

Linear Algebra: Foundations to Frontiers

A Collection of Notes

on Numerical Linear Algebra

Robert A. van de Geijn

Release Date October 26, 2018

Kindly do not share this PDF

Point others to  <http://www.ulaff.net> instead

This is a work in progress

Copyright © 2014, 2015, 2016 by Robert A. van de Geijn.

10 9 8 7 6 5 4 3 2 1

All rights reserved. No part of this book may be reproduced, stored, or transmitted in any manner without the written permission of the publisher. For information, contact any of the authors.

No warranties, express or implied, are made by the publisher, authors, and their employers that the programs contained in this volume are free of error. They should not be relied on as the sole basis to solve a problem whose incorrect solution could result in injury to person or property. If the programs are employed in such a manner, it is at the user's own risk and the publisher, authors, and their employers disclaim all liability for such misuse.

Trademarked names may be used in this book without the inclusion of a trademark symbol. These names are used in an editorial context only; no infringement of trademark is intended.

Library of Congress Cataloging-in-Publication Data not yet available

Draft Edition, November 2014, 2015, 2016

This "Draft Edition" allows this material to be used while we sort out through what mechanism we will publish the book.

Contents

Preface	xi
0 Notes on Setting Up	1
0.1 Opening Remarks	1
0.1.1 Launch	1
0.1.2 Outline	2
0.1.3 What You Will Learn	3
0.2 Setting Up to Learn	4
0.2.1 How to Navigate These Materials	4
0.2.2 Setting Up Your Computer	4
0.3 MATLAB	4
0.3.1 Why MATLAB	4
0.3.2 Installing MATLAB	4
0.3.3 MATLAB Basics	4
0.4 Wrapup	7
0.4.1 Additional Homework	7
0.4.2 Summary	7
I Basics	9
1 Notes on Simple Vector and Matrix Operations	11
1.1 Opening Remarks	11
1.1.1 Launch	11
1.1.2 Outline	12
1.1.3 What You Will Learn	13
1.2 Notation	14
1.3 (Hermitian) Transposition	15
1.3.1 Conjugating a complex scalar	15
1.3.2 Conjugate of a vector	15
1.3.3 Conjugate of a matrix	15
1.3.4 Transpose of a vector	15
1.3.5 Hermitian transpose of a vector	16

1.3.6	Transpose of a matrix	16
1.3.7	Hermitian transpose (adjoint) of a matrix	16
1.3.8	Exercises	16
1.4	Vector-vector Operations	18
1.4.1	Scaling a vector (scal)	18
1.4.2	Scaled vector addition (axpy)	20
1.4.3	Dot (inner) product (dot)	21
1.5	Matrix-vector Operations	22
1.5.1	Matrix-vector multiplication (product)	22
1.5.2	Rank-1 update	25
1.6	Matrix-matrix multiplication (product)	28
1.6.1	Element-by-element computation	28
1.6.2	Via matrix-vector multiplications	29
1.6.3	Via row-vector times matrix multiplications	31
1.6.4	Via rank-1 updates	32
1.7	Enrichments	33
1.7.1	The Basic Linear Algebra Subprograms (BLAS)	33
1.8	Wrapup	34
1.8.1	Additional exercises	34
1.8.2	Summary	34

=■

2	Notes on Vector and Matrix Norms	37
2.1	Opening Remarks	37
2.1.1	Launch	37
2.1.2	Outline	38
2.1.3	What You Will Learn	39
2.2	Absolute Value	40
2.3	Vector Norms	40
2.3.1	Vector 2-norm (Euclidean length)	40
2.3.2	Vector 1-norm	42
2.3.3	Vector ∞ -norm (infinity norm)	42
2.3.4	Vector p -norm	43
2.4	Matrix Norms	43
2.4.1	Frobenius norm	43
2.4.2	Induced matrix norms	44
2.4.3	Special cases used in practice	45
2.4.4	Discussion	47
2.4.5	Submultiplicative norms	47
2.5	An Application to Conditioning of Linear Systems	48
2.6	Equivalence of Norms	49
2.7	Enrichments	50
2.7.1	Practical computation of the vector 2-norm	50
2.8	Wrapup	50
2.8.1	Additional exercises	50
2.8.2	Summary	53

II Orthogonality	55
3 Notes on Orthogonality and the Singular Value Decomposition	57
Video	57
3.1 Opening Remarks	57
3.1.1 Launch: Orthogonal projection and its application	57
3.1.2 Outline	66
3.1.3 What you will learn	67
3.2 Orthogonality and Unitary Matrices	68
3.3 Toward the SVD	71
3.4 <i>The Theorem</i>	73
3.5 Geometric Interpretation	73
3.6 Consequences of the SVD Theorem	77
3.7 Projection onto the Column Space	81
3.8 Low-rank Approximation of a Matrix	83
3.9 An Application	84
3.10 SVD and the Condition Number of a Matrix	86
3.11 An Algorithm for Computing the SVD?	87
3.12 Wrapup	88
3.12.1 Additional exercises	88
3.12.2 Summary	88
=■	
4 Notes on Gram-Schmidt QR Factorization	89
4.1 Opening Remarks	89
4.1.1 Launch	89
4.1.2 Outline	90
4.1.3 What you will learn	91
4.2 Classical Gram-Schmidt (CGS) Process	92
4.3 Modified Gram-Schmidt (MGS) Process	97
4.4 In Practice, MGS is More Accurate	101
4.5 Cost	103
4.5.1 Cost of CGS	104
4.5.2 Cost of MGS	104
4.6 Wrapup	105
4.6.1 Additional exercises	105
4.6.2 Summary	105
5 Notes on the FLAME APIs	107
Video	107
5.0.1 Outline	108
5.1 Motivation	108
5.2 Install FLAME@lab	108
5.3 An Example: Gram-Schmidt Orthogonalization	108
5.3.1 The Spark Webpage	108
5.3.2 Implementing CGS with FLAME@lab	109

5.3.3	Editing the code skeleton	111
5.3.4	Testing	112
5.4	Implementing the Other Algorithms	113
6	Notes on Householder QR Factorization	115
6.1	Opening Remarks	115
6.1.1	Launch	115
6.1.2	Outline	117
6.1.3	What you will learn	118
6.2	Householder Transformations (Reflectors)	119
6.2.1	The general case	119
6.2.2	As implemented for the Householder QR factorization (real case)	120
6.2.3	The complex case (optional)	121
6.2.4	A routine for computing the Householder vector	122
6.3	Householder QR Factorization	123
6.4	Forming Q	126
6.5	Applying Q^H	131
6.6	Blocked Householder QR Factorization	133
6.6.1	The UT transform: Accumulating Householder transformations	133
6.6.2	The WY transform	135
6.6.3	A blocked algorithm	135
6.6.4	Variations on a theme	139
6.7	Enrichments	142
6.8	Wrapup	142
6.8.1	Additional exercises	142
6.8.2	Summary	146
7	Notes on Rank Revealing Householder QR Factorization	147
7.1	Opening Remarks	147
7.1.1	Launch	147
7.1.2	Outline	148
7.1.3	What you will learn	149
7.2	Modifying MGS to Compute QR Factorization with Column Pivoting	150
7.3	Unblocked Householder QR Factorization with Column Pivoting	151
7.3.1	Basic algorithm	151
7.3.2	Alternative unblocked Householder QR factorization with column pivoting	151
7.4	Blocked HQRP	155
7.5	Computing Q	155
7.6	Enrichments	155
7.6.1	QR factorization with randomization for column pivoting	155
7.7	Wrapup	158
7.7.1	Additional exercises	158
7.7.2	Summary	158
8	Notes on Solving Linear Least-Squares Problems	159
8.0.1	Launch	159

8.0.2	Outline	160
8.0.3	What you will learn	161
8.1	The Linear Least-Squares Problem	162
8.2	Method of Normal Equations	162
8.3	Solving the LLS Problem Via the QR Factorization	163
8.3.1	Simple derivation of the solution	163
8.3.2	Alternative derivation of the solution	164
8.4	Via Householder QR Factorization	165
8.5	Via the Singular Value Decomposition	166
8.5.1	Simple derivation of the solution	166
8.5.2	Alternative derivation of the solution	167
8.6	What If A Does Not Have Linearly Independent Columns?	167
8.7	Exercise: Using the the <i>LQ</i> factorization to solve underdetermined systems	174
8.8	Wrapup	174
8.8.1	Additional exercises	174
8.8.2	Summary	174
9	Notes on the Condition of a Problem	177
9.1	Opening Remarks	177
9.1.1	Launch	177
Video	177
9.1.2	Outline	178
9.1.3	What you will learn	179
9.2	Notation	180
9.3	The Prototypical Example: Solving a Linear System	180
9.4	Condition Number of a Rectangular Matrix	184
9.5	Why Using the Method of Normal Equations Could be Bad	185
9.6	Why Multiplication with Unitary Matrices is a Good Thing	186
9.7	Balancing a Matrix	186
9.8	Wrapup	188
9.8.1	Additional exercises	188
9.9	Wrapup	189
9.9.1	Additional exercises	189
9.9.2	Summary	189
10	Notes on the Stability of an Algorithm	191
10.0.1	Launch	191
10.0.2	Outline	192
10.0.3	What you will learn	193
10.1	Motivation	194
10.2	Floating Point Numbers	195
10.3	Notation	197
10.4	Floating Point Computation	197
10.4.1	Model of floating point computation	197
10.4.2	Stability of a numerical algorithm	198
10.4.3	Absolute value of vectors and matrices	198

10.5	Stability of the Dot Product Operation	199
10.5.1	An algorithm for computing DOT	199
10.5.2	A simple start	199
10.5.3	Preparation	201
10.5.4	Target result	203
10.5.5	A proof in traditional format	204
10.5.6	A weapon of math induction for the war on (backward) error (optional)	204
10.5.7	Results	206
10.6	Stability of a Matrix-Vector Multiplication Algorithm	207
10.6.1	An algorithm for computing GEMV	207
10.6.2	Analysis	207
10.7	Stability of a Matrix-Matrix Multiplication Algorithm	209
10.7.1	An algorithm for computing GEMM	209
10.7.2	Analysis	209
10.7.3	An application	210
10.8	Wrapup	210
10.8.1	Additional exercises	210
10.8.2	Summary	210

11 Notes on Performance	211
--------------------------------	------------

12 Notes on Gaussian Elimination and LU Factorization	213	
12.1	Opening Remarks	213
12.1.1	Launch	213
12.1.2	Outline	214
12.1.3	What you will learn	216
12.2	Definition and Existence	217
12.3	LU Factorization	217
12.3.1	First derivation	217
12.3.2	Gauss transforms	218
12.3.3	Cost of LU factorization	220
12.4	LU Factorization with Partial Pivoting	221
12.4.1	Permutation matrices	221
12.4.2	The algorithm	223
12.5	Proof of Theorem 12.3	228
12.6	LU with Complete Pivoting	230
12.7	Solving $Ax = y$ Via the LU Factorization with Pivoting	231
12.8	Solving Triangular Systems of Equations	231
12.8.1	$Lz = y$	231
12.8.2	$Ux = z$	234
12.9	Other LU Factorization Algorithms	234
12.9.1	Variant 1: Bordered algorithm	239
12.9.2	Variant 2: Left-looking algorithm	239
12.9.3	Variant 3: Up-looking variant	240
12.9.4	Variant 4: Crout variant	240
12.9.5	Variant 5: Classical LU factorization	241

12.9.6	All algorithms	241
12.9.7	Formal derivation of algorithms	241
12.10	Numerical Stability Results	243
12.11	Is LU with Partial Pivoting Stable?	244
12.12	Blocked Algorithms	244
12.12.1	Blocked classical LU factorization (Variant 5)	244
12.12.2	Blocked classical LU factorization with pivoting (Variant 5)	247
12.13	Variations on a Triple-Nested Loop	248
12.14	Inverting a Matrix	249
12.14.1	Basic observations	249
12.14.2	Via the LU factorization with pivoting	249
12.14.3	Gauss-Jordan inversion	250
12.14.4	(Almost) never, ever invert a matrix	251
12.15	Efficient Condition Number Estimation	252
12.15.1	The problem	252
12.15.2	Insights	252
12.15.3	A simple approach	253
12.15.4	Discussion	255
12.16	Wrapup	256
12.16.1	Additional exercises	256
12.16.2	Summary	256

13 Notes on Cholesky Factorization	257	
13.1	Opening Remarks	257
13.1.1	Launch	257
13.1.2	Outline	258
13.1.3	What you will learn	259
13.2	Definition and Existence	260
13.3	Application	260
13.4	An Algorithm	261
13.5	Proof of the Cholesky Factorization Theorem	262
13.6	Blocked Algorithm	263
13.7	Alternative Representation	263
13.8	Cost	266
13.9	Solving the Linear Least-Squares Problem via the Cholesky Factorization	267
13.10	Other Cholesky Factorization Algorithms	267
13.11	Implementing the Cholesky Factorization with the (Traditional) BLAS	269
13.11.1	What are the BLAS?	269
13.11.2	A simple implementation in Fortran	272
13.11.3	Implementation with calls to level-1 BLAS	272
13.11.4	Matrix-vector operations (level-2 BLAS)	272
13.11.5	Matrix-matrix operations (level-3 BLAS)	276
13.11.6	Impact on performance	276
13.12	Alternatives to the BLAS	277
13.12.1	The FLAME/C API	277
13.12.2	BLIS	277

13.13	Wrapup	278
13.13.1	Additional exercises	278
13.13.2	Summary	278
14 Notes on Eigenvalues and Eigenvectors		279
Video		279
14.0.1	Outline	280
14.1	Definition	280
14.2	The Schur and Spectral Factorizations	283
14.3	Relation Between the SVD and the Spectral Decomposition	285
15 Notes on the Power Method and Related Methods		287
Video		287
15.0.1	Outline	288
15.1	The Power Method	288
15.1.1	First attempt	289
15.1.2	Second attempt	289
15.1.3	Convergence	290
15.1.4	Practical Power Method	293
15.1.5	The Rayleigh quotient	294
15.1.6	What if $ \lambda_0 \geq \lambda_1 $?	294
15.2	The Inverse Power Method	294
15.3	Rayleigh-quotient Iteration	295
16 Notes on the QR Algorithm and other Dense Eigensolvers		299
Video		299
16.0.1	Outline	300
16.1	Preliminaries	301
16.2	Subspace Iteration	301
16.3	The QR Algorithm	306
16.3.1	A basic (unshifted) QR algorithm	306
16.3.2	A basic shifted QR algorithm	306
16.4	Reduction to Tridiagonal Form	308
16.4.1	Householder transformations (reflectors)	308
16.4.2	Algorithm	309
16.5	The QR algorithm with a Tridiagonal Matrix	311
16.5.1	Givens' rotations	311
16.6	QR Factorization of a Tridiagonal Matrix	312
16.7	The Implicitly Shifted QR Algorithm	314
16.7.1	Upper Hessenberg and tridiagonal matrices	314
16.7.2	The Implicit Q Theorem	315
16.7.3	The Francis QR Step	316
16.7.4	A complete algorithm	318
16.8	Further Reading	322
16.8.1	More on reduction to tridiagonal form	322
16.8.2	Optimizing the tridiagonal QR algorithm	322

16.9	Other Algorithms	322
16.9.1	Jacobi's method for the symmetric eigenvalue problem	322
16.9.2	Cuppen's Algorithm	325
16.9.3	The Method of Multiple Relatively Robust Representations (MRRR)	325
16.10	The Nonsymmetric QR Algorithm	325
16.10.1	A variant of the Schur decomposition	325
16.10.2	Reduction to upperHessenberg form	327
16.10.3	The implicitly double-shifted QR algorithm	329
17 Notes on the Method of Relatively Robust Representations (MRRR)		331
17.0.1	Outline	332
17.1	MRRR, from 35,000 Feet	332
17.2	Cholesky Factorization, Again	335
17.3	The LDL^T Factorization	335
17.4	The UDU^T Factorization	337
17.5	The UDU^T Factorization	340
17.6	The Twisted Factorization	340
17.7	Computing an Eigenvector from the Twisted Factorization	343
18 Notes on Computing the SVD of a Matrix		345
18.0.1	Outline	346
18.1	Background	346
18.2	Reduction to Bidiagonal Form	346
18.3	The QR Algorithm with a Bidiagonal Matrix	350
18.4	Putting it all together	352
Answers		357
1. Notes on Simple Vector and Matrix Operations		357
2. Notes on Vector and Matrix Norms		364
3. Notes on Orthogonality and the SVD		374
4. Notes on Gram-Schmidt QR Factorization		382
6. Notes on Householder QR Factorization		385
8. Notes on Solving Linear Least-squares Problems (Answers)		395
8. Notes on the Condition of a Problem		397
9. Notes on the Stability of an Algorithm		401
10. Notes on Performance		406
11. Notes on Gaussian Elimination and LU Factorization		407
12. Notes on Cholesky Factorization		416
13. Notes on Eigenvalues and Eigenvectors		421
14. Notes on the Power Method and Related Methods		426
16. Notes on the Symmetric QR Algorithm		427
17. Notes on the Method of Relatively Robust Representations		431
18. Notes on Computing the SVD		441
A How to Download		443
B LAFF Routines (FLAME@lab)		445

Preface

This document was created over the course of many years, as I periodically taught an introductory course titled “Numerical Analysis: Linear Algebra,” cross-listed in the departments of Computer Science, Math, and Statistics and Data Sciences (SDS), as well as the Computational Science Engineering Mathematics (CSEM) graduate program.

Over the years, my colleagues and I have used many different books for this course: *Matrix Computations* by Golub and Van Loan [22], *Fundamentals of Matrix Computation* by Watkins [47], *Numerical Linear Algebra* by Trefethen and Bau [38], and *Applied Numerical Linear Algebra* by Demmel [12]. All are books with tremendous strengths and depth. Nonetheless, I found myself writing materials for my students that add insights that are often drawn from our own research experiences in the field. These became a series of notes that are meant to supplement rather than replace any of the mentioned books.

Fundamental to our exposition is the *FLAME notation* [25, 40], which we use to present algorithms hand-in-hand with theory. For example, in Figure 1 (left), we present a commonly encountered LU factorization algorithm, which we will see performs exactly the same computations as does Gaussian elimination. By abstracting away from the detailed indexing that are required to implement an algorithm in, for example, Matlab’s M-script language, the reader can focus on the mathematics that justifies the algorithm rather than the indices used to express the algorithm. The algorithms can be easily translated into code with the help of a FLAME Application Programming Interface (API). Such interfaces have been created for C, M-script, and Python, to name a few [25, 6, 30]. In Figure 1 (right), we show the LU factorization algorithm implemented with the FLAME@lab API for M-script. The C API is used extensively in our implementation of the libflame dense linear algebra library [41, 42] for sequential and shared-memory architectures and the notation also inspired the API used to implement the Elemental dense linear algebra library [31] that targets distributed memory architectures. Thus, the reader is exposed to the abstractions that experts use to translate algorithms to high-performance software.

These notes may at times appear to be a vanity project, since I often point the reader to our research papers for a glimpse at the cutting edge of the field. The fact is that over the last two decades, we have helped further the understanding of many of the topics discussed in a typical introductory course on numerical linear algebra. Since we use the FLAME notation in many of our papers, they should be relatively easy to read once one familiarizes oneself with these notes. Let’s be blunt: these notes do not do the field justice when it comes to also giving a historic perspective. For that, we recommend any of the above mentioned texts, or the wonderful books by G.W. Stewart [36, 37]. This is yet another reason why they should be used to supplement other texts.

Algorithm: $A := L \setminus U = LU(A)$

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

where A_{TL} is 0×0

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

Repartition

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

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

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

Continue with

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

endwhile

```

function [ A_out ] = LU_unb_var4( A )
    [ ATL, ATR, ...,
      ABL, ABR ] = FLA_Part_2x2( A, ...
                                    0, 0, 'FLA_TL' );
    while ( size( ATL, 1 ) < size( A, 1 ) )
        [ A00, a01, A02, ...
          a10t, alphall, a12t, ...
          A20, a21, A22 ] = ...
            FLA_Repart_2x2_to_3x3( ...
                ATL, ATR, ...
                ABL, ABR, 1, 1, 'FLA_BR' );
        %-----
        a21 = a21 / alphall;
        A22 = A22 - a21 * a12t;
        %-----
        [ ATL, ATR, ...
          ABL, ABR ] = ...
            FLA_Cont_with_3x3_to_2x2( ...
                A00, a01, A02, ...
                a10t, alphall, a12t, ...
                A20, a21, A22, 'FLA_TL' );
    end
    A_out = [ ATL, ATR
              ABL, ABR ];
    return

```

Figure 1: LU factorization represented with the FLAME notation and the FLAME@lab API.

The order of the chapters should not be taken too seriously. They were initially written as separate, relatively self-contained notes. The sequence on which I settled roughly follows the order that the topics are encountered in *Numerical Linear Algebra* by Trefethen and Bau [38]. The reason is the same as the one they give: It introduces orthogonality and the Singular Value Decomposition early on, leading with important material that many students will not yet have seen in the undergraduate linear algebra classes they took. However, one could just as easily rearrange the chapters so that one starts with a more traditional topic: solving dense linear systems.

The notes frequently refer the reader to another resource of ours titled *Linear Algebra: Foundations to Frontiers - Notes to LAFF With* (LAFF Notes) [30]. This is a 900+ page document with more than 270 videos that was created for the Massive Open Online Course (MOOC) *Linear Algebra: Foundations to Frontiers* (LAFF), offered by the edX platform. That course provides an appropriate undergraduate background for these notes.

I (tried to) video tape my lectures during Fall 2014. Unlike the many short videos that we created for the Massive Open Online Course (MOOC) titled “Linear Algebra: Foundations to Frontiers” that are now part of the notes for that course, I simply set up a camera, taped the entire lecture, spent minimal time editing, and uploaded the result for the world to see. Worse, I did not prepare particularly well for the lectures, other than feverishly writing these notes in the days prior to the presentation. Sometimes, I forgot to turn on the microphone and/or the camera. Sometimes the memory of the camcorder was full. Sometimes I

forgot to shave. Often I forgot to comb my hair. You are welcome to watch, but don't expect too much!

One should consider this a living document. As time goes on, I will be adding more material. Ideally, people with more expertise than I have on, for example, solving sparse linear systems will contributed notes of their own.

Acknowledgments

These notes use notation and tools developed by the FLAME project at The University of Texas at Austin (USA), Universidad Jaume I (Spain), and RWTH Aachen University (Germany). This project involves a large and ever expanding number of very talented people, to whom I am indebted. Over the years, it has been supported by a number of grants from the National Science Foundation and industry. The most recent and most relevant funding came from NSF Award ACI-1148125 titled “SI2-SSI: A Linear Algebra Software Infrastructure for Sustained Innovation in Computational Chemistry and other Sciences”¹

In Texas, behind every successful man there is a woman who really pulls the strings. For more than thirty years, my research, teaching, and other pedagogical activities have been greatly influenced by my wife, Dr. Maggie Myers. For parts of these notes that are particularly successful, the credit goes to her. Where they fall short, the blame is all mine!

¹ Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Chapter 0

Notes on Setting Up

0.1 Opening Remarks

0.1.1 Launch

0.1.2 Outline

0.1	Opening Remarks	1
0.1.1	Launch	1
0.1.2	Outline	2
0.1.3	What You Will Learn	3
0.2	Setting Up to Learn	4
0.2.1	How to Navigate These Materials	4
0.2.2	Setting Up Your Computer	4
0.3	MATLAB	4
0.3.1	Why MATLAB	4
0.3.2	Installing MATLAB	4
0.3.3	MATLAB Basics	4
0.4	Wrapup	7
0.4.1	Additional Homework	7
0.4.2	Summary	7

0.1.3 What You Will Learn

0.2 Setting Up to Learn

0.2.1 How to Navigate These Materials

To be filled in.

0.2.2 Setting Up Your Computer

It helps if we all set up our environment in a consistent fashion. To achieve this, follow the following steps:

- Download the file [LAFF-NLA.zip](#) to a directory on your computer. (I chose to place it in my home directory `rvdg`).
- Unzip this file. This will create the directory LAFF-NLA.

The above steps create a directory structure with various files as illustrated in Figure 1.

You will want to put this PDF in that directory in the indicated place! Opening it with Acrobat Reader will ensure that hyperlinks work properly.

This is a work in progress. When instructed, you will want to load new versions of this PDF, replacing the existing one. There will also be other files you will be asked to place in various subdirectories.

0.3 MATLAB

0.3.1 Why MATLAB

We use **MATLAB** as a tool because it was invented to support learning about matrix computations. You will find that the syntax of the language used by MATLAB, called M-script, very closely resembles the mathematical expressions in linear algebra.

Those not willing to invest in MATLAB will want to consider **GNU Octave** instead.

0.3.2 Installing MATLAB

All students at UT-Austin can get a free MATLAB license. Let's discuss on Piazza where to find it. Everyone else: you can find instructions on how to purchase and install MATLAB from **MathWorks**.

0.3.3 MATLAB Basics

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 clicking "Launch Tutorial".

HOWEVER, you need very little familiarity with MATLAB in order to learn what we want you to learn about how abstraction in mathematics is linked to abstraction in algorithms. So, you could just skip

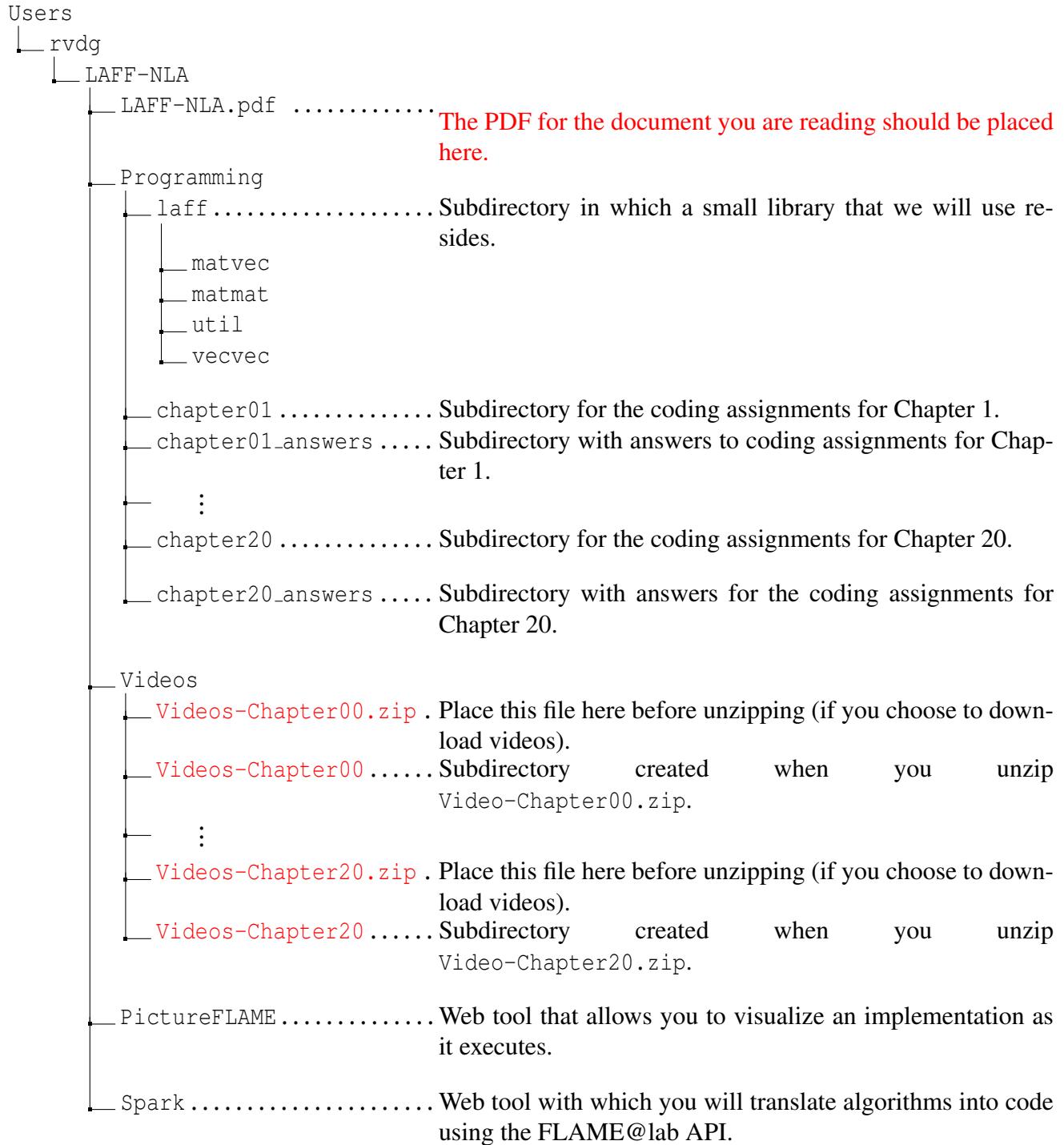


Figure 1: Directory structure for each of the steps (illustrated for Step 1 in subdirectory step1).

these tutorials altogether, and come back to them if you find you want to know more about MATLAB and its programming language (M-script).

What is MATLAB?



What is MATLAB?
Created by MathWorks for
Linear Algebra: Foundations to Frontiers
MathWorks

YouTube
Downloaded Video

A screenshot of a video player window. The video title is "What is MATLAB?". Below the title, it says "Created by MathWorks for Linear Algebra: Foundations to Frontiers". The MathWorks logo is visible. A progress bar at the bottom shows 0:01 / 1:23. To the right of the video frame, there are two download options: "YouTube" (red icon) and "Downloaded Video" (blue icon).

The MATLAB Environment

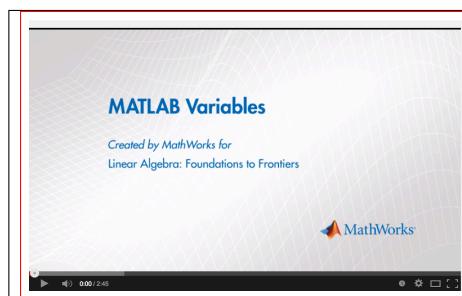


The MATLAB Environment
Created by MathWorks for
Linear Algebra: Foundations to Frontiers
MathWorks

YouTube
Downloaded Video

A screenshot of a video player window. The video title is "The MATLAB Environment". Below the title, it says "Created by MathWorks for Linear Algebra: Foundations to Frontiers". The MathWorks logo is visible. A progress bar at the bottom shows 0:00 / 1:14. To the right of the video frame, there are two download options: "YouTube" (red icon) and "Downloaded Video" (blue icon).

MATLAB Variables

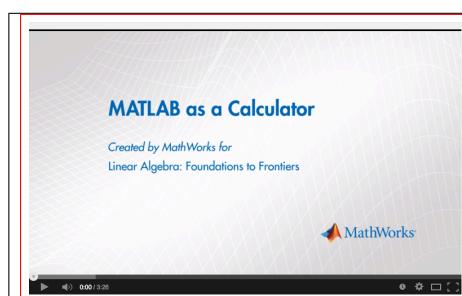


MATLAB Variables
Created by MathWorks for
Linear Algebra: Foundations to Frontiers
MathWorks

YouTube
Downloaded Video

A screenshot of a video player window. The video title is "MATLAB Variables". Below the title, it says "Created by MathWorks for Linear Algebra: Foundations to Frontiers". The MathWorks logo is visible. A progress bar at the bottom shows 0:00 / 2:48. To the right of the video frame, there are two download options: "YouTube" (red icon) and "Downloaded Video" (blue icon).

MATLAB as a Calculator

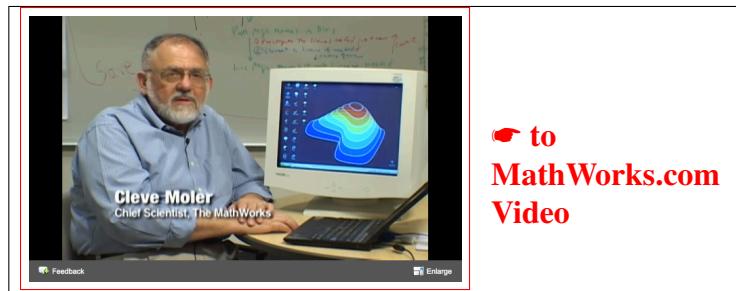


MATLAB as a Calculator
Created by MathWorks for
Linear Algebra: Foundations to Frontiers
MathWorks

YouTube
Downloaded Video

A screenshot of a video player window. The video title is "MATLAB as a Calculator". Below the title, it says "Created by MathWorks for Linear Algebra: Foundations to Frontiers". The MathWorks logo is visible. A progress bar at the bottom shows 0:00 / 3:28. To the right of the video frame, there are two download options: "YouTube" (red icon) and "Downloaded Video" (blue icon).

The Origins of MATLAB



0.4 Wrapup

0.4.1 Additional Homework

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

0.4.2 Summary

You will see that we develop a lot of the theory behind the various topics in linear algebra via a sequence of homework exercises. At the end of each week, we summarize theorems and insights for easy reference.

Chapter **1**

Notes on Simple Vector and Matrix Operations

1.1 Opening Remarks

1.1.1 Launch

We assume that the reader is quite familiar with vectors, linear transformations, and matrices. If not, we suggest the reader reviews the first five weeks of

[Linear Algebra: Foundations to Frontiers - Notes to LAFF With \[30\].](#)

Since undergraduate courses tend to focus on real valued matrices and vectors, we mostly focus on the case where they are complex valued as we review.

Read disclaimer regarding the videos in the preface!

The below first video is actually for the first lecture of the semester.

- 👉 [YouTube](#)
- 👉 [Download from UT Box](#)
- 👉 [View After Local Download](#)

(For help on viewing, see Appendix A.)

(In the downloadable version, the lecture starts about 27 seconds into the video. I was still learning how to do the editing...) The video for this particular note didn't turn out, so you will want to read instead. As I pointed out in the preface: these videos aren't refined by any measure!

