 \\ 0.0 & 0.1 & 0.2 \end{pmatrix} \\ \hline \widetilde{P}(\begin{pmatrix} 1 \end{pmatrix}) \begin{pmatrix} 1.0 & 1.1 & 1.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \end{array}\right) \\
\quad = \quad < \text{ swap current first row with row indexed with 1 relative to that row } > \\
\left(\begin{array}{c} \begin{pmatrix} 2.0 & 2.1 & 2.2 \\ 0.0 & 0.1 & 0.2 \\ 3.0 & 3.1 & 3.2 \end{pmatrix} \\ \hline \begin{pmatrix} 1.0 & 1.1 & 1.2 \end{pmatrix} \end{array}\right) \\
\quad = \\
\begin{pmatrix} 2.0 & 2.1 & 2.2 \\ 0.0 & 0.1 & 0.2 \\ 3.0 & 3.1 & 3.2 \\ 1.0 & 1.1 & 1.2 \end{pmatrix}
\end{array}
$$

□

The relation between $\widetilde{P}(\cdot)$ and $P(\cdot)$ is tricky to specify:

$$
\widetilde{P}(\begin{pmatrix} \pi_0 \\ \pi_1 \\ \vdots \\ \pi_{k-1} \end{pmatrix}) = P(\begin{pmatrix} I_{k-1} & 0 \\ 0 & \widetilde{P}(\pi_{k-1}) \end{pmatrix} \cdots \begin{pmatrix} 1 & 0 \\ 0 & \widetilde{P}(\pi_1) \end{pmatrix} \widetilde{P}(\pi_0) \begin{pmatrix} 0 \\ 1 \\ \vdots \\ k-1 \end{pmatrix}).
$$

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 $\widetilde{P}(p)A = LU$. We represent this operation by

$$[A, p] := \text{LUpiv}A,$$

where upon completion A has been overwritten by $\{L\backslash U\}$, 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 $\widetilde{P}(\pi_1)$ 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 $\pi_1 = \text{maxi}\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}$.
- Permute the rows: $\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} := \widetilde{P}(\pi_1) \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix}$.
- 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
$$\pi_1 := \text{maxi}\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right).$$
- Permute the rows:
$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{c|c} I & 0 \\ \hline 0 & \widetilde{P}(\pi_1) \end{array}\right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right)$$
- Update
$$l_{21} := a_{21}/\alpha_{11}.$$
- Update
$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{c|cc} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array}\right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array}\right)$$
$$= \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array}\right).$$

This algorithm is summarized in Figure 5.3.3.1. In that algorithm, the lower triangular matrix L is accumulated below the diagonal.

$[A, p] = \text{LUpiv-right-looking}(A)$

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

A_{TL} is $0 \times 0, p_T$ has 0 elements

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

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

$\pi_1 := \text{maxi}\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)$

$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array}\right) \left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right)$

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

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

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

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 $L_0, \ldots, L_{n-1}$ and permutations $P_0, \ldots, P_{n-1}$ 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 $P_k = \left(\begin{array}{c|c} I_{k\times k} & 0 \\ \hline 0 & \widetilde{P}(\pi) \end{array}\right)$ 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 $L_0^\star, \ldots L_{n-1}^\star$ such that

$$A = P_0 \cdots P_{n-1} \underbrace{L_0^\star \cdots L_{n-1}^\star}_{L} U$$

or, equivalently,

$$\widetilde{P}(p)A = P_{n-1}\cdots P_0 A = \underbrace{L_0^\star \cdots L_{n-1}^\star}_{L} U,$$

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

$$\left(\begin{array}{cc} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{array}\right)$$

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: $\widetilde{P}(p)A$.

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:

  $$[A, p] := \text{LUpiv}(A).$$

  A now contains L and U and $\widetilde{P}(p)A = LU$.

- We notice that $Ax = b$ is equivalent to $\widetilde{P}(p)Ax = \widetilde{P}(p)b$. Thus, we compute $y := \widetilde{P}(p)b$. Usually, y overwrites b.

- Next, we recognize that $\widetilde{P}(p)Ax = y$ 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

$$\begin{array}{rcl} \zeta_1 & = & \psi_1 \\ \zeta_1 l_{21} + L_{22} z_2 & = & y_2, \end{array}$$

which can be rearranged as

$$\begin{array}{rcl} \zeta_1 & = & \psi_1 \\ L_{22} z_2 & = & y_2 - \zeta_1 l_{21}. \end{array}$$

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)

$L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$

L_{TL} is 0×0 and y_T has 0 elements

while $n(L_{TL}) < n(L)$

$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ \hline y_2 \end{array}\right)$

$y_2 := y_2 - \psi_1 l_{21}$

$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ \hline y_2 \end{array}\right)$

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)

$$U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right), z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right)$$

U_{BR} is 0×0 and z_B has 0 elements

while $n(U_{BR}) < n(U)$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{ccc} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array}\right)$$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{ccc} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ z_2 \end{array}\right)$$

endwhile

Hint: Partition

$$\left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{array}\right)\left(\begin{array}{c} x_0 \\ \chi_1 \end{array}\right) = \left(\begin{array}{c} z_0 \\ \zeta_1 \end{array}\right).$$

Solution. Multiplying this out yields

$$\left(\begin{array}{c} U_{00}x_0 + u_{01}\chi_1 \\ \upsilon_{11}\chi_1 \end{array}\right) = \left(\begin{array}{c} z_0 \\ \zeta_1 \end{array}\right).$$

So, $\chi_1 = \zeta_1/\upsilon_{11}$ 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)

$$U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right), z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right)$$

U_{BR} is 0×0 and z_B has 0 elements

while $n(U_{BR}) < n(U)$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ \hline z_2 \end{array}\right)$$

$\zeta_1 := \zeta_1/\upsilon_{11}$

$z_0 := z_0 - \zeta_1 u_{01}$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_0 \\ \zeta_1 \\ \hline z_2 \end{array}\right)$$

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

$$L \rightarrow \begin{pmatrix} L_{00} & 0 \\ l_{10}^T & 1 \end{pmatrix}, \quad z \rightarrow \begin{pmatrix} z_0 \\ \zeta_1 \end{pmatrix} \quad \text{and} \quad y \rightarrow \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}.$$

Then

$$\underbrace{\begin{pmatrix} L_{00} & 0 \\ l_{10}^T & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} z_0 \\ \zeta_1 \end{pmatrix}}_{z} = \underbrace{\begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}}_{y}.$$

Multiplying out the left-hand side yields

$$\begin{pmatrix} L_{00} z_0 \\ l_{10}^T z_0 + \zeta_1 \end{pmatrix} = \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}$$

and the equalities

$$\begin{array}{rcl} L_{00} z_0 & = & y_0 \\ l_{10}^T z_0 + \zeta_1 & = & \psi_1. \end{array}$$

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)

$L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$

L_{TL} is 0×0 and y_T has 0 elements

while $n(L_{TL}) < n(L)$

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ \hline y_2 \end{array}\right)$$

$\psi_1 := \psi_1 - l_{10}^T y_0$

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ \hline y_2 \end{array}\right)$$

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)

$$U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right), z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right)$$

U_{BR} is 0×0 and z_B has 0 elements

while $n(U_{BR}) < n(U)$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{ccc} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ \hline z_2 \end{array}\right)$$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{ccc} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ \hline z_2 \end{array}\right)$$

endwhile

Hint: Partition

$$U \rightarrow \begin{pmatrix} \upsilon_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix}.$$

Solution. Partition

$$\begin{pmatrix} \upsilon_{11} & u_{12}^T \\ 0 & U_{22} \end{pmatrix} \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \zeta_1 \\ z_2 \end{pmatrix}.$$

Multiplying this out yields

$$\begin{pmatrix} \upsilon_{11}\chi_1 + u_{12}^T x_2 \\ U_{22}x_2 \end{pmatrix} = \begin{pmatrix} \zeta_1 \\ z_2 \end{pmatrix}.$$

So, if we assume that x_2 has already been computed and has overwritten z_2, then χ_1 can be computed as

$$\chi_1 = (\zeta_1 - u_{12}^T x_2)/\upsilon_{11}$$

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 $n(U_{BR}) < n(U)$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ \hline z_2 \end{array}\right)$$

$\zeta_1 := \zeta_1 - u_{12}^T z_2$

$\zeta_1 := \zeta_1/\upsilon_{11}$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|cc} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & \upsilon_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ \hline z_2 \end{array}\right)$$

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:

$$y := \alpha x + y.$$

"*a*lpha times x *p*lus 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). ◊

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 $\alpha\chi^2 > 0$. □

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 jth 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

$$\begin{array}{l} 0 \\ \quad < \quad < A \text{ is HPD } > \\ x^H A x \\ \quad = \quad < \text{ partition } > \\ \begin{pmatrix} 0 \\ x_2 \end{pmatrix}^H \left(\begin{array}{c|c} \alpha_{11} & a_{21}^H \\ \hline a_{21} & A_{22} \end{array} \right) \begin{pmatrix} 0 \\ x_2 \end{pmatrix} \\ \quad = \quad < \text{ multiply out } > \\ x_2^H A_{22} x_2. \end{array}$$

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 $A = U^H U$ 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
  $$A = \begin{pmatrix} \alpha_{11} & \star \\ a_{21} & A_{22} \end{pmatrix} \quad \text{and} \quad L = \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix}. \tag{5.4.1}$$

  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
  $$\begin{array}{rcl} \begin{pmatrix} \alpha_{11} & \star \\ a_{21} & A_{22} \end{pmatrix} & = & \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix}^H = \begin{pmatrix} \lambda_{11} & 0 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} \bar{\lambda}_{11} & l_{21}^H \\ 0 & L_{22}^H \end{pmatrix} \\ & = & \begin{pmatrix} |\lambda_{11}|^2 & \star \\ \bar{\lambda}_{11} l_{21} & l_{21} l_{21}^H + L_{22} L_{22}^H \end{pmatrix}, \end{array}$$

  from which we conclude that
  $$\begin{array}{ll} \alpha_{11} = |\lambda_{11}|^2 & \star \\ a_{21} = \lambda_{11} l_{21} & A_{22} = l_{21} l_{21}^H + L_{22} L_{22}^H \end{array}$$

  or, equivalently,
  $$\begin{array}{ll} \lambda_{11} = \pm\sqrt{\alpha_{11}} & \star \\ l_{21} = a_{21} / \bar{\lambda}_{11} & L_{22} = \text{Chol}(A_{22} - l_{21} l_{21}^H) \end{array}.$$

- 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 = 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)

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

endwhile

Figure 5.4.3.1 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 kth 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

$$
\begin{array}{l}
C_{\rm Chol}(n) \\
\quad \approx \quad < \text{ sum over all iterations } > \\
\quad \underbrace{\sum_{k=0}^{n-1}(n-k-1)^2}_{\text{(Due to update of } A_{22})} + \underbrace{\sum_{k=0}^{n-1}(n-k-1)}_{\text{(Due to update of } a_{21})} \\
\quad = \quad < \text{ change of variables } j = n-k-1 > \\
\sum_{j=0}^{n-1} j^2 + \sum_{j=0}^{n-1} j \\
\quad \approx \quad < \sum_{j=0}^{n-1} j^2 \approx n^3/3; \sum_{j=0}^{n-1} j \approx n^2/2 > \\
\frac{1}{3}n^3 + \frac{1}{2}n^2 \\
\quad \approx \quad < \text{ remove lower order term } > \\
\frac{1}{3}n^3.
\end{array}
$$

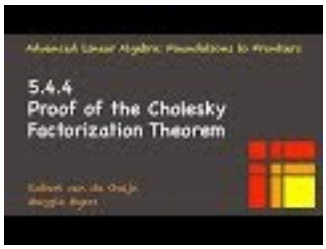
Remark 5.4.3.2 Comparing the cost of the Cholesky factorization to that of the LU factorization in Homework 5.2.2.1, we see that taking advantage of symmetry cuts the cost approximately in half.

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 Factorizaton Theorem



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

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 $x \neq 0$,

$$
\begin{array}{l}
0 \\
\quad < \quad < A \text{ is HPD } > \\
x^H A x \\
\quad = \quad < \text{ partition } > \\
\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix}^H \begin{pmatrix} \alpha_{11} & a_{21}^H \\ a_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \\
\quad = \quad < \text{ partitioned multiplication } > \\
\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix}^H \begin{pmatrix} \alpha_{11}\chi_1 + a_{21}^H x_2 \\ a_{21}\chi_1 + A_{22}x_2 \end{pmatrix} \\
\quad = \quad < \text{ partitioned multiplication } > \\
\alpha_{11}\overline{\chi}_1\chi_1 + \bar{\chi}_1 a_{21}^H x_2 + x_2^H a_{21}\chi_1 + x_2^H A_{22}x_2 \\
\quad = \quad < \text{ linear algebra } > \\
\alpha_{11}\frac{a_{21}^H x_2}{\alpha_{11}}\frac{x_2^H a_{21}}{\alpha_{11}} - \frac{x_2^H a_{21}}{\alpha_{11}}a_{21}^H x_2 - x_2^H a_{21}\frac{a_{21}^H x_2}{\alpha_{11}} + x_2^H A_{22}x_2 \\
\quad = \quad < \text{ since } x_2^H a_{21}, a_{21}^H x_2 \text{ are scalars and hence can move around; } \alpha_{11}/\alpha_{11} = 1 > \\
x_2^H a_{21}\frac{a_{21}^H x_2}{\alpha_{11}} - x_2^H a_{21}\frac{a_{21}^H x_2}{\alpha_{11}} - x_2^H a_{21}\frac{a_{21}^H x_2}{\alpha_{11}} + x_2^H A_{22}x_2 \\
\quad = \quad < \text{ cancel terms; factor out } x_2^H \text{ and } x_2 > \\
x_2^H(A_{22} - \frac{a_{21}a_{21}^H}{\alpha_{11}})x_2 \\
\quad = \quad < \text{ simplify } > \\
x_2^H(A_{22} - l_{21}l_{21}^H)x_2.
\end{array}
$$

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

$$\|b - A\hat{x}\|_2 = \min_{x \in \mathbb{C}^n} \|b - Ax\|_2 \tag{5.4.2}$$

equals the solution to the normal equations

$$\underbrace{A^H A}_{B} \widehat{x} = \underbrace{A^H b}_{y} .$$

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 $j := 0, \ldots, n-1$
  $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$
  for $i := j+1, \ldots, n-1$
    $\alpha_{i,j} := \alpha_{i,j} / \alpha_{j,j}$
  end
  for $k := j+1, \ldots, n-1$
    for $i := k, \ldots, n-1$
      $\alpha_{i,k} := \alpha_{i,k} - \alpha_{i,j} \alpha_{k,j}$
    endfor
  endfor
endfor

```
do j=1, n
  A(j,j) = sqrt(A(j,j))

  do i=j+1,n
    A(i,j) = A(i,j) / A(j,j)
  enddo

  do k=j+1,n
    do i=k,n
      A(i,k) = A(i,k) - A(i,j) * A(k,j)
    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

for $j := 0, \ldots, n-1$
  $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$
  $\alpha_{j+1:n-1,j} := \alpha_{j+1:n-1,j} / \alpha_{j,j}$
  for $k := j+1, \ldots, n-1$
    $\alpha_{k:n-1,k} := \alpha_{k:n-1,k} - \alpha_{k,j} \alpha_{k:n-1,j}$
  endfor
endfor

```
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 )

  do k=j+1,n
    call daxpy( n-k+1, -A(k,j), A(k,j), 1,
                                A(k,k), 1 )
  enddo
enddo
```

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 $\alpha x x^T + A$ where x is stored at address x with increment incx and α is stored in alpha. Since both A and $\alpha x x^T + A$ are symmetric, only the triangular part indicated by uplo is updated. This is captured by the below algorithm and implementation.

for $j := 0, \ldots, n-1$
  $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$
  $\alpha_{j+1:n-1,j} := \alpha_{j+1:n-1,j} / \alpha_{j,j}$
  $\alpha_{j+1:n-1,j+1:n-1} :=$
    $\alpha_{j+1:n-1,j+1:n-1}$
      $- \text{tril}(\alpha_{j+1:n-1,j} \alpha_{j+1:n-1,j}^T)$
endfor

```
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 )

  call dsyr( "Lower triangular", n-j+1, -1.0d00,
                 A(j+1,j), 1,  A(j+1,j+1), ldA )
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 = 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 $n(A_{TL}) < n(A)$

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12}^T \\ A_{20} & A_{21} & A_{22} \end{array}\right)$

  $A_{11} := L_{11} = \text{Chol}(()A_{11})$
  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

$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12}^T \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$

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.

$A =$ 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 $n(A_{TL}) < n(A)$

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

$\vdots$

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

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 $\widehat{A}$ to denote the original contents of A. We will say that the precondition for the algorithm is that

$$A = \widehat{A}$$

  (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

$$A = L\backslash U \wedge LU = \widehat{A}$$

  (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 comformal to (in the same way) as is A.

- Thus, before and after each iteration of the loop the matrices are viewed as quadrants:

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right), L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right), \text{and} \quad U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right).$$

  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

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L\backslash U_{TL} & U_{TR} \\ \hline L_{BL} & L\backslash U_{BR} \end{array}\right)$$
$$\wedge \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) = \left(\begin{array}{c|c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right)$$

- Manipulating this yields what we call the Partitioned Matrix Expression (PME), which can be viewed as a recursive definition of the LU factorization:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L\backslash U_{TL} & U_{TR} \\ \hline L_{BL} & L\backslash U_{BR} \end{array}\right)$$
$$\wedge \begin{array}{c|c} L_{TL}U_{TL} = \widehat{A}_{TL} & L_{TL}U_{TR} = \widehat{A}_{TR} \\ \hline L_{BL}U_{TL} = \widehat{A}_{BL} & L_{BR}U_{BR} = \widehat{A}_{BR} - L_{BL}U_{TR}. \end{array}$$

- 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:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L\backslash U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} - L_{BL}U_{TR} \end{array}\right)$$
$$\wedge \begin{array}{c|c} L_{TL}U_{TL} = \widehat{A}_{TL} & L_{TL}U_{TR} = \widehat{A}_{TR} \\ \hline L_{BL}U_{TL} = \widehat{A}_{BL} & \end{array}.$$

  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 $\widehat{A}_{BR} - L_{BL}U_{TR}$, and $\widehat{A}_{BR} - L_{BL}U_{TR} = L_{BR}U_{BR}$.

- 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:

$\left(\begin{array}{c\|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c\|c} L\backslash U_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right)$ Invariant 1	$\left(\begin{array}{c\|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c\|c} L\backslash U_{TL} & U_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right)$ Invariant 2
$\left(\begin{array}{c\|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c\|c} L\backslash U_{TL} & \widehat{A}_{TR} \\ \hline \widehat{L}_{BL} & \widehat{A}_{BR} \end{array}\right)$ Invariant 3	$\left(\begin{array}{c\|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c\|c} L\backslash U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} \end{array}\right)$ Invariant 4

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L\backslash U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} - L_{BL}U_{TR} \end{array}\right)$$
Invariant 5

- 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:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L\backslash U_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right) \wedge L_{TL}U_{TL} = \widehat{A}_{TL},$$

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

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|cc} L\backslash U_{00} & \widehat{a}_{01} & \widehat{A}_{02} \\ \hline \widehat{a}_{10}^T & \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array}\right) \wedge L_{00}U_{00} = \widehat{A}_{00}$$

while after updating A it must contain

$$\left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{cc|c} L\backslash U_{00} & u_{01} & \widehat{A}_{02} \\ l_{10}^T & \upsilon_{11} & \widehat{a}_{12}^T \\ \hline \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22}, \end{array}\right)$$
$$\wedge \underbrace{\left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array}\right)\left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{array}\right) = \left(\begin{array}{cc} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{array}\right)}_{\begin{array}{cc} L_{00}U_{00} = \widehat{A}_{00} & L_{00}u_{01} = \widehat{a}_{01} \\ l_{10}^T U_{00} = \widehat{a}_{10}^T & l_{10}^T u_{01} + \upsilon_{11} = \widehat{\alpha}_{11} \end{array}}$$

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 $a_{01} = \widehat{a}_{01}$ 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 $a_{10}^T = \widehat{a}_{10}^T$ before this update.)
- Update $\alpha_{11} := \upsilon_{11} = \alpha_{11} - l_{10}^T u_{01}$. (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

$\alpha_{11} := \upsilon_{11} = \alpha_{11} - a_{10}^T a_{01}$

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

endwhile

Figure 5.5.1.3 Variant 1 (bordered) LU factorization algorithm. Here A_{00} stores $L\backslash U_{00}$.