1.1.2 Outline

1.1	Opening Remarks	11
1.1.1	Launch	11
1.1.2	Outline	12
1.1.3	What You Will Learn	13
1.2	Notation	14
1.3	(Hermitian) Transposition	15
1.3.1	Conjugating a complex scalar	15
1.3.2	Conjugate of a vector	15
1.3.3	Conjugate of a matrix	15
1.3.4	Transpose of a vector	15
1.3.5	Hermitian transpose of a vector	16
1.3.6	Transpose of a matrix	16
1.3.7	Hermitian transpose (adjoint) of a matrix	16
1.3.8	Exercises	16
1.4	Vector-vector Operations	18
1.4.1	Scaling a vector (<code>scal</code>)	18
1.4.2	Scaled vector addition (<code>axpy</code>)	20
1.4.3	Dot (inner) product (<code>dot</code>)	21
1.5	Matrix-vector Operations	22
1.5.1	Matrix-vector multiplication (<code>product</code>)	22
1.5.2	Rank-1 update	25
1.6	Matrix-matrix multiplication (<code>product</code>)	28
1.6.1	Element-by-element computation	28
1.6.2	Via matrix-vector multiplications	29
1.6.3	Via row-vector times matrix multiplications	31
1.6.4	Via rank-1 updates	32
1.7	Enrichments	33
1.7.1	The Basic Linear Algebra Subprograms (BLAS)	33
1.8	Wrapup	34
1.8.1	Additional exercises	34
1.8.2	Summary	34

1.1.3 What You Will Learn

1.2 Notation

Throughout our notes we will adopt notation popularized by Alston Householder. As a rule, we will use lower case Greek letters (α, β , etc.) to denote scalars. For vectors, we will use lower case Roman letters (a, b , etc.). Matrices are denoted by upper case Roman letters (A, B , etc.).

If $x \in \mathbb{C}^n$, and $A \in \mathbb{C}^{m \times n}$ then we expose the elements of x and A as

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \quad \text{and} \quad A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix}.$$

If vectors x is partitioned into N subvectors, we may denote this by

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

It is possible for a subvector to be of size zero (no elements) or one (a scalar). If we want to emphasize that a specific subvector is a scalar, then we may choose to use a lower case Greek letter for that scalar, as in

$$x = \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}.$$

We will see frequent examples where a matrix, $A \in \mathbb{C}^{m \times n}$, is partitioned into columns or rows:

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

Here we add the $\widehat{\cdot}$ to be able to distinguish between the identifiers for the columns and rows. When from context it is obvious whether we refer to a row or column, we may choose to skip the $\widehat{\cdot}$. Sometimes, we partition matrices into submatrices:

$$A = \left(\begin{array}{c|c|c|c} A_0 & A_1 & \cdots & A_{N-1} \end{array} \right) = \left(\begin{array}{c} \widehat{A}_0 \\ \hline \widehat{A}_1 \\ \vdots \\ \hline \widehat{A}_{M-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & & \vdots \\ \hline A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{array} \right).$$

1.3 (Hermitian) Transposition

1.3.1 Conjugating a complex scalar

Recall that if $\alpha = \alpha_r + i\alpha_c$ then its (complex) conjugate is given by

$$\bar{\alpha} = \alpha_r - i\alpha_c$$

and its length (absolute value) by

$$|\alpha| = |\alpha_r + i\alpha_c| = \sqrt{\alpha_r^2 + \alpha_c^2} = \sqrt{(\alpha_r + i\alpha_c)(\alpha_r - i\alpha_c)} = \sqrt{\alpha\bar{\alpha}} = \sqrt{\bar{\alpha}\alpha} = |\bar{\alpha}|.$$

1.3.2 Conjugate of a vector

The (*complex*) conjugate of x is given by

$$\bar{x} = \begin{pmatrix} \bar{x}_0 \\ \bar{x}_1 \\ \vdots \\ \bar{x}_{n-1} \end{pmatrix}.$$

1.3.3 Conjugate of a matrix

The (*complex*) conjugate of A is given by

$$\bar{A} = \begin{pmatrix} \bar{a}_{0,0} & \bar{a}_{0,1} & \cdots & \bar{a}_{0,n-1} \\ \bar{a}_{1,0} & \bar{a}_{1,1} & \cdots & \bar{a}_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{a}_{m-1,0} & \bar{a}_{m-1,1} & \cdots & \bar{a}_{m-1,n-1} \end{pmatrix}.$$

1.3.4 Transpose of a vector

The *transpose* of x is given by

$$x^T = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}^T = (x_0 | x_1 | \cdots | x_{n-1}).$$

Notice that transposing a (column) vector rearranges its elements to make a row vector.

1.3.5 Hermitian transpose of a vector

The *Hermitian transpose* of x is given by

$$x^H (= x^c) = (\bar{x})^T = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T = \begin{pmatrix} \bar{\chi}_0 \\ \bar{\chi}_1 \\ \vdots \\ \bar{\chi}_{n-1} \end{pmatrix}^T = \left(\begin{array}{c|c|c|c} \bar{\chi}_0 & \bar{\chi}_1 & \cdots & \bar{\chi}_{n-1} \end{array} \right).$$

In other words, taking the conjugate of a vector is equivalent to taking the conjugate of each of its elements.

1.3.6 Transpose of a matrix

The *transpose* of A is given by

$$A^T = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix}^T = \begin{pmatrix} \alpha_{0,0} & \alpha_{1,0} & \cdots & \alpha_{m-1,0} \\ \alpha_{0,1} & \alpha_{1,1} & \cdots & \alpha_{m-1,1} \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_{0,n-1} & \alpha_{1,n-1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix}.$$

In other words, taking the conjugate of a matrix is equivalent to taking the conjugate of each of its elements.

1.3.7 Hermitian transpose (adjoint) of a matrix

The *Hermitian transpose* of A is given by

$$A^H = \bar{A}^T = \left(\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} \right)^T = \begin{pmatrix} \bar{\alpha}_{0,0} & \bar{\alpha}_{1,0} & \cdots & \bar{\alpha}_{m-1,0} \\ \bar{\alpha}_{0,1} & \bar{\alpha}_{1,1} & \cdots & \bar{\alpha}_{m-1,1} \\ \vdots & \vdots & \cdots & \vdots \\ \bar{\alpha}_{0,n-1} & \bar{\alpha}_{1,n-1} & \cdots & \bar{\alpha}_{m-1,n-1} \end{pmatrix}.$$

In various texts you may see A^H denoted by A^c or A^* instead.

1.3.8 Exercises

Homework 1.1 Partition A

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

Convince yourself that the following hold:

$$\bullet \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)^T = \begin{pmatrix} \overline{a_0^T} \\ \overline{a_1^T} \\ \vdots \\ \overline{a_{n-1}^T} \end{pmatrix}.$$

$$\bullet \begin{pmatrix} \overline{\widehat{a}_0^T} \\ \overline{\widehat{a}_1^T} \\ \vdots \\ \overline{\widehat{a}_{n-1}^T} \end{pmatrix}^T = \left(\begin{array}{c|c|c|c} \widehat{a}_0 & \widehat{a}_1 & \cdots & \widehat{a}_{n-1} \end{array} \right).$$

$$\bullet \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)^H = \begin{pmatrix} \overline{a_0^H} \\ \overline{a_1^H} \\ \vdots \\ \overline{a_{n-1}^H} \end{pmatrix}.$$

$$\bullet \begin{pmatrix} \overline{\widehat{a}_0^T} \\ \overline{\widehat{a}_1^T} \\ \vdots \\ \overline{\widehat{a}_{n-1}^T} \end{pmatrix}^H = \left(\begin{array}{c|c|c|c} \overline{\widehat{a}_0} & \overline{\widehat{a}_1} & \cdots & \overline{\widehat{a}_{n-1}} \end{array} \right).$$

SEE ANSWER

Homework 1.2 Partition x into subvectors:

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

Convince yourself that the following hold:

$$\bullet \bar{x} = \begin{pmatrix} \overline{x_0} \\ \overline{x_1} \\ \vdots \\ \overline{x_{N-1}} \end{pmatrix}.$$

$$\bullet x^T = \left(\begin{array}{c|c|c|c} x_0^T & x_1^T & \cdots & x_{N-1}^T \end{array} \right).$$

- $x^H = \left(\begin{array}{c|c|c|c} x_0^H & x_1^H & \cdots & x_{N-1}^H \end{array} \right).$

SEE ANSWER

Homework 1.3 Partition A

$$A = \begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{pmatrix},$$

where $A_{i,j} \in \mathbb{C}^{m_i \times n_i}$. Here $\sum_{i=0}^{M-1} m_i = m$ and $\sum_{j=0}^{N-1} n_i = n$.

Convince yourself that the following hold:

- $\begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{pmatrix}^T = \begin{pmatrix} A_{0,0}^T & A_{1,0}^T & \cdots & A_{M-1}^T \\ A_{0,1}^T & A_{1,1}^T & \cdots & A_{M-1,1}^T \\ \vdots & \vdots & \cdots & \vdots \\ A_{0,N-1}^T & A_{1,N-1}^T & \cdots & A_{M-1,N-1}^T \end{pmatrix}.$
- $\begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{pmatrix} = \begin{pmatrix} \overline{A_{0,0}} & \overline{A_{0,1}} & \cdots & \overline{A_{0,N-1}} \\ \overline{A_{1,0}} & \overline{A_{1,1}} & \cdots & \overline{A_{1,N-1}} \\ \vdots & \vdots & \cdots & \vdots \\ \overline{A_{M-1,0}} & \overline{A_{M-1,1}} & \cdots & \overline{A_{M-1,N-1}} \end{pmatrix}.$
- $\begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{pmatrix}^H = \begin{pmatrix} A_{0,0}^H & A_{1,0}^H & \cdots & A_{M-1}^H \\ A_{0,1}^H & A_{1,1}^H & \cdots & A_{M-1,1}^H \\ \vdots & \vdots & \cdots & \vdots \\ A_{0,N-1}^H & A_{1,N-1}^H & \cdots & A_{M-1,N-1}^H \end{pmatrix}.$

SEE ANSWER

1.4 Vector-vector Operations

1.4.1 Scaling a vector (`scal`)

Let $x \in \mathbb{C}^n$ and $\alpha \in \mathbb{C}$, with

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}.$$

Then αx equals the vector x “stretched” by a factor α :

$$\alpha x = \alpha \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha \chi_0 \\ \alpha \chi_1 \\ \vdots \\ \alpha \chi_{n-1} \end{pmatrix}.$$

If $y := \alpha x$ with

$$y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix},$$

then the following loop computes y :

```
for  $i := 0, \dots, n - 1$ 
     $\psi_i := \alpha \chi_i$ 
endfor
```

Homework 1.4 Convince yourself of the following:

- $\alpha x^T = (\alpha \chi_0 \mid \alpha \chi_1 \mid \dots \mid \alpha \chi_{n-1})$.
- $(\alpha x)^T = \alpha x^T$.
- $(\alpha x)^H = \overline{\alpha} x^H$.

$$\bullet \alpha \begin{pmatrix} \frac{x_0}{x_1} \\ \frac{x_1}{x_2} \\ \vdots \\ \frac{x_{N-1}}{x_N} \end{pmatrix} = \begin{pmatrix} \frac{\alpha x_0}{x_1} \\ \frac{\alpha x_1}{x_2} \\ \vdots \\ \frac{\alpha x_{N-1}}{x_N} \end{pmatrix}$$

☞ SEE ANSWER

Cost

Scaling a vector of size n requires, approximately, n multiplications. Each of these becomes a floating point operation (flop) when the computation is performed as part of an algorithm executed on a computer that performs floating point computation. We will thus say that scaling a vector costs n flops.

It should be noted that arithmetic with complex numbers is roughly 4 times as expensive as is arithmetic with real numbers. In the chapter “Notes on Performance” (page 211) we also discuss that the cost of moving data impacts the cost of a flop. Thus, not all flops are created equal!

1.4.2 Scaled vector addition (axpy)

Let $x, y \in \mathbb{C}^n$ and $\alpha \in \mathbb{C}$, with

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix}.$$

Then $\alpha x + y$ equals the vector

$$\alpha x + y = \alpha \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} + \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha\chi_0 \\ \alpha\chi_1 \\ \vdots \\ \alpha\chi_{n-1} \end{pmatrix} + \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix}.$$

This operation is known as the axpy operation: alpha times x plus y . Typically, the vector y is overwritten with the result:

$$y := \alpha x + y = \alpha \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} + \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha\chi_0 \\ \alpha\chi_1 \\ \vdots \\ \alpha\chi_{n-1} \end{pmatrix} + \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix}$$

so that the following loop updates y :

```
for  $i := 0, \dots, n - 1$ 
     $\psi_i := \alpha\chi_i + \psi_i$ 
endfor
```

Homework 1.5 Convince yourself of the following:

$$\bullet \alpha \begin{pmatrix} \frac{x_0}{1} \\ \frac{x_1}{1} \\ \vdots \\ \frac{x_{N-1}}{1} \end{pmatrix} + \begin{pmatrix} \frac{y_0}{1} \\ \frac{y_1}{1} \\ \vdots \\ \frac{y_{N-1}}{1} \end{pmatrix} = \begin{pmatrix} \frac{\alpha x_0 + y_0}{1} \\ \frac{\alpha x_1 + y_1}{1} \\ \vdots \\ \frac{\alpha x_{N-1} + y_{N-1}}{1} \end{pmatrix}. \text{(Provided } x_i, y_i \in \mathbb{C}^{n_i} \text{ and } \sum_{i=0}^{N-1} n_i = n\text{.)}$$

 SEE ANSWER

Cost

The axpy with two vectors of size n requires, approximately, n multiplies and n additions or $2n$ flops.

1.4.3 Dot (inner) product (dot)

Let $x, y \in \mathbb{C}^n$ with

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix}.$$

Then the dot product of x and y is defined by

$$\begin{aligned} x^H y &= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^H \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \left(\overline{\chi_0} \ \overline{\chi_1} \ \cdots \ \overline{\chi_{n-1}} \right) \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} \\ &= \overline{\chi_0}\psi_0 + \overline{\chi_1}\psi_1 + \cdots + \overline{\chi_{n-1}}\psi_{n-1} = \sum_{i=0}^{n-1} \overline{\chi_i}\psi_i. \end{aligned}$$

The following loop computes $\alpha := x^H y$:

```

 $\alpha := 0$ 
for  $i := 0, \dots, n-1$ 
     $\alpha := \overline{\chi_i}\psi + \alpha$ 
endfor

```

Homework 1.6 Convince yourself of the following:

$$\bullet \quad \begin{pmatrix} \frac{x_0}{\cdot} \\ \frac{x_1}{\cdot} \\ \vdots \\ \frac{x_{N-1}}{\cdot} \end{pmatrix}^H \begin{pmatrix} \frac{y_0}{\cdot} \\ \frac{y_1}{\cdot} \\ \vdots \\ \frac{y_{N-1}}{\cdot} \end{pmatrix} = \sum_{i=0}^{N-1} x_i^H y_i. \quad (\text{Provided } x_i, y_i \in \mathbb{C}^{n_i} \text{ and } \sum_{i=0}^{N-1} n_i = n.)$$

SEE ANSWER

Homework 1.7 Prove that $x^H y = \overline{y^H x}$.

SEE ANSWER

As we discuss matrix-vector multiplication and matrix-matrix multiplication, the closely related operation $x^T y$ is also useful:

$$\begin{aligned} x^T y &= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \left(\chi_0 \ \chi_1 \ \cdots \ \chi_{n-1} \right) \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} \\ &= \chi_0\psi_0 + \chi_1\psi_1 + \cdots + \chi_{n-1}\psi_{n-1} = \sum_{i=0}^{n-1} \chi_i\psi_i. \end{aligned}$$

We will sometimes refer to this operation as a “dot” product since it is like the dot product $x^H y$, but without conjugation. In the case of real valued vectors, it is the dot product.

Cost

The dot product of two vectors of size n requires, approximately, n multiplies and n additions. Thus, a dot product cost, approximately, $2n$ flops.

1.5 Matrix-vector Operations

1.5.1 Matrix-vector multiplication (product)

Be sure to understand the relation between linear transformations and matrix-vector multiplication by reading Week 2 of

[Linear Algebra: Foundations to Frontiers - Notes to LAFF With \[30\]](#).

Let $y \in \mathbb{C}^m$, $A \in \mathbb{C}^{m \times n}$, and $x \in \mathbb{C}^n$ with

$$y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix}, \quad A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix}, \quad \text{and} \quad x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}.$$

Then $y = Ax$ means that

$$\begin{aligned} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} &= \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} \alpha_{0,0}\chi_0 + \alpha_{0,1}\chi_1 + \cdots + \alpha_{0,n-1}\chi_{n-1} \\ \alpha_{1,0}\chi_0 + \alpha_{1,1}\chi_1 + \cdots + \alpha_{1,n-1}\chi_{n-1} \\ \vdots \\ \alpha_{m-1,0}\chi_0 + \alpha_{m-1,1}\chi_1 + \cdots + \alpha_{m-1,n-1}\chi_{n-1} \end{pmatrix}. \end{aligned}$$

This is the definition of the matrix-vector product Ax , sometimes referred to as a general matrix-vector multiplication (`gemv`) when no special structure or properties of A are assumed.

Now, partition

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

Focusing on how A can be partitioned by columns, we find that

$$\begin{aligned} y &= Ax = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \\ &= a_0\chi_0 + a_1\chi_1 + \cdots + a_{n-1}\chi_{n-1} \\ &= \chi_0a_0 + \chi_1a_1 + \cdots + \chi_{n-1}a_{n-1} \\ &= \chi_{n-1}a_{n-1} + (\cdots + (\chi_1a_1 + (\chi_0a_0 + 0)) \cdots), \end{aligned}$$

where 0 denotes the zero vector of size m . This suggests the following loop for computing $y := Ax$:

```

 $y := 0$ 
for  $j := 0, \dots, n - 1$ 
     $y := \chi_j a_j + y$       (axpy)
endfor
```

In Figure 1.1 (left), we present this algorithm using the FLAME notation ([LAFF Notes Week 3 \[30\]](#)).

Focusing on how A can be partitioned by rows, we find that

$$y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = Ax = \begin{pmatrix} \widehat{a}_0^T \\ \widehat{a}_1^T \\ \vdots \\ \widehat{a}_{m-1}^T \end{pmatrix} x = \begin{pmatrix} \widehat{a}_0^T x \\ \widehat{a}_1^T x \\ \vdots \\ \widehat{a}_{m-1}^T x \end{pmatrix}.$$

This suggests the following loop for computing $y := Ax$:

```

 $y := 0$ 
for  $i := 0, \dots, m - 1$ 
     $\psi_i := \widehat{a}_i^T x + \psi_i$       (''dot'')
endfor
```

Here we use the term “dot” because for complex valued matrices it is not really a dot product. In Figure 1.1 (right), we present this algorithm using the FLAME notation ([LAFF Notes Week 3 \[30\]](#)).

It is important to notice that this first “matrix-vector” operation (matrix-vector multiplication) can be “layered” upon vector-vector operations (axpy or ‘‘dot’’).

Cost

Matrix-vector multiplication with a $m \times n$ matrix costs, approximately, $2mn$ flops. This can be easily argued in three different ways:

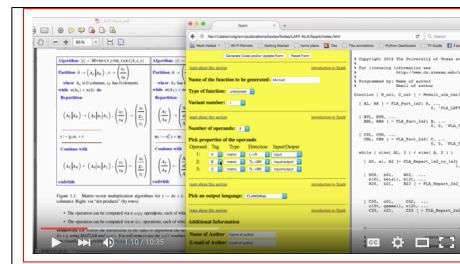
- The computation requires a multiply and an add with each element of the matrix. There are mn such elements.

<p>Algorithm: $[y] := \text{MVMULT_UNB_VAR1}(A, x, y)$</p> <p>Partition $A \rightarrow \left(A_L \middle A_R \right)$, $x \rightarrow \begin{pmatrix} x_T \\ x_B \end{pmatrix}$</p> <p>where A_L is 0 columns, x_T has 0 elements</p> <p>while $n(A_L) < n(A)$ do</p> <p style="padding-left: 20px;">Repartition</p> <p style="text-align: center;">$\left(A_L \middle A_R \right) \rightarrow \left(A_0 \middle a_1 \middle A_2 \right)$, $\begin{pmatrix} x_T \\ x_B \end{pmatrix} \rightarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}$</p> <hr/> <p style="margin-top: 10px;">$y := \chi_1 a_1 + y$</p> <hr/> <p>Continue with</p> <p style="text-align: center;">$\left(A_L \middle A_R \right) \leftarrow \left(A_0 \middle a_1 \middle A_2 \right)$, $\begin{pmatrix} x_T \\ x_B \end{pmatrix} \leftarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}$</p> <p>endwhile</p>	<p>Algorithm: $[y] := \text{MVMULT_UNB_VAR2}(A, x, y)$</p> <p>Partition $A \rightarrow \begin{pmatrix} A_T \\ A_B \end{pmatrix}$, $y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix}$</p> <p>where A_T has 0 rows, y_T has 0 elements</p> <p>while $m(A_T) < m(A)$ do</p> <p style="padding-left: 20px;">Repartition</p> <p style="text-align: center;">$\begin{pmatrix} A_T \\ A_B \end{pmatrix} \rightarrow \begin{pmatrix} A_0 \\ a_1^T \\ A_2 \end{pmatrix}$, $\begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ y_2 \end{pmatrix}$</p> <hr/> <p style="margin-top: 10px;">$\Psi_1 := a_1^T x + \Psi_1$</p> <hr/> <p>Continue with</p> <p style="text-align: center;">$\begin{pmatrix} A_T \\ A_B \end{pmatrix} \leftarrow \begin{pmatrix} A_0 \\ a_1^T \\ A_2 \end{pmatrix}$, $\begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ y_2 \end{pmatrix}$</p> <p>endwhile</p>
--	--

Figure 1.1: Matrix-vector multiplication algorithms for $y := Ax + y$. Left: via `axpy` operations (by columns). Right: via “dot products” (by rows).

- The operation can be computed via n `axpy` operations, each of which requires $2m$ flops.
- The operation can be computed via m dot operations, each of which requires $2n$ flops.

Homework 1.8 Follow the instructions in the below video to implement the two algorithms for computing $y := Ax + y$, using MATLAB and **Spark**. You will want to use the `laff` routines summarized in Appendix B. You can visualize the algorithm with **PictureFLAME**.



YouTube
Downloaded Video

SEE ANSWER

1.5.2 Rank-1 update

Let $y \in \mathbb{C}^m$, $A \in \mathbb{C}^{m \times n}$, and $x \in \mathbb{C}^n$ with

$$y = \begin{pmatrix} \Psi_0 \\ \Psi_1 \\ \vdots \\ \Psi_{m-1} \end{pmatrix}, \quad A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix}, \quad \text{and} \quad x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}.$$

The outerproduct of y and x is given by

$$\begin{aligned} yx^T &= \begin{pmatrix} \Psi_0 \\ \Psi_1 \\ \vdots \\ \Psi_{m-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T = \begin{pmatrix} \Psi_0 \\ \Psi_1 \\ \vdots \\ \Psi_{m-1} \end{pmatrix} \begin{pmatrix} \chi_0 & \chi_1 & \cdots & \chi_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} \Psi_0\chi_0 & \Psi_0\chi_1 & \cdots & \Psi_0\chi_{n-1} \\ \Psi_1\chi_0 & \Psi_1\chi_1 & \cdots & \Psi_1\chi_{n-1} \\ \vdots & \vdots & & \vdots \\ \Psi_{m-1}\chi_0 & \Psi_{m-1}\chi_1 & \cdots & \Psi_{m-1}\chi_{n-1} \end{pmatrix}. \end{aligned}$$

Also,

$$yx^T = y \begin{pmatrix} \chi_0 & \chi_1 & \cdots & \chi_{n-1} \end{pmatrix} = \begin{pmatrix} \chi_0y & \chi_1y & \cdots & \chi_{n-1}y \end{pmatrix}.$$

This shows that all columns are a multiple of vector y . Finally,

$$yx^T = \begin{pmatrix} \Psi_0 \\ \Psi_1 \\ \vdots \\ \Psi_{m-1} \end{pmatrix} x^T = \begin{pmatrix} \Psi_0 x^T \\ \Psi_1 x^T \\ \vdots \\ \Psi_{m-1} x^T \end{pmatrix},$$

which shows that all columns are a multiple of row vector x^T . This motivates the observation that the matrix yx^T has rank at most equal to one (**LAFF Notes Week 10** [30]).

The operation $A := yx^T + A$ is called a rank-1 update to matrix A and is often referred to as underline-general rank-1 update (ger):

$$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} :=$$

$$\begin{pmatrix} \Psi_0\chi_0 + \alpha_{0,0} & \Psi_0\chi_1 + \alpha_{0,1} & \cdots & \Psi_0\chi_{n-1} + \alpha_{0,n-1} \\ \Psi_1\chi_0 + \alpha_{1,0} & \Psi_1\chi_1 + \alpha_{1,1} & \cdots & \Psi_1\chi_{n-1} + \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \Psi_{m-1}\chi_0 + \alpha_{m-1,0} & \Psi_{m-1}\chi_1 + \alpha_{m-1,1} & \cdots & \Psi_{m-1}\chi_{n-1} + \alpha_{m-1,n-1} \end{pmatrix}.$$

Now, partition

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

Focusing on how A can be partitioned by columns, we find that

$$\begin{aligned} yx^T + A &= \left(\begin{array}{c|c|c|c} \chi_0y & \chi_1y & \cdots & \chi_{n-1}y \end{array} \right) + \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) \\ &= \left(\begin{array}{c|c|c|c} \chi_0y + a_0 & \chi_1y + a_1 & \cdots & \chi_{n-1}y + a_{n-1} \end{array} \right). \end{aligned}$$

Notice that each column is updated with an axpy operation. This suggests the following loop for computing $A := yx^T + A$:

```
for  $j := 0, \dots, n-1$ 
     $a_j := \chi_j y + a_j$       (axpy)
endfor
```

In Figure 1.2 (left), we present this algorithm using the FLAME notation ([LAFF Notes Week 3](#)).

Focusing on how A can be partitioned by rows, we find that

$$\begin{aligned} yx^T + A &= \begin{pmatrix} \Psi_0x^T \\ \Psi_1x^T \\ \vdots \\ \Psi_{m-1}x^T \end{pmatrix} + \begin{pmatrix} \widehat{a}_0^T \\ \widehat{a}_1^T \\ \vdots \\ \widehat{a}_{m-1}^T \end{pmatrix} \\ &= \begin{pmatrix} \Psi_0x^T + \widehat{a}_0^T \\ \Psi_1x^T + \widehat{a}_1^T \\ \vdots \\ \Psi_{m-1}x^T + \widehat{a}_{m-1}^T \end{pmatrix}. \end{aligned}$$

Notice that each row is updated with an axpy operation. This suggests the following loop for computing $A := yx^T + A$:

```
for  $i := 0, \dots, m-1$ 
     $\widehat{a}_j^T := \psi_i x^T + \widehat{a}_j^T$       (axpy)
endfor
```

Algorithm: $[A] := \text{RANK1_UNB_VAR1}(y, x, A)$

Partition $x \rightarrow \begin{pmatrix} x_T \\ x_B \end{pmatrix}, A \rightarrow \left(A_L \middle| A_R \right)$

where x_T has 0 elements, A_L has 0 columns

while $m(x_T) < m(x)$ **do**

Repartition

$$\begin{pmatrix} x_T \\ x_B \end{pmatrix} \rightarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}, \left(A_L \middle| A_R \right) \rightarrow \left(A_0 \middle| a_1 \middle| A_2 \right)$$

$$a_1 := \chi_1 y + a_1 \quad (\text{axpy})$$

Continue with

$$\begin{pmatrix} x_T \\ x_B \end{pmatrix} \leftarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}, \left(A_L \middle| A_R \right) \leftarrow \left(A_0 \middle| a_1 \middle| A_2 \right)$$

endwhile

Algorithm: $[A] := \text{RANK1_UNB_VAR2}(y, x, A)$

Partition $y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix}, A \rightarrow \begin{pmatrix} A_T \\ A_B \end{pmatrix}$

where y_T has 0 elements, A_T has 0 rows

while $m(y_T) < m(y)$ **do**

Repartition

$$\begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ y_2 \end{pmatrix}, \begin{pmatrix} A_T \\ A_B \end{pmatrix} \rightarrow \begin{pmatrix} A_0 \\ \frac{a_1^T}{A_2} \end{pmatrix}$$

$$a_1^T := \Psi_1 x^T + a_1^T \quad (\text{axpy})$$

Continue with

$$\begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ y_2 \end{pmatrix}, \begin{pmatrix} A_T \\ A_B \end{pmatrix} \leftarrow \begin{pmatrix} A_0 \\ \frac{a_1^T}{A_2} \end{pmatrix}$$

endwhile

Figure 1.2: Rank-1 update algorithms for computing $A := yx^T + A$. Left: by columns. Right: by rows.

In Figure 1.2 (right), we present this algorithm using the FLAME notation (LAFF Notes Week 3).

Again, it is important to notice that this “matrix-vector” operation (rank-1 update) can be “layered” upon the `axpy` vector-vector operation.

Cost

A rank-1 update of a $m \times n$ matrix costs, approximately, $2mn$ flops. This can be easily argued in three different ways:

- The computation requires a multiply and an add with each element of the matrix. There are mn such elements.
- The operation can be computed one column at a time via n `axpy` operations, each of which requires $2m$ flops.
- The operation can be computed one row at a time via m `axpy` operations, each of which requires $2n$ flops.

Homework 1.9 Implement the two algorithms for computing $A := yx^T + A$, using MATLAB and Spark.

You will want to use the laff routines summarized in Appendix B. You can visualize the algorithm with *PictureFLAME*.

 SEE ANSWER

Homework 1.10 Prove that the matrix xy^T where x and y are vectors has rank at most one, thus explaining the name “rank-1 update”.

 SEE ANSWER

1.6 Matrix-matrix multiplication (product)

Be sure to understand the relation between linear transformations and matrix-matrix multiplication ([LAFF Notes Weeks 3 and 4 \[30\]](#)).

We will now discuss the computation of $C := AB + C$, where $C \in \mathbb{C}^{m \times n}$, $A \in \mathbb{C}^{m \times k}$, and $B \in \mathbb{C}^{k \times n}$. (If one wishes to compute $C := AB$, one can always start by setting $C := 0$, the zero matrix.) This is the definition of the matrix-matrix product AB , sometimes referred to as a general matrix-matrix multiplication (gemm) when no special structure or properties of A and B are assumed.

1.6.1 Element-by-element computation

Let

$$C = \begin{pmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \\ \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,n-1} \end{pmatrix}, A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,k-1} \end{pmatrix}$$

$$B = \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \\ \beta_{1,0} & \beta_{1,1} & \cdots & \beta_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{k-1,0} & \beta_{k-1,1} & \cdots & \beta_{k-1,n-1} \end{pmatrix}.$$

Then

$$C := AB + C$$

$$= \begin{pmatrix} \sum_{p=0}^{k-1} \alpha_{0,p} \beta_{p,0} + \gamma_{0,0} & \sum_{p=0}^{k-1} \alpha_{0,p} \beta_{p,1} + \gamma_{0,1} & \cdots & \sum_{p=0}^{k-1} \alpha_{0,p} \beta_{p,n-1} + \gamma_{0,n-1} \\ \sum_{p=0}^{k-1} \alpha_{1,p} \beta_{p,0} + \gamma_{1,0} & \sum_{p=0}^{k-1} \alpha_{1,p} \beta_{p,1} + \gamma_{1,1} & \cdots & \sum_{p=0}^{k-1} \alpha_{1,p} \beta_{p,n-1} + \gamma_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{p=0}^{k-1} \alpha_{m-1,p} \beta_{p,0} + \gamma_{m-1,0} & \sum_{p=0}^{k-1} \alpha_{m-1,p} \beta_{p,1} + \gamma_{m-1,1} & \cdots & \sum_{p=0}^{k-1} \alpha_{m-1,p} \beta_{p,n-1} + \gamma_{m-1,n-1} \end{pmatrix}.$$

This motivates the algorithm

```

for  $i := 0, \dots, m-1$ 
  for  $J := 0, \dots, n-1$ 
    for  $p := 0, \dots, k-1$ 
       $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$  (dot)
    endfor
  endfor
endfor

```

Notice that the **for** loops can be ordered in $3! = 6$ different ways.

How to update each element of C via dot products can be more elegantly expressed by partitioning

$$A = \begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix} \quad \text{and} \quad B = \left(b_0 \mid b_1 \mid \cdots \mid b_{n-1} \right).$$

Then

$$\begin{aligned} C &:= AB + C = \begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix} \left(b_0 \mid b_1 \mid \cdots \mid b_{n-1} \right) + \begin{pmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \\ \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,n-1} \end{pmatrix} \\ &= \begin{pmatrix} \hat{a}_0^T b_0 + \gamma_{0,0} & \hat{a}_0^T b_1 + \gamma_{0,1} & \cdots & \hat{a}_0^T b_{n-1} + \gamma_{0,n-1} \\ \hat{a}_1^T b_0 + \gamma_{1,0} & \hat{a}_1^T b_1 + \gamma_{1,1} & \cdots & \hat{a}_1^T b_{n-1} + \gamma_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{a}_{m-1}^T b_0 + \gamma_{m-1,0} & \hat{a}_{m-1}^T b_1 + \gamma_{m-1,1} & \cdots & \hat{a}_{m-1}^T b_{n-1} + \gamma_{m-1,n-1} \end{pmatrix}. \end{aligned}$$

This suggests the following algorithms for computing $C := AB + C$:

for $i := 0, \dots, m-1$ for $J := 0, \dots, n-1$ $\gamma_{i,j} := \hat{a}_i^T b_j + \gamma_{i,j}$ (dot) endfor endfor	for $j := 0, \dots, n-1$ for $i := 0, \dots, m-1$ $\gamma_{i,j} := \hat{a}_i^T b_j + \gamma_{i,j}$ (dot) endfor endfor
--	--

The cost of any of these algorithms is $2mnk$ flops, which can be explained by the fact that mn elements of C must be updated with a dot product of length k .

1.6.2 Via matrix-vector multiplications

Partition

$$C = \left(c_0 \mid c_1 \mid \cdots \mid c_{n-1} \right) \quad \text{and} \quad B = \left(b_0 \mid b_1 \mid \cdots \mid b_{n-1} \right).$$

Algorithm: $C := \text{GEMM_UNB_VAR1}(A, B, C)$

Partition $B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array} \right), C \rightarrow \left(\begin{array}{c|c} C_L & C_R \end{array} \right)$
where B_L has 0 columns, C_L has 0 columns

while $n(B_L) < n(B)$ **do**

Repartition

$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right), \left(\begin{array}{c|c} C_L & C_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} C_0 & c_1 & C_2 \end{array} \right)$

$c_1 := Ab_1 + c_1$

Continue with

$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right), \left(\begin{array}{c|c} C_L & C_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} C_0 & c_1 & C_2 \end{array} \right)$

endwhile

Figure 1.3: Algorithm for computing $C := AB + C$ one column at a time.

Then

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

which shows that each column of C is updated with a matrix-vector multiplication: $c_j := Ab_j + c_j$:

```

for  $j := 0, \dots, n - 1$ 
     $c_j := Ab_j + c_j$       (matrix-vector multiplication)
endfor

```

In Figure 1.3, we present this algorithm using the FLAME notation ([LAFF Notes, Week 3 and 4](#)).

The given matrix-matrix multiplication algorithm with $m \times n$ matrix C , $m \times k$ matrix A , and $k \times n$ matrix B costs, approximately, $2mnk$ flops. This can be easily argued by noting that each update of a column of matrix C costs, approximately, $2mk$ flops. There are n such columns to be computed.

Homework 1.11 Implement $C := AB + C$ via matrix-vector multiplications, using MATLAB and [Spark](#). You will want to use the laff routines summarized in Appendix B. You can visualize the algorithm with [PictureFLAME](#).

SEE ANSWER

Algorithm: $C := \text{GEMM_UNB_VAR2}(A, B, C)$
Partition $A \rightarrow \begin{pmatrix} A_T \\ A_B \end{pmatrix}, C \rightarrow \begin{pmatrix} C_T \\ C_B \end{pmatrix}$
where A_T has 0 rows, C_T has 0 rows
while $m(A_T) < m(A)$ do
Repartition
$\begin{pmatrix} A_T \\ A_B \end{pmatrix} \rightarrow \begin{pmatrix} A_0 \\ a_1^T \\ A_2 \end{pmatrix}, \begin{pmatrix} C_T \\ C_B \end{pmatrix} \rightarrow \begin{pmatrix} C_0 \\ c_1^T \\ C_2 \end{pmatrix}$
<hr/> $c_1^T := a_1^T B + c_1^T$ <hr/>
Continue with
$\begin{pmatrix} A_T \\ A_B \end{pmatrix} \leftarrow \begin{pmatrix} A_0 \\ a_1^T \\ A_2 \end{pmatrix}, \begin{pmatrix} C_T \\ C_B \end{pmatrix} \leftarrow \begin{pmatrix} C_0 \\ c_1^T \\ C_2 \end{pmatrix}$
endwhile

Figure 1.4: Algorithm for computing $C := AB + C$ one row at a time.

1.6.3 Via row-vector times matrix multiplications

Partition

$$C = \begin{pmatrix} \hat{c}_0^T \\ \hat{c}_1^T \\ \vdots \\ \hat{c}_{m-1}^T \end{pmatrix} \quad \text{and} \quad A = \begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix}.$$

Then

$$C := AB + C = \begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix} B + \begin{pmatrix} \hat{c}_0^T \\ \hat{c}_1^T \\ \vdots \\ \hat{c}_{m-1}^T \end{pmatrix} = \begin{pmatrix} \hat{a}_0^T B + \hat{c}_0^T \\ \hat{a}_1^T B + \hat{c}_1^T \\ \vdots \\ \hat{a}_{m-1}^T B + \hat{c}_{m-1}^T \end{pmatrix}$$

which shows that each row of C is updated with a row-vector time matrix multiplication: $\hat{c}_i^T := \hat{a}_i^T B + \hat{c}_i^T$:

```

for  $i := 0, \dots, m - 1$ 
   $\hat{c}_i^T := \hat{a}_i^T B + \hat{c}_i^T$       (row-vector times matrix-vector multiplication)
endfor

```

In Figure 1.4, we present this algorithm using the FLAME notation ([LAFF Notes, Week 3 and 4](#)).

Homework 1.12 Argue that the given matrix-matrix multiplication algorithm with $m \times n$ matrix C , $m \times k$ matrix A , and $k \times n$ matrix B costs, approximately, $2mnk$ flops.

☞ [SEE ANSWER](#)

Homework 1.13 Implement $C := AB + C$ via row vector-matrix multiplications, using MATLAB and [Spark](#). You will want to use the `laff` routines summarized in Appendix B. You can visualize the algorithm with [PictureFLAME](#). Hint: $y^T := x^T A + y^T$ is not supported by a `laff` routine. You can use `laff_gemv` instead.

An implementation can be found in

`Programming/chapter01_answers/Gemm_unb_var2.m` ([see file only](#)) ([view in MATLAB](#))

☞ [SEE ANSWER](#)

1.6.4 Via rank-1 updates

Partition

$$A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{k-1} \end{array} \right) \quad \text{and} \quad B = \left(\begin{array}{c} \widehat{b}_0^T \\ \widehat{b}_1^T \\ \vdots \\ \widehat{b}_{k-1}^T \end{array} \right).$$

The

$$\begin{aligned} C := AB + C &= \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{k-1} \end{array} \right) \left(\begin{array}{c} \widehat{b}_0^T \\ \widehat{b}_1^T \\ \vdots \\ \widehat{b}_{k-1}^T \end{array} \right) + C \\ &= a_0 \widehat{b}_0^T + a_1 \widehat{b}_1^T + \cdots + a_{k-1} \widehat{b}_{k-1}^T + C \\ &= a_{k-1} \widehat{b}_{k-1}^T + (\cdots (a_1 \widehat{b}_1^T + (a_0 \widehat{b}_0^T + C)) \cdots) \end{aligned}$$

which shows that C can be updated with a sequence of rank-1 update, suggesting the loop

```

for  $p := 0, \dots, k - 1$ 
   $C := a_p \widehat{b}_p^T + C$       (rank-1 update)
endfor

```

In Figure 1.5, we present this algorithm using the FLAME notation ([LAFF Notes, Week 3 and 4](#)).

<p>Algorithm: $[C] := \text{GEMM_UNB_VAR3}(A, B, C)$</p> <p>Partition $A \rightarrow \left(\begin{array}{c c} A_L & A_R \end{array} \right), B \rightarrow \left(\begin{array}{c} B_T \\ B_B \end{array} \right)$ where A_L has 0 columns, B_T has 0 rows</p> <p>while $n(A_L) < n(A)$ do</p> <p style="margin-left: 20px;">Repartition</p> $\left(\begin{array}{c c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c c c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c} B_T \\ B_B \end{array} \right) \rightarrow \left(\begin{array}{c} B_0 \\ \frac{B_T}{b_1^T} \\ B_2 \end{array} \right)$ <hr style="border: 1px solid red; margin-top: 10px;"/> <p style="margin-left: 20px;">$C := a_1 b_1^T + C$</p> <hr style="border: 1px solid red; margin-top: 10px;"/> <p>Continue with</p> $\left(\begin{array}{c c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c c c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c} B_T \\ B_B \end{array} \right) \leftarrow \left(\begin{array}{c} B_0 \\ \frac{B_T}{b_1^T} \\ B_2 \end{array} \right)$ <p>endwhile</p>

Figure 1.5: Algorithm for computing $C := AB + C$ via rank-1 updates.

Homework 1.14 Argue that the given matrix-matrix multiplication algorithm with $m \times n$ matrix C , $m \times k$ matrix A , and $k \times n$ matrix B costs, approximately, $2mnk$ flops.

► SEE ANSWER

Homework 1.15 Implement $C := AB + C$ via rank-1 updates, using MATLAB and **Spark**. You will want to use the laff routines summarized in Appendix B. You can visualize the algorithm with **PictureFLAME**.

► SEE ANSWER

1.7 Enrichments

1.7.1 The Basic Linear Algebra Subprograms (BLAS)

Any user or practitioner of numerical linear algebra must be familiar with the Basic Linear Algebra Subprograms (BLAS), a standardized interface for commonly encountered linear algebra operations. A recommended reading is

Robert van de Geijn and Kazushige Goto

► **BLAS (Basic Linear Algebra Subprograms)**

Encyclopedia of Parallel Computing , Part 2, Pages 157-164. 2011.

For my graduate class I have posted this article on  Canvas. Others may have access to this article via their employer or university.

1.8 Wrapup

1.8.1 Additional exercises

Homework 1.16 Implement GS_unb_var1 using MATLAB and Spark. You will want to use the laff routines summarized in Appendix B. (I'm not sure if you can visualize the algorithm with PictureFLAME. Try it!)

 [SEE ANSWER](#)

Homework 1.17 Implement MGS_unb_var1 using MATLAB and Spark. You will want to use the laff routines summarized in Appendix B. (I'm not sure if you can visualize the algorithm with PictureFLAME. Try it!)

 [SEE ANSWER](#)

1.8.2 Summary

It is important to realize that almost all operations that we discussed in this chapter are special cases of matrix-matrix multiplication. This is summarized in Figure 1.6. A few notes:

- Row-vector times matrix is matrix-vector multiplication in disguise: If $y^T := x^T A$ then $y := A^T x$.
- For a similar reason $y^T := \alpha x^T + y^T$ is an axpy in disguise: $y := \alpha x + y$.
- The operation $y := x\alpha + y$ is the same as $y := \alpha x + y$ since multiplication of a vector by a scalar commutes.
- The operation $\gamma := \alpha\beta + \gamma$ is known as a Multiply-ACcumulate (MAC) operation. Often floating point hardware implements this as an integrated operation.

Observations made about operations with partitioned matrices and vectors can be summarized by partitioning matrices into blocks: Let

$$C = \begin{pmatrix} C_{0,0} & C_{0,1} & \cdots & C_{0,N-1} \\ C_{1,0} & C_{1,1} & \cdots & C_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ C_{M-1,0} & C_{M-1,1} & \cdots & C_{M-1,N-1} \end{pmatrix}, \quad A = \begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,K-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,K-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,K-1} \end{pmatrix}$$

$$B = \begin{pmatrix} B_{0,1} & B_{0,1} & \cdots & B_{0,N-1} \\ B_{1,0} & B_{1,1} & \cdots & B_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ B_{K-1,0} & B_{K-1,1} & \cdots & B_{K-1,N-1} \end{pmatrix}.$$

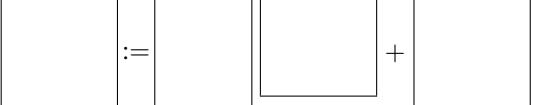
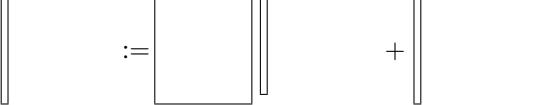
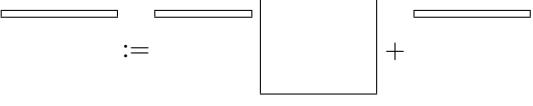
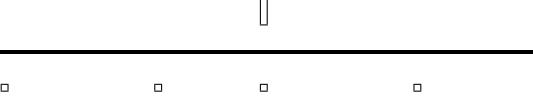
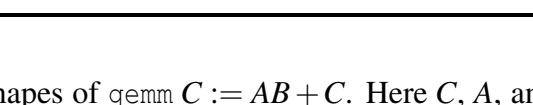
m	n	k	Illustration	Label	Operation
large	large	large		gemm	General matrix-matrix multiplication
large	large	1		ger	General rank-1 update (outer product if initially $C = 0$)
large	1	large		gemv	General matrix-vector multiplication
1	large	large		gemv	General matrix-vector multiplication
1	large	1		axpy	
large	1	1		axpy	
1	1	large		dot	
1	1	1		MAC	Multiply accumulate

Figure 1.6: Special shapes of `gemm` $C := AB + C$. Here C , A , and B are $m \times n$, $m \times k$, and $k \times n$ matrices, respectively.

Then

$$C := AB + C =$$

$$\begin{pmatrix} \sum_{p=0}^{K-1} A_{0,p} B_{p,0} + C_{0,0} & \sum_{p=0}^{K-1} A_{0,p} B_{p,1} + C_{0,1} & \cdots & \sum_{p=0}^{K-1} A_{0,p} B_{p,N-1} + C_{0,N-1} \\ \sum_{p=0}^{K-1} A_{1,p} B_{p,0} + C_{1,0} & \sum_{p=0}^{K-1} A_{1,p} B_{p,1} + C_{1,1} & \cdots & \sum_{p=0}^{K-1} A_{1,p} B_{p,N-1} + C_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ \sum_{p=0}^{K-1} A_{M-1,p} B_{p,0} + C_{M-1,0} & \sum_{p=0}^{k-1} A_{M-1,p} B_{p,1} + C_{M-1,1} & \cdots & \sum_{p=0}^{k-1} A_{M-1,p} B_{p,N-1} + C_{M-1,N-1} \end{pmatrix}.$$

(Provided the partitionings of C , A , and B are “conformal”.)

Notice that multiplication with partitioned matrices is exactly like regular matrix-matrix multiplication with scalar elements, *except that multiplication of two blocks does not necessarily commute*.

Chapter 2

Notes on Vector and Matrix Norms

2.1 Opening Remarks

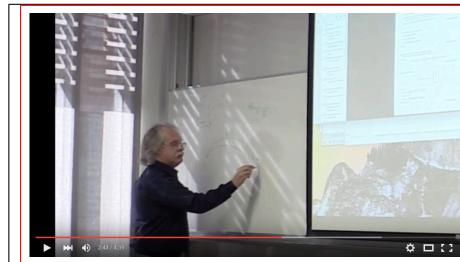
Video from Fall 2014

Read disclaimer regarding the videos in the preface!

- 👉 YouTube
- 👉 Download from UT Box
- 👉 View After Local Download

(For help on viewing, see Appendix A.)

2.1.1 Launch



- 👉 YouTube
- 👉 Downloaded Video

2.1.2 Outline

2.1	Opening Remarks	37
2.1.1	Launch	37
2.1.2	Outline	38
2.1.3	What You Will Learn	39
2.2	Absolute Value	40
2.3	Vector Norms	40
2.3.1	Vector 2-norm (Euclidean length)	40
2.3.2	Vector 1-norm	42
2.3.3	Vector ∞ -norm (infinity norm)	42
2.3.4	Vector p -norm	43
2.4	Matrix Norms	43
2.4.1	Frobenius norm	43
2.4.2	Induced matrix norms	44
2.4.3	Special cases used in practice	45
2.4.4	Discussion	47
2.4.5	Submultiplicative norms	47
2.5	An Application to Conditioning of Linear Systems	48
2.6	Equivalence of Norms	49
2.7	Enrichments	50
2.7.1	Practical computation of the vector 2-norm	50
2.8	Wrapup	50
2.8.1	Additional exercises	50
2.8.2	Summary	53

2.1.3 What You Will Learn

2.2 Absolute Value

Recall that if $\alpha \in \mathbb{C}$, then $|\alpha|$ equals its absolute value. In other words, if $\alpha = \alpha_r + i\alpha_c$, then

$$|\alpha| = \sqrt{\alpha_r^2 + \alpha_c^2} = \sqrt{(\alpha_r - i\alpha_c)(\alpha_r + i\alpha_c)} = \sqrt{\alpha\bar{\alpha}}.$$

This 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).

2.3 Vector Norms

YouTube
Downloaded Video

A (vector) norm extends the notion of an absolute value (length) to vectors:

Definition 2.1 Let $v : \mathbb{C}^n \rightarrow \mathbb{R}$. Then v is a (vector) norm if for all $x, y \in \mathbb{C}^n$ and all $\alpha \in \mathbb{C}$

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

Homework 2.2 Prove that if $v : \mathbb{C}^n \rightarrow \mathbb{R}$ is a norm, then $v(0) = 0$ (where the first 0 denotes the zero vector in \mathbb{C}^n).

SEE ANSWER

Note: often we will use $\|\cdot\|$ to denote a vector norm.

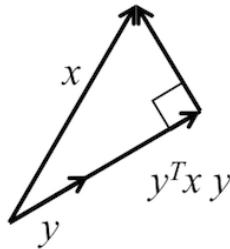
2.3.1 Vector 2-norm (Euclidean length)

The length of a vector is most commonly measured by the “square root of the sum of the square of the elements.” For reasons that will become clear later in this section, we will call this the vector 2-norm.

Definition 2.3 The vector 2-norm $\|\cdot\|_2 : \mathbb{C}^n \rightarrow \mathbb{R}$ is defined for $x \in \mathbb{C}^n$ by

$$\|x\|_2 = \sqrt{x^H x} = \sqrt{\chi_0 \chi_0 + \dots + \chi_{n-1} \chi_{n-1}} = \sqrt{|\chi_0|^2 + \dots + |\chi_{n-1}|^2}.$$

In an undergraduate course, you should have learned that, given real valued vector x and unit length real valued vector y , the component of x in the direction of y is given by $x^T y$:



(in this picture, $\|y\|_2 = 2$.)

The length of x is $\|x\|_2$ and the length of $x^T y$ is $|x^T y|$. Thus $|x^T y| \leq \|x\|_2$. Now, if y is not of unit length, then $|x^T(y/\|y\|_2)| \leq \|x\|_2$ or, equivalently, $|x^T y| \leq \|x\|_2 \|y\|_2$. This is known as the Cauchy-Schwartz inequality for real valued vectors. Here is its generalization for complex valued vectors.

To show that the vector 2-norm is a norm, we will need the following theorem:

Theorem 2.4 (Cauchy-Schwartz inequality) Let $x, y \in \mathbb{C}^n$. Then $|x^H y| \leq \|x\|_2 \|y\|_2$.

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

Pick $\alpha \in \mathbb{C}$ with $|\alpha| = 1$ so that $\alpha \hat{x}^H \hat{y}$ is real and nonnegative. Another way of saying this is that $\alpha \hat{x}^H \hat{y} = |x^H y|$. Note that since it is real we also know that $\alpha \hat{x}^H \hat{y} = \overline{\alpha \hat{x}^H \hat{y}} = \overline{\alpha} \hat{y}^H \hat{x}$.

Now,

$$\begin{aligned}
 0 &\leq \|\hat{x} - \alpha \hat{y}\|_2^2 \\
 &= (x - \alpha y)^H (\hat{x} - \alpha \hat{y}) && (\|z\|_2^2 = z^H z) \\
 &= \hat{x}^H \hat{x} - \overline{\alpha} \hat{y}^H \hat{x} - \alpha \hat{x}^H \hat{y} + \overline{\alpha} \alpha \hat{y}^H \hat{y} && (\text{multiplying out}) \\
 &= 1 - 2\alpha \hat{x}^H \hat{y} + |\alpha|^2 && (\|\hat{x}\|_2 = \|\hat{y}\|_2 = 1 \text{ and } \alpha \hat{x}^H \hat{y} = \overline{\alpha \hat{x}^H \hat{y}} = \overline{\alpha} \hat{y}^H \hat{x}) \\
 &= 2 - 2\alpha \hat{x}^H \hat{y} && (|\alpha| = 1) \\
 &= 2 - 2|\hat{x}^H \hat{y}| && (\alpha \hat{x}^H y = |x^H y|).
 \end{aligned}$$

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

QED

Theorem 2.5 The vector 2-norm is a norm.

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

Let $x, y \in \mathbb{C}^n$ 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 $x_j \neq 0$. Then

$$\|x\|_2 = \sqrt{|x_0|^2 + \cdots + |x_{n-1}|^2} \geq \sqrt{|x_j|^2} = |x_j| > 0.$$

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

$$\begin{aligned}
 \|\alpha x\|_2 &= \sqrt{|\alpha \chi_0|^2 + \cdots + |\alpha \chi_{n-1}|^2} \\
 &= \sqrt{|\alpha|^2 |\chi_0|^2 + \cdots + |\alpha|^2 |\chi_{n-1}|^2} \\
 &= \sqrt{|\alpha|^2 (\chi_0^2 + \cdots + \chi_{n-1}^2)} \\
 &= |\alpha| \sqrt{\chi_0^2 + \cdots + \chi_{n-1}^2} \\
 &= |\alpha| \|x\|_2.
 \end{aligned}$$

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

$$\begin{aligned}
 \|x+y\|_2^2 &= (x+y)^H (x+y) \\
 &= x^H x + y^H x + x^H y + y^H y \\
 &\leq \|x\|_2^2 + 2\|x\|_2 \|y\|_2 + \|y\|_2^2 \\
 &= (\|x\|_2 + \|y\|_2)^2.
 \end{aligned}$$

Taking the square root of both sides yields the desired result.

QED

2.3.2 Vector 1-norm

Definition 2.6 The vector 1-norm $\|\cdot\|_1 : \mathbb{C}^n \rightarrow \mathbb{R}$ is defined for $x \in \mathbb{C}^n$ by

$$\|x\|_1 = |\chi_0| + |\chi_1| + \cdots + |\chi_{n-1}|.$$

Homework 2.7 The vector 1-norm is a norm.

☞ SEE ANSWER

The vector 1-norm is sometimes referred to as the “taxi-cab norm”. It is the distance that a taxi travels along the streets of a city that has square city blocks.

2.3.3 Vector ∞ -norm (infinity norm)

Definition 2.8 The vector ∞ -norm $\|\cdot\|_\infty : \mathbb{C}^n \rightarrow \mathbb{R}$ is defined for $x \in \mathbb{C}^n$ by $\|x\|_\infty = \max_i |\chi_i|$.

Homework 2.9 The vector ∞ -norm is a norm.

☞ SEE ANSWER

2.3.4 Vector p -norm

Definition 2.10 The vector p -norm $\|\cdot\|_p : \mathbb{C}^n \rightarrow \mathbb{R}$ is defined for $x \in \mathbb{C}^n$ by

$$\|x\|_p = \sqrt[p]{|\chi_0|^p + |\chi_1|^p + \cdots + |\chi_{n-1}|^p}.$$

Proving that the p -norm is a norm is a little tricky and not particularly relevant to this course. To prove the triangle inequality requires the following classical result:

Theorem 2.11 (Hölder inequality) Let $x, y \in \mathbb{C}^n$ and $\frac{1}{p} + \frac{1}{q} = 1$ with $1 \leq p, q \leq \infty$. Then $|x^H y| \leq \|x\|_p \|y\|_q$.

Clearly, the 1-norm and 2 norms are special cases of the p -norm. Also, $\|x\|_\infty = \lim_{p \rightarrow \infty} \|x\|_p$.

2.4 Matrix Norms

It is not hard to see that vector norms are all measures of how “big” the vectors are. Similarly, we want to have measures for how “big” matrices are. We will start with one that are somewhat artificial and then move on to the important class of induced matrix norms.



▶ YouTube

▶ Downloaded Video

2.4.1 Frobenius norm

Definition 2.12 The Frobenius norm $\|\cdot\|_F : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is defined for $A \in \mathbb{C}^{m \times n}$ by

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

Notice that 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 2.13 Show that the Frobenius norm is a norm.

▶ SEE ANSWER

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.

2.4.2 Induced matrix norms

Definition 2.14 Let $\|\cdot\|_\mu : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ be vector norms. Define $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

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

Let us start by interpreting this. How “big” 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 a vector (x) is measured with norm $\|\cdot\|_\nu$ and the length of a transformed vector (Ax) is measured with norm $\|\cdot\|_\mu$.

Two comments are in order. First,

$$\sup_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_\mu}{\|x\|_\nu} = \sup_{\|x\|_\nu=1} \|Ax\|_\mu.$$

This follows immediately from the following sequence of equivalences:

$$\sup_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_\mu}{\|x\|_\nu} = \sup_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \left\| \frac{Ax}{\|x\|_\nu} \right\|_\mu = \sup_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \left\| A \frac{x}{\|x\|_\nu} \right\|_\mu = \sup_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \|Ay\|_\mu = \sup_{\substack{y \in \mathbb{C}^n \\ y \neq 0}} \|Ay\|_\mu = \sup_{\|x\|_\nu=1} \|Ax\|_\mu.$$

$y = \frac{x}{\|x\|_\nu}$

Also the “sup” (which stands for supremum) is used because we can’t claim yet that there is a vector x with $\|x\|_\nu = 1$ for which

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

In other 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 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 set. Since real analysis is not a prerequisite for this course, the reader may have to take this on faith! From real analysis we also learn that if the supremum is attained by an element in S , then $\sup_{x \in S} f(x) = \max_{x \in S} f(x)$. Thus, we 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 2.15 Let $\|\cdot\|_\mu : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ be vector norms. Define $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

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

and

Definition 2.16 Let $\|\cdot\|_\mu : \mathbb{C}^m \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ be vector norms. Define $\|\cdot\|_{\mu,\nu} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

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

In this course, we will often encounter proofs involving norms. Such proofs are often much cleaner if one starts by strategically picking the most convenient of these two definitions.

Theorem 2.17 $\|\cdot\|_{\mu,v} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is a norm.

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

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

- $A \neq 0 \Rightarrow \|A\|_{\mu,v} > 0$ ($\|\cdot\|_{\mu,v}$ 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). Let us assume it is the j th column, a_j , that is nonzero. Then

$$\|A\|_{\mu,v} = \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_\mu}{\|x\|_v} \geq \frac{\|Ae_j\|_\mu}{\|e_j\|_v} = \frac{\|a_j\|_\mu}{\|e_j\|_v} > 0.$$

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

$$\|\alpha A\|_{\mu,v} = \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|\alpha Ax\|_\mu}{\|x\|_v} = \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} |\alpha| \frac{\|Ax\|_\mu}{\|x\|_v} = |\alpha| \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_\mu}{\|x\|_v} = |\alpha| \|A\|_{\mu,v}.$$

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

$$\begin{aligned} \|A + B\|_{\mu,v} &= \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|(A + B)x\|_\mu}{\|x\|_v} = \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax + Bx\|_\mu}{\|x\|_v} \leq \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_\mu + \|Bx\|_\mu}{\|x\|_v} \\ &\leq \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \left(\frac{\|Ax\|_\mu}{\|x\|_v} + \frac{\|Bx\|_\mu}{\|x\|_v} \right) \leq \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_\mu}{\|x\|_v} + \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Bx\|_\mu}{\|x\|_v} = \|A\|_{\mu,v} + \|B\|_{\mu,v}. \end{aligned}$$

QED

2.4.3 Special cases used in practice

The most important case of $\|\cdot\|_{\mu,v} : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ uses the same norm for $\|\cdot\|_\mu$ and $\|\cdot\|_v$ (except that m may not equal n).

Definition 2.18 Define $\|\cdot\|_p : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ by

$$\|A\|_p = \max_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p=1} \|Ax\|_p.$$

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 the matrix 2-norm is difficult to compute in practice while the 1-norm, ∞ -norm, and Frobenius norms are straightforward and relatively cheap to compute.

The following theorem shows how to practically compute the matrix 1-norm:

Theorem 2.19 Let $A \in \mathbb{C}^{m \times n}$ and partition $A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$. Show that

$$\|A\|_1 = \max_{0 \leq j < n} \|a_j\|_1.$$

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

$$\begin{aligned} \max_{\|x\|_1=1} \|Ax\|_1 &= \max_{\|x\|_1=1} \left\| \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \right\|_1 \\ &= \max_{\|x\|_1=1} \|\chi_0 a_0 + \chi_1 a_1 + \cdots + \chi_{n-1} a_{n-1}\|_1 \\ &\leq \max_{\|x\|_1=1} (\|\chi_0 a_0\|_1 + \|\chi_1 a_1\|_1 + \cdots + \|\chi_{n-1} a_{n-1}\|_1) \\ &= \max_{\|x\|_1=1} (|\chi_0| \|a_0\|_1 + |\chi_1| \|a_1\|_1 + \cdots + |\chi_{n-1}| \|a_{n-1}\|_1) \\ &\leq \max_{\|x\|_1=1} (|\chi_0| \|a_J\|_1 + |\chi_1| \|a_J\|_1 + \cdots + |\chi_{n-1}| \|a_J\|_1) \\ &= \max_{\|x\|_1=1} (|\chi_0| + |\chi_1| + \cdots + |\chi_{n-1}|) \|a_J\|_1 \\ &= \|a_J\|_1. \end{aligned}$$

Also,

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

Hence

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

which implies that

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

QED

Similarly, the following exercise shows how to practically compute the matrix ∞ -norm:

Homework 2.20 Let $A \in \mathbb{C}^{m \times n}$ and partition $A = \left(\begin{array}{c} \widehat{a}_0^T \\ \hline \widehat{a}_1^T \\ \vdots \\ \hline \widehat{a}_{m-1}^T \end{array} \right)$. Show that

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

SEE ANSWER

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

Homework 2.21 Let $y \in \mathbb{C}^m$ and $x \in \mathbb{C}^n$. Show that $\|yx^H\|_2 = \|y\|_2 \|x\|_2$.

SEE ANSWER

2.4.4 Discussion

While $\|\cdot\|_2$ is a very important matrix norm, it is in practice often difficult to compute. The matrix norms, $\|\cdot\|_F$, $\|\cdot\|_1$, and $\|\cdot\|_\infty$ are more easily computed and hence more practical in many instances.

2.4.5 Submultiplicative norms

Definition 2.22 A matrix norm $\|\cdot\|_v : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is said to be submultiplicative (consistent) if it also satisfies

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

Theorem 2.23 Let $\|\cdot\|_v : \mathbb{C}^n \rightarrow \mathbb{R}$ be a vector norm and given any matrix $C \in \mathbb{C}^{m \times n}$ define the corresponding induced matrix norm as

$$\|C\|_v = \max_{x \neq 0} \frac{\|Cx\|_v}{\|x\|_v} = \max_{\|x\|_v=1} \|Cx\|_v.$$

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

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

Lemma 2.24 Let $\|\cdot\|_v : \mathbb{C}^n \rightarrow \mathbb{R}$ be a vector norm and given any matrix $C \in \mathbb{C}^{m \times n}$ define the corresponding induced matrix norm as

$$\|C\|_v = \max_{x \neq 0} \frac{\|Cx\|_v}{\|x\|_v} = \max_{\|x\|_v=1} \|Cx\|_v.$$

Then for any $A \in \mathbb{C}^{m \times n}$ and $y \in \mathbb{C}^n$ the inequality $\|Ay\|_v \leq \|A\|_v \|y\|_v$ holds.

Proof: If $y = 0$, the result obviously holds since then $\|Ay\|_v = 0$ and $\|y\|_v = 0$. Let $y \neq 0$. Then

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

Rearranging this yields $\|Ay\|_v \leq \|A\|_v \|y\|_v$.

QED

We can now prove the theorem:

Proof:

$$\|AB\|_v = \max_{\|x\|_v=1} \|ABx\|_v = \max_{\|x\|_v=1} \|A(Bx)\|_v \leq \max_{\|x\|_v=1} \|A\|_v \|Bx\|_v \leq \max_{\|x\|_v=1} \|A\|_v \|B\|_v \|x\|_v = \|A\|_v \|B\|_v.$$

QED

Homework 2.25 Show that $\|Ax\|_\mu \leq \|A\|_{\mu,v} \|x\|_v$.

☞ SEE ANSWER

Homework 2.26 Show that $\|AB\|_\mu \leq \|A\|_{\mu,v} \|B\|_v$.

☞ SEE ANSWER

Homework 2.27 Show that the Frobenius norm, $\|\cdot\|_F$, is submultiplicative.

☞ SEE ANSWER

2.5 An Application to Conditioning of Linear Systems

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 $b + \delta b$. (Notice how that 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 propagates into a potential error in the solution x .

This is summarized as follows:

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

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

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

We will assume that A has an inverse. To find an expression for $\kappa(A, b, \delta b)$, we notice that

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

and from this

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

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

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

From this we conclude that

$$\begin{aligned} \frac{1}{\|x\|} &\leq \|A\| \frac{1}{\|b\|} \\ \|\delta x\| &\leq \|A^{-1}\| \|\delta b\|. \end{aligned}$$

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 :

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

$\kappa(A) = \|A\| \|A^{-1}\|$ is called the *condition number* of 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, unfortunately, “yes!”, as we will show next.

Notice that

- There is an \hat{x} for which

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

namely the x for which the maximum is attained. Pick $\hat{b} = A\hat{x}$.

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

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

again, the x for which the maximum is attained.

It is when solving the perturbed system

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

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

Homework 2.28 Let $\|\cdot\|$ be a matrix norm induced by the $\|\cdot\|$ vector norm. Show that $\kappa(A) = \|A\| \|A^{-1}\| \geq 1$.

 [SEE ANSWER](#)

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 $\kappa(A) = 1$).

2.6 Equivalence of Norms

Many results we encounter show that the norm of a particular vector or matrix is small. Obviously, it would be unfortunate if a vector or matrix is large in one norm and small in another norm. The following result shows that, modulo a constant, all norms are equivalent. Thus, if the vector is small in one norm, it is small in other norms as well.

Theorem 2.29 Let $\|\cdot\|_\mu : \mathbb{C}^n \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^n \rightarrow \mathbb{R}$ be vector norms. Then there exist constants $\alpha_{\mu,\nu}$ and $\beta_{\mu,\nu}$ such that for all $x \in \mathbb{C}^n$

$$\alpha_{\mu,\nu} \|x\|_\mu \leq \|x\|_\nu \leq \beta_{\mu,\nu} \|x\|_\mu.$$

The proof of this result again uses the fact that the supremum of a function on a compact set is attained. A similar result holds for matrix norms:

Theorem 2.30 Let $\|\cdot\|_\mu : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ and $\|\cdot\|_\nu : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ be matrix norms. Then there exist constants $\alpha_{\mu,\nu}$ and $\beta_{\mu,\nu}$ such that for all $A \in \mathbb{C}^{m \times n}$

$$\alpha_{\mu,\nu} \|A\|_\mu \leq \|A\|_\nu \leq \beta_{\mu,\nu} \|A\|_\mu.$$

2.7 Enrichments

2.7.1 Practical computation of the vector 2-norm

Consider the computation $\gamma = \sqrt{\alpha^2 + \beta^2}$ where $\alpha, \beta \in \mathbb{R}$. When computing this with floating point numbers, a fundamental problem is that the intermediate values α^2 and β^2 may overflow (become larger than the largest number that can be stored) or underflow (become smaller than the smallest positive number that can be stored), even if the resulting γ itself does not overflow or underflow.

The solution is to first determine the largest value, $\mu = \max(|\alpha|, |\beta|)$ and then compute

$$\gamma = \mu \sqrt{\left(\frac{\alpha}{\mu}\right)^2 + \left(\frac{\beta}{\mu}\right)^2}$$

instead. A careful analysis shows that if γ does not overflow, neither do any of the intermediate values encountered during its computation. While one of the terms $\left(\frac{\alpha}{\mu}\right)^2$ or $\left(\frac{\beta}{\mu}\right)^2$ may underflow, the other one equals one and hence the overall result does not underflow. A complete discussion of all the intricacies go beyond this note.