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 kth iteration, A_{00} is $k \times k$, for $k = 0, \ldots, n-1$. 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 $\upsilon_{11} = \alpha_{11} - l_{10}^T u_{01}$, overwriting α_{11} with the result. Cost: $2k$ flops.

Thus, the total cost is given by

$$\sum_{k=0}^{n-1} \left(k^2 + k^2 + 2k\right) \approx 2\sum_{k=0}^{n-1} k^2 \approx 2\frac{1}{3}n^3 = \frac{2}{3}n^3.$$

5.5.1.2 Variant 2: Up-looking algorithm

Consider next the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L\backslash U_{TL} & U_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array}\right) \wedge \; L_{TL}U_{TL} = \widehat{A}_{TL} \;\Big|\; L_{TL}U_{TR} = \widehat{A}_{TR}$$

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

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|cc} L\backslash U_{00} & u_{01} & U_{02} \\ \hline \widehat{a}_{10}^T & \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array}\right)$$
$$\wedge \underbrace{\begin{array}{c} L_{00}U_{00} = \widehat{A}_{00} \;\Big|\; L_{00}\begin{pmatrix} u_{01} & U_{02} \end{pmatrix} = \begin{pmatrix} \widehat{a}_{01} & \widehat{A}_{02} \end{pmatrix} \\ \hline L_{00}U_{00} = \widehat{A}_{00} \;\Big|\; L_{00}u_{01} = \widehat{a}_{01} \quad L_{00}U_{02} = \widehat{A}_{02} \end{array}}$$

while after updating A it must contain

$$\left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{cc|c} L\backslash U_{00} & u_{01} & U_{02} \\ l_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22}, \end{array}\right)$$
$$\wedge \underbrace{\begin{array}{c} \begin{pmatrix} L_{00} & 0 \\ l_{10}^T & 1 \end{pmatrix}\begin{pmatrix} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{pmatrix} = \begin{pmatrix} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{pmatrix} \;\Big|\; \begin{pmatrix} L_{00} & 0 \\ l_{10}^T & 1 \end{pmatrix}\begin{pmatrix} U_{02} \\ u_{12}^T \end{pmatrix} = \begin{pmatrix} \widehat{A}_{00} \\ \widehat{a}_{12}^T \end{pmatrix} \\ \hline \begin{array}{cc|c} L_{00}U_{00} = \widehat{A}_{00} & L_{00}u_{01} = \widehat{a}_{01} & L_{00}U_{02} = \widehat{A}_{02} \\ l_{10}^T U_{00} = \widehat{a}_{10}^T & l_{10}^T u_{01} + \upsilon_{11} = \widehat{\alpha}_{11} & l_{10}^T U_{02} + u_{12}^T = \widehat{a}_{12}^T \end{array} \end{array}}$$

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 $\alpha_{11} := \upsilon_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$.
- 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

$\alpha_{11} := \upsilon_{11} = \alpha_{11} - a_{10}^T a_{01}$

$a_{12}^T := u_{12}^T = a_{12}^T - l_{10}^T U_{02}$

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

endwhile

Figure 5.5.1.4 Variant 2 (up-looking) LU factorization algorithm. Here A_{00} stores $L \backslash U_{00}$.

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 kth iteration, A_{00} is $k \times k$, for $k = 0, \ldots, n-1$. 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 $\alpha_{11} := \upsilon_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$. 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

$$
\begin{array}{ll}
\sum_{k=0}^{n-1} \left(k^2 + 2k + 2k(n-k-1)\right) & \\
\quad = & < \text{simplify} > \\
\sum_{k=0}^{n-1} \left(2kn - k^2\right) & \\
\quad = & < \text{algebra} > \\
2n \sum_{k=0}^{n-1} k - \sum_{k=0}^{n-1} k^2 & \\
\quad \approx & < \sum_{k=0}^{n-1} k \approx n^2/2; \sum_{k=0}^{n-1} k^2 \approx k^3/3 > \\
n^3 - \frac{n^3}{3} & \\
\quad = & < \text{simplify} > \\
\frac{2}{3}n^3. &
\end{array}
$$

5.5.1.3 Variant 3: Left-looking algorithm

Consider the loop invariant:

$$
\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L\backslash U_{TL} & \widehat{A}_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} \end{array}\right) \wedge \begin{array}{|c} L_{TL}U_{TL} = \widehat{A}_{TL} \\ \hline L_{BL}U_{TL} = \widehat{A}_{BxL}. \end{array}
$$

At the top of the loop, after repartitioning, A then contains

$$
\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|cc} L\backslash U_{00} & \widehat{a}_{01} & \widehat{A}_{02} \\ \hline l_{10}^T & \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ L_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array}\right) \wedge \begin{array}{|c} L_{00}U_{00} = \widehat{A}_{00} \\ \hline l_{10}^T U_{00} = \widehat{a}_{10}^T \\ \hline L_{20}U_{00} = \widehat{A}_{20} \end{array}
$$

while after updating A it must contain

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c|c} L\backslash U_{00} & u_{01} & \widehat{A}_{02} \\ \hline l_{10}^T & \upsilon_{11} & \widehat{a}_{12}^T \\ \hline L_{20} & l_{21} & \widehat{A}_{22}, \end{array}\right)$$

$$\wedge \underbrace{\begin{array}{c} \left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array}\right)\left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{array}\right) = \left(\begin{array}{cc} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{array}\right) \\ \hline \left(\begin{array}{cc} L_{20} & l_{21} \end{array}\right)\left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{array}\right) = \left(\begin{array}{cc} \widehat{A}_{20} & \widehat{a}_{21} \end{array}\right) \end{array}}_{\begin{array}{c|c} L_{00}U_{00} = \widehat{A}_{00} & L_{00}u_{01} = \widehat{a}_{01} \\ \hline l_{10}^T U_{00} = \widehat{a}_{10}^T & l_{10}^T u_{01} + \upsilon_{11} = \widehat{\alpha}_{11} \\ \hline L_{20}U_{00} = \widehat{A}_{20} & L_{20}u_{01} + l_{21}\upsilon_{11} = \widehat{a}_{21} \end{array}}$$

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 $\alpha_{11} := \upsilon_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$.
- Update $a_{21} := l_{21} = (a_{21} - L_{20}u_{01})/\upsilon_{11} = (a_{21} - A_{20}a_{10})/\alpha_{11}$.

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

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

Solve $L_{00}u_{01} = a_{01}$ overwriting a_{01} with the result

$\alpha_{11} := \upsilon_{11} = \alpha_{11} - a_{10}^T a_{01}$

$a_{21} := l_{21} = (a_{21} - A_{20}a_{10})/\alpha_{11}$

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

endwhile

Figure 5.5.1.5 Variant 3 (left-looking) LU factorization algorithm. Here A_{00} stores $L\backslash U_{00}$.

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 kth iteration, A_{00} is $k \times k$, for $k = 0, \ldots, n-1$. 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 $\alpha_{11} := \upsilon_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$. Approximate cost: $2k$ flops.
- Update $a_{21} := l_{21} = (a_{21} - L_{20}u_{01})/\upsilon_{11} = (a_{21} - A_{20}a_{10})/\alpha_{11}$. Approximate cost: $2(n-k-1)$ flops.

Thus, the total cost is approximately given by

$$
\begin{array}{l}
\sum_{k=0}^{n-1} \left(k^2 + 2k + 2k(n-k-1) \right) \\
\quad = \quad < \text{ simplify } > \\
\sum_{k=0}^{n-1} \left(2kn - k^2 \right) \\
\quad = \quad < \text{ algebra } > \\
2n \sum_{k=0}^{n-1} k - \sum_{k=0}^{n-1} k^2 \\
\quad \approx \quad < \sum_{k=0}^{n-1} k \approx n^2/2; \sum_{k=0}^{n-1} k^2 \approx k^3/3 > \\
n^3 - \frac{n^3}{3} \\
\quad = \quad < \text{ simplify } > \\
\frac{2}{3} n^3.
\end{array}
$$

5.5.1.4 Variant 4: Crout variant

Consider next the loop invariant:

$$
\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L\backslash U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} \end{array} \right) \wedge \begin{array}{c|c} L_{TL}U_{TL} = \widehat{A}_{TL} & L_{TL}U_{TR} = \widehat{A}_{TR} \\ \hline L_{BL}U_{TL} = \widehat{A}_{BL}. & \end{array}
$$

At the top of the loop, after repartitioning, A then contains

$$
\left(\begin{array}{cc|cc} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{cc|cc} L\backslash U_{00} & u_{01} & U_{02} \\ l_{10}^T & \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ \hline L_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array} \right)
$$

$$
\wedge \begin{array}{c|cc} L_{00}U_{00} = \widehat{A}_{00} & L_{00}u_{01} = \widehat{a}_{01} & L_{00}U_{02} = \widehat{A}_{02} \\ \hline l_{10}^T U_{00} = \widehat{a}_{10}^T & & \\ L_{20}U_{00} = \widehat{A}_{20} & & \end{array}
$$

while after updating A it must contain

$$
\left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{cc|c} L\backslash U_{00} & u_{01} & U_{02} \\ l_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline L_{20} & l_{21} & \widehat{A}_{22}, \end{array} \right)
$$

$$
\wedge \underbrace{\begin{array}{c|c} \left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{array} \right) = \left(\begin{array}{cc} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{array} \right) & \left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c} U_{02} \\ u_{12}^T \end{array} \right) = \left(\begin{array}{c} \widehat{A}_{02} \\ \widehat{a}_{12}^T \end{array} \right) \\ \hline \left(\begin{array}{cc} L_{20} & l_{21} \end{array} \right) \left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{array} \right) = \left(\begin{array}{cc} \widehat{A}_{20} & \widehat{a}_{21} \end{array} \right) & \end{array}}_{\begin{array}{c|cc} L_{00}U_{00} = \widehat{A}_{00} & L_{00}u_{01} = \widehat{a}_{01} & L_{00}U_{02} = \widehat{A}_{02} \\ l_{10}^T U_{00} = \widehat{a}_{10}^T & l_{10}^T u_{01} + \upsilon_{11} = \widehat{\alpha}_{11} & l_{10}^T U_{02} + u_{12}^T = \widehat{a}_{12}^T \\ \hline L_{20}U_{00} = \widehat{A}_{20} & L_{20}u_{01} + l_{21}\upsilon_{11} = \widehat{a}_{21} & \end{array}}
$$

for the loop invariant to again hold after the iteration. With this, we can compute the desired parts of L and U:

- Update $\alpha_{11} := \upsilon_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$.
- Update $a_{12}^T := u_{12}^T = a_{12}^T - l_{10}^T U_{02} = a_{12}^T - a_{10}^T A_{02}$.
- Update $a_{21} := l_{21} = (a_{21} - L_{20}u_{01})/\upsilon_{11} = (a_{21} - A_{20}a_{10})/\alpha_{11}$.

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

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

$\alpha_{11} := \upsilon_{11} = \alpha_{11} - a_{10}^T a_{01}$

$a_{12}^T := u_{12}^T = a_{12}^T - a_{10}^T A_{02}$

$a_{21} := l_{21} = (a_{21} - A_{20}a_{10})/\alpha_{11}$

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

endwhile

Figure 5.5.1.6 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 kth iteration, A_{00} is $k \times k$, for $k = 0, \ldots, n-1$. Then the (approximate) cost of each of the steps is given by

- Update $\alpha_{11} := \upsilon_{11} = \alpha_{11} - l_{10}^T u_{01} = \alpha_{11} - a_{10}^T a_{01}$. 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 $a_{21} := l_{21} = (a_{21} - L_{20}u_{01})/\upsilon_{11} = (a_{21} - A_{20}a_{10})/\alpha_{11}$. 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

$$\begin{array}{ll} & \sum_{k=0}^{n-1} 4k(n-k-1) \\ \approx & < \text{ remove lower order term } > \sum_{k=0}^{n-1} 4k(n-k) \\ = & < \text{ algebra } > \\ & 4n\sum_{k=0}^{n-1} k - 4\sum_{k=0}^{n-1} k^2 \\ \approx & < \sum_{k=0}^{n-1} k \approx n^2/2; \sum_{k=0}^{n-1} k^2 \approx k^3/3 > \\ & 2n^3 - 4\frac{n^3}{3} \\ = & < \text{ simplify } > \\ & \frac{2}{3}n^3. \end{array}$$

5.5.1.5 Variant 5: Classical Gaussian elimination

Consider final loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L\backslash U_{TL} & U_{TR} \\ \hline L_{BL} & \widehat{A}_{BR} - L_{BL}U_{TR} \end{array}\right) \wedge \begin{array}{c|c} L_{TL}U_{TL} = \widehat{A}_{TL} & L_{TL}U_{TR} = \widehat{A}_{TR} \\ \hline L_{BL}U_{TL} = \widehat{A}_{BL}. & \end{array}$$

At the top of the loop, after repartitioning, A then contains

$$\left(\begin{array}{c|cc} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|cc} L\backslash U_{00} & u_{01} & U_{02} \\ \hline l_{10}^T & \widehat{\alpha}_{11} - l_{10}^T u_{01} & \widehat{a}_{12}^T - l_{10}^T U_{02} \\ L_{20} & \widehat{a}_{21} - L_{20}u_{01} & \widehat{A}_{22} - L_{20}U_{02} \end{array}\right)$$

$$\wedge \left.\begin{array}{l} L_{00}U_{00} = \widehat{A}_{00} \\ l_{10}^T U_{00} = \widehat{a}_{10}^T \\ L_{20}U_{00} = \widehat{A}_{20} \end{array}\right| \left.\begin{array}{cc} L_{00}u_{01} = \widehat{a}_{01} & L_{00}U_{02} = \widehat{A}_{02} \end{array}\right|$$

while after updating A it must contain

$$\left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{cc|c} L\backslash U_{00} & u_{01} & U_{02} \\ l_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline L_{20} & l_{21} & \widehat{A}_{22} - l_{21}u_{12}^T, \end{array}\right)$$

$$\wedge \underbrace{\left.\begin{array}{l} \left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array}\right)\left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{array}\right) = \left(\begin{array}{cc} \widehat{A}_{00} & \widehat{a}_{01} \\ \widehat{a}_{10}^T & \widehat{\alpha}_{11} \end{array}\right) \\ \hline \left(\begin{array}{cc} L_{20} & l_{21} \end{array}\right)\left(\begin{array}{cc} U_{00} & u_{01} \\ 0 & \upsilon_{11} \end{array}\right) = \left(\begin{array}{cc} \widehat{A}_{20} & \widehat{a}_{21} \end{array}\right) \end{array}\right| \left.\begin{array}{l} \left(\begin{array}{cc} L_{00} & 0 \\ l_{10}^T & 1 \end{array}\right)\left(\begin{array}{c} U_{02} \\ u_{12}^T \end{array}\right) = \left(\begin{array}{c} \widehat{A}_{02} \\ \widehat{a}_{12}^T \end{array}\right) \\ \hline \end{array}\right.}_{\begin{array}{cc|c} L_{00}U_{00} = \widehat{A}_{00} & L_{00}u_{01} = \widehat{a}_{01} & L_{00}U_{02} = \widehat{A}_{02} \\ l_{10}^T U_{00} = \widehat{a}_{10}^T & l_{10}^T u_{01} + \upsilon_{11} = \widehat{\alpha}_{11} & l_{10}^T U_{02} + u_{12}^T = \widehat{a}_{12}^T \\ \hline L_{20}U_{00} = \widehat{A}_{20} & L_{20}u_{01} + l_{21}\upsilon_{11} = \widehat{a}_{21} & \end{array}}$$

for the loop invariant to again hold after the iteration. With this, we can compute the desired parts of L and U:

- $\alpha_{11} := \upsilon_{11} = \widehat{\alpha}_{11} - l_{10}^T u_{01} = \alpha_{11}$ (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 $a_{21} := (\widehat{a}_{21} - L_{20}u_{01})/\upsilon_{11} = a_{21}/\alpha_{11}$.
  (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 $n(A_{TL}) < n(A)$

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

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

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

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

endwhile

Figure 5.5.1.7 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 kth iteration, A_{00} is $k \times k$, for $k = 0, \ldots, n-1$. 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

$$\begin{array}{ll} & \sum_{k=0}^{n-1} 2(n-k-1)^2 \\ = & < \text{ change of variable } j = n-k-1 > \\ & 2\sum_{j=0}^{n-1} j^2 \\ \approx & < \sum_{k=0}^{n-1} k^2 \approx k^3/3 > \\ & 2\frac{n^3}{3} \end{array}$$

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 multple 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,

$$\underbrace{L \begin{pmatrix} x_0 & x_1 & \cdots \end{pmatrix}}_{\begin{pmatrix} Lx_0 & Lx_1 & \cdots \end{pmatrix}} = \begin{pmatrix} b_0 & b_1 & \cdots \end{pmatrix},$$

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 multple 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 $j = 0, \ldots, m-1$
   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

$$L = \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right).$$

1. Show that
$$L^{-1} = \left(\begin{array}{c|c} 1 & 0 \\ \hline -L_{22}^{-1} l_{21} & L_{22}^{-1} \end{array} \right).$$

2. Use the insight from the last part to complete the following algorithm for computing the inverse of a unit lower triangular matrix:

   $[L] = inv(L)$
   $L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right)$
   L_{TL} is 0×0
   while $n(L_{TL}) < n(L)$
   $\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right)$
   $l_{21} :=$
   $\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right)$
   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 trianguular 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 nonzeroes 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.*

$A =$ 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 $n(A_{TL}) < n(A)$

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

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

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

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

endwhile

Figure 5.6.2.5 Right-looking LU factorization algorithm.

The right-looking algorithm performs the same computations as the algorithm

for $j := 0, \ldots, n-1$
  for $i := j+1, \ldots, n-1$
    $\lambda_{i,j} := \alpha_{i,j}/\alpha_{j,j}$
    $\alpha_{i,j} := 0$
  endfor
} compute multipliers
  for $i := j+1, \ldots, n-1$
    for $k = j+1, \ldots, n-1$
      $\alpha_{i,k} := \alpha_{i,k} - \lambda_{i,j}\alpha_{j,k}$
    endfor
} subtract $\lambda_{i,j}$ times row j from row k
  endfor
endfor

$A =$ 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 $n(A_{TL}) < n(A)$

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

Solve $L_{00}u_{01} = a_{01}$ overwriting a_{01} with u_{01}

$\alpha_{11} := \upsilon_{11} = \alpha_{11} - a_{10}^T a_{01}$

$a_{21} := a_{21} - A_{20}a_{01}$

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

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

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 appropropriate size" is called a Gauss transform. ◊

$$L_k^{-1} = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & I \end{array}\right)^{-1} = \left(\begin{array}{c|cc} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array}\right).$$

Definition 5.6.2.8 Given

$$p = \begin{pmatrix} \pi_0 \\ \vdots \\ \pi_{n-1} \end{pmatrix},$$

where $\{\pi_0, \pi_1, \ldots, \pi_{n-1}\}$ is a permutation (rearrangement) of the integers $\{0, 1, \ldots, n-1\}$, we define the permutation matrix $P(p)$ by

$$P(p) = \begin{pmatrix} e_{\pi_0}^T \\ \vdots \\ e_{\pi_{n-1}}^T \end{pmatrix}.$$

◊

If P is a permutation matrix then $P^{-1} = P^T$.