This insight generalizes to the computation of $\|x\|_2$ where $x \in \mathbb{C}^n$. Rather than computing it as

$$\|x\|_2 = \sqrt{|\chi_0|^2 + |\chi_1|^2 + \cdots + |\chi_{n-1}|^2}$$

and risk overflow or underflow, instead the following computation is used:

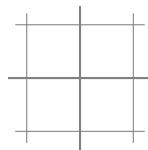
$$\begin{aligned} \mu &= \|x\|_\infty \\ \|x\|_2 &= \mu \sqrt{\left(\frac{|\chi_0|}{\mu}\right)^2 + \left(\frac{|\chi_1|}{\mu}\right)^2 + \cdots + \left(\frac{|\chi_{n-1}|}{\mu}\right)^2}. \end{aligned}$$

2.8 Wrapup

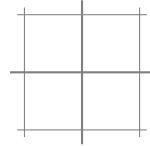
2.8.1 Additional exercises

Homework 2.31 A vector $x \in \mathbb{R}^2$ can be represented by the point to which it points when rooted at the origin. For example, the vector $x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ can be represented by the point $(2, 1)$. With this in mind, plot

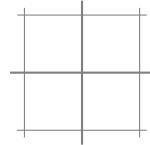
1. The points corresponding to the set $\{x \mid \|x\|_2 = 1\}$.



2. The points corresponding to the set $\{x \mid \|x\|_1 = 1\}$.



3. The points corresponding to the set $\{x \mid \|x\|_\infty = 1\}$.



SEE ANSWER

Homework 2.32 Consider

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

1. $\|A\|_1 =$

2. $\|A\|_\infty =$

3. $\|A\|_F =$

SEE ANSWER

Homework 2.33 Show that for all $x \in \mathbb{C}^n$

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

2. $\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty.$

3. $(1/\sqrt{n})\|x\|_2 \leq \|x\|_\infty \leq \sqrt{n}\|x\|_2.$

SEE ANSWER

Homework 2.34 (I need to double check that this is true!)

Prove that if for all x $\|x\|_v \leq \beta\|x\|_\mu$ then $\|A\|_v \leq \beta\|A\|_{\mu,v}.$

SEE ANSWER

Homework 2.35 Partition

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

Prove that

$$1. \|A\|_F = \|A^T\|_F.$$

$$2. \|A\|_F = \sqrt{\|a_0\|_2^2 + \|a_1\|_2^2 + \cdots + \|a_{n-1}\|_2^2}.$$

$$3. \|A\|_F = \sqrt{\|\tilde{a}_0\|_2^2 + \|\tilde{a}_1\|_2^2 + \cdots + \|\tilde{a}_{m-1}\|_2^2}.$$

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

[SEE ANSWER](#)

Homework 2.36 Let for $e_j \in \mathbb{R}^n$ (a standard basis vector), compute

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

[SEE ANSWER](#)

Homework 2.37 Let for $I \in \mathbb{R}^{n \times n}$ (the identity matrix), compute

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

[SEE ANSWER](#)

Homework 2.38 Let $\|\cdot\|$ be a vector norm defined for a vector of any size (arbitrary n). Let $\|\cdot\|$ be the induced matrix norm. Prove that $\|I\| = 1$ (where I equals the identity matrix).

Conclude that $\|I\|_p = 1$ for any p -norm.

[SEE ANSWER](#)

Homework 2.39 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

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

[SEE ANSWER](#)

Homework 2.40 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). Then

$$\|D\|_p =$$

Prove your answer.

 SEE ANSWER

Homework 2.41 Let $y \in \mathbb{C}^n$. Show that $\|y^H\|_2 = \|y\|_2$.

 SEE ANSWER

2.8.2 Summary

Chapter 3

Notes on Orthogonality and the Singular Value Decomposition

If you need to review the basics of orthogonal vectors, orthogonal spaces, and related topics, you may want to consult weeks Weeks 9-11 of

[Linear Algebra: Foundations to Frontiers - Notes to LAFF With \[30\].](#)

Video

The following videos were recorded for the Fall 2014 offering of “Numerical Analysis: Linear Algebra”.
Read disclaimer regarding the videos in the preface!

 [YouTube Part 1](#)

 [YouTube Part 2](#)

 [Download Part 1 from UT Box](#)

 [Download Part 2 from UT Box](#)

 [View Part 1 After Local Download](#)

 [View Part 2 After Local Download](#)

(For help on viewing, see Appendix A.)

3.1 Opening Remarks

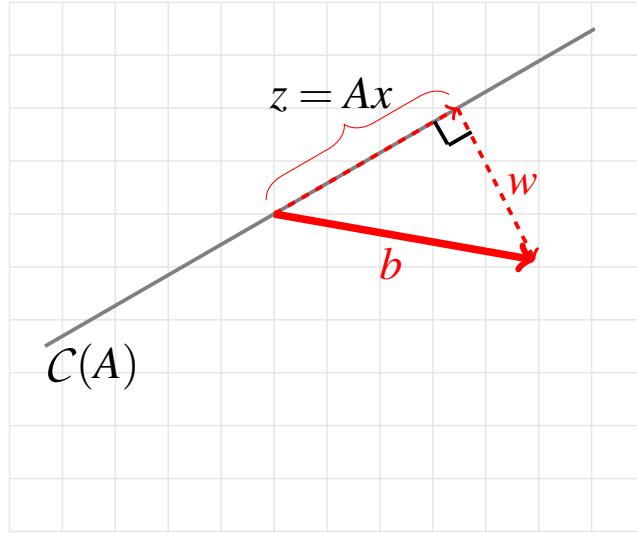
3.1.1 Launch: Orthogonal projection and its application



 [YouTube](#)
 [Downloaded Video](#)

A quick review of orthogonal projection

Consider the following picture:



Here we consider

- A , a matrix in $\mathbb{R}^{m \times n}$.
- $C(A)$, the space spanned by the columns of A (the column space of A).
- b , a vector in \mathbb{R}^m .
- z , the component of b in $C(A)$ which is also the vector in $C(A)$ closest to the vector b . We know that since this vector is in the column space of A it equals $z = Ax$ for some vector $x \in \mathbb{R}^n$.
- w , the component of b orthogonal to $C(A)$.

The vectors b, z, w , all exist in the same planar subspace since $b = z + w$, which is the page on which these vectors are drawn in the above picture.

Thus,

$$b = z + w,$$

where

- $z = Ax$ with $x \in \mathbb{R}^n$; and
- $A^T w = 0$ since w is orthogonal to the column space of A and hence in $\mathcal{N}(A^T)$ (the left null space of A).

Noting that $w = b - z$ we find that

$$0 = A^T w = A^T(b - z) = A^T(b - Ax)$$

or, equivalently,

$$A^T Ax = A^T b.$$

This is known as the normal equation for finding the vector x that “best” solves $Ax = b$ in the linear least-squares sense. More on this later in our course.

Then, provided $(A^T A)^{-1}$ exists (which happens when A has linearly independent columns),

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

Thus, the component of b in $\mathcal{C}(A)$ is given by

$$z = Ax = A(A^T A)^{-1} A^T b$$

while the component of b orthogonal (perpendicular) to $\mathcal{C}(A)$ is given by

$$w = b - z = b - A(A^T A)^{-1} A^T b = Ib - A(A^T A)^{-1} A^T b = (I - A(A^T A)^{-1} A^T) b.$$

Summarizing:

$$\begin{aligned} z &= A(A^T A)^{-1} A^T b \\ w &= (I - A(A^T A)^{-1} A^T) b. \end{aligned}$$

Now, we say that, given matrix A with linearly independent columns, the matrix that *projects* (orthogonally) a given vector b onto the column space of A is given by

$$A(A^T A)^{-1} A^T$$

since $A(A^T A)^{-1} A^T b$ is the component of b in $\mathcal{C}(A)$. Similarly, given matrix A with linearly independent columns, the matrix that *projects* a given vector b onto the space orthogonal to the column space of A (which, recall, is the *left null space* of A) is given by

$$I - A(A^T A)^{-1} A^T$$

since $(I - A(A^T A)^{-1} A^T) b$ is the component of b in $\mathcal{C}(A)^\perp = \mathcal{N}(A^T)$.

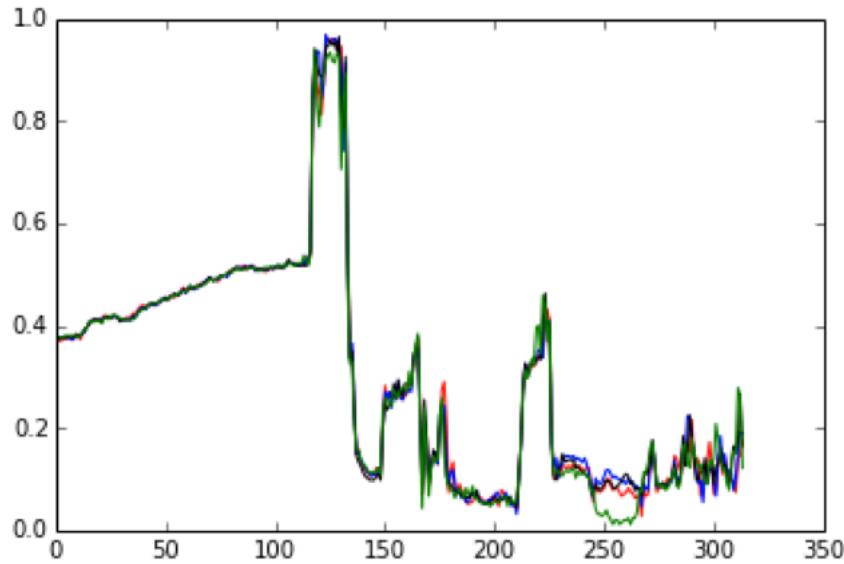
An application to data compression

Consider the picture



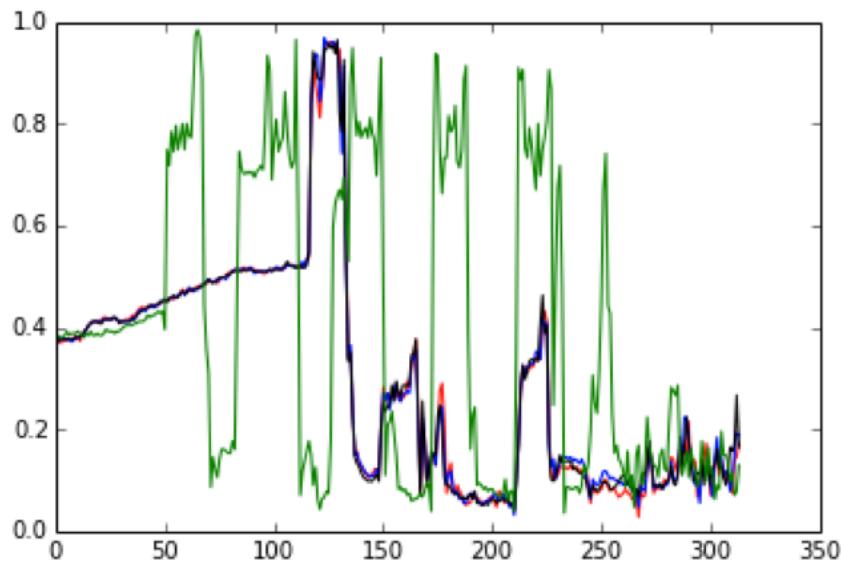
This picture can be thought of as a matrix $B \in \mathbb{R}^{m \times n}$ where each element in the matrix encodes a pixel in the picture. The j th column of B then encodes the j th column of pixels in the picture.

Now, let's focus on the first few columns. Notice that there is a lot of similarity in those columns. This can be illustrated by plotting the values in the column as a function of the index of the element in the column:

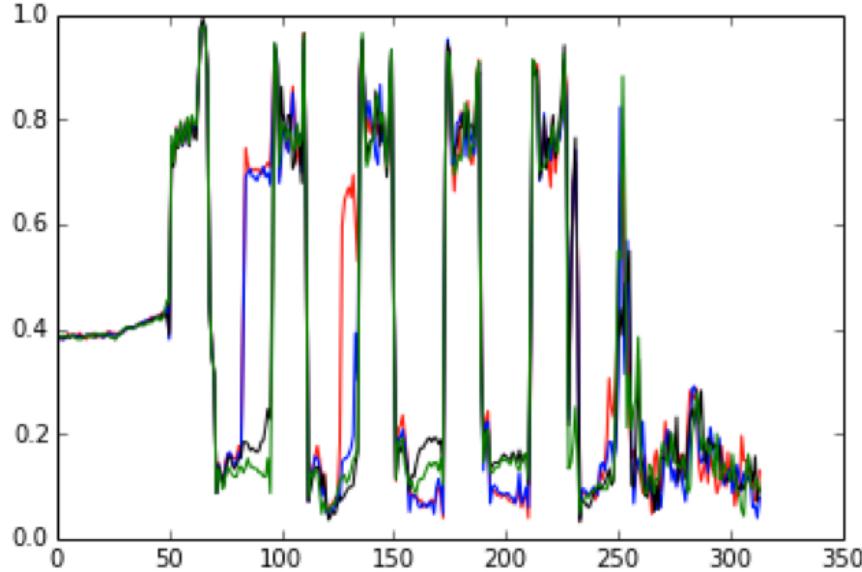


We plot $\beta_{i,j}$, the value of the (i, j) pixel, for $j = 0, 1, 2, 3$ in different colors. The green line corresponds to $j = 3$ and you notice that it is starting to deviate some for i near 250.

If we now instead look at columns $j = 0, 1, 2, 100$, where the green line corresponds to $j = 100$, we see that the curve corresponding to that column is dramatically different:



Changing this to plotting $j = 100, 101, 102, 103$ and we notice a lot of similarity again:



Approximating the picture with one column

Now, let's think about this from the point of view taking one vector, say the first column of B , and projecting the other columns onto the span of that vector. The hope is that the vector is representatives and that therefore the projections of the columns onto that span of that vector is a good approximation for each of the columns. What does this mean?

- Partition B into columns: $B = \left(b_0 \mid b_1 \mid \cdots \mid b_{n-1} \right)$.
- Pick $a = b_0$ to be representative vector for all columns of B .
- Focus on projecting b_0 onto $\text{Span}(\{a\})$. Another way of thinking of this is that we take $A = \left(a \right)$ and project onto $C(A)$.

$$A(A^T A)^{-1} A^T b_0 = a(a^T a)^{-1} a^T b_0 = \underbrace{a(a^T a)^{-1} a^T a}_{\text{Since } b_0 = a} = a.$$

- Next, focus on projecting b_1 onto $\text{Span}(\{a\})$:

$$a(a^T a)^{-1} a^T b_1 \approx a$$

since b_1 is very close to b_0 .

- Do this for all columns, and create a picture with all of the projected vectors by viewing the result again as a matrix:

$$\left(a(a^T a)^{-1} a^T b_0 \mid a(a^T a)^{-1} a^T b_1 \mid a(a^T a)^{-1} a^T b_2 \mid \cdots \right)$$

- Now, remember that if T is some matrix, then

$$TB = \left(Tb_0 \mid Tb_1 \mid Tb_2 \mid \cdots \right).$$

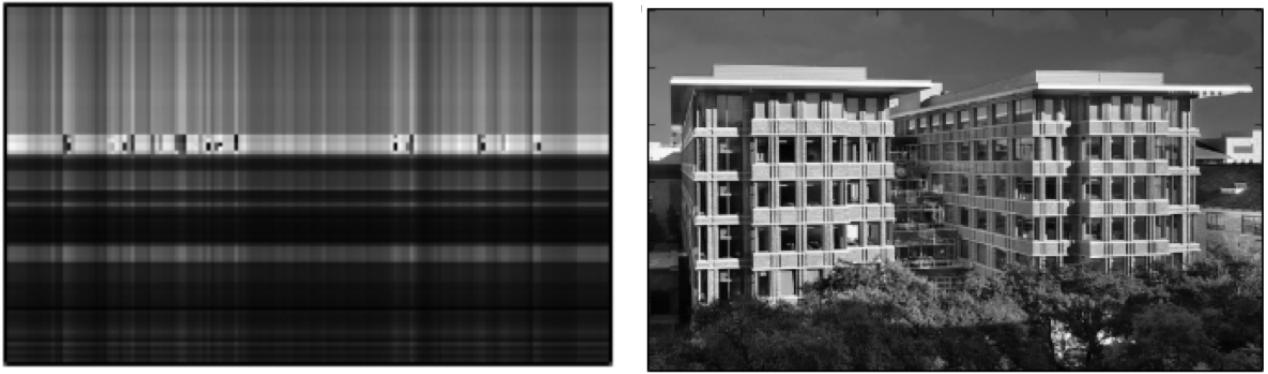
If we let $T = a(a^T a)^{-1}a^T$ (the matrix that projects onto $\text{Span}(\{a\})$), then

$$a(a^T a)^{-1}a^T \left(\begin{array}{c|c|c|c} b_0 & b_1 & b_2 & \cdots \end{array} \right) = a(a^T a)^{-1}a^T B.$$

- We can manipulate this further:

$$a(a^T a)^{-1}a^T B = a \underbrace{\left((a^T a)^{-1} B^T a \right)}_w^T = aw^T$$

- Finally, we recognize aw^T as an outer product (a column vector times a row vector) and hence a matrix of rank one.
- If we do this for our picture, we get the approximation on the left:



Notice how it seems like each column is the same, except with some constant change in the gray-scale. The same is true for rows. Why is this? If you focus on the left-most columns in the picture, they almost look correct (comparing to the left-most columns in the picture on the right). Why is this?

- The benefit of the approximation on the left is that it can be described with two vectors: a and w ($n + m$ floating point numbers) while the original matrix on the right required an entire matrix ($m \times n$ floating point numbers).

The disadvantage of the approximation on the left is that it is hard to recognize the original picture...

What if we take two vectors instead, say column $j = 0$ and $j = n/2$, and projected each of the columns onto the subspace spanned by those two vectors?

- Partition B into columns: $B = \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right)$.
- Pick $A = \left(\begin{array}{c|c} a_0 & a_1 \end{array} \right) = \left(\begin{array}{c|c} b_0 & b_{n/2} \end{array} \right)$.
- Focus on projecting b_0 onto $\text{Span}(\{a_0, a_1\}) = \mathcal{C}(A)$:

$$A(A^T A)^{-1}A^T b_0 = a$$

because a is in $\mathcal{C}(A)$ and a is therefore the best vector in $\mathcal{C}(A)$.

- Next, focus on projecting b_1 onto $\text{Span}(\{a\})$:

$$A(A^T A)^{-1} A^T b_1 \approx a$$

since b_1 is very close to b_0 .

- Do this for all columns

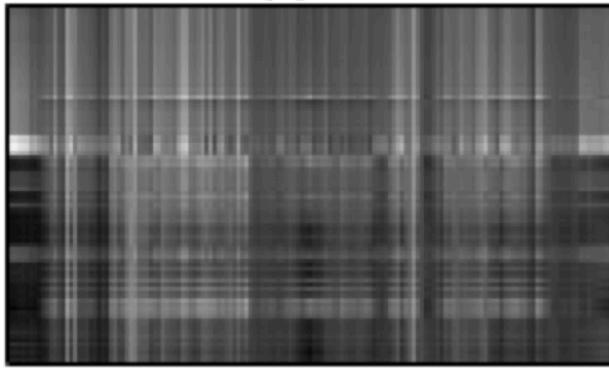
$$\begin{aligned} \left(\begin{array}{c|c|c|c} A(A^T A)^{-1} A^T b_0 & A(A^T A)^{-1} A^T b_1 & A(A^T A)^{-1} A^T b_2 & \dots \end{array} \right) &= A(A^T A)^{-1} A^T \left(\begin{array}{c|c|c|c} b_0 & b_1 & b_2 & \dots \end{array} \right) \\ &= A \underbrace{(A^T A)^{-1} A^T B}_{W^T} = AW^T. \end{aligned}$$

- Notice that A and W each have two columns and AW^T is the sum of two outer products:

$$AW^T = \left(\begin{array}{c|c} a_0 & a_1 \end{array} \right) \underbrace{\left(\begin{array}{c|c} w_0 & w_1 \end{array} \right)^T}_{W^T} = \left(\begin{array}{c|c} a_0 & a_1 \end{array} \right) \begin{pmatrix} w_0^T \\ w_1^T \end{pmatrix} = a_0 w_0^T + a_1 w_1^T.$$

It can be easily shown that this matrix has rank at most two, which is why AW^T is called a rank-2 approximation of B .

- We can visualize this with the following picture on the left:



We are starting to see some more detail.

- We now have to store only $n \times 2$ and $m \times 2$ matrices A and W .

Rank-k approximations

We can continue the above by picking progressively more columns for A . The progression of pictures in Figure 3.1 shows the improvement as more and more columns are used, where k indicates the number of columns.

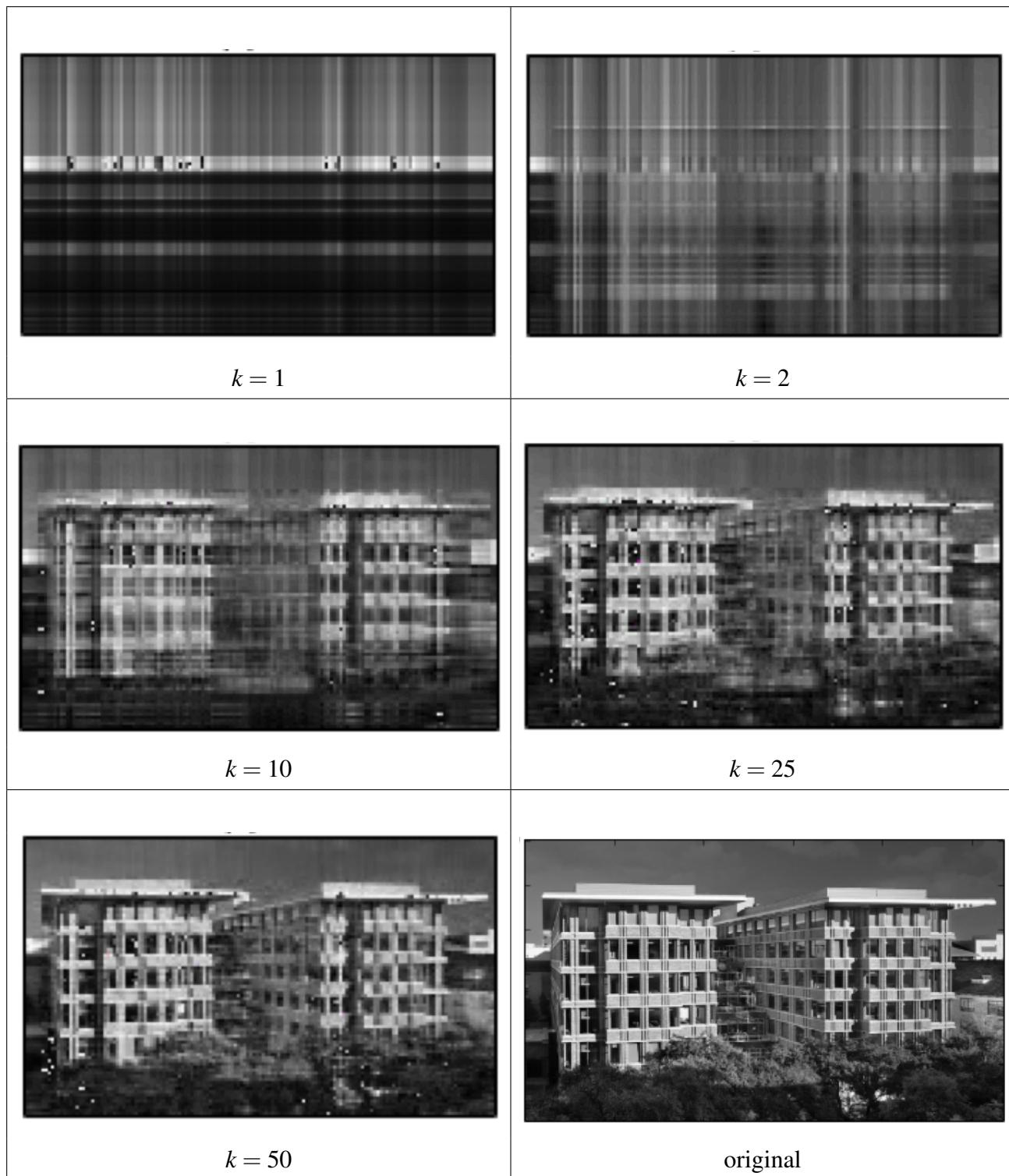


Figure 3.1: Progression of approximations.

The Singular Value Decomposition

Picking vectors for matrix A from the columns of the original picture does not usually yield an optimal approximation. How then does one compute the *best* rank- k approximation of a matrix so that the amount of data that must be stored best captures the matrix? This is where the Singular Value Decomposition (SVD) comes in.

The SVD is probably the most important result in linear algebra.

3.1.2 Outline

Video	57
3.1 Opening Remarks	57
3.1.1 Launch: Orthogonal projection and its application	57
3.1.2 Outline	66
3.1.3 What you will learn	67
3.2 Orthogonality and Unitary Matrices	68
3.3 Toward the SVD	71
3.4 <i>The Theorem</i>	73
3.5 Geometric Interpretation	73
3.6 Consequences of the SVD Theorem	77
3.7 Projection onto the Column Space	81
3.8 Low-rank Approximation of a Matrix	83
3.9 An Application	84
3.10 SVD and the Condition Number of a Matrix	86
3.11 An Algorithm for Computing the SVD?	87
3.12 Wrapup	88
3.12.1 Additional exercises	88
3.12.2 Summary	88

3.1.3 What you will learn

3.2 Orthogonality and Unitary Matrices

In this section, we extend what you learned in an undergraduate course about orthogonality to complex valued vectors and matrices.

Definition 3.1 Let $u, v \in \mathbb{C}^m$. These vectors are said to be orthogonal (perpendicular) if $u^H v = 0$.

Definition 3.2 Let $q_0, q_1, \dots, q_{n-1} \in \mathbb{C}^m$. These vectors are said to be mutually orthonormal if for all $0 \leq i, j < n$

$$q_i^H q_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}.$$

The definition implies that $\|q_i\|_2 = \sqrt{q_i^H q_i} = 1$ and hence each of the vectors is of unit length in addition to being orthogonal to each other.

For n vectors of length 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 length 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 3.3 Let $Q \in \mathbb{C}^{m \times n}$ (with $n \leq m$). Then Q is said to be an orthonormal matrix if $Q^H Q = I$.

The subsequent exercise makes the connection between mutually orthonormal vectors and an orthogonal matrix.

Homework 3.4 Let $Q \in \mathbb{C}^{m \times n}$ (with $n \leq m$). Partition $Q = \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array} \right)$. Show that Q is an orthonormal matrix if and only if q_0, q_1, \dots, q_{n-1} are mutually orthonormal.

 [SEE ANSWER](#)

If an orthogonal matrices is square, then it is called a *unitary* matrix.

Definition 3.5 Let $Q \in \mathbb{C}^{m \times m}$. Then Q is said to be a unitary matrix if $Q^H Q = I$ (the identity).

Unitary matrices are always square and only square matrices can be unitary. 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 exercise.

Homework 3.6 Let $Q \in \mathbb{C}^{m \times m}$. Show that if Q is unitary then $Q^{-1} = Q^H$ and $QQ^H = I$.

 [SEE ANSWER](#)

Homework 3.7 Let $Q_0, Q_1 \in \mathbb{C}^{m \times m}$ both be unitary. Show that their product, $Q_0 Q_1$, is unitary.

 [SEE ANSWER](#)

Homework 3.8 Let $Q_0, Q_1, \dots, Q_{k-1} \in \mathbb{C}^{m \times m}$ all be unitary. Show that their product, $Q_0 Q_1 \cdots Q_{k-1}$, is unitary.

[SEE ANSWER](#)

The following is a very important observation: Let Q be a unitary matrix with

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

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

$$\begin{aligned} x &= QQ^H x = \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{m-1} \end{array} \right) \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{m-1} \end{array} \right)^H x \\ &= \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{m-1} \end{array} \right) \begin{pmatrix} q_0^H \\ q_1^H \\ \vdots \\ q_{m-1}^H \end{pmatrix} x \\ &= \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{m-1} \end{array} \right) \begin{pmatrix} q_0^H x \\ q_1^H x \\ \vdots \\ q_{m-1}^H x \end{pmatrix} \\ &= (q_0^H x)q_0 + (q_1^H x)q_1 + \cdots + (q_{m-1}^H x)q_{m-1}. \end{aligned}$$

What does this mean?

- The vector $x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{m-1} \end{pmatrix}$ gives the coefficients when the vector x is written as a linear combination of the unit basis vectors:

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

- The vector

$$Q^H x = \begin{pmatrix} q_0^H x \\ q_1^H x \\ \vdots \\ q_{m-1}^H x \end{pmatrix}$$

gives the coefficients when the vector x is written as a linear combination of the orthonormal vectors q_0, q_1, \dots, q_{m-1} :

$$x = (q_0^H x)q_0 + (q_1^H x)q_1 + \cdots + (q_{m-1}^H x)q_{m-1}.$$

- The vector $(q_i^H x)q_i$ equals the component of x in the direction of vector q_i .

Another way of looking at this is that if q_0, q_1, \dots, q_{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:

$$x = \alpha_0 q_0 + \alpha_1 q_1 + \cdots + \alpha_{m-1} q_{m-1}.$$

Now,

$$\begin{aligned} q_i^H x &= q_i^H (\alpha_0 q_0 + \alpha_1 q_1 + \cdots + \alpha_{i-1} q_{i-1} + \alpha_i q_i + \alpha_{i+1} q_{i+1} + \cdots + \alpha_{m-1} q_{m-1}) \\ &= \underbrace{\alpha_0 q_i^H q_0}_0 + \underbrace{\alpha_1 q_i^H q_1}_0 + \cdots + \underbrace{\alpha_{i-1} q_i^H q_{i-1}}_0 \\ &\quad + \underbrace{\alpha_i q_i^H q_i}_1 + \underbrace{\alpha_{i+1} q_i^H q_{i+1}}_0 + \cdots + \underbrace{\alpha_{m-1} q_i^H q_{m-1}}_0 \\ &= \alpha_i. \end{aligned}$$

Thus $q_i^H x = \alpha_i$, the coefficient that multiplies q_i .

The point is that given vector x and unitary matrix Q , $Q^H x$ computes the coefficients for the orthonormal basis consisting of the columns of matrix Q . Unitary matrices allow one to elegantly change between orthonormal basis.

Multiplication by a unitary matrix preserves length (since changing the basis for a vector does not change its length when the basis is orthonormal).

Homework 3.9 Let $U \in \mathbb{C}^{m \times m}$ be unitary and $x \in \mathbb{C}^m$, then $\|Ux\|_2 = \|x\|_2$.

 [SEE ANSWER](#)

Homework 3.10 Let $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ be unitary matrices and $A \in \mathbb{C}^{m \times n}$. Then

$$\|UA\|_2 = \|AV\|_2 = \|A\|_2.$$

 [SEE ANSWER](#)

Homework 3.11 Let $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ be unitary matrices and $A \in \mathbb{C}^{m \times n}$. Then

$$\|UA\|_F = \|AV\|_F = \|A\|_F.$$

 [SEE ANSWER](#)

3.3 Toward the SVD

▶ YouTube
▶ Downloaded Video

In this section, we lay the foundation for the Singular Value Decomposition. We come very close: the following lemma differs from the Singular Value Decomposition Theorem (discussed later in this chapter) only in that it doesn't yet guarantee that the diagonal elements of D are ordered from largest to smallest.

Lemma 3.12 *Given $A \in \mathbb{C}^{m \times n}$ there exists unitary $U \in \mathbb{C}^{m \times m}$, unitary $V \in \mathbb{C}^{n \times n}$, and diagonal $D \in \mathbb{R}^{m \times n}$ such that $A = UDV^H$ where $D = \begin{pmatrix} D_{TL} & 0 \\ 0 & 0 \end{pmatrix}$ with $D_{TL} = \text{diag}(\delta_0, \dots, \delta_{r-1})$ and $\delta_i > 0$ for $0 \leq i < r$.*

Proof: First, let us observe that if $A = 0$ (the zero matrix) then the theorem trivially holds: $A = UDV^H$ where $U = I_{m \times m}$, $V = I_{n \times n}$, and $D = \begin{pmatrix} & \\ & \end{pmatrix}$, so that D_{TL} is 0×0 . Thus, w.l.o.g. assume that $A \neq 0$.

We will prove this for $m \geq n$, leaving the case where $m \leq n$ as an exercise. The proof employs induction on n .

- **Base case:** $n = 1$. In this case $A = \begin{pmatrix} a_0 \end{pmatrix}$ where $a_0 \in \mathbb{R}^m$ is its only column. By assumption, $a_0 \neq 0$. Then

$$A = \begin{pmatrix} a_0 \end{pmatrix} = \begin{pmatrix} u_0 \end{pmatrix} (\|a_0\|_2) \begin{pmatrix} 1 \end{pmatrix}^H$$

where $u_0 = a_0 / \|a_0\|_2$. Choose $U_1 \in \mathbb{C}^{m \times (m-1)}$ so that $U = \begin{pmatrix} u_0 & | & U_1 \end{pmatrix}$ is unitary. Then

$$A = \begin{pmatrix} a_0 \end{pmatrix} = \begin{pmatrix} u_0 \end{pmatrix} (\|a_0\|_2) \begin{pmatrix} 1 \end{pmatrix}^H = \begin{pmatrix} u_0 & | & U_1 \end{pmatrix} \begin{pmatrix} \frac{\|a_0\|_2}{0} & | & \\ & | & \end{pmatrix} \begin{pmatrix} 1 \end{pmatrix}^H = UDV^H$$

where $D_{TL} = \begin{pmatrix} \delta_0 \end{pmatrix} = \begin{pmatrix} \|a_0\|_2 \end{pmatrix}$ and $V = \begin{pmatrix} 1 \end{pmatrix}$.

- **Inductive step:** Assume the result is true for all matrices with $1 \leq k < n$ columns. Show that it is true for matrices with n columns.

Let $A \in \mathbb{C}^{m \times n}$ with $n \geq 2$. W.l.o.g., $A \neq 0$ so that $\|A\|_2 \neq 0$. Let δ_0 and $v_0 \in \mathbb{C}^n$ have the property that $\|v_0\|_2 = 1$ and $\delta_0 = \|Av_0\|_2 = \|A\|_2$. (In other words, v_0 is the vector that maximizes $\max_{\|x\|_2=1} \|Ax\|_2$.) Let $u_0 = Av_0 / \delta_0$. Note that $\|u_0\|_2 = 1$. Choose $U_1 \in \mathbb{C}^{m \times (m-1)}$ and $V_1 \in \mathbb{C}^{n \times (n-1)}$ so that $\tilde{U} = \begin{pmatrix} u_0 & | & U_1 \end{pmatrix}$ and $\tilde{V} = \begin{pmatrix} v_0 & | & V_1 \end{pmatrix}$ are unitary. Then

$$\tilde{U}^H A \tilde{V} = \begin{pmatrix} u_0 & | & U_1 \end{pmatrix}^H A \begin{pmatrix} v_0 & | & V_1 \end{pmatrix}$$

$$= \begin{pmatrix} u_0^H A v_0 & | & u_0^H A V_1 \\ U_1^H A v_0 & | & U_1^H A V_1 \end{pmatrix} = \begin{pmatrix} \delta_0 u_0^H u_0 & | & u_0^H A V_1 \\ \delta_0 U_1^H u_0 & | & U_1^H A V_1 \end{pmatrix} = \begin{pmatrix} \delta_0 & | & w^H \\ 0 & | & B \end{pmatrix},$$

where $w = V_1^H A^H u_0$ and $B = U_1^H A V_1$. Now, we will argue that $w = 0$, the zero vector of appropriate size:

$$\begin{aligned} \delta_0^2 = \|A\|_2^2 &= \|U^H A V\|_2^2 = \max_{x \neq 0} \frac{\|U^H A V x\|_2^2}{\|x\|_2^2} = \max_{x \neq 0} \frac{\left\| \begin{pmatrix} \delta_0 & | & w^H \\ 0 & | & B \end{pmatrix} x \right\|_2^2}{\|x\|_2^2} \\ &\geq \frac{\left\| \begin{pmatrix} \delta_0 & | & w^H \\ 0 & | & B \end{pmatrix} \begin{pmatrix} \delta_0 \\ w \end{pmatrix} \right\|_2^2}{\left\| \begin{pmatrix} \delta_0 \\ w \end{pmatrix} \right\|_2^2} = \frac{\left\| \begin{pmatrix} \delta_0^2 + w^H w \\ Bw \end{pmatrix} \right\|_2^2}{\left\| \begin{pmatrix} \delta_0 \\ w \end{pmatrix} \right\|_2^2} \\ &\geq \frac{(\delta_0^2 + w^H w)^2}{\delta_0^2 + w^H w} = \delta_0^2 + w^H w. \end{aligned}$$

Thus $\delta_0^2 \geq \delta_0^2 + w^H w$ which means that $w = 0$ and $\tilde{U}^H A \tilde{V} = \begin{pmatrix} \delta_0 & | & 0 \\ 0 & | & B \end{pmatrix}$.

By the induction hypothesis, there exists unitary $\check{U} \in \mathbb{C}^{(m-1) \times (m-1)}$, unitary $\check{V} \in \mathbb{C}^{(n-1) \times (n-1)}$, and $\check{D} \in \mathbb{R}^{(m-1) \times (n-1)}$ such that $B = \check{U} \check{D} \check{V}^H$ where $\check{D} = \begin{pmatrix} \check{D}_{TL} & | & 0 \\ 0 & | & 0 \end{pmatrix}$ with $\check{D}_{TL} = \text{diag}(\delta_1, \dots, \delta_{r-1})$.

Now, let

$$U = \tilde{U} \begin{pmatrix} 1 & | & 0 \\ 0 & | & \check{U} \end{pmatrix}, V = \tilde{V} \begin{pmatrix} 1 & | & 0 \\ 0 & | & \check{V} \end{pmatrix}, \text{ and } D = \begin{pmatrix} \delta_0 & | & 0 \\ 0 & | & \check{D} \end{pmatrix}.$$

(There are some really tough to see "checks" in the definition of U , V , and D !!) Then $A = UDV^H$ where U , V , and D have the desired properties.

- **By the Principle of Mathematical Induction** the result holds for all matrices $A \in \mathbb{C}^{m \times n}$ with $m \geq n$.

Homework 3.13 Let $D = \text{diag}(\delta_0, \dots, \delta_{n-1})$. Show that $\|D\|_2 = \max_{i=0}^{n-1} |\delta_i|$.

SEE ANSWER

Homework 3.14 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 .

SEE ANSWER

Homework 3.15 Let $A \in \mathbb{C}^{m \times n}$ with $A = \begin{pmatrix} \sigma_0 & 0 \\ 0 & B \end{pmatrix}$ and assume that $\|A\|_2 = \sigma_0$. Show that $\|B\|_2 \leq \|A\|_2$. (Hint: Use the SVD of B .)

☞ SEE ANSWER

Homework 3.16 Prove Lemma 3.12 for $m \leq n$.

☞ SEE ANSWER

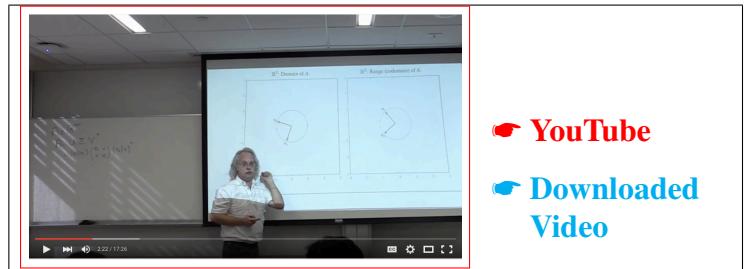
3.4 The Theorem

Theorem 3.17 (Singular Value Decomposition) Given $A \in \mathbb{C}^{m \times n}$ there exists 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 $A = U\Sigma V^H$ where $\Sigma = \begin{pmatrix} \Sigma_{TL} & 0 \\ 0 & 0 \end{pmatrix}$ with $\Sigma_{TL} = \text{diag}(\sigma_0, \dots, \sigma_{r-1})$ and $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{r-1} > 0$. The $\sigma_0, \dots, \sigma_{r-1}$ are known as the singular values of A .

Proof: Notice that the proof of the above theorem is identical to that of Lemma 3.12. However, thanks to the above exercises, we can conclude that $\|B\|_2 \leq \sigma_0$ in the proof, which then can be used to show that the singular values are found in order.

Proof: (Alternative) An alternative proof uses Lemma 3.12 to conclude that $A = UDV^H$. If the entries on the diagonal of D are not ordered from largest to smallest, then this can be fixed by permuting the rows and columns of D , and correspondingly permuting the columns of U and V .

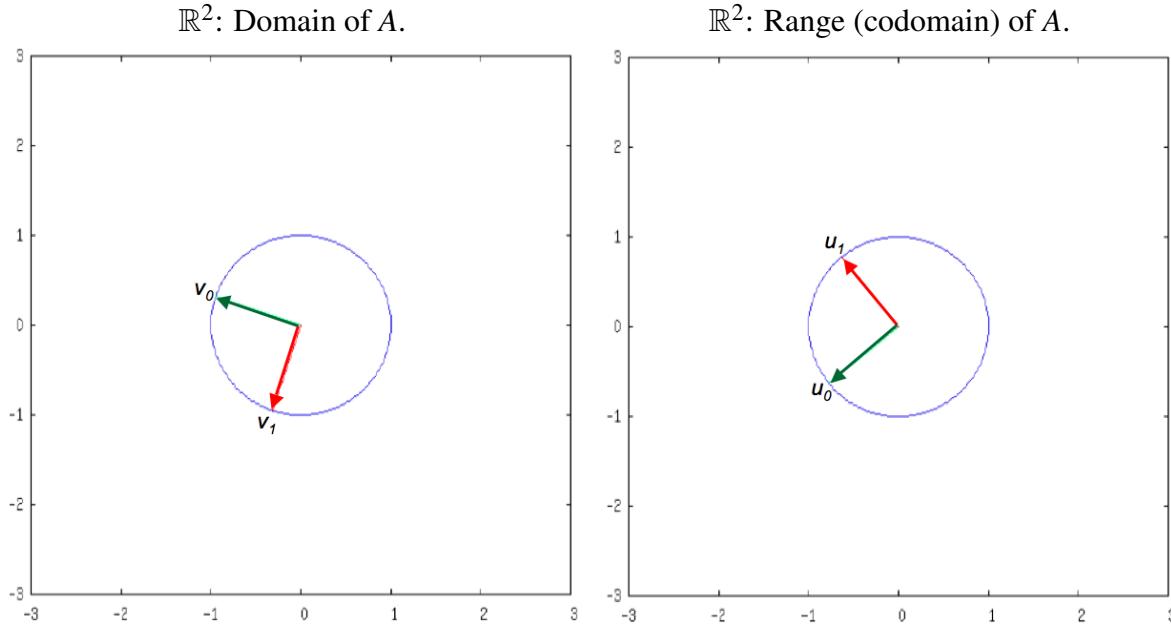
3.5 Geometric Interpretation



We will now quickly illustrate what the SVD Theorem tells us about matrix-vector multiplication (linear transformations) by examining the case where $A \in \mathbb{R}^{2 \times 2}$. Let $A = U\Sigma V^T$ be its SVD. (Notice that all matrices are now real valued, and hence $V^H = V^T$.) Partition

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

Since U and V are unitary matrices, $\{u_0, u_1\}$ and $\{v_0, v_1\}$ form orthonormal bases for the range and domain of A , respectively:



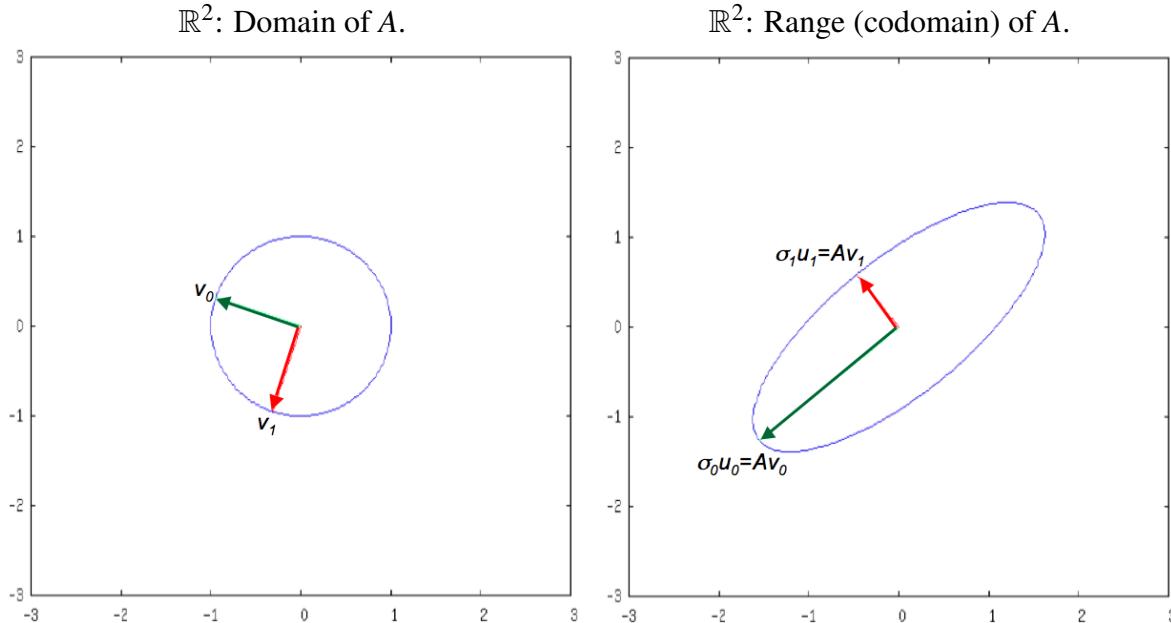
Let us manipulate the decomposition a little:

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

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

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

and similarly $Av_1 = \sigma_1 u_1$. This motivates the pictures



Now let us look at how A transforms any vector with (Euclidean) unit length. Notice that $x = \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}$ means that

$$x = \chi_0 e_0 + \chi_1 e_1,$$

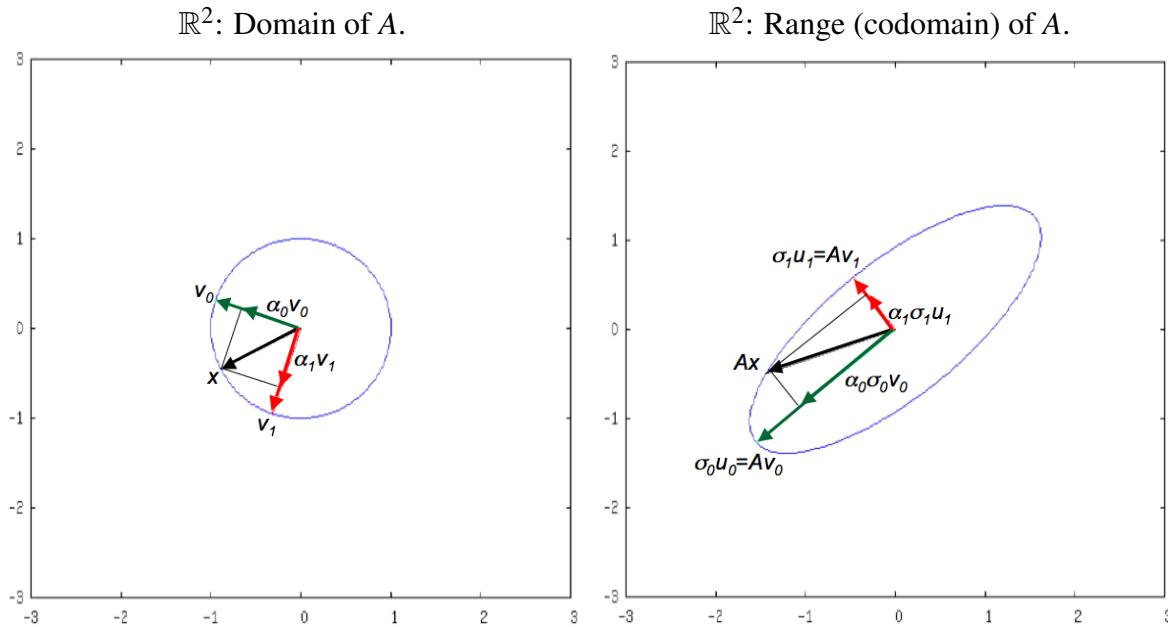
where e_0 and e_1 are the unit basis vectors. Thus, χ_0 and χ_1 are the coefficients when x is expressed using e_0 and e_1 as basis. However, we can also express x in the basis given by v_0 and v_1 :

$$\begin{aligned} x &= \underbrace{VV^T}_{I} x = \begin{pmatrix} v_0 & | & v_1 \end{pmatrix} \begin{pmatrix} v_0 & | & v_1 \end{pmatrix}^T x = \begin{pmatrix} v_0 & | & v_1 \end{pmatrix} \begin{pmatrix} v_0^T x \\ v_1^T x \end{pmatrix} \\ &= \underbrace{v_0^T x}_{\alpha_0} v_0 + \underbrace{v_1^T x}_{\alpha_1} v_1 = \alpha_0 v_0 + \alpha_1 v_1 = \begin{pmatrix} v_0 & | & v_1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}. \end{aligned}$$

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

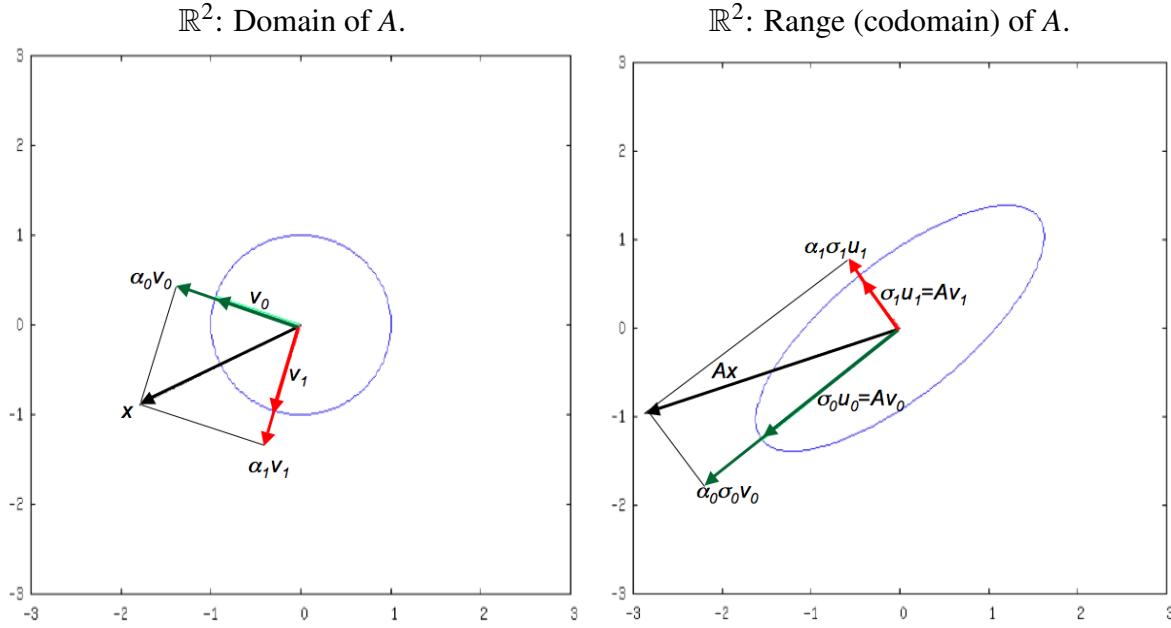
$$\begin{aligned} Ax &= \begin{pmatrix} \sigma_0 u_0 & | & \sigma_1 u_1 \end{pmatrix} \begin{pmatrix} v_0 & | & v_1 \end{pmatrix}^T x = \begin{pmatrix} \sigma_0 u_0 & | & \sigma_1 u_1 \end{pmatrix} \begin{pmatrix} v_0 & | & v_1 \end{pmatrix}^T \begin{pmatrix} v_0 & | & v_1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \\ &= \begin{pmatrix} \sigma_0 u_0 & | & \sigma_1 u_1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \alpha_0 \sigma_0 u_0 + \alpha_1 \sigma_1 u_1. \end{aligned}$$

This is illustrated by the following picture, which also captures the fact that the unit ball is mapped to an “ellipse”¹ with major axis equal to $\sigma_0 = \|A\|_2$ and minor axis equal to σ_1 :



¹It is not clear that it is actually an ellipse and this is not important to our observations.

Finally, we show the same insights for general vector x (not necessarily of unit length).



Another observation is that if one picks the right basis for the domain and codomain, then the computation Ax simplifies to a matrix multiplication with a diagonal matrix. Let us again illustrate this for nonsingular $A \in \mathbb{R}^{2 \times 2}$ with

$$A = \underbrace{\begin{pmatrix} u_0 & | & u_1 \end{pmatrix}}_U \underbrace{\begin{pmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} v_0 & | & v_1 \end{pmatrix}}_V^T.$$

Now, if we chose to express y using u_0 and u_1 as the basis and express x using v_0 and v_1 as the basis, then

$$\begin{aligned} \underbrace{UU^T}_I y &= U \underbrace{U^T y}_{\hat{y}} = (u_0^T y)u_0 + (u_1^T y)u_1 = \begin{pmatrix} u_0 & | & u_1 \end{pmatrix} \begin{pmatrix} u_0^T y \\ u_1^T y \end{pmatrix} = U \underbrace{\begin{pmatrix} \hat{\Psi}_0 \\ \hat{\Psi}_1 \end{pmatrix}}_{\hat{y}} \\ \underbrace{VV^T}_I x &= V \underbrace{V^T x}_{\hat{x}} = (v_0^T x)v_0 + (v_1^T x)v_1 = \begin{pmatrix} v_0 & | & v_1 \end{pmatrix} \begin{pmatrix} v_0^T x \\ v_1^T x \end{pmatrix} = V \underbrace{\begin{pmatrix} \hat{\chi}_0 \\ \hat{\chi}_1 \end{pmatrix}}_{\hat{x}}. \end{aligned}$$

If $y = Ax$ then

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

so that $\hat{y} = \Sigma\hat{x}$ and

$$\begin{pmatrix} \hat{\Psi}_0 \\ \hat{\Psi}_1. \end{pmatrix} = \begin{pmatrix} \sigma_0 \hat{\chi}_0 \\ \sigma_1 \hat{\chi}_1. \end{pmatrix}.$$

These observations generalize to $A \in \mathbb{C}^{m \times m}$.

3.6 Consequences of the SVD Theorem

Throughout this section we will assume that

- $A = U\Sigma V^H$ is the SVD of $A \in \mathbb{C}^{m \times n}$, with U and V unitary and Σ diagonal.
- $\Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right)$ where $\Sigma_{TL} = \text{diag}(\sigma_0, \dots, \sigma_{r-1})$ with $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{r-1} > 0$.
- $U = \left(\begin{array}{c|c} U_L & U_R \end{array} \right)$ with $U_L \in \mathbb{C}^{m \times r}$.
- $V = \left(\begin{array}{c|c} V_L & V_R \end{array} \right)$ with $V_L \in \mathbb{C}^{n \times r}$.

We first generalize the observations we made for $A \in \mathbb{R}^{2 \times 2}$. Let us track what the effect of $Ax = U\Sigma V^H x$ is on vector x . We assume that $m \geq n$.

- Let $U = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right)$ and $V = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right)$.
- Let

$$\begin{aligned} x &= VV^H x = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right) \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right)^H x = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right) \begin{pmatrix} v_0^H x \\ \vdots \\ v_{n-1}^H x \end{pmatrix} \\ &= v_0^H x v_0 + \cdots + v_{n-1}^H x v_{n-1}. \end{aligned}$$

This can be interpreted as follows: vector x can be written in terms of the usual basis of \mathbb{C}^n as $\chi_0 e_0 + \cdots + \chi_{n-1} e_{n-1}$ or in the orthonormal basis formed by the columns of V as $v_0^H x v_0 + \cdots + v_{n-1}^H x v_{n-1}$.

- Notice that $Ax = A(v_0^H x v_0 + \cdots + v_{n-1}^H x v_{n-1}) = v_0^H x A v_0 + \cdots + v_{n-1}^H x A v_{n-1}$ so that we next look at how A transforms each v_i : $A v_i = U\Sigma V^H v_i = U\Sigma e_i = \sigma_i U e_i = \sigma_i u_i$.
- Thus, another way of looking at Ax is

$$\begin{aligned} Ax &= v_0^H x A v_0 + \cdots + v_{n-1}^H x A v_{n-1} \\ &= v_0^H x \sigma_0 u_0 + \cdots + v_{n-1}^H x \sigma_{n-1} u_{n-1} \\ &= \sigma_0 u_0 v_0^H x + \cdots + \sigma_{n-1} u_{n-1} v_{n-1}^H x \\ &= (\sigma_0 u_0 v_0^H + \cdots + \sigma_{n-1} u_{n-1} v_{n-1}^H) x. \end{aligned}$$

Corollary 3.18 $A = U_L \Sigma_{TL} V_L^H$. This is called the reduced SVD of A .

Proof:

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

Homework 3.19 Let $A = \begin{pmatrix} A_T \\ 0 \end{pmatrix}$. Use the SVD of A_T to show that $\|A\|_2 = \|A_T\|_2$.

☞ SEE ANSWER

Any matrix with rank r can be written as the sum of r rank-one matrices:

Corollary 3.20 Let $A = U_L \Sigma_{TL} V_L^H$ be the reduced SVD with

$$U_L = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{r-1} \end{array} \right), \quad \Sigma_{TL} = \text{diag}(\sigma_0, \dots, \sigma_{r-1}), \quad \text{and} \quad V_L = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{r-1} \end{array} \right).$$

Then

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

(Each term is a nonzero matrix and an outer product, and hence a rank-1 matrix.)

Proof: We leave the proof as an exercise.

Corollary 3.21 $\mathcal{C}(A) = \mathcal{C}(U_L)$.

Proof:

- Let $y \in \mathcal{C}(A)$. Then there exists $x \in \mathbb{C}^n$ such that $y = Ax$ (by the definition of $y \in \mathcal{C}(A)$). But then

$$y = Ax = U_L \underbrace{\Sigma_{TL} V_L^H x}_z = U_L z,$$

i.e., there exists $z \in \mathbb{C}^r$ such that $y = U_L z$. This means $y \in \mathcal{C}(U_L)$.

- Let $y \in \mathcal{C}(U_L)$. Then there exists $z \in \mathbb{C}^r$ such that $y = U_L z$. But then

$$y = U_L z = U_L \underbrace{\Sigma_{TL} \Sigma_{TL}^{-1}}_I z = U_L \Sigma_{TL} \underbrace{V_L^H V_L}_I \Sigma_{TL}^{-1} z = A \underbrace{V_L \Sigma_{TL}^{-1} z}_x = Ax$$

so that there exists $x \in \mathbb{C}^n$ such that $y = Ax$, i.e., $y \in \mathcal{C}(A)$.

Corollary 3.22 Let $A = U_L \Sigma_{TL} V_L^H$ be the reduced SVD of A where U_L and V_L have r columns. Then the rank of A is r .

Proof: The rank of A equals the dimension of $\mathcal{C}(A) = \mathcal{C}(U_L)$. But the dimension of $\mathcal{C}(U_L)$ is clearly r .

Corollary 3.23 $\mathcal{N}(A) = \mathcal{C}(V_R)$.

Proof:

- Let $x \in \mathcal{N}(A)$. Then

$$\begin{aligned} x &= \underbrace{VV^H}_{I} x = \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H x = \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c} V_L^H \\ V_R^H \end{array} \right) x \\ &= \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c} V_L^H x \\ V_R^H x \end{array} \right) = V_L V_L^H x + V_R V_R^H x. \end{aligned}$$

If we can show that $V_L^H x = 0$ then $x = V_R z$ where $z = V_R^H x$. Assume that $V_L^H x \neq 0$. Then $\Sigma_{TL}(V_L^H x) \neq 0$ (since Σ_{TL} is nonsingular) and $U_L(\Sigma_{TL}(V_L^H x)) \neq 0$ (since U_L has linearly independent columns). But that contradicts the fact that $Ax = U_L \Sigma_{TL} V_L^H x = 0$.

- Let $x \in \mathcal{C}(V_R)$. Then $x = V_R z$ for some $z \in \mathbb{C}^r$ and $Ax = U_L \Sigma_{TL} \underbrace{V_L^H V_R}_{0} z = 0$.

Corollary 3.24 For all $x \in \mathbb{C}^n$ there exists $z \in \mathcal{C}(V_L)$ such that $Ax = Az$.

Proof:

$$\begin{aligned} Ax &= A \underbrace{VV^H}_{I} x = A \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H x \\ &= A(V_L V_L^H x + V_R V_R^H x) = AV_L V_L^H x + AV_R V_R^H x \\ &= AV_L V_L^H x + U_L \Sigma_{TL} \underbrace{V_L^H V_R}_{0} V_R^H x = A \underbrace{V_L V_L^H}_{z} x. \end{aligned}$$

Alternative proof (which uses the last corollary):

$$\begin{aligned} Ax &= A(V_L V_L^H x + V_R V_R^H x) = AV_L V_L^H x + A \underbrace{V_R V_R^H x}_{\in \mathcal{N}(A)} = A \underbrace{V_L V_L^H x}_{z}. \end{aligned}$$

The proof of the last corollary also shows that

Corollary 3.25 Any vector $x \in \mathbb{C}^n$ can be written as $x = z + x_n$ where $z \in \mathcal{C}(V_L)$ and $x_n \in \mathcal{N}(A) = \mathcal{C}(V_R)$.

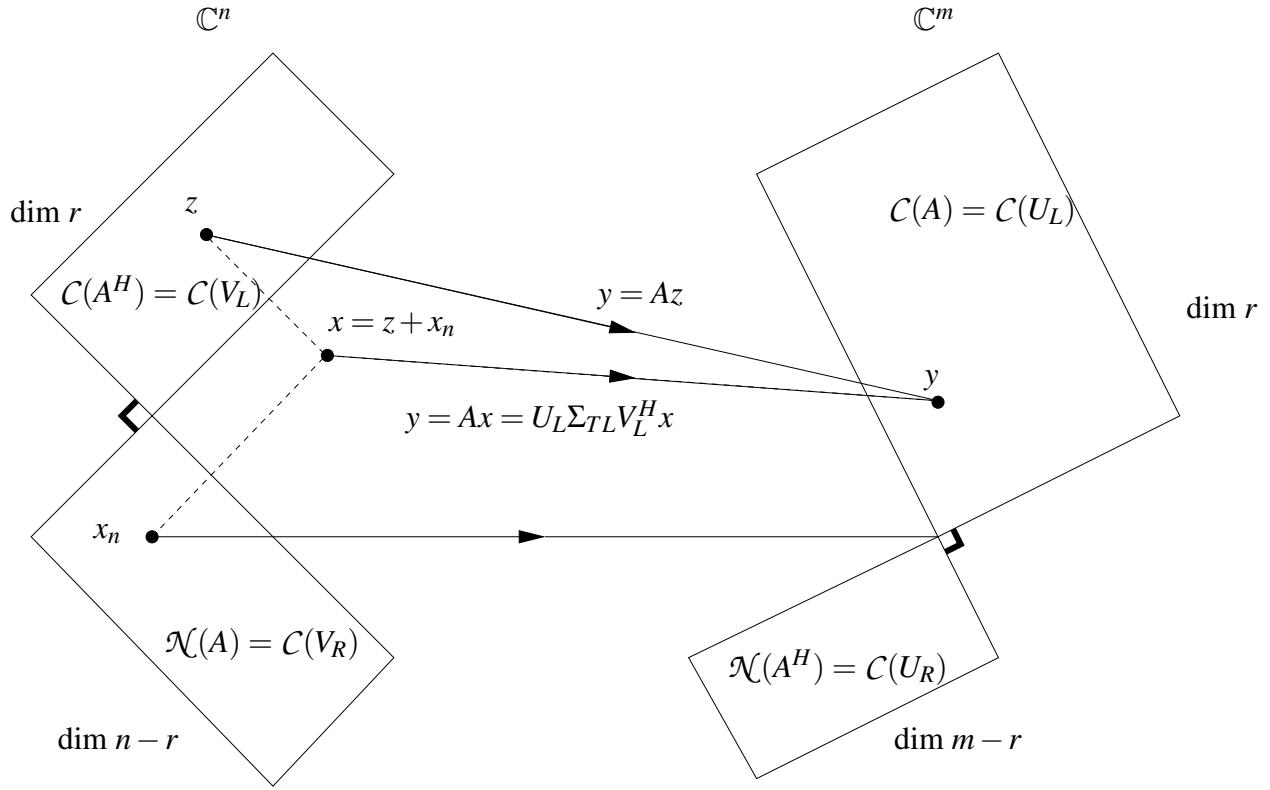


Figure 3.2: A pictorial description of how $x = z + x_n$ is transformed by $A \in \mathbb{C}^{m \times n}$ into $y = Ax = A(z + x_n)$. We see that the spaces $\mathcal{C}(V_L)$ and $\mathcal{C}(V_R)$ are orthogonal complements of each other within \mathbb{C}^n . Similarly, the spaces $\mathcal{C}(U_L)$ and $\mathcal{C}(U_R)$ are orthogonal complements of each other within \mathbb{C}^m . Any vector x can be written as the sum of a vector $z \in \mathcal{C}(V_R)$ and $x_n \in \mathcal{C}(V_C) = \mathcal{N}(A)$.

Corollary 3.26 $A^H = V_L \Sigma_{TL} U_L^H$ so that $\mathcal{C}(A^H) = \mathcal{C}(V_L)$ and $\mathcal{N}(A^H) = \mathcal{C}(U_R)$.

The above corollaries are summarized in Figure 3.2.

Theorem 3.27 Let $A \in \mathbb{C}^{n \times n}$ be nonsingular. Let $A = U \Sigma V^H$ be its SVD. Then

1. The SVD is the reduced SVD.

2. $\sigma_{n-1} \neq 0$.

3. If

$$U = \left(\begin{array}{c|c|c} u_0 & \cdots & u_{n-1} \end{array} \right), \Sigma = \text{diag}(\sigma_0, \dots, \sigma_{n-1}), \text{ and } V = \left(\begin{array}{c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right),$$

then

$$A^{-1} = (V P^T)(P \Sigma^{-1} P^T)(U P^T)^H = \left(\begin{array}{c|c|c} v_{n-1} & \cdots & v_0 \end{array} \right) \text{diag}\left(\frac{1}{\sigma_{n-1}}, \dots, \frac{1}{\sigma_0}\right) \left(\begin{array}{c|c|c} u_{n-1} & \cdots & u_0 \end{array} \right),$$

where $P = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{pmatrix}$ is the permutation matrix such that Px reverses the order of the entries

in x . (Note: for this permutation matrix, $P^T = P$. In general, this is not the case. What is the case for all permutation matrices P is that $P^T P = P P^T = I$.)

$$4. \|A^{-1}\|_2 = 1/\sigma_{n-1}.$$

Proof: The only item that is less than totally obvious is (3). Clearly $A^{-1} = V\Sigma^{-1}U^H$. The problem is that in Σ^{-1} the diagonal entries are not ordered from largest to smallest. The permutation fixes this.

Corollary 3.28 If $A \in \mathbb{C}^{m \times n}$ has linearly independent columns then $A^H A$ is invertible (nonsingular) and $(A^H A)^{-1} = V_L (\Sigma_{TL}^2)^{-1} V_L^H$.

Proof: Since A has linearly independent columns, $A = U_L \Sigma_{TL} V_L^H$ is the reduced SVD where U_L has n columns and V_L is unitary. Hence

$$A^H A = (U_L \Sigma_{TL} V_L^H)^H U_L \Sigma_{TL} V_L^H = V_L \Sigma_{TL}^H U_L^H U_L \Sigma_{TL} V_L^H = V_L \Sigma_{TL} \Sigma_{TL} V_L^H = V_L \Sigma_{TL}^2 V_L^H.$$

Since V_L is unitary and Σ_{TL} is diagonal with nonzero diagonal entries, they are both nonsingular. Thus

$$(V_L \Sigma_{TL}^2 V_L^H) (V_L (\Sigma_{TL}^2)^{-1} V_L^H) = I.$$

This means $A^T A$ is invertible and $(A^T A)^{-1}$ is as given.

3.7 Projection onto the Column Space

Definition 3.29 Let $U_L \in \mathbb{C}^{m \times k}$ have orthonormal columns. The projection of a vector $y \in \mathbb{C}^m$ onto $\mathcal{C}(U_L)$ is the vector $U_L x$ that minimizes $\|y - U_L x\|_2$, where $x \in \mathbb{C}^k$. We will also call this vector y the component of y in $\mathcal{C}(U_L)$.

Theorem 3.30 Let $U_L \in \mathbb{C}^{m \times k}$ have orthonormal columns. The projection of y onto $\mathcal{C}(U_L)$ is given by $U_L U_L^H y$.