Definition 5.6.2.9 Elementary pivot matrix. Given $\pi \in \{0, \ldots, n-1\}$ define the elementary pivot matrix

$$\widetilde{P}(\pi) = \left(\begin{array}{c} e_\pi^T \\ \hline e_1^T \\ \vdots \\ e_{\pi-1}^T \\ \hline e_0^T \\ \hline e_{\pi+1}^T \\ \vdots \\ e_{n-1}^T \end{array}\right)$$

or, equivalently,

$$\widetilde{P}(\pi) = \begin{cases} I_n & \text{if } \pi = 0 \\ \left(\begin{array}{c|c|c|c} 0 & 0 & 1 & 0 \\ \hline 0 & I_{\pi-1} & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & I_{n-\pi-1} \end{array}\right) & \text{otherwise,} \end{cases}$$

where n is the size of the permutation matrix. ◊

$[A, p] = \text{LUpiv-right-looking}(A)$

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

A_{TL} is $0 \times 0, p_T$ has 0 elements

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

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

$\pi_1 := \text{maxi}\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)$

$$\left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array}\right) \left(\begin{array}{cc|c} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

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

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

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

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 $n(L_{TL}) < n(L)$

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ \hline y_2 \end{array}\right)$$

$y_2 := y_2 - \psi_1 l_{21}$

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ \hline y_2 \end{array}\right)$$

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)

$L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$

L_{TL} is 0×0 and y_T has 0 elements

while $n(L_{TL}) < n(L)$

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array}\right)$$

$\psi_1 := \psi_1 - l_{10}^T y_0$

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array}\right)$$

endwhile

Figure 5.6.2.12 Lower triangular solve (with unit lower triangular matrix), Variant 2

Solve $Ux = z$, overwriting z with x (Variant 1)

$U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right), z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right)$

U_{BR} is 0×0 and z_B has 0 elements

while $n(U_{BR}) < n(U)$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ z_2 \end{array}\right)$$

$\zeta_1 := \zeta_1 / \upsilon_{11}$

$z_0 := z_0 - \zeta_1 u_{01}$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ z_2 \end{array}\right)$$

endwhile

Figure 5.6.2.13 Upper triangular solve Variant 1

Solve $Ux = z$, overwriting z with x (Variant 2)

$U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right), z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right)$

U_{BR} is 0×0 and z_B has 0 elements

while $n(U_{BR}) < n(U)$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ z_2 \end{array}\right)$$

$\zeta_1 := \zeta_1 - u_{12}^T z_2$

$\zeta_1 := \zeta_1 / \upsilon_{11}$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ z_2 \end{array}\right)$$

endwhile

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 $\widehat{\delta x}$ is an approximate solution to $A\delta x = b - A\widehat{x}$,
- Then $\widehat{x} + \widehat{\delta x}$, 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). ◊

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 = 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)

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

endwhile

Figure 5.6.2.17 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

$$\|b - A\widehat{x}\|_2 = \min_{x\in\mathbb{C}^n} \|b - Ax\|_2, \tag{5.6.1}$$

where A has linearly independent columns, equals the solution to the normal equations

$$\underbrace{A^H A}_{B}\ \widehat{x} = \underbrace{A^H b}_{y}.$$

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 be 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 scenerios.

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

6.1.3 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

- $\beta = 2$,
- $\mu = \pm.\delta_0\delta_1 \cdots \delta_{t-1}$ (μ has only t (binary) digits), where $\delta_j \in \{0, 1\}$),
- $\delta_0 = 0$ iff $\mu = 0$ (the mantissa is normalized), and
- $-L \leq e \leq U$.

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

$$\begin{array}{l} \mu \times \beta^e \\ \quad = \\ .101 \times 2^1 \\ \quad = \\ \left(1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}\right) \times 2^1 \\ \quad = \\ \left(\frac{1}{2} + \frac{0}{4} + \frac{1}{8}\right) \times 2 \\ \quad = \\ 1.25 \end{array}$$

□

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

$$\|x\|_2 = \sqrt{\sum_{i=0}^{n-1} \chi_i^2}. \tag{6.2.1}$$

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. □

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

$$\begin{array}{rcl} \delta\chi & = & \underbrace{.\delta_0 \delta_1 \cdots \delta_{t-1} \delta_t \cdots \times 2^e}_{\chi} - \underbrace{.\delta_0 \delta_1 \cdots \delta_{t-1} \times 2^e}_{\check{\chi}} \\ & = & \underbrace{.0 \cdots 00}_{t} \; \delta_t \cdots \times 2^e \\ & < & \underbrace{.0 \cdots 01}_{t} \; \times 2^e = 2^{-t} 2^e. \end{array}$$

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 $4/3 = 1.3333\cdots$ can be written as

$$
\begin{array}{l}
1.3333\cdots \\
\quad = \\
1 + \frac{0}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} + \cdots \\
\quad = \quad < \text{ convert to binary representation } > \\
1.0101\cdots \times 2^0 \\
\quad = \quad < \text{ normalize } > \\
.10101\cdots \times 2^1
\end{array}
$$

Now, if $t = 4$ then this would be truncated to

$$.1010 \times 2^1,$$

which equals the number

$$
\begin{array}{l}
.101 \times 2^1 \quad = \\
\frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{0}{16} \times 2^1 \\
\quad = \\
0.625 \times 2 \quad = \quad < \text{ convert to decimal } > \\
1.25
\end{array}
$$

The relative error equals

$$\frac{1.333\cdots - 1.25}{1.333\cdots} = 0.0625.$$

□

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

$$
\begin{array}{l}
1.3333\cdots \\
\quad = \\
1 + \frac{0}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} + \cdots \\
\quad = \quad < \text{ convert to binary representation } > \\
1.0101\cdots \times 2^0 \\
\quad = \quad < \text{ normalize } > \\
.10101\cdots \times 2^1
\end{array}
$$

Now, if $t = 4$ then this would be rounded to

$$.1011 \times 2^1,$$

which is equals the number

$$
\begin{array}{l}
.1011 \times 2^1 \quad = \\
\frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} \times 2^1 \\
\quad = \\
0.6875 \times 2 \quad = \qquad < \text{ convert to decimal } > \\
1.375
\end{array}
$$

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 roundoff), 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:

  $$\underbrace{.\underbrace{10\cdots0}_{t \text{ digits}}}_{1} \times 2^1.$$

- What is the ϵ_{mach} for this system?

  Answer:

  $$\underbrace{.\underbrace{10\cdots0}_{t \text{ digits}}}_{1} \times 2^1 \quad + \quad \underbrace{.\underbrace{00\cdots1}_{t \text{ digits}}}_{2^{-(t-1)}} \times 2^1 \quad = \quad \underbrace{.\underbrace{10\cdots1}_{t \text{ digits}}}_{>1} \times 2^1$$

  and

  $$\underbrace{.\underbrace{10\cdots0}_{t \text{ digits}}}_{1} \times 2^1 \quad + \quad \underbrace{.\underbrace{00\cdots0}_{t \text{ digits}} 11\cdots}_{< 2^{-(t-1)}} \times 2^1 \quad = \quad \underbrace{.\underbrace{10\cdots0}_{t \text{ digits}} 11\cdots}_{\text{truncates to } 1} \times 2^1$$

  Notice that

  $$.\underbrace{00\cdots1}_{t \text{ digits}} \times 2^1$$

  can be represented as

  $$.\underbrace{10\cdots0}_{t \text{ digits}} \times 2^{-(t-2)}$$

  and

  $$.\underbrace{00\cdots0}_{t \text{ digits}} 11\cdots \times 2^1$$

as

$$\underbrace{.11\cdots1}_{t \text{ digits}} \times 2^{-(t-1)}$$

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 $\text{fl}(\cdot)$.

6.2.3.2 Standard Computational Model (SCM)

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

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 $(\epsilon_+, \epsilon_*, \cdots)$ 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 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

$$\begin{array}{l} \check{\kappa} \\ \quad = \\ \text{fl}(4/3) \\ \quad = \\ 1.25 \\ \quad = \\ 1.333\cdots + (-0.0833\cdots) \\ \quad = \\ 1.333\cdots \times \left(1 + \frac{-0.08333\cdots}{1.333\cdots}\right) \\ \quad = \\ 4/3 \times (1 + (-0.0625)) \\ \quad = \\ \kappa(1 + \epsilon_/), \end{array}$$

where

$$|\epsilon_/| = 0.0625 \leq \underbrace{0.125}_{\epsilon_{\text{mach}} = 2^{-(t-1)}}.$$

□

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 $f : \mathcal{D} \to \mathcal{R}$ 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 $\check{f} : \mathcal{D} \to \mathcal{R}$ 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.

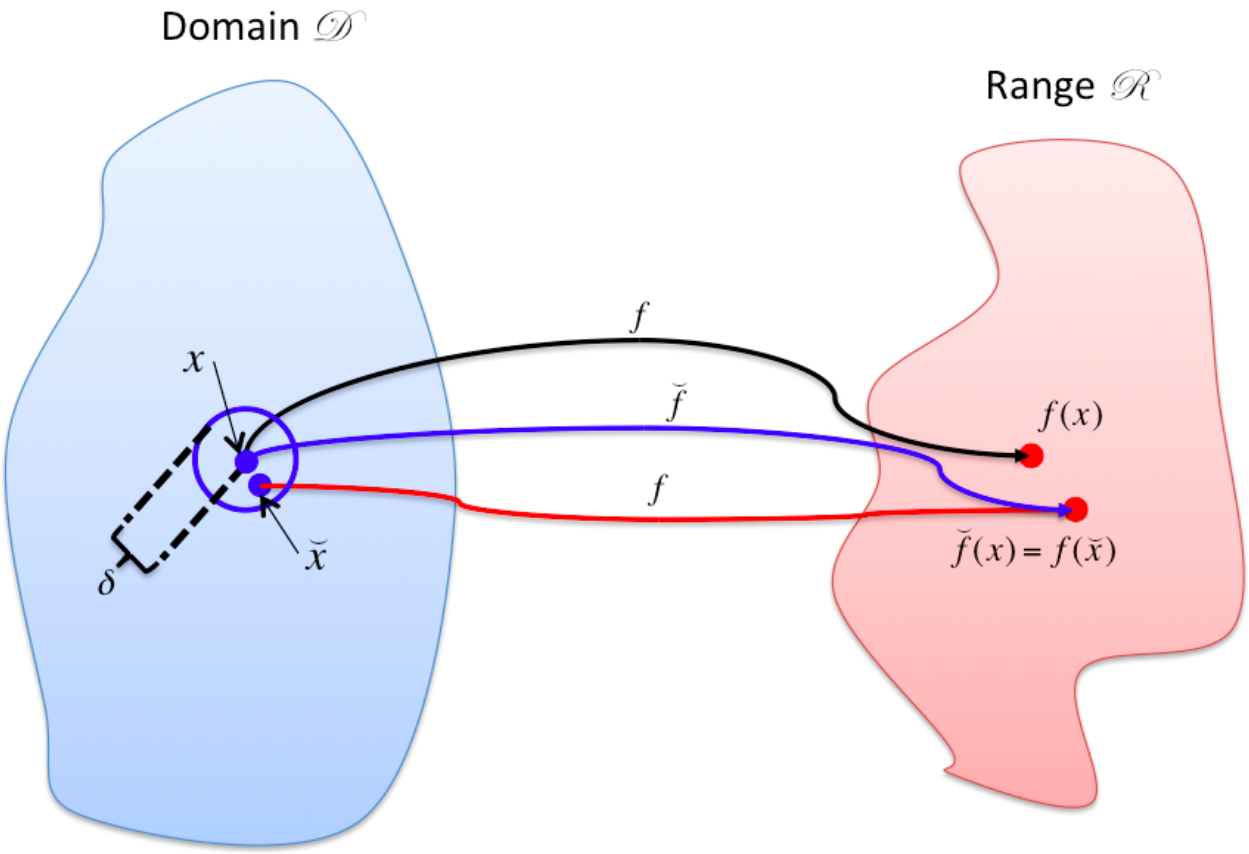


Figure 6.2.4.1 In this illustation, $f : \mathcal{D} \to \mathcal{R}$ is a function to be evaluated. The function $\check{f}$ represents the implementation of the function that uses floating point arithmetic, thus incurring errors. The fact that for a nearby value, $\check{x}$, the computed value equals 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 $f : D \to R$, where $D \subset \mathbb{R}^n$ is the domain and $R \subset \mathbb{R}^m$ is the range (codomain), let $\check{f} : D \to R$ 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 $\check{f}(x) = f(\check{x})$. ◊

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 implemention 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 that case that $\check{f}(x) \approx f(x)$. 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.

$$
\begin{array}{l}
\check{\kappa} \\
\quad = \quad < \text{ computed value for } \kappa > \\
[\chi + \psi] \\
\quad = \quad < \text{ SCM } > \\
(\chi + \psi)(1 + \epsilon_+) \\
\quad = \quad < \text{ distribute } > \\
\chi(1 + \epsilon_+) + \psi(1 + \epsilon_+) \\
\quad = \\
(\chi + \delta\chi) + (\psi + \delta\psi)
\end{array}
$$

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. □

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.

$$
\begin{array}{l}
\check{\kappa} \\
\quad = \quad < \text{ computed value for } \kappa > \\
[\chi - \psi] \\
\quad = \quad < \text{ SCM } > \\
(\chi - \psi)(1 + \epsilon_-) \\
\quad = \quad < \text{ distribute } > \\
\chi(1 + \epsilon_-) - \psi(1 + \epsilon_-) \\
\quad = \\
(\chi + \delta\chi) - (\psi + \delta\psi)
\end{array}
$$

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.

$$
\begin{array}{l}
\check{\kappa} \\
\quad = \quad < \text{ computed value for } \kappa > \\
[\chi \times \psi] \\
\quad = \quad < \text{ SCM } > \\
(\chi \times \psi)(1 + \epsilon_\times) \\
\quad = \quad < \text{ associative property } > \\
\chi \times \psi(1 + \epsilon_\times) \\
\quad = \\
\chi(\psi + \delta\psi)
\end{array}
$$

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.

$$
\begin{array}{l}
\check{\kappa} \\
\quad = \quad < \text{ computed value for } \kappa > \\
[\chi/\psi] < \\
\quad = \quad \text{SCM } > \\
(\chi/\psi)(1 + \epsilon_/) \\
\quad = \quad < \text{ commutative property } > \\
\chi(1 + \epsilon_/)/\psi \\
\quad = \\
(\chi + \delta\chi)/\psi
\end{array}
$$

where

- $|\epsilon_/| \leq \epsilon_{\text{mach}}$,
- $\delta\chi = \chi\epsilon_/$,

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 $A \in \mathbb{R}^{m \times n}$,

$$|x| = \begin{pmatrix} |\chi_0| \\ |\chi_1| \\ \vdots \\ |\chi_{n-1}| \end{pmatrix} \quad \text{and} \quad |A| = \begin{pmatrix} |\alpha_{0,0}| & |\alpha_{0,1}| & \cdots & |\alpha_{0,n-1}| \\ |\alpha_{1,0}| & |\alpha_{1,1}| & \cdots & |\alpha_{1,n-1}| \\ \vdots & \vdots & \ddots & \vdots \\ |\alpha_{m-1,0}| & |\alpha_{m-1,1}| & \cdots & |\alpha_{m-1,n-1}| \end{pmatrix}.$$

$\Diamond$

Definition 6.2.6.2 Let $\triangle \in \{<, \leq, =, \geq, >\}$ and $x, y \in \mathbb{R}^n$. Then

$$|x| \triangle |y| \quad \text{iff} \quad |\chi_i| \triangle |\psi_i|,$$

for all $i = 0, \ldots, n-1$. Similarly, given A and $B \in \mathbb{R}^{m \times n}$,

$$|A| \vartriangle |B| \quad \text{iff} \quad |\alpha_{ij}| \vartriangle |\beta_{ij}|,$$

for all $i = 0, \ldots, m-1$ and $j = 0, \ldots, n-1$. $\Diamond$

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 $\| \|_F$ 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

$$A = \left(\begin{array}{c|c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right) \quad \text{and} \quad B = \left(\begin{array}{c|c|c|c} b_0 & \cdots & b_{n-1} \end{array} \right).$$

  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 $>$
$\left(\sum_{i=0}^{m-1} |\alpha_{i,k}| \right)$
$\leq$ $<$ 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 $>$
$\|b_k\|_1$
$\leq$ $<$ 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

  $$\|A\|_\infty = \|A^T\|_1 \leq \|B^T\|_1 = \|B\|_\infty.$$

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

$$\check{\kappa} = \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) & 0 \\ 0 & (1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix}. \tag{6.3.1}$$

Solution.

$$
\begin{array}{l}
\check{\kappa} \\
\quad = \quad < \check{\kappa} = \left[x^T y \right] > \\
\left[\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right] \\
\quad = \quad < \text{ definition of } x^T y > \\
[\chi_0 \psi_0 + \chi_1 \psi_1] \\
\quad = \quad < \text{ each suboperation is performed in floating point arithmetic } > \\
[[\chi_0 \psi_0] + [\chi_1 \psi_1]] \\
\quad = \quad < \text{ apply SCM multiple times } > \\
\left[\chi_0 \psi_0 (1 + \epsilon_*^{(0)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)}) \right] \\
\quad = \quad < \text{ apply SCM } > \\
(\chi_0 \psi_0 (1 + \epsilon_*^{(0)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)}))(1 + \epsilon_+^{(1)}) \\
\quad = \quad < \text{ distribute } > \\
\chi_0 \psi_0 (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)}) \\
\quad = \quad < \text{ commute } > \\
\chi_0 (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)}) \psi_0 + \chi_1 (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)}) \psi_1 \\
\quad = \quad < \text{ (perhaps too) slick way of expressing the final result } > \\
\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1 + \epsilon_*^{(0)})(1 + \epsilon_+^{(1)}) & 0 \\ 0 & (1 + \epsilon_*^{(1)})(1 + \epsilon_+^{(1)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \\
,
\end{array}
$$

where $|\epsilon_*^{(0)}|, |\epsilon_*^{(1)}|, |\epsilon_+^{(1)}| \leq \epsilon_{\text{mach}}$. □

An important insight from this example is that the result in (6.3.1) can be manipulated to associate the accummulated 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.

$$
\begin{array}{l}
\check{\kappa} \\
\quad = \quad < \check{\kappa} = \left[x^T y \right] > \\
\left[(\chi_0\psi_0 + \chi_1\psi_1) + \chi_2\psi_2 \right] \\
\quad = \quad < \text{ each suboperation is performed in floating point arithmetic } > \\
\left[[\chi_0\psi_0 + \chi_1\psi_1] + [\chi_2\psi_2] \right] \\
\quad = \quad < \text{ reformulate so we can use result from Example 6.3.1.1 } > \\
\left[\left[\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right] + [\chi_2\psi_2] \right] \\
\quad = \quad < \text{ use Example 6.3.1.1; twice SCM } > \\
\left(\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) & 0 \\ 0 & (1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right. \\
\qquad \left. + \chi_2\psi_2(1+\epsilon_*^{(2)}) \right) (1+\epsilon_+^{(2)}) \\
\quad = \quad < \text{ distribute, commute } > \\
\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) & 0 \\ 0 & (1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix} (1+\epsilon_+^{(2)}) \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \\
\qquad + \chi_2(1+\epsilon_*^{(2)})(1+\epsilon_+^{(2)})\psi_2 \\
\quad = \quad < \text{ (perhaps too) slick way of expressing the final result } > \\
\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix}^T \begin{pmatrix} (1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)})(1+\epsilon_+^{(2)}) & 0 & 0 \\ 0 & (1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)})(1+\epsilon_+^{(2)}) & 0 \\ 0 & 0 & (1+\epsilon_*^{(2)})(1+\epsilon_+^{(2)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \end{pmatrix} \\
,
\end{array}
$$

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

$$
\kappa := x^T y = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-2} \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix} = \Big(\big((\chi_0\psi_0 + \chi_1\psi_1) + \cdots \big) + \chi_{n-2}\psi_{n-2} \Big) + \chi_{n-1}\psi_{n-1}.
$$

Under the computational model given in Subsection 6.2.3 the computed result, $\check{\kappa}$, satisfies

$$\begin{array}{rcl}
\check{\kappa} & = & \Big(\big((\chi_0\psi_0(1+\epsilon_*^{(0)}) + \chi_1\psi_1(1+\epsilon_*^{(1)}))(1+\epsilon_+^{(1)}) + \cdots \big)(1+\epsilon_+^{(n-2)}) \\
& & \quad + \chi_{n-1}\psi_{n-1}(1+\epsilon_*^{(n-1)}) \Big)(1+\epsilon_+^{(n-1)}) \\
& = & \chi_0\psi_0(1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)})(1+\epsilon_+^{(2)})\cdots(1+\epsilon_+^{(n-1)}) \\
& & + \chi_1\psi_1(1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)})(1+\epsilon_+^{(2)})\cdots(1+\epsilon_+^{(n-1)}) \\
& & + \chi_2\psi_2(1+\epsilon_*^{(2)}) \qquad (1+\epsilon_+^{(2)})\cdots(1+\epsilon_+^{(n-1)}) \\
& & + \cdots \\
& & + \chi_{n-1}\psi_{n-1}(1+\epsilon_*^{(n-1)}) \qquad (1+\epsilon_+^{(n-1)}) \\
& = & \sum_{i=0}^{n-1} \left(\chi_i\psi_i(1+\epsilon_*^{(i)}) \prod_{j=i}^{n-1}(1+\epsilon_+^{(j)}) \right)
\end{array}$$

so that

$$\check{\kappa} = \sum_{i=0}^{n-1} \left(\chi_i\psi_i(1+\epsilon_*^{(i)}) \prod_{j=i}^{n-1}(1+\epsilon_+^{(j)}) \right), \tag{6.3.2}$$

where $\epsilon_+^{(0)} = 0$ and $|\epsilon_*^{(0)}|, |\epsilon_*^{(j)}|, |\epsilon_+^{(j)}| \leq \epsilon_{\text{mach}}$ for $j = 1, \ldots, n-1$.