Proof: Proof 1: The vector $U_L x$ that we want must satisfy

$$\|U_L x - y\|_2 = \min_{w \in \mathbb{C}^k} \|U_L w - y\|_2.$$

Now, the 2-norm is invariant under multiplication by the unitary matrix $U^H = \left(\begin{array}{c|c} U_L & U_R \end{array} \right)^H$

$$\begin{aligned} \|U_L x - y\|_2^2 &= \min_{w \in \mathbb{C}^k} \|U_L w - y\|_2^2 \\ &= \min_{w \in \mathbb{C}^k} \|U^H (U_L w - y)\|_2^2 \quad (\text{since the two norm is preserved}) \\ &= \min_{w \in \mathbb{C}^k} \left\| \left(\begin{array}{c|c} U_L & U_R \end{array} \right)^H (U_L w - y) \right\|_2^2 \\ &= \min_{w \in \mathbb{C}^k} \left\| \left(\begin{array}{c} U_L^H \\ U_R^H \end{array} \right) (U_L w - y) \right\|_2^2 \end{aligned}$$

$$\begin{aligned}
&= \min_{w \in \mathbb{C}^k} \left\| \left(\begin{pmatrix} U_L^H \\ U_R^H \end{pmatrix} U_L w - \begin{pmatrix} U_L^H \\ U_R^H \end{pmatrix} y \right) \right\|_2^2 \\
&= \min_{w \in \mathbb{C}^k} \left\| \left(\begin{pmatrix} U_L^H U_L w \\ U_R^H U_L w \end{pmatrix} - \begin{pmatrix} U_L^H y \\ U_R^H y \end{pmatrix} \right) \right\|_2^2 \\
&= \min_{w \in \mathbb{C}^k} \left\| \left(\begin{pmatrix} w \\ 0 \end{pmatrix} - \begin{pmatrix} U_L^H y \\ U_R^H y \end{pmatrix} \right) \right\|_2^2 \\
&= \min_{w \in \mathbb{C}^k} \left\| \begin{pmatrix} w - U_L^H y \\ -U_R^H y \end{pmatrix} \right\|_2^2 \\
&= \min_{w \in \mathbb{C}^k} \left(\|w - U_L^H y\|_2^2 + \|-U_R^H y\|_2^2 \right) \quad (\text{since } \left\| \begin{pmatrix} u \\ v \end{pmatrix} \right\|_2^2 = \|u\|_2^2 + \|v\|_2^2) \\
&= \left(\min_{w \in \mathbb{C}^k} \|w - U_L^H y\|_2^2 \right) + \|U_R^H y\|_2^2.
\end{aligned}$$

This is minimized when $w = U_L^H y$. Thus, the vector that is closest to y in the space spanned by U_L is given by $x = U_L w = U_L U_L^H y$.

Corollary 3.31 Let $A \in \mathbb{C}^{m \times n}$ and $A = U_L \Sigma_{TL} V_L^H$ be its reduced SVD. Then the projection of $y \in \mathbb{C}^m$ onto $\mathcal{C}(A)$ is given by $U_L U_L^H y$.

Proof: This follows immediately from the fact that $\mathcal{C}(A) = \mathcal{C}(U_L)$.

Corollary 3.32 Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns. Then the projection of $y \in \mathbb{C}^m$ onto $\mathcal{C}(A)$ is given by $A(A^H A)^{-1} A^H y$.

Proof: From Corollary 3.28, we know that $A^H A$ is nonsingular and that $(A^H A)^{-1} = V_L (\Sigma_{TL}^2)^{-1} V_L^H$. Now,

$$\begin{aligned}
A(A^H A)^{-1} A^H y &= (U_L \Sigma_{TL} V_L^H)(V_L (\Sigma_{TL}^2)^{-1} V_L^H)(U_L \Sigma_{TL} V_L^H)^H y \\
&= U_L \Sigma_{TL} \underbrace{V_L^H V_L}_I \underbrace{\Sigma_{TL}^{-1} \Sigma_{TL}^{-1}}_I \underbrace{V_L^H V_L}_I \Sigma_{TL} U_L^H y = U_L U_L^H y.
\end{aligned}$$

Hence the projection of y onto $\mathcal{C}(A)$ is given by $A(A^H A)^{-1} A^H y$.

We saw this result, for real-valued matrices, already in Section 3.1.1.

Definition 3.33 Let A have linearly independent columns. Then $(A^H A)^{-1} A^H$ is called the pseudo-inverse or Moore-Penrose generalized inverse of matrix A .

3.8 Low-rank Approximation of a Matrix

We are now ready to answer the question “How then does one compute the *best* rank-k approximation of a matrix so that the amount of data that must be stored best captures the matrix?” posed in Section 3.1.1.

Theorem 3.34 *Let $A \in \mathbb{C}^{m \times n}$ have SVD $A = U\Sigma V^H$ and assume A has rank r . Partition*

$$U = \left(\begin{array}{c|c} U_L & U_R \end{array} \right), \quad V = \left(\begin{array}{c|c} V_L & V_R \end{array} \right), \quad \text{and} \quad \Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right),$$

where $U_L \in \mathbb{C}^{m \times k}$, $V_L \in \mathbb{C}^{n \times k}$, and $\Sigma_{TL} \in \mathbb{R}^{k \times k}$ with $k \leq r$. 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:

$$\begin{aligned} \|A - B\|_2 &= \min_{C \in \mathbb{C}^{m \times n}} \|A - C\|_2 = \sigma_k. \\ \text{rank}(C) &\leq k \end{aligned}$$

In other words, B is the rank-k matrix closest to A as measured by the 2-norm.

Proof: First, if B is as defined, then clearly $\|A - B\|_2 = \sigma_k$:

$$\begin{aligned} \|A - B\|_2 &= \|U^H(A - B)V\|_2 = \|U^H A V - U^H B V\|_2 \\ &= \left\| \Sigma - \left(\begin{array}{c|c} U_L & U_R \end{array} \right)^H B \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \right\|_2 = \left\| \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right) - \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right) \right\|_2 \\ &= \left\| \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right) \right\|_2 = \|\Sigma_{BR}\|_2 = \sigma_k \end{aligned}$$

Next, assume that C has rank $t \leq k$ 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 $\dim(\mathcal{N}(C)) + \text{rank}(C) = n$.
- If $x \in \mathcal{N}(C)$ then

$$\|Ax\|_2 = \|(A - C)x\|_2 \leq \|A - C\|_2 \|x\|_2 < \sigma_k \|x\|_2.$$

- Partition $U = \left(\begin{array}{c|c|c|c} u_0 & \cdots & u_{m-1} \end{array} \right)$ and $V = \left(\begin{array}{c|c|c|c} v_0 & \cdots & v_{n-1} \end{array} \right)$. Then $\|Av_j\|_2 = \|\sigma_j u_j\|_2 = \sigma_j \geq \sigma_s$ for $j = 0, \dots, k$. Now, let x be any linear combination of v_0, \dots, v_k : $x = \alpha_0 v_0 + \cdots + \alpha_k v_k$. Notice that

$$\|x\|_2^2 = \|\alpha_0 v_0 + \cdots + \alpha_k v_k\|_2^2 \leq |\alpha_0|^2 + \cdots + |\alpha_k|^2.$$

Then

$$\begin{aligned} \|Ax\|_2^2 &= \|A(\alpha_0 v_0 + \cdots + \alpha_k v_k)\|_2^2 = \|\alpha_0 Av_0 + \cdots + \alpha_k Av_k\|_2^2 \\ &= \|\alpha_0 \sigma_0 u_0 + \cdots + \alpha_k \sigma_k u_k\|_2^2 = \|\alpha_0 \sigma_0 u_0\|_2^2 + \cdots + \|\alpha_k \sigma_k u_k\|_2^2 \\ &= |\alpha_0|^2 \sigma_0^2 + \cdots + |\alpha_k|^2 \sigma_k^2 \geq (|\alpha_0|^2 + \cdots + |\alpha_k|^2) \sigma_k^2 \end{aligned}$$

so that $\|Ax\|_2 \geq \sigma_k \|x\|_2$. In other words, vectors in the subspace of all linear combinations of $\{v_0, \dots, v_k\}$ satisfy $\|Ax\|_2 \geq \sigma_k \|x\|_2$. The dimension of this subspace is $k+1$ (since $\{v_0, \dots, v_k\}$ form an orthonormal basis).

- Both these subspaces are subspaces of \mathbb{C}^n . Since their dimensions add up to more than 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.

The above theorem tells us how to pick the best approximation to a given matrix of a given desired rank. In Section 3.1.1 we discussed how a low rank matrix can be used to compress data. The SVD thus gives the *best* such rank- k approximation. In the next section, we revisit this.

3.9 An Application

Let us revisit data compression. Let $Y \in \mathbb{R}^{m \times n}$ be a matrix that, for example, stores a picture. In this case, the (i, j) entry in Y is, for example, a number that represents the grayscale value of pixel (i, j) . The following instructions, executed in octave or matlab, generate the picture of Mexican artist Frida Kahlo in Figure 3.3(top-left). The file `FridaPNG.png` can be found in [Programming/chapter03/](#).

```
octave> IMG = imread( 'FridaPNG.png' ); % this reads the image
octave> Y = IMG( :, :, 1 );
octave> imshow( Y ) % this dispays the image
```

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 $Y = \text{IMG}(:, :, 1)$ 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.)

Now, let $Y = U\Sigma V^T$ be the SVD of matrix Y . Partition, conformally,

$$U = \left(\begin{array}{c|c} U_L & U_R \end{array} \right), \quad V = \left(\begin{array}{c|c} V_L & V_R \end{array} \right), \quad \text{and} \quad \Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right),$$

where U_L and V_L have k columns and Σ_{TL} is $k \times k$. so that

$$\begin{aligned} Y &= \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^T \\ &= \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & \Sigma_{BR} \end{array} \right) \left(\begin{array}{c} V_L^T \\ V_R^T \end{array} \right) \\ &= \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c} \Sigma_{TL} V_L^T \\ \hline \Sigma_{BR} V_R^T \end{array} \right) \\ &= U_L \Sigma_{TL} V_L^T + U_R \Sigma_{BR} V_R^T. \end{aligned}$$

Recall that then $U_L \Sigma_{TL} V_L^T$ is the best rank- k approximation to Y .

Let us approximate the matrix that stores the picture with $U_L \Sigma_{TL} V_L^T$:

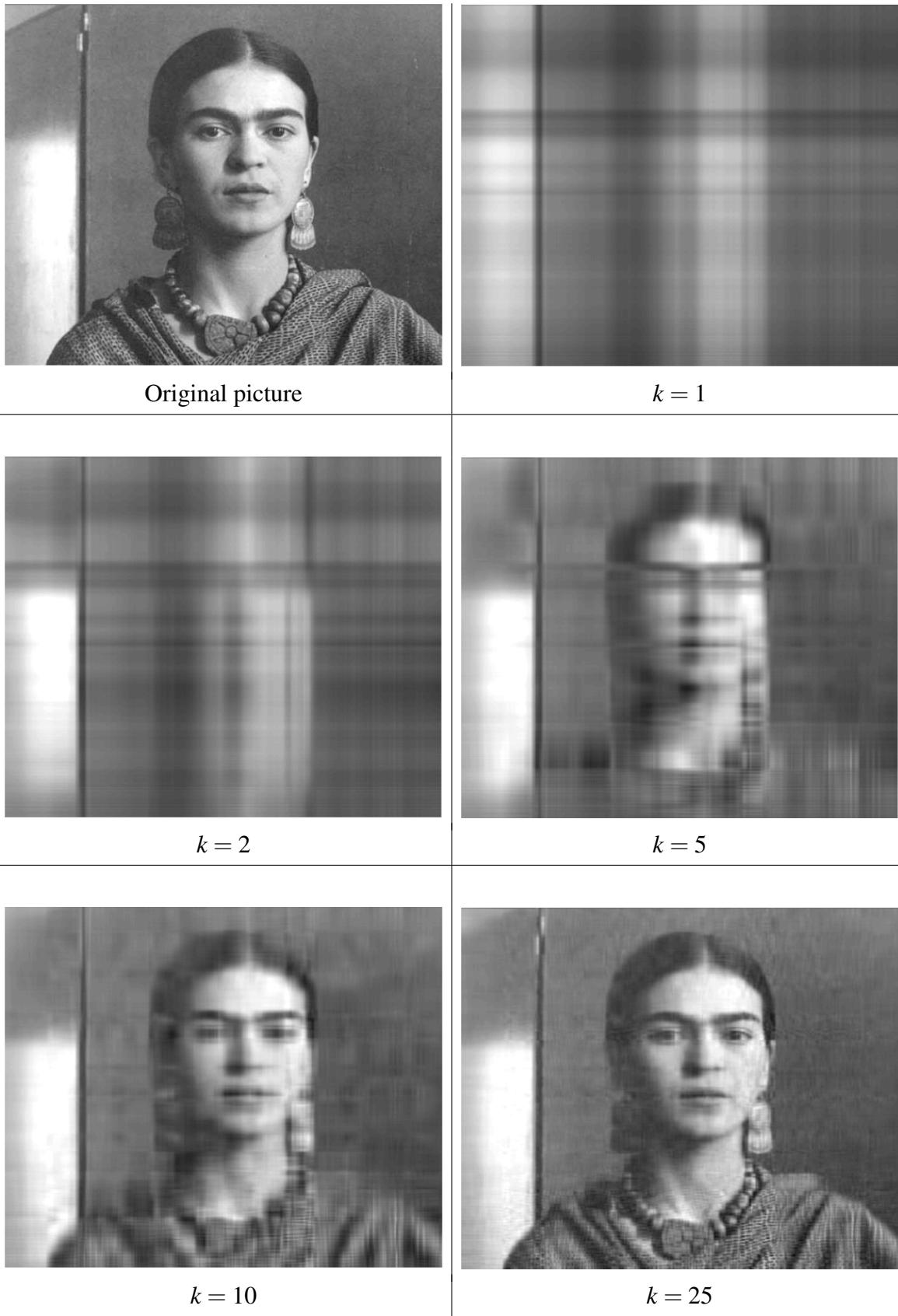


Figure 3.3: Multiple pictures as generated by the code

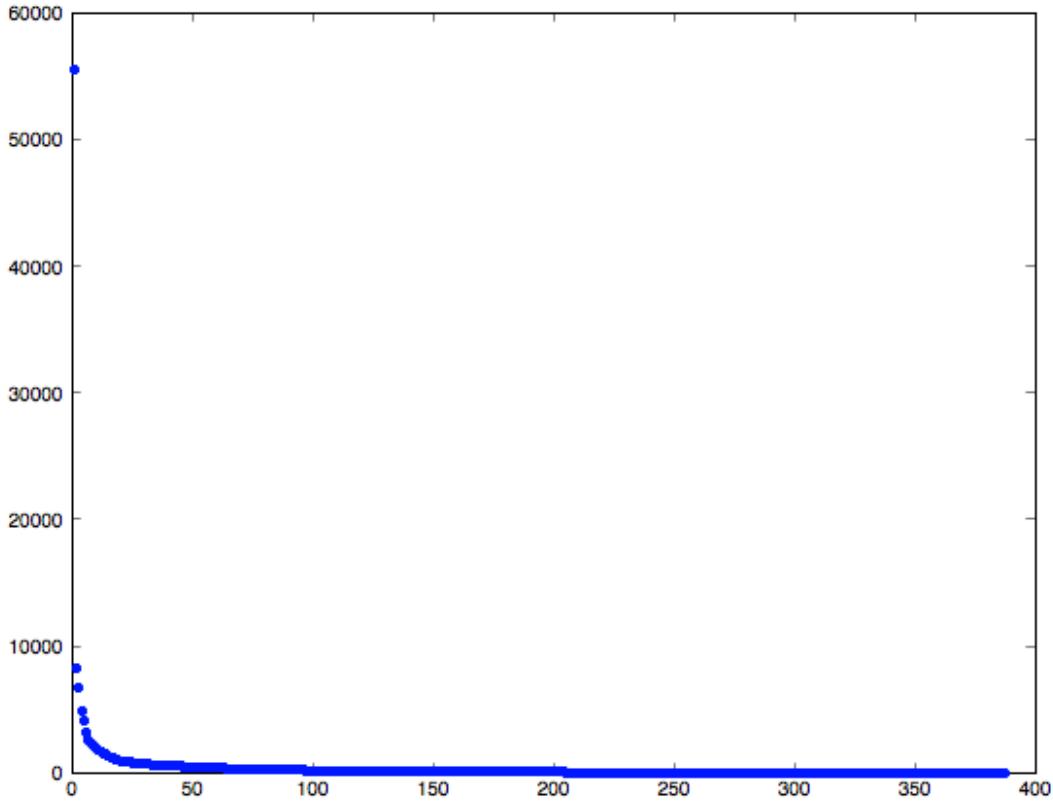


Figure 3.4: Distribution of singular values for the picture.

```

>> IMG = imread( 'FridaPNG.png' ); % read the picture
>> Y = IMG( :, :, 1 );
>> imshow( Y ); % this displays the image
>> k = 1;
>> [ U, Sigma, V ] = svd( Y );
>> UL = U( :, 1:k ); % first k columns
>> VL = V( :, 1:k ); % first k columns
>> SigmaTL = Sigma( 1:k, 1:k ); % TL submatrix of Sigma
>> Yapprox = uint8( UL * SigmaTL * VL' );
>> imshow( Yapprox );

```

As one increases k , the approximation gets better, as illustrated in Figure 3.3. The graph in Figure 3.4 helps explain. The original matrix Y is 387×469 , with 181,503 entries. When $k = 10$, matrices U , V , and Σ are 387×10 , 469×10 and 10×10 , respectively, requiring only 8,660 entries to be stored.

3.10 SVD and the Condition Number of a Matrix

In “Notes on Norms” we saw that if $Ax = b$ and $A(x + \delta x) = b + \delta b$, then

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq \kappa_2(A) \frac{\|\delta b\|_2}{\|b\|_2},$$

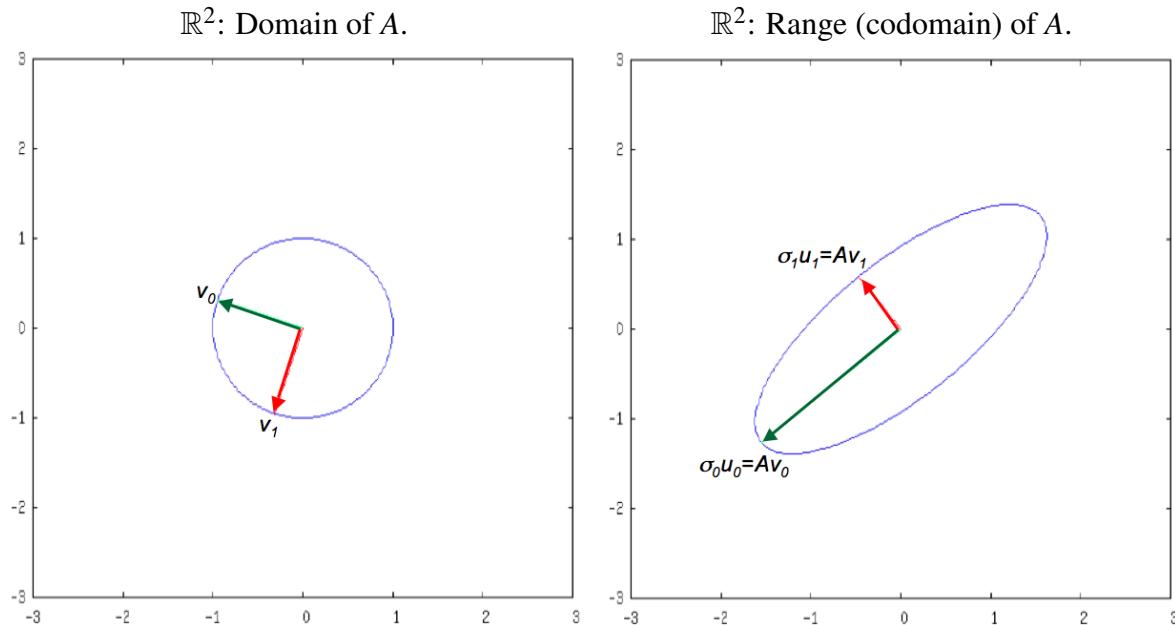
where $\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2$ is the condition number of A , using the 2-norm.

Homework 3.35 Show that if $A \in \mathbb{C}^{m \times m}$ is nonsingular, then

- $\|A\|_2 = \sigma_0$, the largest singular value;
- $\|A^{-1}\|_2 = 1/\sigma_{m-1}$, the inverse of the smallest singular value; and
- $\kappa_2(A) = \sigma_0/\sigma_{m-1}$.

 [SEE ANSWER](#)

If we go back to the example of $A \in \mathbb{R}^{2 \times 2}$, recall the following pictures that shows how A transforms the unit circle:



In this case, the ratio σ_0/σ_{n-1} represents the ratio between the major and minor axes of the “ellipse” on the right. Notice that the more elongated the “ellipse” on the right is, the worse (larger) the condition number.

3.11 An Algorithm for Computing the SVD?

It would seem that the proof of the existence of the SVD is constructive in the sense that it provides an algorithm for computing the SVD of a given matrix $A \in \mathbb{C}^{m \times m}$. **Not so fast!** Observe that

- Computing $\|A\|_2$ is nontrivial.
- Computing the vector that maximizes $\max_{\|x\|_2=1} \|Ax\|_2$ is nontrivial.
- Given a vector q_0 computing vectors q_1, \dots, q_{m-1} so that q_0, \dots, q_{m-1} are mutually orthonormal is expensive (as we will see when we discuss the QR factorization).

Towards the end of the course we will discuss algorithms for computing the eigenvalues and eigenvectors of a matrix, and related algorithms for computing the SVD.

3.12 Wrapup

3.12.1 Additional exercises

Homework 3.36 Let $D \in \mathbb{R}^{n \times n}$ be a diagonal matrix with entries $\delta_0, \delta_1, \dots, \delta_{n-1}$ on its diagonal. Show that

1. $\max_{x \neq 0} \|Dx\|_2 / \|x\|_2 = \max_{0 \leq i < n} |\delta_i|.$
2. $\min_{x \neq 0} \|Dx\|_2 / \|x\|_2 = \min_{0 \leq i < n} |\delta_i|.$

 [SEE ANSWER](#)

Homework 3.37 Let $A \in \mathbb{C}^{n \times n}$ have singular values $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{n-1}$ (where σ_{n-1} may equal zero). Prove that $\sigma_{n-1} \leq \|Ax\|_2 / \|x\|_2 \leq \sigma_0$ (assuming $x \neq 0$).

 [SEE ANSWER](#)

Homework 3.38 Let $A \in \mathbb{C}^{n \times n}$ have the property that for all vectors $x \in \mathbb{C}^n$ it holds that $\|Ax\|_2 = \|x\|_2$. Use the SVD to prove that A is unitary.

 [SEE ANSWER](#)

Homework 3.39 Use the SVD to prove that $\|A\|_2 = \|A^T\|_2$

 [SEE ANSWER](#)

Homework 3.40 Compute the SVD of $\begin{pmatrix} 1 \\ -2 \end{pmatrix} \begin{pmatrix} 2 & 1 \end{pmatrix}$.

 [SEE ANSWER](#)

3.12.2 Summary

Chapter **4**

Notes on Gram-Schmidt QR Factorization

A classic problem in linear algebra is the computation of an orthonormal basis for the space spanned by a given set of linearly independent vectors: Given a linearly independent set of vectors $\{a_0, \dots, a_{n-1}\} \subset \mathbb{C}^m$ we would like to find a set of mutually orthonormal vectors $\{q_0, \dots, q_{n-1}\} \subset \mathbb{C}^m$ so that

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

This problem is equivalent to the problem of, given a matrix $A = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right)$, computing a matrix $Q = \left(\begin{array}{c|c|c} q_0 & \cdots & q_{n-1} \end{array} \right)$ with $Q^H Q = I$ so that $C(A) = C(Q)$, where (A) denotes the column space of A .

A review at the undergraduate level of this topic (with animated illustrations) can be found in Week 11 of

[Linear Algebra: Foundations to Frontiers - Notes to LAFF With \[30\].](#)

4.1 Opening Remarks

Video from Fall 2014

Read disclaimer regarding the videos in the preface!

- 👉 [YouTube](#)
- 👉 [Download from UT Box](#)
- 👉 [View After Local Download](#)

(For help on viewing, see Appendix A.)

4.1.1 Launch

4.1.2 Outline

4.1	Opening Remarks	89
4.1.1	Launch	89
4.1.2	Outline	90
4.1.3	What you will learn	91
4.2	Classical Gram-Schmidt (CGS) Process	92
4.3	Modified Gram-Schmidt (MGS) Process	97
4.4	In Practice, MGS is More Accurate	101
4.5	Cost	103
4.5.1	Cost of CGS	104
4.5.2	Cost of MGS	104
4.6	Wrapup	105
4.6.1	Additional exercises	105
4.6.2	Summary	105

4.1.3 What you will learn

4.2 Classical Gram-Schmidt (CGS) Process

Given a set of linearly independent vectors $\{a_0, \dots, a_{n-1}\} \subset \mathbb{C}^m$, the Gram-Schmidt process computes an orthonormal basis $\{q_0, \dots, q_{n-1}\}$ that spans the same subspace as the original vectors, i.e.

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

The process proceeds as described in Figure 4.1 and in the algorithms in Figure 4.2.

Homework 4.1 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?

[SEE ANSWER](#)

Homework 4.2 Convince yourself that the relation between the vectors $\{a_j\}$ and $\{q_j\}$ in the algorithms in Figure 4.2 is given by

$$\left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array} \right) \left(\begin{array}{c|c|c|c} \rho_{0,0} & \rho_{0,1} & \cdots & \rho_{0,n-1} \\ \hline 0 & \rho_{1,1} & \cdots & \rho_{1,n-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \rho_{n-1,n-1} \end{array} \right),$$

where

$$q_i^H q_j = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \rho_{i,j} = \begin{cases} q_i^H a_j & \text{for } i < j \\ \|a_j - \sum_{i=0}^{j-1} \rho_{i,j} q_i\|_2 & \text{for } i = j \\ 0 & \text{otherwise.} \end{cases}$$

[SEE ANSWER](#)

Thus, this relationship between the linearly independent vectors $\{a_j\}$ and the orthonormal vectors $\{q_j\}$ can be concisely stated as

$$A = QR,$$

where A and Q are $m \times n$ matrices ($m \geq n$), $Q^H Q = I$, and R is an $n \times n$ upper triangular matrix.

Theorem 4.3 Let A have linearly independent columns, $A = QR$ where $A, Q \in \mathbb{C}^{m \times n}$ with $n \leq m$, $R \in \mathbb{C}^{n \times n}$, $Q^H Q = I$, and R is an upper triangular matrix with nonzero diagonal entries. Then, for $0 < k < n$, the first k columns of A span the same space as the first k columns of Q .

Proof: 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 $A_L, Q_L \in \mathbb{C}^{m \times k}$ and $R_{TL} \in \mathbb{C}^{k \times k}$. Then R_{TL} is nonsingular (since it is upper triangular and has no zero on its diagonal), $Q_L^H Q_L = I$, and $A_L = Q_L R_{TL}$. We want to show that $C(A_L) = C(Q_L)$:

Steps	Comment
$\rho_{0,0} := \ a_0\ _2$ $q_0 =: a_0 / \rho_{0,0}$	Compute the length of vector a_0 , $\rho_{0,0} := \ a_0\ _2$. Set $q_0 := a_0 / \rho_{0,0}$, creating a unit vector in the direction of a_0 . Clearly, $\text{Span}(\{a_0\}) = \text{Span}(\{q_0\})$. (Why?)
$\rho_{0,1} = q_0^H a_1$ $a_1^\perp = a_1 - \rho_{0,1} q_0$ $\rho_{1,1} = \ a_1^\perp\ _2$ $q_1 = a_1^\perp / \rho_{1,1}$	Compute a_1^\perp , the component of vector a_1 orthogonal to q_0 . Compute $\rho_{1,1}$, the length of a_1^\perp . Set $q_1 = a_1^\perp / \rho_{1,1}$, creating a unit vector in the direction of a_1^\perp . Now, q_0 and q_1 are mutually orthonormal and $\text{Span}(\{a_0, a_1\}) = \text{Span}(\{q_0, q_1\})$. (Why?)
$\rho_{0,2} = q_0^H a_2$ $\rho_{1,2} = q_1^H a_2$ $a_2^\perp = a_2 - \rho_{0,2} q_0 - \rho_{1,2} q_1$ $\rho_{2,2} = \ a_2^\perp\ _2$ $q_2 = a_2^\perp / \rho_{2,2}$	Compute a_2^\perp , the component of vector a_2 orthogonal to q_0 and q_1 . Compute $\rho_{2,2}$, the length of a_2^\perp . Set $q_2 = a_2^\perp / \rho_{2,2}$, creating a unit vector in the direction of a_2^\perp . Now, $\{q_0, q_1, q_2\}$ is an orthonormal basis and $\text{Span}(\{a_0, a_1, a_2\}) = \text{Span}(\{q_0, q_1, q_2\})$. (Why?)
And so forth.	

Figure 4.1: Gram-Schmidt orthogonalization.

<pre> for $j = 0, \dots, n - 1$ $a_j^\perp := a_j$ for $k = 0, \dots, j - 1$ $\rho_{k,j} := q_k^H a_j$ end $a_j^\perp := a_j$ for $k = 0, \dots, j - 1$ $a_j^\perp := a_j^\perp - \rho_{k,j} q_k$ end $\rho_{j,j} := \ a_j^\perp\ _2$ $q_j := a_j^\perp / \rho_{j,j}$ end </pre>	<pre> for $j = 0, \dots, n - 1$ for $k = 0, \dots, j - 1$ $\rho_{k,j} := q_k^H a_j$ end $a_j^\perp := a_j$ for $k = 0, \dots, j - 1$ $a_j^\perp := a_j^\perp - \rho_{k,j} q_k$ end $\rho_{j,j} := \ a_j^\perp\ _2$ $q_j := a_j^\perp / \rho_{j,j}$ end </pre>	$\begin{pmatrix} \rho_{0,j} \\ \vdots \\ \rho_{j-1,j} \end{pmatrix} := \begin{pmatrix} q_0^H a_j \\ \vdots \\ q_{j-1}^H a_j \end{pmatrix} = \left(q_0 \mid \cdots \mid q_{j-1} \right)^H a_j$ $a_j^\perp := a_j - \left(q_0 \mid \cdots \mid q_{j-1} \right) \begin{pmatrix} \rho_{0,j} \\ \vdots \\ \rho_{j-1,j} \end{pmatrix}$ $\rho_{j,j} := \ a_j^\perp\ _2$ $q_j := a_j^\perp / \rho_{j,j}$ end
---	---	---

Figure 4.2: Three equivalent (Classical) Gram-Schmidt algorithms.

- We first show that $C(A_L) \subseteq C(Q_L)$. Let $y \in C(A_L)$. Then there exists $x \in \mathbb{C}^k$ such that $y = A_L x$. But then $y = Q_L z$, where $z = R_T L x \neq 0$, which means that $y \in C(Q_L)$. Hence $C(A_L) \subseteq C(Q_L)$.
- We next show that $C(Q_L) \subseteq C(A_L)$. Let $y \in C(Q_L)$. Then there exists $z \in \mathbb{C}^k$ such that $y = Q_L z$. But then $y = A_L x$, where $x = R_T^{-1} L z$, from which we conclude that $y \in C(A_L)$. Hence $C(Q_L) \subseteq C(A_L)$.

Since $C(A_L) \subseteq C(Q_L)$ and $C(Q_L) \subseteq C(A_L)$, we conclude that $C(Q_L) = C(A_L)$.

Theorem 4.4 *Let $A \in \mathbb{C}^{m \times n}$ have linearly independent columns. Then there exist $Q \in \mathbb{C}^{m \times n}$ with $Q^H Q = I$ and upper triangular R with no zeroes on the diagonal such that $A = QR$. This is known as the QR factorization. If the diagonal elements of R are chosen to be real and positive, the QR factorization is unique.*

Proof: (By induction). Note that $n \leq m$ since A has linearly independent columns.

- **Base case:** $n = 1$. In this case $A = \begin{pmatrix} a_0 \end{pmatrix}$ where a_0 is its only column. Since A has linearly independent columns, $a_0 \neq 0$. Then

$$A = \begin{pmatrix} a_0 \end{pmatrix} = (q_0)(\rho_{00}),$$

where $\rho_{00} = \|a_0\|_2$ and $q_0 = a_0 / \rho_{00}$, so that $Q = (q_0)$ and $R = (\rho_{00})$.

- **Inductive step:** Assume that the result is true for all A with $n - 1$ linearly independent columns. We will show it is true for $A \in \mathbb{C}^{m \times n}$ with linearly independent columns.

Let $A \in \mathbb{C}^{m \times n}$. Partition $A \rightarrow \begin{pmatrix} A_0 & | & a_1 \end{pmatrix}$. By the induction hypothesis, there exist Q_0 and R_{00} such that $Q_0^H Q_0 = I$, R_{00} is upper triangular with nonzero diagonal entries and $A_0 = Q_0 R_{00}$. Now,

Algorithm: $[Q, R] := CGS_unb_var1(A)$
Partition $A \rightarrow \left(\begin{array}{c c} A_L & A_R \end{array} \right),$
$Q \rightarrow \left(\begin{array}{c c} Q_L & Q_R \end{array} \right),$
$R \rightarrow \left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right)$
where A_L and Q_L has 0 columns and R_{TL} is 0×0
while $n(A_L) \neq n(A)$ do
Repartition
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c c c} A_0 & a_1 & A_2 \end{array} \right),$
$\left(\begin{array}{c c} Q_L & Q_R \end{array} \right) \rightarrow \left(\begin{array}{c c c} Q_0 & q_1 & Q_2 \end{array} \right),$
$\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c c c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array} \right)$
<hr/>
$r_{01} := Q_0^T a_1$
$a_1^\perp := a_1 - Q_0 r_{01}$
$\rho_{11} := \ a_1^\perp\ _2$
$q_1 := a_1^\perp / \rho_{11}$
<hr/>
Continue with
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c c c} A_0 & a_1 & A_2 \end{array} \right),$
$\left(\begin{array}{c c} Q_L & Q_R \end{array} \right) \leftarrow \left(\begin{array}{c c c} Q_0 & q_1 & Q_2 \end{array} \right),$
$\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c c c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array} \right)$
endwhile

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

$$\underbrace{\begin{pmatrix} \rho_{0,j} \\ \vdots \\ \rho_{j-1,j} \end{pmatrix}}_{r_{01}} := \underbrace{\begin{pmatrix} q_0 & \cdots & q_{j-1} \end{pmatrix}^H}_{Q_0^H} \underbrace{a_j}_{a_1}$$

$$\underbrace{a_j^\perp}_{a_1^\perp} := \underbrace{a_j}_{a_1} - \underbrace{\begin{pmatrix} q_0 & \cdots & q_{j-1} \end{pmatrix}}_{Q_0} \underbrace{\begin{pmatrix} \rho_{0,j} \\ \vdots \\ \rho_{j-1,j} \end{pmatrix}}_{r_{01}}$$

$\rho_{j,j} := \|a_j^\perp\|_2$ ($\rho_{11} := \|a_1^\perp\|_2$)
 $q_j := a_j^\perp / \rho_{j,j}$ ($q_1 := a_1^\perp / \rho_{11}$)

end

Figure 4.3: (Classical) Gram-Schmidt algorithm for computing the QR factorization of a matrix A .

compute $r_{01} = Q_0^H a_1$ and $a_1^\perp = a_1 - Q_0 r_{01}$, the component of a_1 orthogonal to $C(Q_0)$. Because the columns of A are linearly independent, $a_1^\perp \neq 0$. Let $\rho_{11} = \|a_1^\perp\|_2$ and $q_1 = a_1^\perp / \rho_{11}$. Then

$$\begin{aligned} \left(\begin{array}{c|c} Q_0 & q_1 \end{array} \right) \left(\begin{array}{c|c} R_{00} & r_{01} \\ \hline 0 & \rho_{11} \end{array} \right) &= \left(\begin{array}{c|c} Q_0 R_{00} & Q_0 r_{01} + q_1 \rho_{11} \end{array} \right) \\ &= \left(\begin{array}{c|c} A_0 & Q_0 r_{01} + a_1^\perp \end{array} \right) = \left(\begin{array}{c|c} A_0 & a_1 \end{array} \right) = A. \end{aligned}$$

$$\text{Hence } Q = \left(\begin{array}{c|c} Q_0 & q_1 \end{array} \right) \text{ and } R = \left(\begin{array}{c|c} R_{00} & r_{01} \\ \hline 0 & \rho_{11} \end{array} \right).$$

- **By the Principle of Mathematical Induction** the result holds for all matrices $A \in \mathbb{C}^{m \times n}$ with $m \geq n$.

The proof motivates the algorithm in Figure 4.3 (left) in FLAME notation¹.

An alternative for motivating that algorithm is as follows: Consider $A = QR$. Partition A , Q , and R to yield

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

Assume that Q_0 and R_{00} have already been computed. Since corresponding columns of both sides must be equal, we find that

$$a_1 = Q_0 r_{01} + q_1 \rho_{11}. \quad (4.1)$$

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. Next, $a_1^\perp = a_1 - Q_0 r_{01}$ is computed from (4.1). This is the component of a_1 that is perpendicular 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 $q_1 = a_1^\perp / \rho_{11}$, which completes a derivation of the algorithm in Figure 4.3.

Homework 4.5 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 $k \times k$. Show that

1. $A_L = Q_L R_{TL}$: $Q_L R_{TL}$ equals the QR factorization of A_L ,
2. $C(A_L) = C(Q_L)$: 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. $(A_R - Q_L R_{TR})^H Q_L = 0$,
5. $A_R - Q_L R_{TR} = Q_R R_{BR}$, and
6. $C(A_R - Q_L R_{TR}) = C(Q_R)$.

SEE ANSWER

¹ The FLAME notation should be intuitively obvious. If it is not, you may want to review the earlier weeks in [Linear Algebra: Foundations to Frontiers - Notes to LAFF With](#).

$[y^\perp, r] = \text{Proj_orthog_to_Q}_{\text{CGS}}(Q, y)$ (used by classical Gram-Schmidt)	$[y^\perp, r] = \text{Proj_orthog_to_Q}_{\text{MGS}}(Q, y)$ (used by modified Gram-Schmidt)
$y^\perp = y$ for $i = 0, \dots, k - 1$ $\rho_i := q_i^H y$ $y^\perp := y^\perp - \rho_i q_i$ endfor	$y^\perp = y$ for $i = 0, \dots, k - 1$ $\rho_i := q_i^H y^\perp$ $y^\perp := y^\perp - \rho_i q_i$ endfor

Figure 4.4: Two different ways of computing $y^\perp = (I - QQ^H)y$, the component of y orthogonal to $\mathcal{C}(Q)$, where Q has k orthonormal columns.

4.3 Modified Gram-Schmidt (MGS) Process

We start by considering the following problem: Given $y \in \mathbb{C}^m$ and $Q \in \mathbb{C}^{m \times k}$ with orthonormal columns, compute y^\perp , the component of y orthogonal to the columns of Q . This is a key step in the Gram-Schmidt process in Figure 4.3.

Recall that if A has linearly independent columns, then $A(A^H A)^{-1}A^H y$ equals the projection of y onto the columns space of A (i.e., the component of y in $\mathcal{C}(A)$) and $y - A(A^H A)^{-1}A^H y = (I - A(A^H A)^{-1}A^H)y$ equals the component of y orthogonal to $\mathcal{C}(A)$. If Q has orthonormal columns, then $Q^H Q = I$ and hence $QQ^H y$ equals the projection of y onto the columns space of Q (i.e., the component of y in $\mathcal{C}(Q)$) and $y - QQ^H y = (I - QQ^H)y$ equals the component of y orthogonal to $\mathcal{C}(A)$.

Thus, mathematically, the solution to the stated problem is given by

$$\begin{aligned}
y^\perp &= (I - QQ^H)y = y - QQ^H y \\
&= y - \left(\begin{array}{c|c|c} q_0 & \cdots & q_{k-1} \end{array} \right) \left(\begin{array}{c|c|c} q_0 & \cdots & q_{k-1} \end{array} \right)^H y \\
&= y - \left(\begin{array}{c|c|c} q_0 & \cdots & q_{k-1} \end{array} \right) \begin{pmatrix} \frac{q_0^H}{\vdots} \\ \frac{\vdots}{q_{k-1}^H} \end{pmatrix} y \\
&= y - \left(\begin{array}{c|c|c} q_0 & \cdots & q_{k-1} \end{array} \right) \begin{pmatrix} \frac{q_0^H y}{\vdots} \\ \frac{\vdots}{q_{k-1}^H y} \end{pmatrix} \\
&= y - [(q_0^H y)q_0 + \cdots + (q_{k-1}^H y)q_{k-1}] \\
&= y - (q_0^H y)q_0 - \cdots - (q_{k-1}^H y)q_{k-1}.
\end{aligned}$$

This can be computed by the algorithm in Figure 4.4 (left) and is used by what is often called the *Classical* Gram-Schmidt (CGS) algorithm given in Figure 4.3.

An alternative algorithm for computing y^\perp is given in Figure 4.4 (right) and is used by the *Modified* Gram-Schmidt (MGS) algorithm also given in Figure 4.5. This approach is mathematically equivalent to

Algorithm: $[AR] := \text{Gram-Schmidt}(A)$ (overwrites A with Q)

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

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

while $n(A_L) \neq n(A)$ **do**

Repartition

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

where a_1 and ρ_{11} are columns, ρ_{11} is a scalar

CGS

$$r_{01} := A_0^H a_1$$

$$a_1 := a_1 - A_0 r_{01}$$

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

$$a_1 := a_1 / \rho_{11}$$

MGS

$$[a_1, r_{01}] = \text{Proj_orthog_to_Q}_{\text{MGS}}(A_0, a_1)$$

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

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

MGS (alternative)

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

$$a_1 := a_1 / \rho_{11}$$

$$r_{12}^T := a_1^H A_2$$

$$A_2 := A_2 - a_1 r_{12}^T$$

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

Figure 4.5: Left: Classical Gram-Schmidt algorithm. Middle: Modified Gram-Schmidt algorithm. Right: Modified Gram-Schmidt algorithm where every time a new column of Q , q_1 is computed the component of all future columns in the direction of this new vector are subtracted out.

the algorithm to its left for the following reason:

The algorithm on the left in Figure 4.4 computes

$$y^\perp := y - (q_0^H y) q_0 - \cdots - (q_{k-1}^H y) q_{k-1}$$

by in the i th step computing the component of y in the direction of q_i , $(q_i^H y) q_i$, and then subtracting this

```

for  $j = 0, \dots, n - 1$ 
   $a_j^\perp := a_j$ 
  for  $k = 0, \dots, j - 1$ 
     $\rho_{k,j} := q_k^H a_j^\perp$ 
     $a_j^\perp := a_j^\perp - \rho_{k,j} q_k$ 
  end
   $\rho_{j,j} := \|a_j^\perp\|_2$ 
   $q_j := a_j^\perp / \rho_{j,j}$ 
end

```

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

```

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

```

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

```

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

```

(c) MGS algorithm that normalizes the j th column to have unit length to compute q_j (overwriting a_j with the result) and then subtracts the component in the direction of q_j off the rest of the columns (a_{j+1}, \dots, a_{n-1}).

```

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

```

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

```

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

```

(e) Algorithm in (d) rewritten without loops.

```

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

```

(f) Algorithm in (e) rewritten to expose the row-vector-times matrix multiplication $a_j^H \left(a_{j+1} \Big| \cdots \Big| a_{n-1} \right)$ and rank-1 update $\left(a_{j+1} \Big| \cdots \Big| a_{n-1} \right) - a_j \left(\rho_{j,j+1} \Big| \cdots \Big| \rho_{j,n-1} \right)$.

Figure 4.6: Various equivalent MGS algorithms.

Algorithm: $[A, R] := MGS_unb_var1(A)$
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 and Q_L has 0 columns and R_{TL} is 0×0
while $n(A_L) \neq n(A)$ do
Repartition
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c c c} A_0 & a_1 & A_2 \end{array} \right),$ $\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c c c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array} \right)$
$\rho_{11} := \ a_1\ _2$
$a_1 := a_1 / \rho_{11}$
$r_{12}^T := a_1^H A_2$
$A_2 := A_2 - a_1 r_{12}^T$
Continue with
$\left(\begin{array}{c c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c c c} A_0 & a_1 & A_2 \end{array} \right),$ $\left(\begin{array}{c c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c c c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & r_{12}^T \\ \hline 0 & 0 & R_{22} \end{array} \right)$
endwhile

for $j = 0, \dots, n-1$
 $\rho_{j,j} := \|a_j\|_2$ ($\rho_{11} := \|a_1\|_2$)
 $a_j := a_j / \rho_{j,j}$ ($a_1 := a_1 / \rho_{11}$)

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

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

end

Figure 4.7: Modified Gram-Schmidt algorithm for computing the QR factorization of a matrix A .

off the vector y^\perp that already contains

$$y^\perp = y - (q_0^H y) q_0 - \cdots - (q_{i-1}^H y) q_{i-1},$$

leaving us with

$$y^\perp = y - (q_0^H y) q_0 - \cdots - (q_{i-1}^H y) q_{i-1} - (q_i^H y) q_i.$$

Now, notice that

$$\begin{aligned}
 q_i^H [y - (q_0^H y) q_0 - \cdots - (q_{i-1}^H y) q_{i-1}] &= q_i^H y - q_i^H (q_0^H y) q_0 - \cdots - q_i^H (q_{i-1}^H y) q_{i-1} \\
 &= q_i^H y - (q_0^H y) \underbrace{q_i^H q_0}_0 - \cdots - (q_{i-1}^H y) \underbrace{q_i^H q_{i-1}}_0 \\
 &= q_i^H y.
 \end{aligned}$$

What this means is that we can use y^\perp in our computation of ρ_i instead:

$$\rho_i := q_i^H y^\perp = q_i^H y,$$

an insight that justifies the equivalent algorithm in Figure 4.4 (right).

Next, we massage the MGS algorithm into the third (right-most) algorithm given in Figure 4.5. For this, consider the equivalent algorithms in Figure 4.6 and 4.7.

4.4 In Practice, MGS is More Accurate

In theory, all Gram-Schmidt algorithms discussed in the previous sections are equivalent: they compute the exact same QR factorizations. In practice, in the presence of round-off error, MGS is more accurate than CGS. We will (hopefully) get into detail about this later, but for now we will illustrate it with a classic example.

When storing real (or complex for that matter) 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 are performed. The *machine epsilon* or *unit roundoff error* is defined as the largest positive number ϵ_{mach} such that the stored value of $1 + \epsilon_{\text{mach}}$ is rounded 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. Let $\epsilon = \sqrt{\epsilon_{\text{mach}}}$ and consider the matrix

$$A = \left(\begin{array}{c|c|c} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{array} \right) = \left(\begin{array}{c|c|c} a_0 & a_1 & a_2 \end{array} \right) \quad (4.2)$$

In Figure 4.8 (left) we execute the CGS algorithm. It yields the approximate matrix

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

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. But

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

Clearly, the 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 of each of the algorithms.

<p><u>First iteration</u></p> $\rho_{0,0} = \ a_0\ _2 = \sqrt{1 + \epsilon^2} = \sqrt{1 + \epsilon_{\text{mach}}}$ <p>which is rounded to 1.</p> $q_0 = a_0 / \rho_{0,0} = \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix} / 1 = \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix}$ <p><u>Second iteration</u></p> $\rho_{0,1} = q_0^H a_1 = 1$ $a_1^\perp = a_1 - \rho_{0,1} q_0 = \begin{pmatrix} 0 \\ -\epsilon \\ \epsilon \\ 0 \end{pmatrix}$ $\rho_{1,1} = \ a_1^\perp\ _2 = \sqrt{2\epsilon^2} = \sqrt{2}\epsilon$ $q_1 = a_1^\perp / \rho_{1,1} = \begin{pmatrix} 0 \\ -\epsilon \\ \epsilon \\ 0 \end{pmatrix} / (\sqrt{2}\epsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}$ <p><u>Third iteration</u></p> $\rho_{0,2} = q_0^H a_2 = 1$ $a_2^\perp = a_2 - \rho_{0,2} q_0 - \rho_{1,2} q_1 = \begin{pmatrix} 0 \\ -\epsilon \\ 0 \\ \epsilon \end{pmatrix}$ $\rho_{1,2} = q_1^H a_2^\perp = (\sqrt{2}/2)\epsilon$ $q_2 = a_2^\perp / \rho_{2,2} = \begin{pmatrix} 0 \\ -\epsilon \\ 0 \\ \epsilon \end{pmatrix} / (\sqrt{2}\epsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ 0 \\ \frac{\sqrt{2}}{2} \end{pmatrix}$	<p><u>First iteration</u></p> $\rho_{0,0} = \ a_0\ _2 = \sqrt{1 + \epsilon^2} = \sqrt{1 + \epsilon_{\text{mach}}}$ <p>which is rounded to 1.</p> $q_0 = a_0 / \rho_{0,0} = \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix} / 1 = \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix}$ <p><u>Second iteration</u></p> $\rho_{0,1} = q_0^H a_1 = 1$ $a_1^\perp = a_1 - \rho_{0,1} q_0 = \begin{pmatrix} 0 \\ -\epsilon \\ \epsilon \\ 0 \end{pmatrix}$ $\rho_{1,1} = \ a_1^\perp\ _2 = \sqrt{2\epsilon^2} = \sqrt{2}\epsilon$ $q_1 = a_1^\perp / \rho_{1,1} = \begin{pmatrix} 0 \\ -\epsilon \\ \epsilon \\ 0 \end{pmatrix} / (\sqrt{2}\epsilon) = \begin{pmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix}$ <p><u>Third iteration</u></p> $\rho_{0,2} = q_0^H a_2 = 1$ $a_2^\perp = a_2 - \rho_{0,2} q_0 = \begin{pmatrix} 0 \\ -\epsilon \\ 0 \\ \epsilon \end{pmatrix}$ $\rho_{1,2} = q_1^H a_2^\perp = (\sqrt{2}/2)\epsilon$ $q_2 = a_2^\perp / \rho_{2,2} = \begin{pmatrix} 0 \\ -\frac{\epsilon}{2} \\ -\frac{\epsilon}{2} \\ \epsilon \end{pmatrix} / (\frac{\sqrt{6}}{2}\epsilon) = \begin{pmatrix} 0 \\ \frac{\sqrt{6}}{6} \\ -\frac{\sqrt{6}}{6} \\ -\frac{2\sqrt{6}}{6} \end{pmatrix}$
---	---

Figure 4.8: Execution of the CGS algorithm (left) and MGS algorithm (right) on the example in Eqn. (4.2).

- In the CGS algorithm, we find that

$$a_2^\perp := a_2 - (q_0^H a_2) q_0 - (q_1^H a_2) q_1.$$

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 .

- In the MGS algorithm, we find that

$$a_2^\perp := a_2 - (q_0^H a_2) q_0$$

after which

$$a_2^\perp := a_2^\perp - q_1^H a_2^\perp q_1 = [a_2 - (q_0^H a_2) q_0] - (q_1^H [a_2 - (q_0^H a_2) q_0]) q_1.$$

This time, if $a_2 - q_1^H a_2^\perp q_1$ 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 .

Obviously, 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. This is beyond the scope of this note.

4.5 Cost

Let us examine the cost of computing the QR factorization of an $m \times n$ matrix A . We will count multiplies and adds as each as one floating point operation.

We start by reviewing the cost, in floating point operations (flops), of various vector-vector and matrix-vector operations:

Name	Operation	Approximate cost (in flops)
Vector-vector operations ($x, y \in \mathbb{C}^n, \alpha \in \mathbb{C}$)		
Dot	$\alpha := x^H y$	$2n$
Axpy	$y := \alpha x + y$	$2n$
Scal	$x := \alpha x$	n
Nrm2	$\alpha := \ a\ _2$	$2n$
Matrix-vector operations ($A \in \mathbb{C}^{m \times n}, \alpha, \beta \in \mathbb{C}$, with x and y vectors of appropriate size)		
Matrix-vector multiplication (Gmv)		
$y := \alpha Ax + \beta y$		
$y := \alpha A^H x + \beta y$		
Rank-1 update (Ger)	$A := \alpha y x^H + A$	$2mn$

Now, consider the algorithms in Figure 4.5. Notice that the columns of A are of size m . During the k th iteration ($0 \leq k < n$), A_0 has k columns and A_2 has $n - k - 1$ columns.

4.5.1 Cost of CGS

Operation	Approximate cost (in flops)
$r_{01} := A_0^H a_1$	$2mk$
$a_1 := a_1 - A_0 r_{01}$	$2mk$
$\rho_{11} := \ a_1\ _2$	$2m$
$a_1 := a_1 / \rho_{11}$	m

Thus, the total cost is (approximately)

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

4.5.2 Cost of MGS

Operation	Approximate cost (in flops)
$\rho_{11} := \ a_1\ _2$	$2m$
$a_1 := a_1 / \rho_{11}$	m
$r_{12}^T := a_1^H A_2$	$2m(n-k-1)$
$A_2 := A_2 - a_1 r_{12}^T$	$2m(n-k-1)$

Thus, the total cost is (approximately)

$$\begin{aligned}
 & \sum_{k=0}^{n-1} [2m + m + 2m(n-k-1) + 2m(n-k-1)] \\
 &= \sum_{k=0}^{n-1} [3m + 4m(n-k-1)] \\
 &= 3mn + 4m \sum_{k=0}^{n-1} (n-k-1) \\
 &= 3mn + 4m \sum_{i=0}^{n-1} i \quad (\text{Change of variable: } i = n-k-1) \\
 &\approx 3mn + 4m \frac{n^2}{2} \quad (\sum_{i=0}^{n-1} i = n(n-1)/2 \approx n^2/2) \\
 &= 3mn + 2mn^2 \\
 &\approx 2mn^2 \quad (3mn \text{ is of lower order}).
 \end{aligned}$$

4.6 Wrapup

4.6.1 Additional exercises

Homework 4.6 Implement GS_unb_var1 using MATLAB and *Spark*. You will want to use the laff routines summarized in Appendix B. (I'm not sure if you can visualize the algorithm with *PictureFLAME*. Try it!)

 [SEE ANSWER](#)

Homework 4.7 Implement MGS_unb_var1 using MATLAB and *Spark*. You will want to use the laff routines summarized in Appendix B. (I'm not sure if you can visualize the algorithm with *PictureFLAME*. Try it!)

 [SEE ANSWER](#)

4.6.2 Summary

Notes on the FLAME APIs

Video

Read disclaimer regarding the videos in the preface!

No video.

5.0.1 Outline

Video	107
5.0.1 Outline	108
5.1 Motivation	108
5.2 Install FLAME@lab	108
5.3 An Example: Gram-Schmidt Orthogonalization	108
5.3.1 The Spark Webpage	108
5.3.2 Implementing CGS with FLAME@lab	109
5.3.3 Editing the code skeleton	111
5.3.4 Testing	112
5.4 Implementing the Other Algorithms	113

5.1 Motivation

In the course so far, we have frequently used the “FLAME Notation” to express linear algebra algorithms. In this note we show how to translate such algorithms into code, using various QR factorization algorithms as examples.

5.2 Install FLAME@lab

The API we will use we refer to as the “FLAME@lab” API, which is an API that targets the M-script language used by **Matlab** and **Octave** (an Open Source Matlab implementation). This API is very intuitive, and hence we will spend (almost) no time explaining it.

Download all files from <http://www.cs.utexas.edu/users/flame/Notes/FLAMEatlab/> and place them in the same directory as you will the remaining files that you will create as part of the exercises in this document. (Unless you know how to set up paths in Matlab/Octave, in which case you can put it wherever you please, and set the path.)

5.3 An Example: Gram-Schmidt Orthogonalization

Let us start by considering the various Gram-Schmidt based QR factorization algorithms from “Notes on Gram-Schmidt QR Factorization”, typeset using the FLAME Notation in Figure 5.2.

5.3.1 The Spark Webpage

We wish to typeset the code so that it closely resembles the algorithms in Figure 5.2. The FLAME notation itself uses “white space” to better convey the algorithms. We want to do the same for the codes that implement the algorithms. However, typesetting that code is somewhat bothersome because of the

$[y^\perp, r] = \text{Proj_orthog_to_Q}_{\text{CGS}}(Q, y)$ (used by classical Gram-Schmidt)	$[y^\perp, r] = \text{Proj_orthog_to_Q}_{\text{MGS}}(Q, y)$ (used by modified Gram-Schmidt)
$y^\perp = y$ for $i = 0, \dots, k - 1$ $\rho_i := q_i^H y$ $y^\perp := y^\perp - \rho_i q_i$ endfor	$y^\perp = y$ for $i = 0, \dots, k - 1$ $\rho_i := q_i^H y^\perp$ $y^\perp := y^\perp - \rho_i q_i$ endfor

Figure 5.1: Two different ways of computing $y^\perp = (I - QQ^H)y$, the component of y orthogonal to $\mathcal{C}(Q)$, where Q has k orthonormal columns.

careful spacing that is required. For this reason, we created a webpage that creates a “code skeleton.”. We call this page the “Spark” page:

<http://www.cs.utexas.edu/users/flame/Spark/>.

When you open the link, you will get a page that looks something like the picture in Figure 5.3.

5.3.2 Implementing CGS with FLAME@lab

We will focus on the Classical Gram-Schmidt algorithm on the left, which we show by itself in Figure 5.4 (left). To its right, we show how the menu on the left side of the Spark webpage needs to be filled out.

Some comments:

Name: Choose a name that describes the algorithm/operation being implemented.

Type of function: Later you will learn about “blocked” algorithms. For now, we implement “unblocked” algorithms.

Variant number: Notice that there are a number of algorithmic variants for implementing the Gram-Schmidt algorithm. We choose to call the first one “Variant 1”.

Number of operands: This routine requires two operands: one each for matrices A and R . (A will be overwritten by the matrix Q .)

Operand 1: We indicate that A is a matrix through which we “march” from left to right ($L \rightarrow R$) and it is both input and output.

Operand 2: We indicate that R is a matrix through which we “march” from top-left to bottom-right ($TL \rightarrow BR$) and it is both input and output. Our API requires you to pass in the array in which to put an output, so an appropriately sized R must be passed in.

Pick and output language: A number of different representations are supported, including APIs for M-script (FLAME@lab), C (FLAMEC), LATEX(FLaTeX), and Python (FlamePy). Pick FLAME@lab.

Algorithm: $[AR] := \text{Gram-Schmidt}(A)$ (overwrites A with Q)

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

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

while $n(A_L) \neq n(A)$ **do**

Repartition

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

where a_1 and ρ_{11} are columns, ρ_{11} is a scalar

CGS

$$r_{01} := A_0^H a_1$$

$$a_1 := a_1 - A_0 r_{01}$$

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

$$a_1 := a_1 / \rho_{11}$$

MGS

$$[a_1, r_{01}] = \text{Proj_orthog_to_Q}_{\text{MGS}}(A_0, a_1)$$

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

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

MGS (alternative)

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

$$a_1 := a_1 / \rho_{11}$$

$$r_{12}^T := a_1^H A_2$$

$$A_2 := A_2 - a_1 r_{12}^T$$

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

Figure 5.2: Left: Classical Gram-Schmidt algorithm. Middle: Modified Gram-Schmidt algorithm. Right: Modified Gram-Schmidt algorithm where every time a new column of Q , q_1 is computed the component of all future columns in the direction of this new vector are subtracted out.

To the left of the menu, you now find what we call a code skeleton for the implementation, as shown in Figure 5.5. In Figure 5.6 we show the algorithm and generated code skeleton side-by-side.

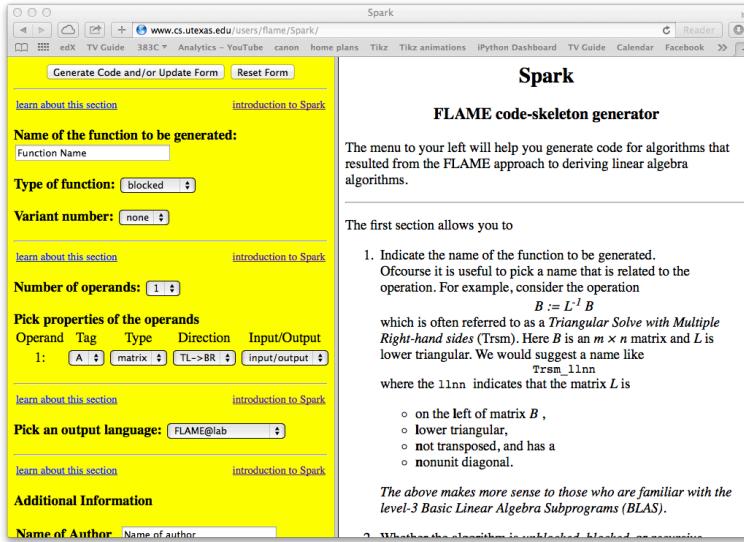


Figure 5.3: The Spark webpage.

5.3.3 Editing the code skeleton

At this point, one should copy the code skeleton into one's favorite text editor. (We highly recommend emacs for the serious programmer.) Once this is done, there are two things left to do:

Fix the code skeleton: The Spark webpage “guesses” the code skeleton. One detail that it sometimes gets wrong is the “stopping criteria”. In this case, the algorithm should stay in the loop as long as $n(A_L) \neq n(A)$ (the width of A_L is not yet the width of A). In our example, the Spark webpage guessed that the column size of matrix A is to be used for the stopping criteria:

```
while ( size( AL, 2 ) < size( A, 2 ) )
```

which happens to be correct. (When you implement the Householder QR factorization, you may not be so lucky...)

The “update” statements: The Spark webpage can't guess what the actual updates to the various parts of matrices A and R should be. It fills in

%	update line 1	%
%	:	%
%	update line n	%

Thus, one has to manually translate

$r_{01} := A_0^H a_1$ $a_1 := a_1 - A_0 r_{01}$ $\rho_{11} := \ a_1\ _2$ $a_1 := a_1 / \rho_{11}$
--

into appropriate M-script code:

$r01 = A0' * a1;$ $a1 = a1 - A0 * r01;$ $rho11 = norm(a1);$ $a1 = a1 / rho11;$

Algorithm: $[A, R] := \text{Gram-Schmidt}(A)$ (overwrites A with Q)

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 $n(A_L) \neq n(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right),$$

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

$r_{01} := A_0^H a_1$
 $a_1 := a_1 - A_0 r_{01}$
 $\rho_{11} := \|a_1\|_2$
 $a_1 := a_1 / \rho_{11}$

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

Name of the function to be generated: CGS

Type of function: unblocked

Variant number: 1

[learn about this section](#) [introduction to Spark](#)

Number of operands: 2

Pick properties of the operands

Operand	Tag	Type	Direction	Input/Output
1:	A	matrix	L->R	input/output
2:	R	matrix	TL->BR	input/output

[learn about this section](#) [introduction to Spark](#)

Pick an output language: FLAME@lab

Figure 5.4: Left: Classical Gram-Schmidt algorithm. Right: Generated code-skeleton for CGS.

(Notice: if one forgets the “;”, when executed the results of the assignment will be printed by Matlab/Octave.)

At this point, one saves the resulting code in the file CGS_unb_var1.m. The “.m” ending is important since the name of the file is used to find the routine when using Matlab/Octave.

5.3.4 Testing

To now test the routine, one starts octave and, for example, executes the commands

```
> A = rand( 5, 4 )
> R = zeros( 4, 4 )
> [ Q, R ] = CGS_unb_var1( A, R )
> A - Q * triu( R )
```

The result should be (approximately) a 5×4 zero matrix.

<div style="background-color: #ffffcc; padding: 10px;"> <p style="margin: 0;">Generate Code and/or Update Form Reset Form</p> <p>learn about this section introduction to Spark</p> <p>Name of the function to be generated: CGS</p> <p>Type of function: unblocked</p> <p>Variant number: 1</p> <p>learn about this section introduction to Spark</p> <p>Number of operands: 2</p> <p>Pick properties of the operands</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Operand</th> <th>Tag</th> <th>Type</th> <th>Direction</th> <th>Input/Output</th> </tr> </thead> <tbody> <tr> <td>1:</td> <td>A</td> <td>matrix</td> <td>L->R</td> <td>input/output</td> </tr> <tr> <td>2:</td> <td>R</td> <td>matrix</td> <td>TL->BR</td> <td>input/output</td> </tr> </tbody> </table> <p>learn about this section introduction to Spark</p> <p>Pick an output language: FLAME@lab</p> <p>learn about this section introduction to Spark</p> <p>Additional Information</p> <p>Name of Author Name of author</p> </div>	Operand	Tag	Type	Direction	Input/Output	1:	A	matrix	L->R	input/output	2:	R	matrix	TL->BR	input/output	<pre> function [A_out, R_out] = CGS_unb_var1(A, R) [AL, AR] = FLA_Part_1x2(A, ... 0, 'FLA_LEFT'); [RTL, RTR, ... RBL, RBR] = FLA_Part_2x2(R, ... 0, 0, 'FLA_TL'); while (size(AL, 2) < size(A, 2)) [A0, a1, A2] = FLA_Report_1x2_to_1x3(AL, AR, ... 1, 'FLA_RIGHT'); [R00, r01, R02, ... r10t, rho11, r12t, ... R20, r21, R22] = FLA_Report_2x2_to_3x3(RTL, RTR, ... RBL, RBR, ... 1, 1, 'FLA_BR'); %-----% % update line 1 % : % update line n %-----% [AL, AR] = FLA_Cont_with_1x3_to_1x2(A0, a1, A2, ... 'FLA_LEFT'); [RTL, RTR, ... RBL, RBR] = FLA_Cont_with_3x3_to_2x2(R00, r01, R02, ... r10t, rho11, r12t, ... R20, r21, R22, ... 'FLA_TL'); end </pre>
Operand	Tag	Type	Direction	Input/Output												
1:	A	matrix	L->R	input/output												
2:	R	matrix	TL->BR	input/output												

Figure 5.5: The Spark webpage filled out for CGS Variant 1.

(The first time you execute the above, you may get a bunch of warnings from Octave. Just ignore those.)

5.4 Implementing the Other Algorithms

Next, we leave it to the reader to implement

- Modified Gram Schmidt algorithm, (MGS_unb_var1, corresponding to the **right-most** algorithm in Figure 5.2), respectively.
- The Householder QR factorization algorithm and algorithm to form Q from “Notes on Householder QR Factorization”.

The routine for computing a Householder transformation (similar to Figure 5.1) can be found at

<http://www.cs.utexas.edu/users/flame/Notes/FLAMEatlab/Housev.m>

That routine implements the algorithm on the left in Figure 5.1). Try and see what happens if you replace it with the algorithm to its right.

Note: For the Householder QR factorization and “form Q ” algorithm how to start the algorithm when the matrix is not square is a bit tricky. Thus, you may assume that the matrix *is* square.

Algorithm: $[A, R] := \text{Gram-Schmidt}(A)$ (overwrites A with Q)

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 $n(A_L) \neq n(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right),$$

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

$$r_{01} := A_0^H a_1$$

$$a_1 := a_1 - A_0 r_{01}$$

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

$$a_1 := a_1 / \rho_{11}$$

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

```

function [ A_out, R_out ] = CGS_unb_var1( A, R )
    [ AL, AR ] = FLA_Part_1x2( A, ...
        0, 'FLA_LEFT' );
    [ RTL, RTR, ...
        RBL, RBR ] = FLA_Part_2x2( R, ...
        0, 0, 'FLA_TL' );
    while ( size( AL, 2 ) < size( A, 2 ) )
        [ A0, a1, A2 ] = FLA_Repart_1x2_to_1x3( AL, AR, ...
            1, 'FLA_RIGHT' );
        [ R00, r01, R02, ...
            r10t, rho11, r12t, ...
            R20, r21, R22 ] = FLA_Repart_2x2_to_3x3( RTL, RTR, ...
            RBL, RBR, ...
            1, 1, 'FLA_BR' );
        %-----%
        %           update line 1 %
        %           : %
        %           update line n %
        %-----%
        [ AL, AR ] = FLA_Cont_with_1x3_to_1x2( A0, a1, A2, ...
            'FLA_LEFT' );
        [ RTL, RTR, ...
            RBL, RBR ] = FLA_Cont_with_3x3_to_2x2( R00, r01, R02, ...
            r10t, rho11, r12t, ...
            R20, r21, R22, ...
            'FLA_TL' );
    end

```

Figure 5.6: Left: Classical Gram-Schmidt algorithm. Right: Generated code-skeleton for CGS.

Chapter **6**

Notes on Householder QR Factorization

Video

Read disclaimer regarding the videos in the preface! This is the video recorded in Fall 2014.

- [▶ YouTube](#)
- [▶ Download from UT Box](#)
- [◀ View After Local Download](#)

(For help on viewing, see Appendix A.)