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

  $$\begin{array}{ll}
  & |\theta_{n+1}| \\
  = & \quad < \text{ definition of } \theta_{n+1} > \\
  & |(\theta_n-\epsilon_n)/(1+\epsilon_n)| \\
  \leq & \quad < |\theta_n-\epsilon_n| \leq |\theta_n|+|\epsilon_n| \leq |\theta_n|+\epsilon_{\text{mach}} > \\
  & (|\theta_n|+\epsilon_{\text{mach}})/(|1+\epsilon_n|) \\
  \leq & \quad < |1+\epsilon_n| \geq 1-|\epsilon_n| \geq 1-\epsilon_{\text{mach}} > \\
  & (|\theta_n|+\epsilon_{\text{mach}})/(1-\epsilon_{\text{mach}}) \\
  \leq & \quad < \text{ bound } |\theta_n| \text{ using I.H. } > \\
  & (\frac{n\epsilon_{\text{mach}}}{1-n\epsilon_{\text{mach}}}+\epsilon_{\text{mach}})/(1-\epsilon_{\text{mach}}) \\
  = & \quad < \text{ algebra } > \\
  & (n\epsilon_{\text{mach}}+(1-n\epsilon_{\text{mach}})\epsilon_{\text{mach}})/((1-n\epsilon_{\text{mach}})(1-\epsilon_{\text{mach}})) \\
  & \\
  = & \quad < \text{ algebra } > \\
  & ((n+1)\epsilon_{\text{mach}}-n\epsilon_{\text{mach}}^2)/(1-(n+1)\epsilon_{\text{mach}}+n\epsilon_{\text{mach}}^2) \\
  \leq & \quad < \text{ increase numerator; decrease denominator } > \\
  & ((n+1)\epsilon_{\text{mach}})/(1-(n+1)\epsilon_{\text{mach}}).
  \end{array}$$

- 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

$$
\begin{array}{l}
\check{\kappa} \\
\quad = \\
\chi_0\psi_0(1+\theta_n) + \chi_1\psi_1(1+\theta_n) + \cdots + \chi_{n-1}\psi_{n-1}(1+\theta_2) \\
\quad = \\
\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} (1+\theta_n) & 0 & 0 & \cdots & 0 \\ 0 & (1+\theta_n) & 0 & \cdots & 0 \\ 0 & 0 & (1+\theta_{n-1}) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (1+\theta_2) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix} \\
\quad = \\
\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \underbrace{\left(I + \begin{pmatrix} \theta_n & 0 & 0 & \cdots & 0 \\ 0 & \theta_n & 0 & \cdots & 0 \\ 0 & 0 & \theta_{n-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \theta_2 \end{pmatrix} \right)}_{I + \Sigma^{(n)}} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix} \\
= \\
x^T (I + \Sigma^{(n)}) y,
\end{array}
\tag{6.3.3}
$$

where $|\theta_j| \leq \gamma_j$, $j = 2, \ldots, n$.

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.

$$
\begin{array}{ll}
\gamma_n & \\
\quad = & < \text{ definition } > \\
(n\epsilon_{\text{mach}})/(1 - n\epsilon_{\text{mach}}) & \\
\quad \leq & < b \geq 1 > \\
((n + b)\epsilon_{\text{mach}})/(1 - n\epsilon_{\text{mach}}) & \\
\quad \leq & < 1/(1 - n\epsilon_{\text{mach}}) \leq 1/(1 - (n + b)\epsilon_{\text{mach}}) \text{ if } (n + b)\epsilon_{\text{mach}} < 1 > \\
((n + b)\epsilon_{\text{mach}})/(1 - (n + b)\epsilon_{\text{mach}}) & \\
\quad = & < \text{ definition } > \\
\gamma_{n+b}. &
\end{array}
$$

and

$$
\begin{array}{ll}
\gamma_n + \gamma_b + \gamma_n\gamma_b & \\
\quad = & < \text{ definition } > \\
\frac{n\epsilon_{\text{mach}}}{1-n\epsilon_{\text{mach}}} + \frac{b\epsilon_{\text{mach}}}{1-b\epsilon_{\text{mach}}} + \frac{n\epsilon_{\text{mach}}}{(1-n\epsilon_{\text{mach}})} \frac{b\epsilon_{\text{mach}}}{(1-b\epsilon_{\text{mach}})} & \\
\quad = & < \text{ algebra } > \\
\frac{n\epsilon_{\text{mach}}(1-b\epsilon_{\text{mach}})+(1-n\epsilon_{\text{mach}})b\epsilon_{\text{mach}}+bn\epsilon_{\text{mach}}^2}{(1-n\epsilon_{\text{mach}})(1-b\epsilon_{\text{mach}})} & \\
\quad = & < \text{ algebra } > \\
\frac{n\epsilon_{\text{mach}}-bn\epsilon_{\text{mach}}^2+b\epsilon_{\text{mach}}-bn\epsilon_{\text{mach}}^2+bn\epsilon_{\text{mach}}^2}{1-(n+b)\epsilon_{\text{mach}}+bn\epsilon_{\text{mach}}^2} & \\
\quad = & < \text{ algebra } > \\
\frac{(n+b)\epsilon_{\text{mach}}-bn\epsilon_{\text{mach}}^2}{1-(n+b)\epsilon_{\text{mach}}+bn\epsilon_{\text{mach}}^2} & \\
\quad \leq & < bn\epsilon_{\text{mach}}^2 > 0 > \\
\frac{(n+b)\epsilon_{\text{mach}}}{1-(n+b)\epsilon_{\text{mach}}+bn\epsilon_{\text{mach}}^2} & \\
\quad \leq & < bn\epsilon_{\text{mach}}^2 > 0 > \\
\frac{(n+b)\epsilon_{\text{mach}}}{1-(n+b)\epsilon_{\text{mach}}} & \\
\quad = & < \text{ definition } > \\
\gamma_{n+b}. &
\end{array}
$$

6.3.3 Dot product: error results

YouTube: 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 $\Sigma = \text{diag}(\sigma_0, \ldots, \sigma_{n-1})$. The following special structure of Σ, inspired by (6.3.3) will be used in the remainder of this note:

$$\Sigma^{(n)} = \begin{cases} 0 \times 0 \text{ matrix} & \text{if } n = 0 \\ \theta_1 & \text{if } n = 1 \\ \text{diag}(\theta_n, \theta_n, \theta_{n-1}, \ldots, \theta_2) & \text{otherwise .} \end{cases}$$

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

$$\begin{pmatrix} I + \Sigma^{(k)} & 0 \\ 0 & (1 + \epsilon_1) \end{pmatrix} (1 + \epsilon_2)$$
$$= \quad < k = 0 \text{ means } (I + \Sigma^{(k)}) \text{ is } 0 \times 0 \text{ and } (1 + \epsilon_1) = (1 + 0) >$$
$$(1 + 0)(1 + \epsilon_2)$$
$$=$$
$$(1 + \epsilon_2)$$
$$=$$
$$(1 + \theta_1)$$
$$=$$
$$(I + \Sigma^{(1)}).$$

Case: $k = 1$.

Then

$$\begin{pmatrix} I + \Sigma^{(k)} & 0 \\ 0 & (1 + \epsilon_1) \end{pmatrix} (1 + \epsilon_2)$$
$$=$$
$$\begin{pmatrix} 1 + \theta_1 & 0 \\ 0 & (1 + \epsilon_1) \end{pmatrix} (1 + \epsilon_2)$$
$$=$$
$$\begin{pmatrix} (1 + \theta_1)(1 + \epsilon_2) & 0 \\ 0 & (1 + \epsilon_1)(1 + \epsilon_2) \end{pmatrix}$$
$$=$$
$$\begin{pmatrix} (1 + \theta_2) & 0 \\ 0 & (1 + \theta_2) \end{pmatrix}$$
$$=$$
$$(I + \Sigma^{(2)}).$$

Case: $k > 1$.

Notice that

$$
\begin{array}{l}
(I + \Sigma^{(k)})(1 + \epsilon_2) \\
\quad = \\
\begin{pmatrix}
1 + \theta_k & 0 & 0 & \cdots & 0 \\
0 & 1 + \theta_k & 0 & \cdots & 0 \\
0 & 0 & 1 + \theta_{k-1} & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1 + \theta_2
\end{pmatrix} (1 + \epsilon_2) \\
\quad = \\
\begin{pmatrix}
1 + \theta_{k+1} & 0 & 0 & \cdots & 0 \\
0 & 1 + \theta_{k+1} & 0 & \cdots & 0 \\
0 & 0 & 1 + \theta_k & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1 + \theta_3
\end{pmatrix}
\end{array}
$$

Then

$$
\begin{array}{l}
\begin{pmatrix} I + \Sigma^{(k)} & 0 \\ 0 & (1 + \epsilon_1) \end{pmatrix} (1 + \epsilon_2) \\
\quad = \\
\begin{pmatrix} (I + \Sigma^{(k)})(1 + \epsilon_2) & 0 \\ 0 & (1 + \epsilon_1)(1 + \epsilon_2) \end{pmatrix} \\
\quad = \\
\begin{pmatrix}
\begin{pmatrix}
1 + \theta_{k+1} & 0 & 0 & \cdots & 0 \\
0 & 1 + \theta_{k+1} & 0 & \cdots & 0 \\
0 & 0 & 1 + \theta_k & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1 + \theta_3
\end{pmatrix} & 0 \\
0 & (1 + \theta_2)
\end{pmatrix} \\
\quad = \\
(I + \Sigma^{(k+1)}).
\end{array}
$$

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

$$\check{\kappa} = \left[x^T y\right] = x^T(I + \Sigma^{(n)})y.$$

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

  $$\text{fl}(\text{x}_\text{T}^\text{T}\text{y}_\text{T}) = x_T^T(I + \Sigma_T)y_T, \text{ where } \Sigma_T = \Sigma^{(k)}.$$

  We will show that when $x_T, y_T \in \mathbb{R}^{k+1}$, the equality $\text{fl}(\text{x}_\text{T}^\text{T}\text{y}_\text{T}) = x_T^T(I + \Sigma_T)y_T$ 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

$$
\begin{array}{ll}
& \mathrm{fl}(\begin{pmatrix} x_0 \\ \chi_1 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}) \\
= & \quad < \text{definition} > \\
& \mathrm{fl}((\mathrm{fl}(\mathrm{x}_0^{\mathrm{T}}\mathrm{y}_0) + \mathrm{fl}(\chi_1\psi_1)) \\
= & \quad < \text{I.H. with } x_T = x_0, y_T = y_0, \text{ and } \Sigma_0 = \Sigma^{(k)} > \\
& \mathrm{fl}(\mathrm{x}_0^{\mathrm{T}}(\mathrm{I} + \Sigma_0)\mathrm{y}_0 + \mathrm{fl}(\chi_1\psi_1)) \\
= & \quad < \text{SCM, twice} > \\
& \left(x_0^T(I + \Sigma_0)y_0 + \chi_1\psi_1(1 + \epsilon_*)\right)(1 + \epsilon_+) \\
= & \quad < \text{rearrangement} > \\
& \begin{pmatrix} x_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (I + \Sigma_0) & 0 \\ 0 & (1 + \epsilon_*) \end{pmatrix} (1 + \epsilon_+) \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \\
= & \quad < \text{renaming} > \\
& x_T^T(I + \Sigma_T)y_T
\end{array}
$$

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

$$
\begin{array}{rcl}
|\delta\kappa| & = & |x^T\Sigma^{(n)}y| \\
& \leq & |\chi_0||\theta_n||\psi_0| + |\chi_1||\theta_n||\psi_1| + \cdots + |\chi_{n-1}||\theta_2||\psi_{n-1}| \\
& \leq & \gamma_n|\chi_0||\psi_0| + \gamma_n|\chi_1||\psi_1| + \cdots + \gamma_2|\chi_{n-1}||\psi_{n-1}| \\
& \leq & \gamma_n|x|^T|y|.
\end{array}
$$

■

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

$$\begin{array}{rcl} |\delta x| & = & |\Sigma^{(n)}x| = \left| \begin{pmatrix} \theta_n\chi_0 \\ \theta_n\chi_1 \\ \theta_{n-1}\chi_2 \\ \vdots \\ \theta_2\chi_{n-1} \end{pmatrix} \right| = \begin{pmatrix} |\theta_n\chi_0| \\ |\theta_n\chi_1| \\ |\theta_{n-1}\chi_2| \\ \vdots \\ |\theta_2\chi_{n-1}| \end{pmatrix} = \begin{pmatrix} |\theta_n||\chi_0| \\ |\theta_n||\chi_1| \\ |\theta_{n-1}||\chi_2| \\ \vdots \\ |\theta_2||\chi_{n-1}| \end{pmatrix} \\ & \leq & \begin{pmatrix} |\theta_n||\chi_0| \\ |\theta_n||\chi_1| \\ |\theta_n||\chi_2| \\ \vdots \\ |\theta_n||\chi_{n-1}| \end{pmatrix} \leq \begin{pmatrix} \gamma_n|\chi_0| \\ \gamma_n|\chi_1| \\ \gamma_n|\chi_2| \\ \vdots \\ \gamma_n|\chi_{n-1}| \end{pmatrix} = \gamma_n|x|. \end{array}$$

(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=q7rACPOu4ZQ

Assume $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$. Partition

$$A = \begin{pmatrix} \widetilde{a}_0^T \\ \widetilde{a}_1^T \\ \vdots \\ \widetilde{a}_{m-1}^T \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix}.$$

Then computing $y := Ax$ can be orchestrated as

$$\begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} := \begin{pmatrix} \widetilde{a}_0^T x \\ \widetilde{a}_1^T x \\ \vdots \\ \widetilde{a}_{m-1}^T x \end{pmatrix}. \tag{6.3.4}$$

From R-1B 6.3.3.2 regarding the dot product we know that

$$\check{y} = \begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} (\widetilde{a}_0 + \delta\widetilde{a}_0)^T x \\ (\widetilde{a}_1 + \delta\widetilde{a}_1)^T x \\ \vdots \\ (\widetilde{a}_{m-1} + \delta\widetilde{a}_{m-1})^T x \end{pmatrix}$$
$$= \left(\begin{pmatrix} \widetilde{a}_0^T \\ \widetilde{a}_1^T \\ \vdots \\ \widetilde{a}_{m-1}^T \end{pmatrix} + \begin{pmatrix} \delta\widetilde{a}_0^T \\ \delta\widetilde{a}_1^T \\ \vdots \\ \delta\widetilde{a}_{m-1}^T \end{pmatrix} \right) x = (A + \Delta A)x,$$

where $|\delta\widetilde{a}_i| \leq \gamma_n |\widetilde{a}_i|$, $i = 0, \ldots, m-1$, and hence $|\Delta A| \leq \gamma_n |A|$. .

Also, from Corollary 6.3.3.2 R-1F regarding the dot product we know that

$$\check{y} = \begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} \widetilde{a}_0^T x + \delta\psi_0 \\ \widetilde{a}_1^T x + \delta\psi_1 \\ \vdots \\ \widetilde{a}_{m-1}^T x + \delta\psi_{m-1} \end{pmatrix} = \begin{pmatrix} \widetilde{a}_0^T \\ \widetilde{a}_1^T \\ \vdots \\ \widetilde{a}_{m-1}^T \end{pmatrix} x + \begin{pmatrix} \delta\psi_0 \\ \delta\psi_1 \\ \vdots \\ \delta\psi_{m-1} \end{pmatrix} = Ax + \delta y.$$

where $|\delta\psi_i| \leq \gamma_n |\widetilde{a}_i|^T |x|$ 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

$$\check{\psi}_i = \widetilde{a}_i^T (x + \delta x)$$

which would appear to support that

$$\begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} \widetilde{a}_0^T (x + \delta x) \\ \widetilde{a}_1^T (x + \delta x) \\ \vdots \\ \widetilde{a}_{m-1}^T (x + \delta x) \end{pmatrix}.$$

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*

$\check{C} = AB + \Delta C$, *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

$$\begin{array}{rcl} \left(\begin{array}{c|c|c|c} \check{c}_0 & \check{c}_1 & \cdots & \check{c}_{n-1} \end{array} \right) & = & \left(\begin{array}{c|c|c|c} Ab_0 + \delta c_0 & Ab_1 + \delta c_1 & \cdots & Ab_{n-1} + \delta c_{n-1} \end{array} \right) \\ & = & \left(\begin{array}{c|c|c|c} Ab_0 & Ab_1 & \cdots & Ab_{n-1} \end{array} \right) + \left(\begin{array}{c|c|c|c} \delta c_0 & \delta c_1 & \cdots & \delta c_{n-1} \end{array} \right) \\ & = & AB + \Delta C. \end{array}$$

where $|\delta c_j| \leq \gamma_k |A||b_j|$, $j = 0, \ldots, n-1$, 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 $Lx = y$

$L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right), x \rightarrow \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$

L_{TL} is 0×0 and y_T, x_T have 0 elements

while $n(L_{TL}) < n(L)$

$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \rightarrow \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ x_2 \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array}\right)$

$\chi_1 := (\psi_1 - l_{10}^T x_0)/\lambda_{11}$

$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \leftarrow \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ x_2 \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array}\right)$

endwhile

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

$$\check{\nu} = \frac{\alpha - \beta}{\lambda} \frac{1}{(1 + \epsilon_-)(1 + \epsilon_/)},$$

  where $|\epsilon_-| \leq \epsilon_{\text{mach}}$ and $|\epsilon_/| \leq \epsilon_{\text{mach}}$.

Hence

$$\check{\nu} = \frac{\alpha - (x + \delta x)^T y}{\lambda} \frac{1}{(1 + \epsilon_-)(1 + \epsilon_/)},$$

or, equivalently,

$$\lambda(1 + \epsilon_-)(1 + \epsilon_/)\check{\nu} = \alpha - (x + \delta x)^T y,$$

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 errror 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

$$\check{\chi}_1 = \psi_1/\lambda_{11} \frac{1}{1 + \epsilon_/}$$

Rearranging gives us

$$\lambda_{11}\check{\chi}_1(1+\epsilon_/) = \psi_1$$

or

$$(\lambda_{11}+\delta\lambda_{11})\check{\chi}_1 = \psi_1$$

where $\delta\lambda_{11} = \epsilon_/\lambda_{11}$ and hence

$$\begin{array}{rcl} |\delta\lambda_{11}| & = & |\epsilon_/||\lambda_{11}| \\ & \leq & \gamma_1|\lambda_{11}| \\ & \leq & \gamma_2|\lambda_{11}| \\ & \leq & \max(\gamma_2,\gamma_{n-1})|\lambda_{11}|. \end{array}$$

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

$$\left(\begin{array}{c|c} \lambda_{00} & 0 \\ \hline \lambda_{10} & \lambda_{11} \end{array}\right)\left(\begin{array}{c} \chi_0 \\ \chi_1 \end{array}\right) = \left(\begin{array}{c} \psi_0 \\ \psi_1 \end{array}\right).$$

From the proof of Case 1 we know that

$$(\lambda_{00}+\delta\lambda_{00})\check{\chi}_0 = \psi_0, \text{ where } |\delta\lambda_{00}| \leq \gamma_1|\lambda_{00}|. \tag{6.4.1}$$

Since $\chi_1 = (\psi_1 - \lambda_{10}\check{\chi}_0)/\lambda_{11}$, Lemma 6.4.1.2 tells us that

$$(\lambda_{10}+\delta\lambda_{10})\check{\chi}_0 + (\lambda_{11}+\delta\lambda_{11})\check{\chi}_1 = \psi_1, \tag{6.4.2}$$

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

$$\left(\begin{array}{c|c} \lambda_{00}+\delta\lambda_{00} & 0 \\ \hline \lambda_{10}+\delta\lambda_{10} & \lambda_{11}+\delta\lambda_{11} \end{array}\right)\left(\begin{array}{c} \check{\chi}_0 \\ \check{\chi}_1 \end{array}\right) = \left(\begin{array}{c} \psi_0 \\ \psi_1 \end{array}\right),$$

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
  $$\left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array}\right)\left(\begin{array}{c} x_0 \\ \chi_1 \end{array}\right) = \left(\begin{array}{c} y_0 \\ \psi_1 \end{array}\right), \tag{6.4.3}$$
  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

$$(l_{10} + \delta l_{10})^T \check{x}_0 + (\lambda_{11} + \delta\lambda_{11})\check{\chi}_1 = \psi_1, \tag{6.4.4}$$

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

$$\left(\begin{array}{c|c} L_{00} + \delta L_{00} & 0 \\ \hline (l_{10} + \delta l_{10})^T & \lambda_{11} + \delta\lambda_{11} \end{array}\right) \left(\begin{array}{c} \check{x}_0 \\ \hline \check{\chi}_1 \end{array}\right) = \left(\begin{array}{c} y_0 \\ \hline \psi_1 \end{array}\right),$$

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*

$$(A + \Delta A) = \check{L}\check{U} \quad \textit{with} \quad |\Delta A| \leq \gamma_n |\check{L}||\check{U}|.$$

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 $\check{U}x = \check{z}$, yielding approximate solution $\check{x}$.*

Then

$$(A + \Delta A)\check{x} = y \quad \textit{with} \quad |\Delta A| \leq (3\gamma_n + \gamma_n^2)|\check{L}||\check{U}|.$$

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,

$$\Delta A x + A\delta x + \Delta A \delta x = 0.$$

We can rewrite this as

$$\delta x = A^{-1}(-\Delta A x - \Delta A \delta x)$$

so that

$$\|\delta x\| = \|A^{-1}(-\Delta A x - \Delta A \delta x)\| \leq \|A^{-1}\|\|\Delta A\|\|x\| + \|A^{-1}\|\|\Delta A\|\|\delta x\|.$$

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 contradition.

Since $A + \Delta A$ is singular, there exists $x \neq 0$ such that $(A + \Delta A)x = 0$. We can rewrite this as

$$x = -A^{-1}\Delta A x$$

and hence

$$\|x\| = \|A^{-1}\Delta A x\| \leq \|A^{-1}\|\|\Delta A\|\|x\|.$$

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 extention, 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

- [47] 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.

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 roundoff), 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. ◊

$$\text{fl(expression)} = [\text{expression}]$$

equals the result when computing {\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

$$\text{fl}(\chi \text{ op } \psi) = (\chi \text{ op } \psi)(1+\epsilon), |\epsilon| \leq \epsilon_{\text{mach}}, \text{ and } \text{ op } \in \{+,-,*,/\}.$$

The Alternative Computational Model (ACM) assumes for the basic arithmetic operations that

$$\text{fl}(\chi \text{ op } \psi) = \frac{\chi \text{ op } \psi}{1+\epsilon}, |\epsilon| \leq \epsilon_{\text{mach}}, \text{ and } \text{ op } \in \{+,-,*,/\}.$$

Definition 6.6.2.2 Backward stable implementation. Given the mapping $f : D \to R$, where $D \subset \mathbb{R}^n$ is the domain and $R \subset \mathbb{R}^m$ is the range (codomain), let $\check{f} : D \to R$ 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 $\check{f}(x) = f(\check{x})$. ◊

- *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 $A \in \mathbb{R}^{m\times n}$,

$$|x| = \begin{pmatrix} |\chi_0| \\ |\chi_1| \\ \vdots \\ |\chi_{n-1}| \end{pmatrix} \quad \text{and} \quad |A| = \begin{pmatrix} |\alpha_{0,0}| & |\alpha_{0,1}| & \cdots & |\alpha_{0,n-1}| \\ |\alpha_{1,0}| & |\alpha_{1,1}| & \cdots & |\alpha_{1,n-1}| \\ \vdots & \vdots & \ddots & \vdots \\ |\alpha_{m-1,0}| & |\alpha_{m-1,1}| & \cdots & |\alpha_{m-1,n-1}| \end{pmatrix}.$$

◊

Definition 6.6.2.4 Let $\triangle \in \{<, \leq, =, \geq, >\}$ and $x, y \in \mathbb{R}^n$. Then

$$|x| \triangle |y| \quad \text{iff} \quad |\chi_i| \triangle |\psi_i|,$$

with $i = 0, \ldots, n-1$. Similarly, given A and $B \in \mathbb{R}^{m\times n}$,

$$|A| \triangle |B| \quad \text{iff} \quad |\alpha_{ij}| \triangle |\beta_{ij}|,$$

with $i = 0, \ldots, m-1$ and $j = 0, \ldots, n-1$. ◊

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

$$\kappa := x^T y = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-2} \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix} = \Big(((\chi_0\psi_0 + \chi_1\psi_1) + \cdots) + \chi_{n-2}\psi_{n-2} \Big) + \chi_{n-1}\psi_{n-1}.$$

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 $j = 1, \ldots, n-1$.

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

$$\gamma_n = n\epsilon_{\text{mach}}/(1 - n\epsilon_{\text{mach}}).$$

$\Diamond$

simplifies to

$$\begin{array}{l}
\check{\kappa} \\
\quad = \\
\chi_0\psi_0(1+\theta_n) + \chi_1\psi_1(1+\theta_n) + \cdots + \chi_{n-1}\psi_{n-1}(1+\theta_2) \\
\quad = \\
\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} (1+\theta_n) & 0 & 0 & \cdots & 0 \\ 0 & (1+\theta_n) & 0 & \cdots & 0 \\ 0 & 0 & (1+\theta_{n-1}) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (1+\theta_2) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix} \\
\quad = \\
\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \left(I + \begin{pmatrix} \theta_n & 0 & 0 & \cdots & 0 \\ 0 & \theta_n & 0 & \cdots & 0 \\ 0 & 0 & \theta_{n-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \theta_2 \end{pmatrix} \right) \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix},
\end{array}$$

where $|\theta_j| \leq \gamma_j$, $j = 2, \ldots, n$.

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

$$\check{\kappa} = \left[x^T y\right] = x^T(I + \Sigma^{(n)})y.$$

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* $y := Ax$ *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*

$\check{C} = AB + \Delta C$, *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 factoriztion. *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* $\check{U}x = \check{z}$, *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

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y) \tag{7.1.1}$$

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 $i = 0, 1, \ldots, N-1$ and $\psi_j = (j+1)h$ for $j = 0, 1, \ldots, N-1$.

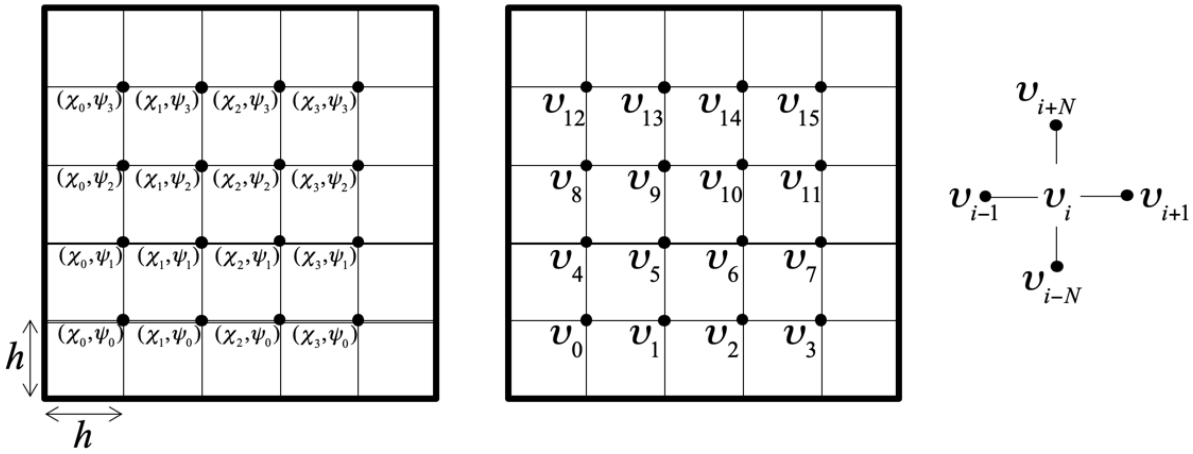


Figure 7.1.1.1 2D mesh.

- 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:

$$\frac{-u(\chi_i, \psi_{j-1}) - u(\chi_{i-1}, \psi_j) + 4u(\chi_i, \psi_j) - u(x_{i+1}, u_j) - u(x_i, \psi_{j+1})}{h^2} = f(\chi_i, \psi_i).$$

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=VYMbSJAqaUM

All these insights can be put together into a system of linear equations:

$$
\begin{array}{ccccccccccc}
4\upsilon_0 & -\upsilon_1 & & & -\upsilon_4 & & & & & & = h^2\phi_0 \\
-\upsilon_0 & +4\upsilon_1 & -\upsilon_2 & & & -\upsilon_5 & & & & & = h^2\phi_1 \\
 & -\upsilon_1 & +4\upsilon_2 & -\upsilon_3 & & & -\upsilon_6 & & & & = h^2\phi_2 \\
 & & -\upsilon_2 & +4\upsilon_3 & & & & -\upsilon_7 & & & = h^2\phi_3 \\
-\upsilon_0 & & & & +4\upsilon_4 & -\upsilon_5 & & & -\upsilon_8 & & = h^2\phi_4 \\
 & \ddots & & & \ddots & \ddots & \ddots & & & \ddots & \vdots
\end{array}
$$

where $\phi_i = f(\chi_i, \psi_j)$ if (χ_i, ψ_j) is the point associated with value υ_i. In matrix notation this becomes

$$
\left(\begin{array}{cccc|cccc|cc}
4 & -1 & & & -1 & & & & & \\
-1 & 4 & -1 & & & -1 & & & & \\
 & -1 & 4 & -1 & & & -1 & & & \\
 & & -1 & 4 & & & & -1 & & \\
\hline
-1 & & & & 4 & -1 & & & -1 & \\
 & -1 & & & -1 & 4 & -1 & & & \ddots \\
 & & -1 & & & -1 & 4 & -1 & & \\
 & & & -1 & & & -1 & 4 & & \\
\hline
 & & & & -1 & & & & 4 & \ddots \\
 & & & & & \ddots & & & \ddots & \ddots
\end{array}\right)
\left(\begin{array}{c}
\upsilon_0 \\ \upsilon_1 \\ \upsilon_2 \\ \upsilon_3 \\ \hline \upsilon_4 \\ \upsilon_5 \\ \upsilon_6 \\ \upsilon_7 \\ \hline \upsilon_8 \\ \vdots
\end{array}\right)
=
\left(\begin{array}{c}
h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots
\end{array}\right).
$$

This demonstrates how solving the discretized Poisson's equation boils down to the solution of a linear system $Au = h^2 f$, where A has a distinct sparsity pattern (pattern of nonzeroes).

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

$$
\begin{array}{l}
U(i,j) = \\
(h^2 \underbrace{F(i,j)}_{f(\chi_i,\psi_j)} + \underbrace{U(i,j-1)}_{u(\chi_i,\psi_{j-1})} + \underbrace{U(i-1,j)}_{u(\chi_{i-1},\psi_j)} + \underbrace{U(i+1,j)}_{u(\chi_{i+1},\psi_j)} + \underbrace{U(i,j+1)}_{u(\chi_i,\psi_{j+1})})/4
\end{array}
$$

  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 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

7.1.3 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

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}. \tag{7.2.1}$$

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

$$\left(\begin{array}{c|ccccc|c} 1 & & & 0 & & & 0 \\ \hline -1 & 2 & -1 & & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & & & -1 & 2 & -1 & 0 \\ 0 & & & & -1 & 2 & -1 \\ \hline 0 & & & 0 & & & 1 \end{array}\right) \left(\begin{array}{c} \chi_{-1} \\ \hline \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-2} \\ \chi_{n-1} \\ \hline \chi_n \end{array}\right) = \left(\begin{array}{c} 0 \\ \hline 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \hline 0 \end{array}\right)$$

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 $\chi_{-1}, \ldots, \chi_{i-1}$ 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

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{1,0} & & & & \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{2,1} & & & \\ & \ddots & \ddots & \ddots & & \\ & & \alpha_{n-2,n-3} & \alpha_{n-2,n-2} & \alpha_{n-1,n-2} \\ & & & \alpha_{n-1,n-2} & \alpha_{n-1,n-1} \end{pmatrix}. \tag{7.2.2}$$

- 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

$$\begin{pmatrix} \alpha_{0,0} & \alpha_{1,0} & & & & \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{2,1} & & & \\ & \ddots & \ddots & \ddots & & \\ & & \alpha_{n-2,n-3} & \alpha_{n-2,n-2} & \alpha_{n-1,n-2} \\ & & & \alpha_{n-1,n-2} & \alpha_{n-1,n-1} \end{pmatrix}$$

$$= \begin{pmatrix} \lambda_{0,0} & & & & \\ \lambda_{1,0} & \lambda_{1,1} & & & \\ \ddots & \ddots & & & \\ & \lambda_{n-2,n-3} & \lambda_{n-2,n-2} & & \\ & & \lambda_{n-1,n-2} & \lambda_{n-1,n-1} \end{pmatrix} \begin{pmatrix} \lambda_{0,0} & \lambda_{1,0} & & & & \\ & \lambda_{1,1} & \lambda_{2,1} & & & \\ & & \ddots & \ddots & & \\ & & & \lambda_{n-2,n-2} & \lambda_{n-1,n-2} \\ & & & & \lambda_{n-1,n-1} \end{pmatrix}$$

$$= \begin{pmatrix} \lambda_{0,0}\lambda_{0,0} & \lambda_{0,0}\lambda_{1,0} & & & & \\ \lambda_{1,0}\lambda_{0,0} & \lambda_{1,0}\lambda_{0,1} + \lambda_{1,1}\lambda_{1,1} & \lambda_{1,1}\lambda_{2,1} & & & \\ & \lambda_{21}\lambda_{11} & \ddots & \ddots & & \\ & & \ddots & \star\star & \lambda_{n-2,n-2}\lambda_{n-1,n-2} & \\ & & & \lambda_{n-1,n-2}\lambda_{n-2,n-2} & \lambda_{n-1,n-2}\lambda_{n-1,n-2} & \\ & & & & + \lambda_{n-1,n-1}\lambda_{n-1,n-1} \end{pmatrix},$$

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 $i = 0, \ldots, n-2$
  $\alpha_{i,i} := \sqrt{\alpha_{i,i}}$
  $\alpha_{i+1,i} := \alpha_{i+1,i} / \alpha_{i,i}$
  $\alpha_{i+1,i+1} := \alpha_{i+1,i+1} - \alpha_{i+1,i}\alpha_{i+1,i}$
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.

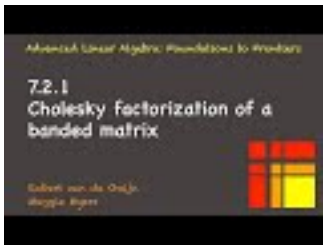
- Use the algorithm from Homework 7.2.1.3 to overwrite A with its Cholesky factor.
- Since $A = LL^T$, we need to solve $Lz = y$ and then $L^T x = z$.
  - Overwriting y with the solution of $Lz = y$ (forward substitution) is accomplished by the following algorithm (here L had overwritten A):

    for $i = 0, \ldots, n-2$
      $\psi_i := \psi_i / \alpha_{i,i}$
      $\psi_{i+1} := \psi_{i+1} - \alpha_{i+1,i}\psi_i$
    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 $n-1 = 0, \ldots, 1$
      $\psi_i := \psi_i / \alpha_{i,i}$
      $\psi_{i-1} := \psi_{i-1} - \alpha_{i,i-1}\psi_i$
    endfor
    $\psi_0 := \psi_0 / \alpha_{0,0}$

The last exercises illustrate how special structure (in terms of patterns of zeroes and nonzeroes) 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 jth super-diagonal and subdiagonal of the matrix equal zero if $j > b$.

- A diagonal matrix has bandwidth 1.
- A tridiagonal matrix has bandwith 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 subdiagonals 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

$A =$ 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)

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

endwhile

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 nonzeroes. 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).

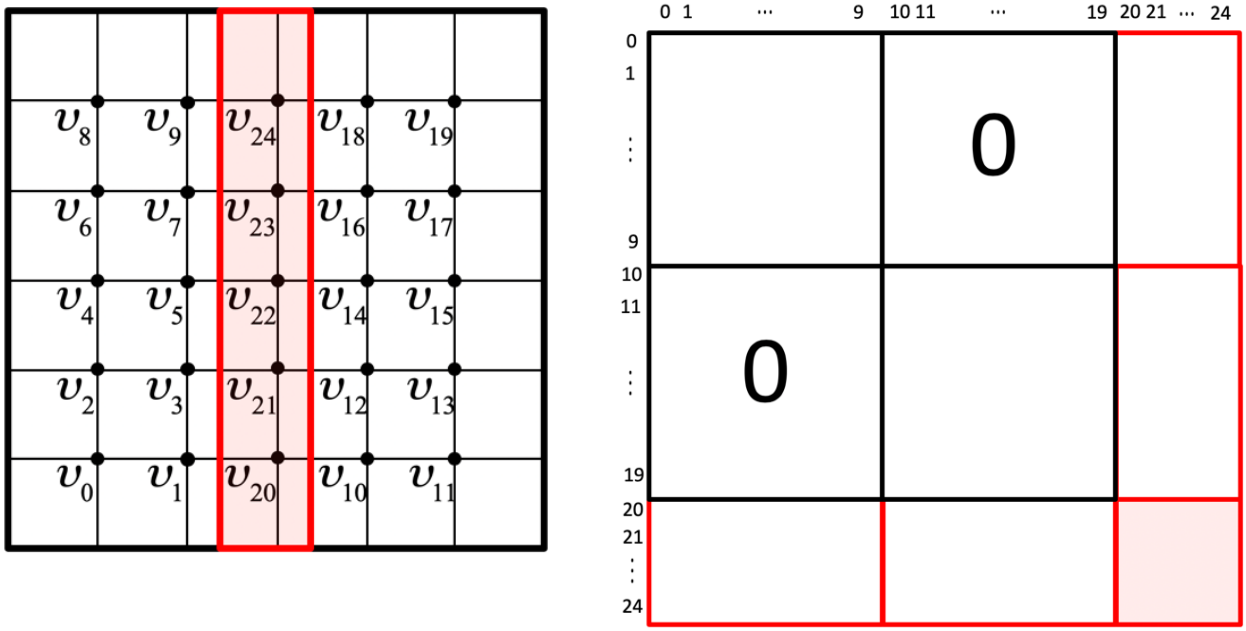


Figure 7.2.2.1 An illustration of nested dissection.

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

$$\left(\begin{array}{c|c|c} A_{00} & 0 & A_{20}^T \\ \hline 0 & A_{11} & A_{21}^T \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right) = \underbrace{\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline 0 & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline 0 & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right)^T}_{\left(\begin{array}{c|c|c} L_{00}L_{00}^T & 0 & \star \\ \hline 0 & L_{11}L_{11}^T & \star \\ \hline L_{20}L_{00}^T & L_{21}L_{11}^T & L_{20}L_{20}^T + L_{21}L_{21}^T + L_{22}L_{22}^T \end{array}\right)},$$

  where the $\star$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.)
  - 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.)
  - Update the lower triangular part of A_{22} with

$$A_{22} - L_{20}L_{20}^T - L_{21}L_{21}^T.$$

  - 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

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline 0 & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right)\left(\begin{array}{c} z_0 \\ \hline z_1 \\ \hline z_2 \end{array}\right) = \left(\begin{array}{c} y_0 \\ \hline y_1 \\ \hline y_2 \end{array}\right).$$

  This can be solved via the steps

- Solve $L_{00}z_0 = y_0$.
- Solve $L_{11}z_1 = y_1$.
- Solve $L_{22}z_2 = y_2 - L_{20}z_0 - L_{21}z_1$.

Similarly,

$$\left(\begin{array}{c|c|c} L_{00}^T & 0 & L_{20}^T \\ \hline 0 & L_{11}^T & L_{21}^T \\ \hline 0 & 0 & L_{22}^T \end{array}\right)^T \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline x_2 \end{array}\right) = \left(\begin{array}{c} z_0 \\ \hline z_1 \\ \hline z_2 \end{array}\right).$$

can be solved via the steps

- Solve $L_{22}^T x_2 = z_2$.
- Solve $L_{11}^T x_1 = z_1 - L_{21}^T x_2$.
- Solve $L_{00}^T x_0 = z_0 - L_{20}^T x_2$.

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.

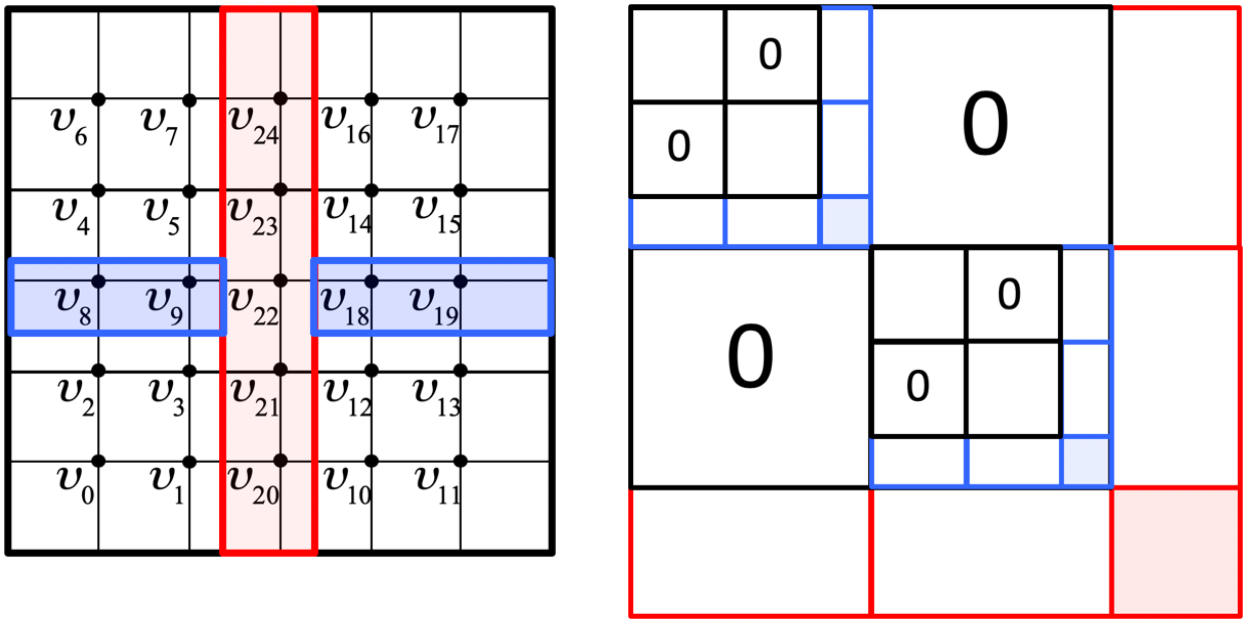


Figure 7.2.2.2 A second level of nested dissection.

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 nonzeroes 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 obervations 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=OMbxk1ihIFo

Let's review what we saw in Subsection 7.1.1. The linear system $Au = f$

$$
\begin{array}{ccccccccccc}
4\upsilon_0 & -\upsilon_1 & & & -\upsilon_4 & & & & & & = h^2\phi_0 \\
-\upsilon_0 & +4\upsilon_1 & -\upsilon_2 & & & -\upsilon_5 & & & & & = h^2\phi_1 \\
 & -\upsilon_1 & +4\upsilon_2 & -\upsilon_3 & & & -\upsilon_6 & & & & = h^2\phi_2 \\
 & & -\upsilon_2 & +4\upsilon_3 & & & & -\upsilon_7 & & & = h^2\phi_3 \\
-\upsilon_0 & & & & +4\upsilon_4 & -\upsilon_5 & & & -\upsilon_8 & & = h^2\phi_4 \\
 & \ddots & & & \ddots & \ddots & \ddots & & & \ddots & \vdots
\end{array}
$$

which can be written in matrix form as

$$\left(\begin{array}{cccc|cccc|cc}
4 & -1 & & & -1 & & & & & \\
-1 & 4 & -1 & & & -1 & & & & \\
 & -1 & 4 & -1 & & & -1 & & & \\
 & & -1 & 4 & & & & -1 & & \\
\hline
-1 & & & & 4 & -1 & & & -1 & \\
 & -1 & & & -1 & 4 & -1 & & & \ddots \\
 & & -1 & & & -1 & 4 & -1 & & \\
 & & & -1 & & & -1 & 4 & & \\
\hline
 & & & & -1 & & & & 4 & \ddots \\
 & & & & & & \ddots & & \ddots & \ddots
\end{array}\right)
\left(\begin{array}{c}
v_0 \\ v_1 \\ v_2 \\ v_3 \\ \hline v_4 \\ v_5 \\ v_6 \\ v_7 \\ \hline v_8 \\ \vdots
\end{array}\right)
=
\left(\begin{array}{c}
h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots
\end{array}\right).$$

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 kth iteration with $v_i^{(k)}$ and state the algorithm more explicitly as

for $k = 0, \ldots,$ convergence
  for $i = 0, \ldots, N \times N - 1$
    $v_i^{(k+1)} = (h^2\phi_i + v_{i-N}^{(k)} + v_{i-1}^{(k)} + v_{i+1}^{(k)} + v_{i+N}^{(k)})/4$
  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

$$\begin{array}{rcccccccccccc}
4v_0^{(k+1)} & = & & v_1^{(k)} & & & +v_4^{(k)} & & & & & & +h^2\phi_0 \\
4v_1^{(k+1)} & = & v_0^{(k)} & & +v_2^{(k)} & & & +v_5^{(k)} & & & & & +h^2\phi_1 \\
4v_2^{(k+1)} & = & & v_1^{(k)} & & +v_3^{(k)} & & & +v_6^{(k)} & & & & +h^2\phi_2 \\
4v_3^{(k+1)} & = & & & v_2^{(k)} & & & & & +v_7^{(k)} & & & +h^2\phi_3 \\
4v_4^{(k+1)} & = & v_0^{(k)} & & & & & +v_5^{(k)} & & & -v_8^{(k)} & & +h^2\phi_4 \\
\vdots & & & \ddots & & & \ddots & \ddots & \ddots & & & \ddots & \vdots
\end{array}$$

which can be written in matrix form as

$$
\left(\begin{array}{cccc|cccc|c}
4 & & & & & & & & \\
 & 4 & & & & & & & \\
 & & 4 & & & & & & \\
 & & & 4 & & & & & \\
\hline
 & & & & 4 & & & & \\
 & & & & & 4 & & & \\
 & & & & & & 4 & & \\
 & & & & & & & 4 & \\
\hline
 & & & & & & & 4 & \\
 & & & & & & & & \ddots
\end{array}\right)
\left(\begin{array}{c}
v_0^{(k+1)} \\ v_1^{(k+1)} \\ v_2^{(k+1)} \\ v_3^{(k+1)} \\ \hline v_4^{(k+1)} \\ v_5^{(k+1)} \\ v_6^{(k+1)} \\ v_7^{(k+1)} \\ \hline v_8^{(k+1)} \\ \vdots
\end{array}\right)
$$

$$
= \left(\begin{array}{cccc|cccc|c}
0 & 1 & & & 1 & & & & \\
1 & 0 & 1 & & & 1 & & & \\
 & 1 & 0 & 1 & & & 1 & & \\
 & & 1 & 0 & & & & 1 & \\
\hline
1 & & & & 0 & 1 & & & 1 \\
 & 1 & & & 1 & 0 & 1 & & \\
 & & 1 & & & 1 & 0 & 1 & \\
 & & & 1 & & & 1 & 0 & \\
\hline
 & & & & 1 & & & & 0 \\
 & & & & & \ddots & & & \ddots
\end{array}\right)
\left(\begin{array}{c}
v_0^{(k)} \\ v_1^{(k)} \\ v_2^{(k)} \\ v_3^{(k)} \\ \hline v_4^{(k)} \\ v_5^{(k)} \\ v_6^{(k)} \\ v_7^{(k)} \\ \hline v_8^{(k)} \\ \vdots
\end{array}\right)
+
\left(\begin{array}{c}
h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots
\end{array}\right). \tag{7.3.1}
$$

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=ufMUhO1vDew

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 $k = 0, \ldots,$ convergence
  for $i = 0, \ldots, N \times N - 1$
    $v_i^{(k+1)} = (h^2\phi_i + v_{i-N}^{(k+1)} + v_{i-1}^{(k+1)} + v_{i+1}^{(k)} + v_{i+N}^{(k)})/4$
  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:

$$
\left(\begin{array}{cccc|cccc|c}
4 & & & & & & & & \\
 & 4 & & & & & & & \\
 & & 4 & & & & & & \\
 & & & 4 & & & & & \\
\hline
 & & & & 4 & & & & \\
 & & & & & 4 & & & \\
 & & & & & & 4 & & \\
 & & & & & & & 4 & \\
\hline
 & & & & & & & & 4 \\
 & & & & & & & & & \ddots
\end{array}\right)
\left(\begin{array}{c}
v_0^{(k+1)} \\ v_1^{(k+1)} \\ v_2^{(k+1)} \\ v_3^{(k+1)} \\ \hline v_4^{(k+1)} \\ v_5^{(k+1)} \\ v_6^{(k+1)} \\ v_7^{(k+1)} \\ \hline v_8^{(k+1)} \\ \vdots
\end{array}\right)
$$

$$
= \left(\begin{array}{cccc|cccc|c}
0 & 1 & & & 1 & & & & \\
1 & 0 & 1 & & & 1 & & & \\
 & 1 & 0 & 1 & & & 1 & & \\
 & & 1 & 0 & & & & 1 & \\
\hline
1 & & & & 0 & 1 & & & 1 \\
 & 1 & & & 1 & 0 & 1 & & \\
 & & 1 & & & 1 & 0 & 1 & \\
 & & & 1 & & & 1 & 0 & \\
\hline
 & & & & 1 & & & & 0 & \ddots \\
 & & & & & \ddots & & & \ddots & \ddots
\end{array}\right)
\left(\begin{array}{c}
v_0^{(k)} \\ v_1^{(k)} \\ v_2^{(k)} \\ v_3^{(k)} \\ \hline v_4^{(k)} \\ v_5^{(k)} \\ v_6^{(k)} \\ v_7^{(k)} \\ \hline v_8^{(k)} \\ \vdots
\end{array}\right)
+
\left(\begin{array}{c}
h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots
\end{array}\right). \tag{7.3.2}
$$

Modify this to reflect the Gauss-Seidel iteration.

Solution.

$$
\left(\begin{array}{cccc|cccc|c}
4 & & & & & & & & \\
-1 & 4 & & & & & & & \\
 & -1 & 4 & & & & & & \\
 & & -1 & 4 & & & & & \\
\hline
-1 & & & & 4 & & & & \\
 & -1 & & & -1 & 4 & & & \\
 & & -1 & & & -1 & 4 & & \\
 & & & -1 & & & -1 & 4 & \\
\hline
 & & & & -1 & & & & 4 \\
 & & & & & \ddots & & & \ddots & \ddots
\end{array}\right)
\left(\begin{array}{c}
v_0^{(k+1)} \\ v_1^{(k+1)} \\ v_2^{(k+1)} \\ v_3^{(k+1)} \\ \hline v_4^{(k+1)} \\ v_5^{(k+1)} \\ v_6^{(k+1)} \\ v_7^{(k+1)} \\ \hline v_8^{(k+1)} \\ \vdots
\end{array}\right)
$$

$$
:= \left(\begin{array}{cccc|cccc|c}
0 & 1 & & & 1 & & & & \\
 & 0 & 1 & & & 1 & & & \\
 & & 0 & 1 & & & 1 & & \\
 & & & 0 & & & & 1 & \\
\hline
 & & & & 0 & 1 & & & 1 \\
 & & & & & 0 & 1 & & & \ddots \\
 & & & & & & 0 & 1 & \\
 & & & & & & & 0 & \\
\hline
 & & & & & & & & 0 & \ddots \\
 & & & & & & & & & \ddots
\end{array}\right)
\left(\begin{array}{c}
v_0^{(k)} \\ v_1^{(k)} \\ v_2^{(k)} \\ v_3^{(k)} \\ \hline v_4^{(k)} \\ v_5^{(k)} \\ v_6^{(k)} \\ v_7^{(k)} \\ \hline v_8^{(k)} \\ \vdots
\end{array}\right)
+
\left(\begin{array}{c}
h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots
\end{array}\right).
$$

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 $\chi_0^{(k+1)}, \chi_1^{(k+1)}, \ldots$. If $A = D - L - L^T$, this is captured by

$$(D - L)x^{(k+1)} = L^T x^{(k)} + y. \tag{7.3.3}$$

Modify (7.3.3) to yield a "reverse" Gauss-Seidel method that computes the entries of vector $x^{(k+1)}$ in the order $\chi_{n-1}^{(k+1)}, \chi_{n-2}^{(k+1)}, \ldots$.

Solution. The reverse order is given by $\chi_{n-1}^{(k+1)}, \chi_{n-2}^{(k+1)}, \ldots$. 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

$$\begin{array}{rcl} M_F x^{(k+\frac{1}{2})} & = & N_F x^{(k)} + y \\ M_R x^{(k+1)} & = & N_R x^{(k+\frac{1}{2})} + y \end{array}$$

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
$$\begin{array}{l} I + L(D - L)^{-1} \\ \quad = \\ (D - L)(D - L)^{-1} + L(D - L)^{-1} \quad = \\ (D - L + L)(D - L)^{-1} \quad = \\ D(D - L)^{-1}. \end{array}$$
- 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.)
$$(D - L^T)x^{(k+1)} = L(D - L)^{-1}L^T x^{(k)} + D(D - L)^{-1}y$$
$$\underbrace{(D - L)D^{-1}(D - L^T)}_{M}\ x^{(k+1)} = \underbrace{(D - L)D^{-1}L(D - L)^{-1}L^T}_{N}\ x^{(k)} + y$$

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.

$$\begin{array}{l} x^{(k)} + M^{-1}r^{(k)} \\ = \\ x^{(k)} + M^{-1}(y - Ax^{(k)}) \\ = \\ x^{(k)} + M^{-1}(y - (M - N)x^{(k)}) \\ = \\ x^{(k)} + M^{-1}y - M^{-1}(M - N)x^{(k)} \\ = \\ x^{(k)} + M^{-1}y - (I - M^{-1}N)x^{(k)} \\ = \\ M^{-1}(Nx^{(k)} + y) \end{array}$$

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

$$\lim_{k \to \infty} \chi^{(k)} = \chi.$$

A sequence of vectors, $x^{(k)}$, converges to the vector x if for some norm $\| \cdot \|$

$$\lim_{k \to \infty} \|x^{(k)} - x\| = 0.$$

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 $\| \cdot \|$

$$\lim_{k \to \infty} \|A^{(k)} - A\|.$$

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

$$\begin{array}{rcl} Mx & = & Nx + y \\ Mx^{(k+1)} & = & Nx^{(k)} + y \end{array}$$

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 $\chi_i^{\mathrm{GS}\,(i+1)}$ in our subsequent discussion.

What if we pick our next value a bit further:

$$\chi_i^{(k+1)} = \omega\chi_i^{\mathrm{GS}\,(k+1)} + (1-\omega)\chi_i^{(k)},$$

where $\omega \geq 1$. This is known as **over-relaxation**. Then

$$\chi_i^{\mathrm{GS}\,(k+1)} = \frac{1}{\omega}\chi_i^{(k+1)} - \frac{1-\omega}{\omega}\chi_i^{(k)}$$

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:

$$\begin{array}{rcl} x^{(k+\frac{1}{2})} & = & M_F^{-1}(N_F x^{(k)} + y) \\ x^{(k+1)} & = & M_R^{-1}(N_R x^{(k+\frac{1}{2})} + y). \end{array}$$

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.

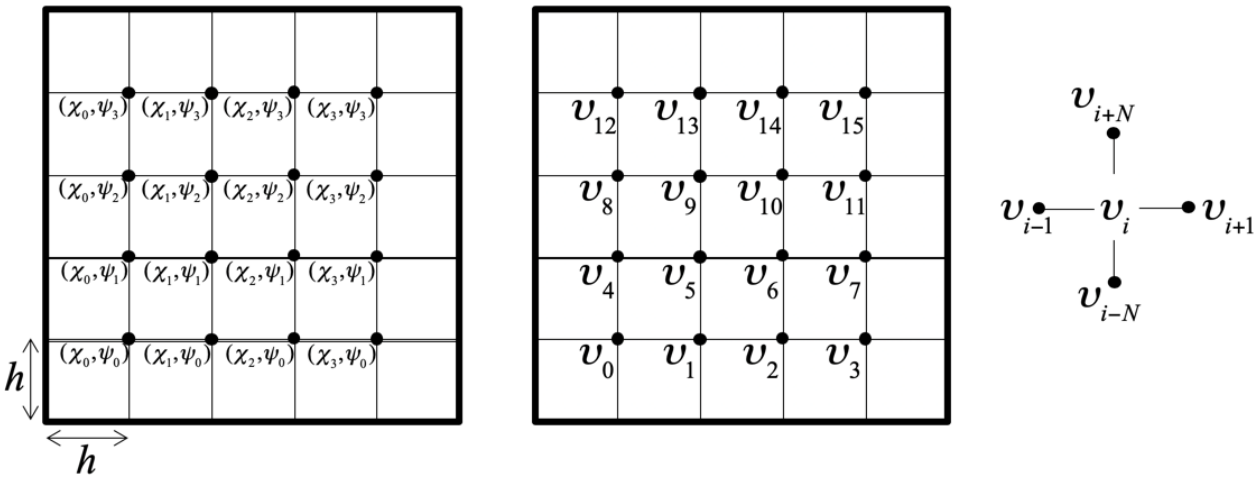


Figure 7.4.1.1 2D mesh.

In that figure, υ_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

$$\begin{array}{ccccccccccc}
4v_0 & -v_1 & & & -v_4 & & & & & & = h^2\phi_0 \\
-v_0 & +4v_1 & -v_2 & & & -v_5 & & & & & = h^2\phi_1 \\
 & -v_1 & +4v_2 & -v_3 & & & -v_6 & & & & = h^2\phi_2 \\
 & & -v_2 & +4v_3 & & & & -v_7 & & & = h^2\phi_3 \\
-v_0 & & & & +4v_4 & -v_5 & & & -v_8 & & = h^2\phi_4 \\
 & v_1 & & & \ddots & \ddots & \ddots & & & \ddots & \vdots
\end{array}$$

In matrix notation this becomes

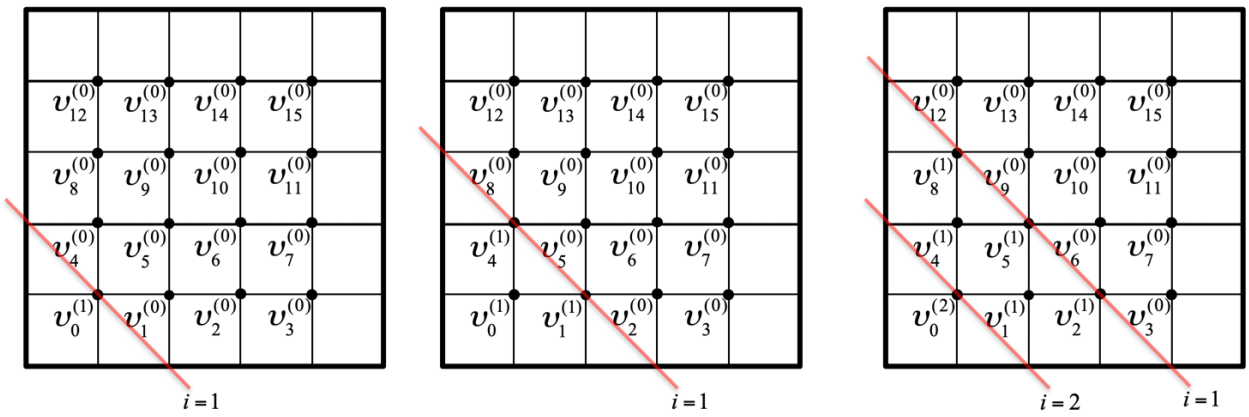
$$\left(\begin{array}{cccc|cccc|cc}
4 & -1 & & & -1 & & & & & \\
-1 & 4 & -1 & & & -1 & & & & \\
 & -1 & 4 & -1 & & & -1 & & & \\
 & & -1 & 4 & & & & -1 & & \\
\hline
-1 & & & & 4 & -1 & & & -1 & \\
 & -1 & & & -1 & 4 & -1 & & & \ddots \\
 & & -1 & & & -1 & 4 & -1 & & \\
 & & & -1 & & & -1 & 4 & & \\
\hline
 & & & & -1 & & & & 4 & \ddots \\
 & & & & & & \ddots & & \ddots & \ddots
\end{array}\right)
\left(\begin{array}{c}
v_0 \\ v_1 \\ v_2 \\ v_3 \\ \hline v_4 \\ v_5 \\ v_6 \\ v_7 \\ \hline v_8 \\ \vdots
\end{array}\right)
=
\left(\begin{array}{c}
h^2\phi_0 \\ h^2\phi_1 \\ h^2\phi_2 \\ h^2\phi_3 \\ \hline h^2\phi_4 \\ h^2\phi_5 \\ h^2\phi_6 \\ h^2\phi_7 \\ \hline h^2\phi_8 \\ \vdots
\end{array}\right). \tag{7.4.1}$$

This demonstrates how solving the discretized Poisson's equation boils down to the solution of a linear system $Au = h^2 f$, 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.

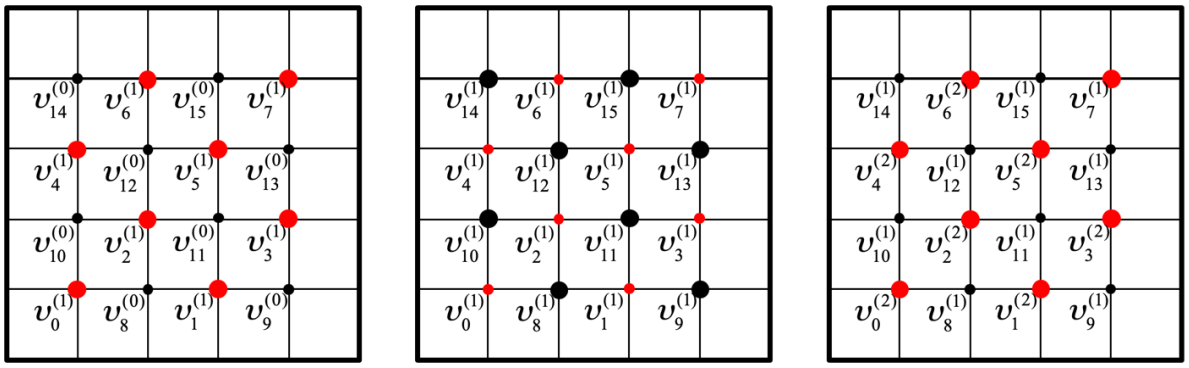
There is actually quite a bit of parallelism to be exploited in the Gauss-Seidel iteration as well. Consider our example of a mesh on a square domain as illustrated by



- 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 $v_1^{(1)}$, $v_N^{0i)}$, 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 $i = 0, \ldots, n-2$
  $\alpha_{i,i} := \sqrt{\alpha_{i,i}}$
  $\alpha_{i+1,i} := \alpha_{i+1,i} / \alpha_{i,i}$
  $\alpha_{i+1,i+1} := \alpha_{i+1,i+1} - \alpha_{i+1,i} \alpha_{i+1,i}$
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 $i = 0, \ldots, n-2$
    $\psi_i := \psi_i / \alpha_{i,i}$
    $\psi_{i+1} := \psi_{i+1} - \alpha_{i+1,i} \psi_i$
  endfor
  $\psi_{n-1} := \psi_{n-1} / \alpha_{n-1,n-1}$

- Overwriting y with the solution of $L^T x = z$, (where z has overwritten y (back substitution).

  for $i = n-1, \ldots, 1$
    $\psi_i := \psi_i / \alpha_{i,i}$
    $\psi_{i-1} := \psi_{i-1} - \alpha_{i,i-1} \psi_i$
  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 subdiagonals 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 nonzeroes).

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 $\| \cdots \|$

$$\|M^{-1}N\| < 1.$$

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 $\widehat{\chi}$ that minimizes this polynomial can be accomplished via the steps:

- Compute the derivative and set it to zero:

$$f'(\widehat{\chi}) = \alpha\widehat{\chi} - \beta = 0.$$

  We notice that computing $\widehat{\chi}$ is equivalent to solving the linear system (of one equation)

$$\alpha\widehat{\chi} = \beta.$$

- It is a minimum if $\alpha > 0$ (the quadratic polynomial is concaved up).

Obviously, you can turn this around: in order to solve $\alpha\widehat{\chi} = \beta$ where $\alpha > 0$, we can instead minimize the polynomial

$$f(\chi) = \frac{1}{2}\alpha\chi^2 - \beta\chi.$$

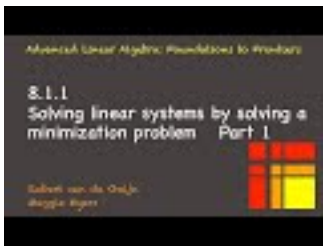
This course does not have multivariate calculus as a prerequisite, so we will walk you through the basic results we will employ. We will focus on finding a solution to $Ax = b$ where A is symmetric positive definite (SPD). (In our discussions we will just focus on real-valued problems). Now, if

$$f(x) = \frac{1}{2}x^T Ax - x^T b,$$

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 $A\widehat{x} = b$ we can instead find the vector $\widehat{x}$ 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 $A\widehat{x} = b$. Then the vector $\widehat{x}$ minimizes the function $f(x) = \frac{1}{2}x^T Ax - x^T b$.*

Proof. This proof does not employ multivariate calculus!

Let $A\widehat{x} = b$. Then

$$\begin{array}{l} f(x) \\ \quad = \quad < \text{ definition of } f(x) > \\ \frac{1}{2}x^T Ax - x^T b \\ \quad = \quad < A\widehat{x} = b > \\ \frac{1}{2}x^T Ax - x^T A\widehat{x} \\ \quad = \quad < \text{ algebra } > \\ \frac{1}{2}x^T Ax - x^T A\widehat{x} + \underbrace{\frac{1}{2}\widehat{x}^T A\widehat{x} - \frac{1}{2}\widehat{x}^T A\widehat{x}}_{0} \\ \quad = \quad < \text{ factor out } > \\ \frac{1}{2}(x - \widehat{x})^T A(x - \widehat{x}) - \frac{1}{2}\widehat{x}^T A\widehat{x}. \end{array}$$

Since $\widehat{x}^T A\widehat{x}$ is independent of x, and A is SPD, this is clearly minimized when $x = \widehat{x}$. ■

8.1.2 Overview

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 $-\widehat{x}A\widehat{x}$ at $\widehat{x}$ 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 kth 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, preferrably, $f(x^{(k+1)}) \leq f(x^{(k)})$. 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 α_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=O1Slxl3oAc8

Now,

$$\begin{array}{l}
f(x^{(k+1)}) \\
\quad = \quad < x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} > \\
f(x^{(k)} + \alpha_k p^{(k)}) \\
\quad = \quad < \text{ evaluate } > \\
\frac{1}{2}\left(x^{(k)} + \alpha_k p^{(k)}\right)^T A \left(x^{(k)} + \alpha_k p^{(k)}\right) - \left(x^{(k)} + \alpha_k p^{(k)}\right)^T b \\
\quad = \quad < \text{ multiply out } > \\
\frac{1}{2}x^{(k)\,T}Ax^{(k)} + \alpha_k p^{(k)\,T}Ax^{(k)} + \frac{1}{2}\alpha_k^2 p^{(k)\,T}Ap^{(k)} - x^{(k)\,T}b - \alpha_k p^{(k)\,T}b \\
\quad = \quad < \text{ rearrange } > \\
\frac{1}{2}x^{(k)\,T}Ax^{(k)} - x^{(k)\,T}b + \frac{1}{2}\alpha_k^2 p^{(k)\,T}Ap^{(k)} + \alpha_k p^{(k)\,T}Ax^{(k)} - \alpha_k p^{(k)\,T}b \\
\quad = \quad < \text{ substitute } f(x^{(k)}) \text{ and factor out common terms } > \\
f(x^{(k)}) + \frac{1}{2}\alpha_k^2 p^{(k)\,T}Ap^{(k)} + \alpha_k p^{(k)\,T}(Ax^{(k)} - b) \\
\quad = \quad < \text{ substitute } r^{(k)} \text{ and commute to expose polynomial in } \alpha_k \\
\frac{1}{2}p^{(k)\,T}Ap^{(k)}\alpha_k^2 - p^{(k)\,T}r^{(k)}\alpha_k + f(x^{(k)}),
\end{array}$$

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

$$f(x^{(k+1)}) = \frac{1}{2}p^{(k)\,T}Ap^{(k)}\alpha_k^2 - p^{(k)\,T}r^{(k)}\alpha_k + f(x^{(k)})$$

exactly requires the deriviative with respect to α_k to be zero:

$$0 = \frac{df(x^{(k)} + \alpha_k p^{(k)})}{d\alpha_k} = p^{(k)\,T}Ap^{(k)}\alpha_k - p^{(k)\,T}r^{(k)}.$$

Hence, for a given choice of p_k

$$\alpha_k = \frac{p^{(k)\,T}r^{(k)}}{p^{(k)\,T}Ap^{(k)}} \quad \text{and} \quad x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}.$$

provides the next approximation to the solution. This leaves us with the question of how to pick the search directions $\{p^{(0)}, p^{(1)}, \ldots\}$.

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
  $\alpha_k := \frac{p^{(k)\,T}r^{(k)}}{p^{(k)\,T}Ap^{(k)}}$
  $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
  $r^{(k+1)} := b - Ax^{(k+1)}$
  $k := k + 1$
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.

$\alpha_k := \frac{p^{(k)\,T}r^{(k)}}{p^{(k)\,T}Ap^{(k)}}$	1 mvmult, 2 dot products
$x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$	1 axpy
$r^{(k+1)} := b - Ax^{(k+1)}$	1 mvmult

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 nonzeroes 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.

$$\begin{aligned} r^{(k+1)} &= b - Ax^{(k+1)} \\ &= \quad < r^{(k)} = b - Ax^{(k)} > \\ r^{(k+1)} &= r^{(k)} + Ax^{(k)} - Ax^{(k+1)} \\ &= \quad < \text{ rearrange, factor } > \\ r^{(k+1)} &= r^{(k)} - A(x^{(k+1)} - x^{(k)}) \\ &= \quad < x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} > \\ r^{(k+1)} &= r^{(k)} - \alpha_k A p^{(k)} \end{aligned}$$

Alternatively:

$$\begin{aligned} r^{(k+1)} &= b - Ax^{(k+1)} \\ &= \quad < x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} > \\ r^{(k+1)} &= b - A(x^{(k)} + \alpha_k p^{(k)}) \\ &= \quad < \text{ distribute } > \\ r^{(k+1)} &= b - Ax^{(k)} - \alpha_k A p^{(k)} \\ &= \quad < \text{ definition of } r^{(k)} > \\ r^{(k+1)} &= r^{(k)} - \alpha_k A p^{(k)} \end{aligned}$$

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 $r^{(k)} \neq 0$
$p^{(k)} =$ 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**

Given : $A, b, x^{(0)}$
$r^{(0)} := b - Ax^{(0)}$
$k := 0$
while $r^{(k)} \neq 0$
$p^{(k)} :=$ next direction

$\alpha_k := \frac{p^{(k)\,T} r^{(k)}}{p^{(k)\,T} A p^{(k)}}$
$x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
$r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$
$k := k + 1$

endwhile

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

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 "axpy" operations (not counting the cost of determining the next descent direction).

Solution.

$q^{(k)} := Ap^{(k)}$	1 mvmult
$\alpha_k := \frac{p^{(k)\,T} r^{(k)}}{p^{(k)\,T} q x^{(k)}}$	2 dot products
$x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$	1 axpy
$r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$	1axpy

Total: 1 mvmults, 2 dot products, 2 axpys

YouTube: https://www.youtube.com/watch?v=OGqV_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=ifwailOB1EI

Let us pick some really simple search directions in the right-most algorithm in Homework 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).
- $r^{(0)} = Ax^{(0)} - b$.
- $p^{(0)\,T} r^{(0)} = e_0^T (b - Ax^{(0)}) = e_0^T b - e_0^T A x^{(0)} = \beta_0 - \widetilde{a}_0^T x^{(0)}$, where $\widetilde{a}_k^T$ denotes the kth row of A.
- $x^{(1)} = x^{(0)} + \alpha_0 p^{(0)} = x^{(0)} + \frac{p^{(0)\,T} r^{(0)}}{p^{(0)\,T} A p^{(0)}} e_0 = x^{(0)} + \frac{\beta_0 - \widetilde{a}_0^T x^{(0)}}{\alpha_{0,0}} e_0$. 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=tOqAd1OhIwc

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 (((Unresolved xref, reference "chapter09-shur-spectral-decomposition"; check spelling or use "provisional" attribute)))). 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 $\widehat{b} = Q^T b$, then

$$g(y) = \frac{1}{2}y^T \Sigma y - y^T \widehat{b}.$$

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 $\sigma_0 = \cdots = \sigma_{n-1}$? 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 $\widehat{b}$ (the transformed right-hand side), then

$$\begin{array}{l} y^{(1)} \\ \quad = \\ y^{(0)} - \frac{r^{(0)\,T} r^{(0)}}{r^{(0)\,T} \sigma_0 I r^{(0)}} r^{(0)} \\ \quad = \\ y^{(0)} - \frac{1}{\sigma_0} r^{(0)} \\ \quad = \\ y^{(0)} - \frac{1}{\sigma_0}(\sigma_0 y^{(0)} - \widehat{b}) \\ \quad = \\ \frac{1}{\sigma_0}\widehat{b}, \end{array}$$

which is the solution to $\sigma_0 I = \widehat{b}$. 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 $L_M^{-1} A L_M^{-T} \approx L_M^{-1} L L^T L_M^{-T} \approx I$. With this, we can transform our linear system $Ax = b$ in to one that has the same solution:

$$\underbrace{L_M^{-1} A L_M^{-T}}_{\tilde{A}} \quad \underbrace{L_M^T x}_{\tilde{x}} \quad = \quad \underbrace{L_M^{-1} b}_{\tilde{b}} .$$

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 $\tilde{A} = L_M^{-1} A L_M^{-T}$, $\tilde{x} = L_M^T x$, 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, $\kappa(L_M^{-1} A L_M^{-T})$ 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 : $A, b, x^{(0)}$, $M = LL^T$

$\tilde{A} = L^{-1} A L^{-T}$
$\tilde{b} = L^{-1} b$
$\tilde{x}^{(0)} = L^T x^{(0)}$
$\tilde{r}^{(0)} := \tilde{b} - \tilde{A}\tilde{x}^{(0)}$
$k := 0$
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 $r^{(k)} \neq 0$
  $p^{(k)} := M^{-1} r^{(k)}$
  $q^{(k)} := Ap^{(k)}$
  $\alpha_k := \frac{p^{(k)\,T} r^{(k)}}{p^{(k)\,T} q^{(k)}}$
  $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
  $r^{(k+1)} := r^{(k)} - \alpha_k q^{(k)}$

  $k := k + 1$
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 $x^{(k)} = l^{-T} x^{(k)}$ hold for all k, where $r^{(k)}$ and $x^{(k)}$ are as computed by the algorithm on the right.

Solution 1. Notice that $\tilde{A} = L^{-1} A L^{-T}$ 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 $\tilde{A}$ $>$
    $L^{-1} b - L^{-1} A x^{(0)}$
    $=$ $<$ 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. $>$
  $\frac{(L^Tp^{(0)})^TL^{-1}r^{(0)}}{(L^Tp^{(0)})^TL^{-1}AL^{-T}L^Tp^{(0)}}$
  $=$ $<$ transpose and cancel $>$
  $\frac{p^{(0)\,T}r^{(0)}}{p^{(0)\,T}Ap^{(0)}}$
  $=$ $<$ right algorithm $>$
  α_0.

- Inductive Step: Assume that $\tilde{x}^{(k)} = L^Tx^{(k)}$, $\tilde{r}^{(k)} = L^{-1}r^{(k)}$, $\tilde{p}^{(k)} = L^Tp^{(k)}$, and $\tilde{\alpha}_k = \alpha_k$. Show that $\tilde{x}^{(k+1)} = L^Tx^{(k+1)}$, $\tilde{r}^{(k+1)} = L^{-1}r^{(k+1)}$, $\tilde{p}^{(k+1)} = L^Tp^{(k+1)}$, 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. $>$
  $L^Tx^{(k)} + \alpha_kL^Tp^{(k)}$
  $=$ $<$ factor out; right algorithm $>$
  $L^Tx^{(k+1)}$

- $\tilde{r}^{(k+1)}$
  $=$ $<$ middle algorithm $>$
  $\tilde{r}^{(k)} - \tilde{\alpha}_k\tilde{A}\tilde{p}^{(k)}$
  $=$ $<$ I.H. $>$
  $L^{-1}r^{(k)} - \alpha_kL^{-1}AL^{-T}L^Tp^{(k)}$
  $=$ $< L^{-T}L^T = I;$ factor out; right algorithmn $>$
  $L^{-1}r^{(k+1)}$
- $\tilde{p}^{(k+1)}$
  $=$ $<$ middle algorithm $>$
  $\tilde{r}^{(k+1)}$
  $=$ $< \tilde{r}^{(k+1)} = L^{-1}r^{(k+1)} >$
  $L^{-1}r^{(k+1)}$
  $=$ $<$ from right algorithm: $r^{(k+1)} = Mp^{(k+1)}$ and $M = LL^T >$
  $L^{-1}LL^Tp^{(k+1)}$
  $=$ $< L^{-1}L = I >$
  $= L^Tp^{(k+1)}$.

- $\tilde{\alpha}_{k+1}$
  $= \quad <$ middle algorithm $>$
  $\frac{\tilde{p}^{(k+1)\,T}\tilde{r}^{(k+1)}}{\tilde{p}^{(k+1)\,T}\tilde{A}\tilde{p}^{(k+1)}}$
  $= \quad <\tilde{p}^{(k+1)} = L^T p^{(k+1)}$ etc. $>$
  $\frac{(L^T p^{(k+1)})^T L^{-1} r^{(k+1)}}{(L^T p^{(k+1)})^T L^{-1} A L^{-T} L^T p^{(k+1)}}$
  $= \quad <$ transpose and cancel $>$
  $\frac{p^{(k+1)\,T} r^{(k+1)}}{p^{(k+1)\,T} A p^{(k+1)}}$
  $= \quad <$ right algorithm $>$
  α_{k+1}.

- 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 $r^{(k)} \neq 0$
$\tilde{p}^{(k)} := L^{-1}r^{(k)}$
$\tilde{q}^{(k)} := L^{-1}AL^{-T}\tilde{p}^{(k)}$
$\tilde{\alpha}_k := \frac{\tilde{p}^{(k)\,T}L^{-1}r^{(k)}}{\tilde{p}^{(k)\,T}\tilde{q}^{(k)}}$
$x^{(k+1)} := x^{(k)} + \tilde{\alpha}_k L^{-T}\tilde{p}^{(k)}$
$r^{(k+1)} := r^{(k)} - \tilde{\alpha}_k L\tilde{q}^{(k)}$
$k := k + 1$
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
$\alpha_k := \frac{p^{(k)\,T} r^{(k)}}{p^{(k)\,T} Ap^{(k)}}$
$x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
$r^{(k+1)} := r^{(k)} - \alpha Ap^{(k)}$
$k := k + 1$
endwhile

Given : A, b
$x := 0$
$r := b$

while $r \neq 0$
$p :=$ next direction
$\alpha := \frac{p^T r}{p^T Ap}$
$x := x + \alpha p$
$r := r - \alpha Ap$

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, $x^{(k+1)} \in \text{Span}(p^{(0)}, \ldots, p^{(k)})$.

It would be nice if after the kth iteration

$$f(x^{(k+1)}) = \min_{x \in \text{Span}(p^{(0)}, \ldots, p^{(k)})} f(x) \tag{8.3.1}$$

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

$$\text{Span}(p^{(0)}, \ldots, p^{(n-1)}) = \mathbb{R}^n$$

so that

$$f(x^{(n)}) = \min_{x \in \text{Span}(p^{(0)}, \ldots, p^{(n-1)})} f(x) = \min_{x \in \mathbb{R}^n} f(x)$$

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 $x^{(j)} + \alpha p^{(j)}$, $j = 0, \ldots, k$. However, unless we pick the search directions very carefully, that is not the same as it minimizing over all vectors in $\text{Span}(p^{(0)}, \ldots, p^{(k)})$.

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

$$x^{(k)} = \begin{pmatrix} p^{(0)} & \cdots & p^{(k-1)} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{k-1} \end{pmatrix} = P^{(k-1)} a_{k-1}. \tag{8.3.2}$$

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

$$\begin{aligned} &\min_{x \in \text{Span}(p^{(0)}, \ldots, p^{(k-1)}, p^{(k)})} f(x) = \min_y f(P^{(k)} y) \\ &= \min_y \left[\tfrac{1}{2} y_0^T P^{(k-1)\,T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)\,T} b \right. \\ &\qquad \left. + \psi_1 y_0^T P^{(k-1)\,T} A p^{(k)} + \tfrac{1}{2} \psi_1^2 p^{(k)\,T} A p^{(k)} - \psi_1 p^{(k)\,T} b \right], \end{aligned}$$

where $y = \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \in \mathbb{R}^{k+1}$.

Hint.

$$x \in \text{Span}(p^{(0)}, \ldots, p^{(k-1)}, p^{(k)})$$

if and only if there exists

$$y = \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \in \mathbb{R}^{k+1} \text{ such that } x = \left(\begin{array}{c|c} P^{(k-1)} & p^{(k)} \end{array} \right) \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}.$$

Solution.

$$
\begin{array}{l}
\min_{x \in \mathrm{Span}(p^{(0)},\ldots,p^{(k-1)},p^{(k)})} f(x) \\
\quad = \quad < \text{ equivalent formulation } > \\
\min_y f(\left(\begin{array}{c|c} P^{(k-1)} & p^{(k)} \end{array}\right) y) \\
\quad = \quad < \text{ partition } y = \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} > \\
\min_y f(\left(\begin{array}{c|c} P^{(k-1)} & p^{(k)} \end{array}\right) \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix}) \\
\quad = \quad < \text{ instantiate } f > \\
\min_y \left[\frac{1}{2} \left[\left(\begin{array}{c|c} P^{(k-1)} & p^{(k)} \end{array}\right) \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \right]^T A \left(\begin{array}{c|c} P^{(k-1)} & p^{(k)} \end{array}\right) \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \right. \\
\qquad \left. - \left[\left(\begin{array}{c|c} P^{(k-1)} & p^{(k)} \end{array}\right) \begin{pmatrix} y_0 \\ \psi_1 \end{pmatrix} \right]^T b \right]. \\
\quad = \quad < \text{ multiply out } > \\
\min_y \left[\frac{1}{2} \left[y_0^T P^{(k-1)\,T} + \psi_1 p^{(k)\,T} \right] A \left[P^{(k-1)} y_0 + \psi_1 p^{(k)} \right] - y_0^T P^{(k-1)\,T} b - \psi_1 p^{(k)\,T} b \right] \\
\quad = \quad < \text{ multiply out some more } > \\
\min_y \left[\frac{1}{2} y_0^T P^{(k-1)\,T} A P^{(k-1)} y_0 + \psi_1 y_0^T P^{(k-1)\,T} A p^{(k)} \right. \\
\qquad \left. + \frac{1}{2} \psi_1^2 p^{(k)\,T} A p^{(k)} - y_0^T P^{(k-1)\,T} b - \psi_1 p^{(k)\,T} b \right] \\
\quad = \quad < \text{ rearrange } > \\
\min_y \left[\frac{1}{2} y_0^T P^{(k-1)\,T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)\,T} b + \psi_1 y_0^T P^{(k-1)\,T} A p^{(k)} \right. \\
\qquad \left. + \frac{1}{2} \psi_1^2 p^{(k)\,T} A p^{(k)} - \psi_1 p^{(k)\,T} b \right].
\end{array}
$$

YouTube: https://www.youtube.com/watch?v=5eNmr776GJY

Now, *if*

$$P^{(k-1)\,T} A p^{(k)} = 0$$

then

$$
\begin{array}{l}
\min_{x \in \mathrm{Span}(p^{(0)},\ldots,p^{(k-1)},p^{(k)})} f(x) \\
\quad = \quad < \text{ from before } > \\
\min_y \left[\frac{1}{2} y_0^T P^{(k-1)\,T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)\,T} b \right. \\
\qquad \left. + \underbrace{\psi_1 y_0^T P^{(k-1)\,T} A p^{(k)}}_{0} + \frac{1}{2} \psi_1^2 p^{(k)\,T} A p^{(k)} - \psi_1 p^{(k)\,T} b \right] \\
\quad = \quad < \text{ remove zero term } > \\
\min_y \left[\frac{1}{2} y_0^T P^{(k-1)\,T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)\,T} b \right. \\
\qquad \left. + \frac{1}{2} \psi_1^2 p^{(k)\,T} A p^{(k)} - \psi_1 p^{(k)\,T} b \right] \\
\quad = \quad < \text{ split into two terms that can be minimized separately } > \\
\min_{y_0} \left[\frac{1}{2} y_0^T P^{(k-1)\,T} A P^{(k-1)} y_0 - y_0^T P^{(k-1)\,T} b \right] + \min_{\psi_1} \left[\frac{1}{2} \psi_1^2 p^{(k)\,T} A p^{(k)} - \psi_1 p^{(k)\,T} b \right] \\
\quad = \quad < \text{ recognize first set of terms as } f(P^{(k-1)} y_0) > \\
\min_{x \in \mathrm{Span}(p^{(0)},\ldots,p^{(k-1)})} f(x) + \min_{\psi_1} \left[\frac{1}{2} \psi_1^2 p^{(k)\,T} A p^{(k)} - \psi_1 p^{(k)\,T} b \right].
\end{array}
$$

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

$$x^{(k+1)} = P^{(k-1)} y_0 + \psi_1 p^{(k)} = \alpha_0 p^{(0)} + \cdots + \alpha_{k-1} p^{(k-1)} + \alpha_k p^{(k)} = x^{(k)} + \alpha_k p^{(k)}.$$

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 $p^{(0)}, \ldots, p^{(k-1)} \in \mathbb{R}^n$ such that $p^{(j)\,T} A p^{(i)} = 0$ if and only if $j \neq i$ is said to be A-conjugate. ◊

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 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.

$$\begin{array}{l}
P^T A P \\
\quad = \quad < \text{ partition } P \text{ by columns } > \\
\left(\begin{array}{c|c|c|c} p_0 & \cdots & p_{k-1} \end{array} \right)^T A \left(\begin{array}{c|c|c|c} p_0 & \cdots & p_{k-1} \end{array} \right) \\
\quad = \quad < \text{ transpose } > \\
\left(\begin{array}{c} p_0^T \\ \hline \vdots \\ \hline p_{k-1}^T \end{array} \right) A \left(\begin{array}{c|c|c|c} p_0 & \cdots & p_{k-1} \end{array} \right) \\
\quad = \quad < \text{ multiply out } > \\
\left(\begin{array}{c} p_0^T \\ \hline \vdots \\ \hline p_{k-1}^T \end{array} \right) \left(\begin{array}{c|c|c|c} Ap_0 & \cdots & Ap_{k-1} \end{array} \right) \\
\quad = \quad < \text{ multiply out } > \\
\left(\begin{array}{c|c|c|c} p_0^T A p_0 & p_0^T A p^{(k)} & \cdots & p_0^T A p_{k-1} \\ \hline p^{(k)\,T} A p_0 & p^{(k)\,T} A p^{(k)} & \cdots & p^{(k)\,T} A p_{k-1} \\ \hline \vdots & & \vdots & \\ \hline p_{k-1}^T A p_0 & p_{k-1}^T A p^{(k)} & \cdots & p_{k-1}^T A p_{k-1} \end{array} \right) \\
\quad = \quad < \text{ multiply out } > \\
\left(\begin{array}{c|c|c|c} p_0^T A p_0 & 0 & \cdots & 0 \\ \hline 0 & p^{(k)\,T} A p^{(k)} & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & p_{k-1}^T A p_{k-1} \end{array} \right),
\end{array}$$

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

$$\begin{array}{ll} 0 & \\ \quad = & < Py = 0 > \\ (Py)^T A(Py) & \\ \quad = & < \text{ multiply out } > \\ y^T P^T APy & \\ \quad = & < P^T AP = D > \\ y^T Dy & \\ \quad > & < D \text{ is SPD} > \\ 0, & \end{array}$$

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 $p^{(k)\,T} r^{(k)} \neq 0$
  $\alpha_k := \frac{p^{(k)\,T} r^{(k)}}{p^{(k)\,T} Ap^{(k)}}$
  $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
  $r^{(k+1)} := r^{(k)} - \alpha_k Ap^{(k)}$
  $k := k + 1$
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 $p^{(0)}, \ldots, p^{(k)}$ are chosen to be A-conjugate, then $x^{(k+1)}$ minimizes not only

$$f(x^{(k)} + \alpha p^{(k)})$$

but also

$$\min_{x \in \text{Span}(p^{(0)}, \ldots, p^{(k-1)})} f(x).$$

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 $p^{(0)}, \ldots, p^{(k-1)}$ were A-conjugate.
- Consider all vectors $p \in \mathbb{R}^n$ that are A-conjugate to $p^{(0)}, \ldots, p^{(k-1)}$. A vector p has this property if and only if $p \perp \text{Span}(Ap^{(0)}, \ldots, Ap^{(k-1)})$.
- For $p \perp \text{Span}(Ap^{(0)}, \ldots, Ap^{(k-1)})$ 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 $p^{(0)}, \ldots, p^{(k-1)}$ are orthogonal to the current residual, $p^T r^{(k)} = 0$ for all p with $P^{(k-1)\,T} Ap = 0$, then

  $$0 = p^T b - pAP^{(k-1)}a^{(k-1)} = p^T b \text{ for all } p \perp \text{Span}(Ap^{(0)}, \ldots, Ap^{(p-1)}).$$

  Let's think about this: b is orthogonal to all vectors that are orthogonal to $\text{Span}(Ap^{(0)}, \ldots, Ap^{(p-1)})$. This means that

  $$b \in \text{Span}(Ap^{(0)}, \ldots, Ap^{(k-1)}).$$

- 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 $\text{Span}(p^{(0)}, \ldots, p^{(k-1)})$. 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

$$\begin{array}{l} r^{(k)} \\ \quad = \\ b - Ax^{(k)} \\ \quad = \\ b - A(\alpha_0 p^{(0)} + \cdots + \alpha_{k-1} p^{(k-1)}) \\ \quad = \\ b - \alpha_0 A p^{(0)} - \cdots - \alpha_{k-1} A p^{(k-1)} \\ \quad = \\ r^{(k-1)} - \alpha_{k-1} A p^{(k-1)}. \end{array}$$

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 $p^{(0)}, \ldots, p^{(k-1)}$ and closest to the direction of steepest descent, $r^{(k)}$:

$$\|p^{(k)} - r^{(k)}\|_2 = \min_{p \perp \text{Span}(Ap^{(0)}, \ldots, Ap^{(k-1)})} \|r^{(k)} - p\|_2.$$

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 $\min_{p \perp \text{Span}(Ap^{(0)}, \ldots, Ap^{(k-1)})} \|r^{(k)} - p\|_2$
  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 8.3.3.1 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 Graduate 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

$$\|r^{(k)} - p^{(k)}\|_2 = \min_{p \perp \text{Span}(Ap^{(0)},\ldots,Ap^{(k-1)})} \|r^{(k)} - p\|_2.$$

Notice that

$$r^{(k)} = v + p^{(k)}$$

where v is the orthogonal projection of $r^{(k)}$ onto $\text{Span}(Ap^{(0)}, \ldots, Ap^{(k-1)})$

$$\|r^{(k)} - v\|_2 = \min_{w \in \text{Span}(Ap^{(0)},\ldots,Ap^{(k-1)})} \|r^{(k)} - w\|_2$$

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,*

- $P^{(k-1)\,T} r^{(k)} = 0.$
- $\text{Span}(p^{(0)}, \ldots, p^{(k-1)}) = \text{Span}(r^{(0)}, \ldots, r^{(k-1)}) = \text{Span}(b, Ab, \ldots, A^{k-1}b).$

Proof.

- Proving that

  $$P^{(k-1)\,T} r^{(k)} = 0.$$

  starts by considering that

  $$\begin{array}{l} f(P^{(k-1)}y) \\ = \\ \frac{1}{2}(P^{(k-1)}y)^T A(P^{(k-1)}y) - (P^{(k-1)}y)^T b \\ = \\ \frac{1}{2}y^T(P^{(k-1)\,T}AP^{(k-1)})y - y^T P^{(k-1)\,T} b \end{array}$$

  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

$$\min_{x \in \text{Span}(p^{(0)},\ldots,p^{(k-1)})} f(x)$$

we conclude that $x = P^{(k-1)}y_0$. But then

$$0 = P^{(k-1)\,T}b - \left(P^{(k-1)\,T}Ax^{(k)}\right) = P^{(k-1)\,T}\left(b - Ax^{(k)}\right) = P^{(k-1)\,T}r^{(k)}.$$

- Show that $\text{Span}(p^{(0)}, \ldots, p^{(k-1)}) = \text{Span}(r^{(0)}, \ldots, r^{(k-1)}) = \text{Span}(b, Ab, \ldots, A^{k-1}b)$.
  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.

      $$r^{(n-1)} \in \text{Span}(b, Ab, \ldots, A^{n-1}b)$$

      and

      $$p^{(n-1)} \in \text{Span}(b, Ab, \ldots, A^{n-1}b).$$

      But then

      $$Ap^{(n-1)} \in \text{Span}(Ab, A^2b, \ldots, A^nb)$$

      and hence

      $$r^{(n)} \in \text{Span}(b, Ab, A^2b, \ldots, A^nb).$$

    - $p^{(n)} = r^{(n)} - AP^{(n-1)}y_0$ and hence

      $$p^{(n)} \in \text{Span}(b, Ab, A^2b, \ldots, A^nb)$$

      since

      $$r^{(n)} \in \text{Span}(b, Ab, A^2b, \ldots, A^nb)$$

      and

      $$AP^{n-1}y_0 \in \text{Span}(Ab, A^2b, \ldots, A^nb).$$

  - 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

$$\mathcal{K}_k(A, b) = \text{Span}(b, Ab, \ldots, A^{k-1}b)$$

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

$$\text{Span}(p^{(0)},\ldots,p^{(j-1)}) = \text{Span}(r^{(0)},\ldots,r^{(j-1)})$$

and hence

$$\text{Span}(r^{(0)},\ldots,r^{(j-1)}) \subset \text{Span}(p^{(0)},\ldots,p^{(k-1)}) =$$

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

$$\begin{array}{ll}
& p^{(k)} \\
= & \quad < \text{(8.3.3)} > \\
& r^{(k)} - AP^{(k-1)} z^{(k-1)} \\
= & \quad < z^{(k-1)} = \begin{pmatrix} z_0 \\ \zeta_1 \end{pmatrix} > \\
& r^{(k)} - AP^{(k-2)} z_0 + \zeta_1 A p^{(k-1)} \\
= & \quad <> \\
& r^{(k)} - \left(AP^{(k-2)} z_0 + \zeta_1 (r^{(k)} - r^{(k-1)})/\alpha_{k-1}\right) \\
= & \quad <> \\
& \left(1 - \frac{\zeta_1}{\alpha_{k-1}}\right) r^{(k)} + \underbrace{\left(\frac{\zeta_1}{\alpha_{k-1}} r^{(k-1)} - AP^{(k-2)} z_0\right)}_{s^{(k)}} \\
= & \quad <> \\
& \left(1 - \frac{\zeta_1}{\alpha_{k-1}}\right) r^{(k)} + s^{(k)}.
\end{array}$$

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

$$\|r^{(k-1)} - p^{(k-1)}\|_2 = \min_{p \perp \text{Span}(p^{(0)},\ldots,p^{(k-2)})} \|r^{(k-1)} - Ap\|_2$$

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,

$$f(x^{(k)}) = \min_{x \in \text{Span}(p^{(0)},\ldots,p^{(k-1)})} f(x),$$

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=jHBK1OQE01s

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 $p^{(k)\,T} r^{(k)} = r^{(k)\,T} r^{(k)}$.

$$\begin{array}{ll} & p^{(k)\,T} r^{(k)} \\ = & < r^{(k)} + \gamma_k p^{(k-1)} > \\ & (r^{(k)} + \gamma_k p^{(k-1)})^T r^{(k)} \\ = & < \text{ distribute } > \\ & r^{(k)\,T} r^{(k)} + \gamma_k p^{(k-1)\,T} r^{(k)} \\ = & < p^{(k-1)\,T} r^{(k)} = 0 > \\ & r^{(k)\,T} r^{(k)}. \end{array}$$

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
$$r^{(k)} = r^{(k-1)} - \alpha_{k-1} A p^{(k-1)}. \tag{8.3.4}$$
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.
$$r^{(k)\,T} r^{(k)} = r^{(k)\,T} r^{(k-1)} - \alpha_{k-1} r^{(k)\,T} A p^{k-1} = -\alpha_{k-1} r^{(k)\,T} A p^{k-1}.$$

$$\begin{array}{l} p^{(k-1)\,T} A p^{(k-1)} \\ \quad = \\ (r^{(k-1)} - \gamma_{k-1} p^{(k-2)})^T A p^{(k-1)} \\ \quad = \\ r^{(k-1)\,T} A p^{(k-1)} \\ \quad = \\ r^{(k-1)\,T} (r^{(k-1)} - r^{(k)})/\alpha_{k-1} \\ \quad = \\ r^{(k-1)\,T} r^{(k-1)}/\alpha_{k-1}. \end{array}$$

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=f3rLky6mIA4

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

$$\underbrace{\tilde{L}^{-1}A\tilde{L}^{-T}}_{\tilde{A}} \quad \underbrace{\tilde{L}^T x}_{\tilde{x}} \quad = \quad \underbrace{\tilde{L}^{-1}b}_{\tilde{b}}.$$

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 nonzeroes).
- 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

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial \chi_0}(x) \\ \frac{\partial f}{\partial \chi_1}(x) \\ \vdots \\ \frac{\partial f}{\partial \chi_{n-1}}(x) \end{pmatrix}.$$

$\nabla f(x)$ 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

$$f(x) = \frac{1}{2} x^T A x - x^T b.$$

The gradient of this function equals

$$\nabla f(x) = Ax - b$$

and $\widehat{x}$ minimizes the function if and only if

$$A\widehat{x} = b.$$

If $x^{(k)}$ is an approximation to $\widehat{x}$ 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 α_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 $\widehat{x}$, $x^{(k+1)}$, 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 $p^{(0)}, \ldots, p^{(k-1)} \in \mathbb{R}^n$ such that $p^{(j)\,T} A p^{(i)} = 0$ if and only if $j \neq i$ is said to be A-conjugate. ◊

The columns of $P \in \mathbb{R}^{n \times k}$ are A-conjugate if and only if $P^T AP = D$ 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 $p^{(k)\,T} A P^{(k-1)} = 0$ and $p^{(k)\,T} r^{(k)} \neq 0$
  $\alpha_k := \frac{p^{(k)\,T} r^{(k)}}{p^{(k)\,T} A p^{(k)}}$
  $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
  $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$
  $k := k + 1$
endwhile

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 $\min_{p \perp \text{Span}(Ap^{(0)}, \ldots, Ap^{(k-1)})} \|r^{(k)} - p\|_2$
  endif
  $\alpha_k := \frac{p^{(k)\,T} r^{(k)}}{p^{(k)\,T} A p^{(k)}}$
  $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$
  $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$
endwhile

The various vectors that appear in the Conjugate Gradient Method have the following properties: If $P^{(p-1)} = \begin{pmatrix} p^{(0)} & \cdots & p^{(k-1)} \end{pmatrix}$ then

- $P^{(k-1)\,T} r^{(k)} = 0$.
- $\text{Span}(p^{(0)}, \ldots, p^{(k-1)}) = \text{Span}(r^{(0)}, \ldots, r^{(k-1)}) = \text{Span}(b, Ab, \ldots, A^{k-1}b)$.
- 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

$$\mathcal{K}_k(A, b) = \text{Span}(b, Ab, \ldots, A^{k-1}b)$$

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

To be released Wednesday April 1.

Week 10

Practical Solution of the Hermitian Eigenvalue Problem

To be released Wednesday April 8.

Week 11

The QR Algorithm: Computing the SVD

To be released Wednesday April 15.

Week 12

Attaining High Performance

To be released Wednesday April 22.

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 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.

$$A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{array} \right)$$

and

$$x = \left(\begin{array}{c} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{m-1} \end{array} \right),$$

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

$$\chi = \left[\frac{\left[-\beta + \left[\sqrt{[[\beta^2] - [4\gamma]]} \right] \right]}{2} \right],$$

truncating every expression within square brackets to three significant digits, to solve

$$\chi^2 + 25\chi + \gamma = 0$$

$$\begin{aligned} \chi &= \left[\frac{\left[-25 + \left[\sqrt{[[25^2] - [4]]} \right] \right]}{2} \right] = \left[\frac{\left[-25 + \left[\sqrt{[625 - 4]} \right] \right]}{2} \right] \\ &= \left[\frac{\left[-25 + \left[\sqrt{621} \right] \right]}{2} \right] = \left[\frac{[-25 + 24.9]}{2} \right] = \left[\frac{-0.1}{2} \right] = -0.05. \end{aligned}$$

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 cancelation: 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. □

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

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 location 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 Groiup 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 Sä kular-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 House-*

holder 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 and Maggie Myers, *Advanced Linear Algebra: Are you ready?*, http://www.cs.utexas.edu/users/flame/laff/alaff/ALAFF-pretest.html, 2020.

[40] Robert van de Geijn, Margaret Myers, and Devangi N. Parikh, *LAFF-On Programming for High Performance*, ulaff.net, 2019.

[41] Robert van de Geijn and Jerrell Watts, *SUMMA: Scalable Universal Matrix Multiplication Algorithm*, Concurrency: Practice and Experience, Volume 9, Number 4, 1997.

[42] Field G. Van Zee, *libflame: The Complete Reference*, http://www.lulu.com, 2009. [free PDF]

[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), 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.

[44] 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.

[45] 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.

[46] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM Journal on Scientific and Statistical Computing, Vol. 9, No. 1, 1988.

[47] 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.

[48] *BLAS-like Library Instantiation Software Framework*, GitHub repository.

[49] *BLIS typed interface*, https://github.com/flame/blis/blob/master/docs/BLISTypedAPI.md.

[50] 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.

[51] 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, 2019. (To appear in ACM Transactions on Mathematical Software.)

[52] 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.

[53] 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

Colophon

This article was authored in, and produced with, PreTeXt.