6.1 Opening Remarks

6.1.1 Launch

A fundamental problem to avoid in numerical codes is the situation where one starts with large values and one ends up with small values with large relative errors in them. This is known as catastrophic cancellation. The Gram-Schmidt algorithms can inherently fall victim to this: column a_j is successively reduced in length as components in the directions of $\{q_0, \dots, q_{j-1}\}$ are subtracted, leaving a small vector if a_j was almost in the span of the first j columns of A . Application of a unitary transformation to a matrix or vector inherently preserves length. Thus, it would be beneficial if the QR factorization can be implemented as the successive application of unitary transformations. The Householder QR factorization accomplishes this.

The first fundamental insight is that the product of unitary matrices is itself unitary. If, given $A \in \mathbb{C}^{m \times n}$ (with $m \geq n$), one could find a sequence of unitary matrices, $\{H_0, \dots, H_{n-1}\}$, such that

$$H_{n-1} \cdots H_0 A = \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where $R \in \mathbb{C}^{n \times n}$ is upper triangular, then

$$A = \underbrace{H_0^H \cdots H_{n-1}^H}_{Q} \begin{pmatrix} R \\ 0 \end{pmatrix} = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_L R,$$

where Q_L equals the first n columns of Q . Then $A = Q_L R$ is the QR factorization of A . The second fundamental insight will be that the desired unitary transformations $\{H_0, \dots, H_{n-1}\}$ can be computed and applied cheaply.

6.1.2 Outline

6.1	Opening Remarks	115
6.1.1	Launch	115
6.1.2	Outline	117
6.1.3	What you will learn	118
6.2	Householder Transformations (Reflectors)	119
6.2.1	The general case	119
6.2.2	As implemented for the Householder QR factorization (real case)	120
6.2.3	The complex case (optional)	121
6.2.4	A routine for computing the Householder vector	122
6.3	Householder QR Factorization	123
6.4	Forming Q	126
6.5	Applying Q^H	131
6.6	Blocked Householder QR Factorization	133
6.6.1	The UT transform: Accumulating Householder transformations	133
6.6.2	The WY transform	135
6.6.3	A blocked algorithm	135
6.6.4	Variations on a theme	139
6.7	Enrichments	142
6.8	Wrapup	142
6.8.1	Additional exercises	142
6.8.2	Summary	146

6.1.3 What you will learn

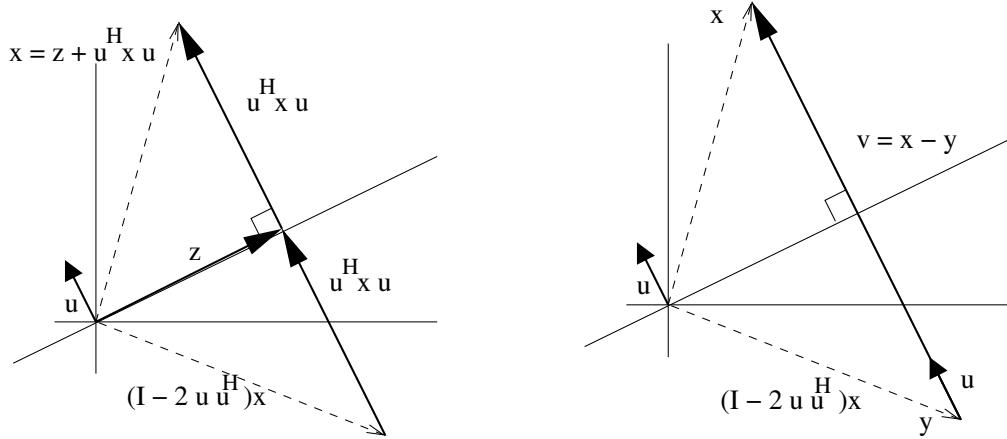


Figure 6.1: Left: Illustration that shows how, given vectors x and unit length vector u , the subspace orthogonal to u becomes a mirror for reflecting x represented by the transformation $(I - 2uu^H)$. Right: Illustration that shows how to compute u given vectors x and y with $\|x\|_2 = \|y\|_2$.

6.2 Householder Transformations (Reflectors)

6.2.1 The general case

In this section we discuss *Householder transformations*, also referred to as *reflectors*.

Definition 6.1 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 reflector or Householder transformation.

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

$$\begin{aligned} (I - 2uu^H)x &= (I - 2uu^H)(z + u^H xu) = z + u^H xu - 2u \underbrace{u^H z}_0 - 2uu^H u^H xu \\ &= z + u^H xu - 2u^H x \underbrace{u^H u}_1 u = z - u^H xu. \end{aligned}$$

This can be interpreted as follows: The space perpendicular to u acts as a “mirror”: any vector in that space (along the mirror) is not reflected, while any other vector has the component that is orthogonal to the space (the component outside, orthogonal to, the mirror) reversed in direction, as illustrated in Figure 6.1. Notice that a reflection preserves the length of the vector.

Homework 6.2 Show that if H is a reflector, then

- $HH = I$ (reflecting the reflection of a vector results in the original vector).
- $H = H^H$.
- $H^H H = I$ (a reflector is unitary).

 SEE ANSWER

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 6.1(right) 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 6.3 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.

In the next subsection, 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).

Let us first discuss how to find H in the case where $x \in \mathbb{R}^n$. We seek v so that $(I - \frac{2}{v^T v}vv^T)x = \pm\|x\|_2 e_0$. Since the resulting vector that we want is $y = \pm\|x\|_2 e_0$, we must choose $v = x - y = x \mp \|x\|_2 e_0$.

Homework 6.4 Show that if $x \in \mathbb{R}^n$, $v = x \mp \|x\|_2 e_0$, and $\tau = v^T v/2$ then $(I - \frac{1}{\tau}vv^T)x = \pm\|x\|_2 e_0$.

 SEE ANSWER

In practice, we choose $v = x + \text{sign}(\chi_1)\|x\|_2 e_0$ where χ_1 denotes the first element of x . The reason is as follows: the first element of v , v_1 , will be $v_1 = \chi_1 \mp \|x\|_2$. If χ_1 is positive and $\|x\|_2$ is almost equal to χ_1 , then $\chi_1 - \|x\|_2$ is a small number and if there is error in χ_1 and/or $\|x\|_2$, this error becomes large relative to the result $\chi_1 - \|x\|_2$, due to catastrophic cancellation. Regardless of whether χ_1 is positive or negative, we can avoid this by choosing $x = \chi_1 + \text{sign}(\chi_1)\|x\|_2 e_0$.

6.2.2 As implemented for the Householder QR factorization (real case)

Next, we discuss a slight variant on the above discussion that is used in practice. To do so, we view x as a vector that consists of its first element, χ_1 , and the rest of the vector, x_2 : More precisely, partition

$$x = \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix},$$

where χ_1 equals the first element of x and x_2 is the rest of x . Then we will wish to find a Householder vector $u = \begin{pmatrix} 1 \\ u_2 \end{pmatrix}$ so that

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

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

$$v = \begin{pmatrix} \chi_1 \mp \|x\|_2 \\ x_2 \end{pmatrix}.$$

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

$$u = \frac{v}{v_1} = \frac{1}{\chi_1 \mp \|x\|_2} \begin{pmatrix} \chi_1 \mp \|x\|_2 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ x_2/v_1 \end{pmatrix}.$$

where $v_1 = \chi_1 \mp \|x\|_2$ equals the first element of v . (Note that if $v_1 = 0$ then u_2 can be set to 0.)

6.2.3 The complex case (optional)

Next, let us work out the complex case, dealing explicitly with x as a vector that consists of its first element, χ_1 , and the rest of the vector, x_2 : More precisely, partition

$$x = \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix},$$

where χ_1 equals the first element of x and x_2 is the rest of x . Then we will wish to find a Householder vector $u = \begin{pmatrix} 1 \\ u_2 \end{pmatrix}$ so that

$$\left(I - \frac{1}{\tau} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \right) \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \oplus\|x\|_2 \\ 0 \end{pmatrix}.$$

Here \oplus denotes a complex scalar on the complex unit circle. By the same argument as before

$$v = \begin{pmatrix} \chi_1 - \oplus\|x\|_2 \\ x_2 \end{pmatrix}.$$

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

$$u = \frac{v}{\|v\|_2} = \frac{1}{\chi_1 - \oplus\|x\|_2} \begin{pmatrix} \chi_1 - \oplus\|x\|_2 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ x_2/v_1 \end{pmatrix}.$$

where $v_1 = \chi_1 - \oplus\|x\|_2$. (If $v_1 = 0$ then we set u_2 to 0.)

Homework 6.5 Verify that

$$\left(I - \frac{1}{\tau} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \right) \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \rho \\ 0 \end{pmatrix}$$

where $\tau = u^H u / 2 = (1 + u_2^H u_2) / 2$ and $\rho = \oplus \|x\|_2$.

Hint: $\rho \bar{\rho} = |\rho|^2 = \|x\|_2^2$ since H preserves the norm. Also, $\|x\|_2^2 = |\chi_1|^2 + \|x_2\|_2^2$ and $\sqrt{\frac{z}{\bar{z}}} = \frac{z}{|\bar{z}|}$.

☞ SEE ANSWER

Again, the choice \oplus is important. For the complex case we choose $\oplus = -\text{sign}(\chi_1) = \frac{\chi_1}{|\chi_1|}$

6.2.4 A routine for computing the Householder vector

We will refer to the vector

$$\begin{pmatrix} 1 \\ u_2 \end{pmatrix}$$

as the Householder vector that reflects x into $\oplus \|x\|_2 e_0$ and introduce the notation

$$\left[\begin{pmatrix} \rho \\ u_2 \end{pmatrix}, \tau \right] := \text{Housev} \left(\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \right)$$

as the computation of the above mentioned vector u_2 , and scalars ρ and τ , from vector x . We will use the notation $H(x)$ for the transformation $I - \frac{1}{\tau}uu^H$ where u and τ are computed by $\text{Housev}(x)$.

The function

```
function [ rho, ...
    u2, tau ] = Housev( chi1, ...
    x2 )
```

implements the function **Housev**. It can be found in

Programming/chapter06/Housev.m (see file only) (view in MATLAB)

Homework 6.6 Function **Housev.m** implements the steps in Figure 6.2 (left). Update this implementation with the equivalent steps in Figure 6.2 (right), which is closer to how it is implemented in practice.

☞ SEE ANSWER

Algorithm:	$\left[\begin{pmatrix} \rho \\ u_2 \end{pmatrix}, \tau \right] = \text{Housev} \left(\begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \right)$
$\rho = -\text{sign}(\chi_1) \ x\ _2$	$\chi_2 := \ x_2\ _2$
$v_1 = \chi_1 + \text{sign}(\chi_1) \ x\ _2$	$\alpha := \left\ \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} \right\ _2 (= \ x\ _2)$
$u_2 = x_2/v_1$	$\rho := -\text{sign}(\chi_1) \alpha$
$\tau = (1 + u_2^H u_2)/2$	$v_1 := \chi_1 - \rho$
	$u_2 := x_2/v_1$
	$\chi_2 = \chi_2/ v_1 (= \ u_2\ _2)$
	$\tau = (1 + \chi_2^2)/2$

Figure 6.2: 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.

6.3 Householder QR Factorization

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 6.3.

In the first iteration, we partition

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

Let

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

be the Householder transform computed from the first column of A . Then applying this Householder transform to A yields

$$\begin{aligned} \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) &:= \left(I - \frac{1}{\tau_1} \begin{pmatrix} 1 \\ u_2 \end{pmatrix} \begin{pmatrix} 1 \\ u_2 \end{pmatrix}^H \right) \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) \\ &= \left(\begin{array}{c|c} \rho_{11} & a_{12}^T - w_{12}^T \\ \hline 0 & A_{22} - u_{21} w_{12}^T \end{array} \right), \end{aligned}$$

where $w_{12}^T = (a_{12}^T + u_{21}^H A_{22})/\tau_1$. Computation of a full QR factorization of A will now proceed with the updated matrix A_{22} .

Original matrix	$\begin{bmatrix} \left(\frac{\rho_{11}}{u_{21}} \right), \tau_1 \\ \text{Housev} \left(\frac{\alpha_{11}}{a_{21}} \right) \end{bmatrix} = \begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \\ \rho_{11} & a_{12}^T - w_{12}^T \\ 0 & A_{22} - u_{21}w_{12}^T \end{pmatrix} :=$	“Move forward”	

Figure 6.3: Illustration of Householder QR factorization.

Now let us assume that after k iterations of the algorithm matrix A contains

$$A \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & A_{BR} \end{array} \right) = \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right),$$

where R_{TL} and R_{00} are $k \times k$ upper triangular matrices. Let

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

and update

$$\begin{aligned} A &:= \left(\begin{array}{c|c} I & 0 \\ \hline 0 & \left(I - \frac{1}{\tau_1} \left(\frac{1}{u_2} \right) \left(\frac{1}{u_2} \right)^H \right) \end{array} \right) \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) \\ &= \left(I - \frac{1}{\tau_1} \left(\frac{0}{1} \right) \left(\frac{0}{1} \right)^H \right) \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) \\ &= \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \rho_{11} & a_{12}^T - w_{12}^T \\ \hline 0 & 0 & A_{22} - u_{21}w_{12}^T \end{array} \right), \end{aligned}$$

where again $w_{12}^T = (a_{12}^T + u_{21}^H A_{22})/\tau_1$.

Let

$$H_k = \left(I - \frac{1}{\tau_1} \left(\frac{0_k}{1} \right) \left(\frac{0_k}{1} \right)^H \right)$$

be the Householder transform so computed during the $(k+1)$ st iteration. Then upon completion matrix A contains

$$R = \left(\begin{array}{c} R_{TL} \\ \hline 0 \end{array} \right) = H_{n-1} \cdots H_1 H_0 \hat{A}$$

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 $\hat{A} = QR$.

Homework 6.7 Show that

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

☞ SEE ANSWER

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 Fig. 6.4.

We will let

$$[\{U \setminus R\}, t] = \text{HQR}(A)$$

denote 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 6.8 Given $A \in \mathbb{R}^{m \times n}$ show that the cost of the algorithm in Figure 6.4 is given by

$$C_{\text{HQR}}(m, n) \approx 2mn^2 - \frac{2}{3}n^3 \text{ flops.}$$

☞ SEE ANSWER

Homework 6.9 Implement the algorithm in Figure 6.4 as

```
function [ Aout, tout ] = HQR_unb_var1( A, t )
```

Input is an $m \times n$ matrix A and vector t of size n . Output is the overwritten matrix A and the vector of scalars that define the Householder transformations. You may want to use Programming/chapter06/test_HQR_unb_var1.m to check your implementation.

☞ SEE ANSWER

6.4 Forming Q

Given $A \in \mathbb{C}^{m \times n}$, let $[A, t] = \text{HQR}(A)$ yield the matrix A with the Householder vectors stored below the diagonal, R stored on and above the diagonal, and the τ_i 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 6.5.

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

$$Q \begin{pmatrix} I_{n \times n} \\ 0 \end{pmatrix} = H_0 \cdots H_{n-1} \begin{pmatrix} I_{n \times n} \\ 0 \end{pmatrix} = H_0 \cdots H_{k-1} \underbrace{H_k \cdots H_{n-1} \begin{pmatrix} I_{n \times n} \\ 0 \end{pmatrix}}_{B_k}.$$

where B_k is defined as indicated.

Algorithm: $[A, t] = \text{HQR_UNB_VAR1}(A, t)$

Partition $A \rightarrow \begin{pmatrix} A_{TL} & | & A_{TR} \\ \hline A_{BL} & | & A_{BR} \end{pmatrix}$ and $t \rightarrow \begin{pmatrix} t_T \\ \hline t_B \end{pmatrix}$

where A_{TL} is 0×0 and t_T has 0 elements

while $n(A_{BR}) \neq 0$ **do**

Repartition

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

where α_{11} and τ_1 are scalars

$$\left[\left(\frac{\alpha_{11}}{a_{21}} \right), \tau_1 \right] := \left[\left(\frac{\rho_{11}}{u_{21}} \right), \tau_1 \right] = \text{Housev} \left(\frac{\alpha_{11}}{a_{21}} \right)$$

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

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

Continue with

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

endwhile

Figure 6.4: Unblocked Householder transformation based QR factorization.

Original matrix	$\left(\begin{array}{c c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \\ \hline 1 - 1/\tau_1 & -(u_{21}^H A_{22})/\tau_1 \\ \hline -u_{21}/\tau_1 & A_{22} + u_{21} a_{12}^T \end{array} \right) :=$	“Move forward”
$\begin{array}{cccc c} 1 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & \\ 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 & \\ \hline 0 & 0 & 0 & 0 & \end{array}$	$\begin{array}{ccc c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \end{array}$	$\begin{array}{ccc c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \end{array}$
	$\begin{array}{cc cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & \times & \times \\ \hline 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{array}$	$\begin{array}{cc cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & \times & \times \\ \hline 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{array}$
	$\begin{array}{c cc cc} 1 & 0 & 0 & 0 & \\ 0 & \times & \times & \times & \\ \hline 0 & \times & \times & \times & \\ 0 & \times & \times & \times & \\ 0 & \times & \times & \times & \end{array}$	$\begin{array}{c cc cc} 1 & 0 & 0 & 0 & \\ 0 & \times & \times & \times & \\ \hline 0 & \times & \times & \times & \\ 0 & \times & \times & \times & \\ 0 & \times & \times & \times & \end{array}$
	$\begin{array}{c cc cc} \times & \times & \times & \times & \\ \hline \times & \times & \times & \times & \\ \times & \times & \times & \times & \\ \times & \times & \times & \times & \\ \times & \times & \times & \times & \end{array}$	$\begin{array}{c cc cc} \times & \times & \times & \times & \\ \times & \times & \times & \times & \\ \times & \times & \times & \times & \\ \times & \times & \times & \times & \\ \times & \times & \times & \times & \end{array}$

Figure 6.5: Illustration of the computation of Q .

Lemma 6.10 B_k has the form

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

Proof: The proof of this is by induction on k :

- **Base case:** $k = n$. Then $B_n = \begin{pmatrix} I_{n \times n} \\ 0 \end{pmatrix}$, which has the desired form.
- **Inductive step:** Assume the result is true for B_k . We show it is true for B_{k-1} :

$$\begin{aligned}
B_{k-1} &= H_{k-1}H_k \cdots H_{n-1} \begin{pmatrix} I_{n \times n} \\ 0 \end{pmatrix} = H_{k-1}B_k = H_{k-1} \begin{pmatrix} I_{k \times k} & 0 \\ 0 & \tilde{B}_k \end{pmatrix}. \\
&= \begin{pmatrix} I_{(k-1) \times (k-1)} & 0 \\ 0 & I - \frac{1}{\tau_k} \begin{pmatrix} 1 \\ u_k^H \end{pmatrix} \begin{pmatrix} 1 & | & u_k^H \end{pmatrix} \end{pmatrix} \begin{pmatrix} I_{(k-1) \times (k-1)} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \tilde{B}_k \end{pmatrix} \\
&= \begin{pmatrix} I_{(k-1) \times (k-1)} & 0 \\ 0 & \left(I - \frac{1}{\tau_k} \begin{pmatrix} 1 \\ u_k^H \end{pmatrix} \begin{pmatrix} 1 & | & u_k^H \end{pmatrix} \right) \begin{pmatrix} 1 & 0 \\ 0 & \tilde{B}_k \end{pmatrix} \end{pmatrix} \\
&= \begin{pmatrix} I_{(k-1) \times (k-1)} & 0 \\ 0 & \left(\begin{pmatrix} 1 & 0 \\ 0 & \tilde{B}_k \end{pmatrix} - \begin{pmatrix} 1 \\ u_k \end{pmatrix} \begin{pmatrix} 1/\tau_k & | & y_k^T \end{pmatrix} \right) \end{pmatrix} \quad \text{where } y_k^T = u_k^H \tilde{B}_k / \tau_k \\
&= \begin{pmatrix} I_{(k-1) \times (k-1)} & 0 \\ 0 & \begin{pmatrix} 1 - 1/\tau_k & -y_k^T \\ -u_k/\tau_k & \tilde{B}_k - u_k y_k^T \end{pmatrix} \end{pmatrix} \\
&= \begin{pmatrix} I_{(k-1) \times (k-1)} & 0 \\ 0 & \begin{pmatrix} 1 - 1/\tau_k & -y_k^T \\ -u_k/\tau_k & \tilde{B}_k - u_k y_k^T \end{pmatrix} \end{pmatrix} = \begin{pmatrix} I_{(k-1) \times (k-1)} & 0 \\ 0 & \tilde{B}_{k-1} \end{pmatrix}.
\end{aligned}$$

- **By the Principle of Mathematical Induction** the result holds for B_0, \dots, B_n .

Theorem 6.11 Given $[A, t] = \text{HQR}(A, t)$ from Figure 6.4, the algorithm in Figure 6.6 overwrites A with the first $n = n(A)$ columns of Q as defined by the Householder transformations stored below the diagonal of A and in the vector t .

Proof: The algorithm is justified by the proof of Lemma 6.10.

Homework 6.12 Implement the algorithm in Figure 6.6 as

```
function Aout = FormQ_unb_var1( A, t )
```

Algorithm: $[A] = \text{FORMQ_UNB_VAR1}(A, t)$

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

where A_{TL} is $n(A) \times n(A)$ and t_T has $n(A)$ elements

while $n(A_{TR}) \neq 0$ **do**

Repartition

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

where α_{11} and τ_1 are scalars

Update $\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) := \left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21}^H \end{array} \right) \left(\begin{array}{c|c} 1 & u_{21}^H \end{array} \right)^T \right) \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & A_{22} \end{array} \right)$

via the steps

- $\alpha_{11} := 1 - 1/\tau_1$
 - $a_{12}^T := -(a_{21}^H A_{22})/\tau_1$
 - $A_{22} := A_{22} + a_{21} a_{12}^T$
 - $a_{21} := -a_{21}/\tau_1$
-

Continue with

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

endwhile

Figure 6.6: Algorithm for overwriting A with Q from the Householder transformations stored as Householder vectors below the diagonal of A (as produced by $[A, t] = \text{HQR_UNB_VAR1}(A, t)$).

You may want to use Programming/chapter06/test_HQR_unb_var1.m to check your implementation.

 SEE ANSWER

Homework 6.13 Given $A \in \mathbb{C}^{m \times n}$ the cost of the algorithm in Figure 6.6 is given by

$$C_{\text{FormQ}}(m, n) \approx 2mn^2 - \frac{2}{3}n^3 \text{ flops.}$$

 SEE ANSWER

Homework 6.14 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 6.6.

 SEE ANSWER

6.5 Applying Q^H

In a future Note, we will see that the QR factorization is used to solve the linear least-squares problem. To do so, we need to be able to compute $\hat{y} = Q^H y$ where $Q^H = H_{n-1} \cdots H_0$.

Let us start by computing $H_0 y$:

$$\begin{aligned} \left(I - \frac{1}{\tau_1} \left(\frac{1}{u_2} \right) \left(\frac{1}{u_2} \right)^H \right) \left(\begin{pmatrix} \Psi_1 \\ y_2 \end{pmatrix} \right) &= \left(\begin{pmatrix} \Psi_1 \\ y_2 \end{pmatrix} \right) - \left(\frac{1}{u_2} \right) \underbrace{\left(\frac{1}{u_2} \right)^H \left(\begin{pmatrix} \Psi_1 \\ y_2 \end{pmatrix} \right) / \tau_1}_{\omega_1} \\ &= \left(\begin{pmatrix} \Psi_1 \\ y_2 \end{pmatrix} \right) - \omega_1 \left(\frac{1}{u_2} \right) = \left(\frac{\Psi_1 - \omega_1}{y_2 - \omega_1 u_2} \right). \end{aligned}$$

More generally, let us compute $H_k y$:

$$\left(I - \frac{1}{\tau_1} \left(\frac{0}{u_2} \right) \left(\frac{0}{u_2} \right)^H \right) \left(\begin{pmatrix} y_0 \\ \Psi_1 \\ y_2 \end{pmatrix} \right) = \left(\begin{pmatrix} y_0 \\ \frac{\Psi_1 - \omega_1}{y_2 - \omega_1 u_2} \end{pmatrix} \right),$$

where $\omega_1 = (\Psi_1 + u_2^H y_2) / \tau_1$. This motivates the algorithm in Figure 6.7 for computing $y := H_{n-1} \cdots H_0 y$ given the output matrix A and vector t from routine HQR.

The cost of this algorithm can be analyzed as follows: When y_T is of length k , the bulk of the computation is in an inner product with vectors of length $m - k$ (to compute ω_1) and an axpy operation with vectors of length $m - k$ to subsequently update Ψ_1 and y_2 . Thus, the cost is approximately given by

$$\sum_{k=0}^{n-1} 4(m - k) \approx 4mn - 2n^2.$$

Notice that this is *much* cheaper than forming Q and then multiplying.

Algorithm: $[y] = \text{APPLYQT_UNB_VAR1}(A, t, y)$

Partition $A \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix}$, $t \rightarrow \begin{pmatrix} t_T \\ t_B \end{pmatrix}$, and $y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix}$

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

while $n(A_{BR}) \neq 0$ **do**

Repartition

$$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{pmatrix}, \begin{pmatrix} t_T \\ t_B \end{pmatrix} \rightarrow \begin{pmatrix} t_0 \\ \tau_1 \\ t_2 \end{pmatrix}, \text{ and } \begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \psi_1 \\ y_2 \end{pmatrix}$$

where α_{11} , τ_1 , and ψ_1 are scalars

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

- $\omega_1 := (\psi_1 + a_{21}^H y_2) / \tau_1$
- $\begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix} := \begin{pmatrix} \psi_1 - \omega_1 \\ y_2 - \omega_1 u_2 \end{pmatrix}$

Continue with

$$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{pmatrix}, \begin{pmatrix} t_T \\ t_B \end{pmatrix} \leftarrow \begin{pmatrix} t_0 \\ \tau_1 \\ t_2 \end{pmatrix}, \text{ and } \begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \psi_1 \\ y_2 \end{pmatrix}$$

endwhile

Figure 6.7: Algorithm for computing $y := H_{n-1} \cdots H_0 y$ given the output from the algorithm HQR_UNB_VAR1.

6.6 Blocked Householder QR Factorization

6.6.1 The UT transform: Accumulating Householder transformations

Through a series of exercises, we will show how the application of a sequence of k Householder transformations can be accumulated. What we exactly mean that this will become clear.

Homework 6.15 Consider $u_1 \in \mathbb{C}^m$ with $u_1 \neq 0$ (the zero vector), $U_0 \in \mathbb{C}^{m \times k}$, and nonsingular $T_{00} \in \mathbb{C}^{k \times k}$. Define $\tau_1 = (u_1^H u_1)/2$, so that

$$H_1 = I - \frac{1}{\tau_1} u_1 u_1^H$$

equals a Householder transformation, and let

$$Q_0 = I - U_0 T_{00}^{-1} U_0^H.$$

Show that

$$Q_0 H_1 = (I - U_0 T_{00}^{-1} U_0^H) (I - \frac{1}{\tau_1} u_1 u_1^H) = I - \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} T_{00} & t_{01} \\ 0 & \tau_1 \end{array} \right)^{-1} \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right)^H,$$

where $t_{01} = Q_0^H u_1$.

SEE ANSWER

Homework 6.16 Consider $u_i \in \mathbb{C}^m$ with $u_i \neq 0$ (the zero vector). Define $\tau_i = (u_i^H u_i)/2$, so that

$$H_i = I - \frac{1}{\tau_i} u_i u_i^H$$

equals a Householder transformation, and let

$$U = \left(\begin{array}{c|c|c|c} u_0 & u_1 & \cdots & u_{k-1} \end{array} \right).$$

Show that

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

where T is an upper triangular matrix.

SEE ANSWER

The above exercises can be summarized in the algorithm for computing T from U in Figure 6.8.

Homework 6.17 Implement the algorithm in Figure 6.8 as

```
function T = FormT_unb_var1( U, t, T )
```

SEE ANSWER

In [28] 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 be computed as

$$T = \text{triu}(U^H U)$$

(the upper triangular part of $U^H U$) followed by either dividing the diagonal elements by two or setting them to $\tau_0, \dots, \tau_{k-1}$ (in order). In that paper, we point out similar published results [10, 35, 46, 32].

Algorithm: $[T] := \text{FORMT_UNB_VAR1}(U, t, T)$

Partition $U \rightarrow \begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix}$, $t \rightarrow \begin{pmatrix} t_T \\ t_B \end{pmatrix}$, $T \rightarrow \begin{pmatrix} T_{TL} & T_{TR} \\ \hline T_{BL} & T_{BR} \end{pmatrix}$

where U_{TL} is 0×0 , t_T has 0 rows, T_{TL} is 0×0

while $m(U_{TL}) < m(U)$ **do**

Repartition

$$\begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{pmatrix}, \quad \begin{pmatrix} t_T \\ t_B \end{pmatrix} \rightarrow \begin{pmatrix} t_0 \\ \tau_1 \\ t_2 \end{pmatrix},$$

$$\begin{pmatrix} T_{TL} & T_{TR} \\ \hline T_{BL} & T_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} T_{00} & t_{01} & T_{02} \\ \hline t_{10}^T & \tau_{11} & t_{12}^T \\ \hline T_{20} & t_{21} & T_{22} \end{pmatrix}$$

where v_{11} is 1×1 , τ_1 has 1 row, τ_{11} is 1×1

$$t_{01} := (u_{10}^T)^H + U_{20}^H u_{21}$$

$$\tau_{11} := \tau_1$$

Continue with

$$\begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{pmatrix}, \quad \begin{pmatrix} t_T \\ t_B \end{pmatrix} \leftarrow \begin{pmatrix} t_0 \\ \tau_1 \\ t_2 \end{pmatrix},$$

$$\begin{pmatrix} T_{TL} & T_{TR} \\ \hline T_{BL} & T_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} T_{00} & t_{01} & T_{02} \\ \hline t_{10}^T & \tau_{11} & t_{12}^T \\ \hline T_{20} & t_{21} & T_{22} \end{pmatrix}$$

endwhile

Figure 6.8: Algorithm that computes T from U and the vector t so that $I - UT^{-1}U^H$ equals the UT transform. Here U is assumed to be the output of, for example, `HouseQR_unb_var1`. This means that it is a lower trapezoidal matrix, with ones on the diagonal.

6.6.2 The WY transform

An alternative way of expressing a Householder transform is

$$I - \beta vv^T,$$

where $\beta = 2/v^T v$ ($= 1/\tau$, where τ is as discussed before). This leads to an alternative accumulation of Householder transforms known as the *compact WY transform* [35] (which is itself a refinement of the WY representation of products of Householder Transformations [10]):

$$I - USU^H$$

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), yielding the algorithm in Figure 6.10.

An algorithm that computes the inverse of an upper triangular matrix T based on the above exercise is given in Figure 6.9.

Homework 6.18 Assuming all inverses exist, show that

$$\left(\begin{array}{c|c} T_{00} & t_{01} \\ \hline 0 & \tau_1 \end{array} \right)^{-1} = \left(\begin{array}{c|c} T_{00}^{-1} & -T_{00}^{-1}t_{01}/\tau_1 \\ \hline 0 & 1/\tau_1 \end{array} \right).$$

☞ SEE ANSWER

6.6.3 A blocked algorithm

A QR factorization that exploits the insights that resulted in the UT transform can now be described:

- Partition

$$A \rightarrow \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right)$$

where A_{11} is $b \times b$.

- We can use the unblocked algorithm in Figure 6.4 to factor the panel $\left(\begin{array}{c} A_{11} \\ A_{21} \end{array} \right)$

$$\left[\left(\begin{array}{c} A_{11} \\ A_{21} \end{array} \right), t_1 \right] := \text{HOUSEQR_UNB_VAR1} \left(\left(\begin{array}{c} A_{11} \\ A_{21} \end{array} \right) \right),$$

overwriting the entries below the diagonal with the Householder vectors $\left(\begin{array}{c} U_{11} \\ U_{21} \end{array} \right)$ (with the ones on the diagonal implicitly stored) and the upper triangular part with R_{11} .

Algorithm: $[T] := \text{UTRINV_UNB_VAR1}(T)$

Partition $T \rightarrow \left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline T_{BL} & T_{BR} \end{array} \right)$

where T_{TL} is 0×0

while $m(T_{TL}) < m(T)$ **do**

Repartition

$$\left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline T_{BL} & T_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} T_{00} & t_{01} & T_{02} \\ \hline t_{10}^T & \tau_{11} & t_{12}^T \\ \hline T_{20} & t_{21} & T_{22} \end{array} \right)$$

where τ_{11} is 1×1

$$t_{01} := -T_{00}t_{01}/\tau_{11}$$

$$\tau_{11} := 1/\tau_{11}$$

Continue with

$$\left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline T_{BL} & T_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} T_{00} & t_{01} & T_{02} \\ \hline t_{10}^T & \tau_{11} & t_{12}^T \\ \hline T_{20} & t_{21} & T_{22} \end{array} \right)$$

endwhile

Figure 6.9: Unblocked algorithm for inverting an upper triangular matrix. The algorithm assumes that T_{00} has already been inverted, and computes the next column of T in the current iteration.

Algorithm: $[S] := \text{FORMS_UNB_VAR1}(U, t, S)$

Partition $U \rightarrow \begin{pmatrix} U_{TL} & U_{TR} \\ U_{BL} & U_{BR} \end{pmatrix}$, $t \rightarrow \begin{pmatrix} t_T \\ t_B \end{pmatrix}$, $S \rightarrow \begin{pmatrix} S_{TL} & S_{TR} \\ S_{BL} & S_{BR} \end{pmatrix}$

where U_{TL} is 0×0 , t_T has 0 rows, S_{TL} is 0×0

while $m(U_{TL}) < m(U)$ **do**

Repartition

$$\begin{pmatrix} U_{TL} & U_{TR} \\ U_{BL} & U_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{pmatrix}, \quad \begin{pmatrix} t_T \\ t_B \end{pmatrix} \rightarrow \begin{pmatrix} t_0 \\ \tau_1 \\ t_2 \end{pmatrix},$$

$$\begin{pmatrix} S_{TL} & S_{TR} \\ S_{BL} & S_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} S_{00} & s_{01} & S_{02} \\ s_{10}^T & \sigma_{11} & s_{12}^T \\ S_{20} & t_{21} & S_{22} \end{pmatrix}$$

where v_{11} is 1×1 , τ_1 has 1 row, σ_{11} is 1×1

$$\tau_{11} := 1/\tau_1$$

$$s_{01} := -\tau_{11} U_{00} ((u_{10}^T)^H + U_{20}^H u_{21})$$

Continue with

$$\begin{pmatrix} U_{TL} & U_{TR} \\ U_{BL} & U_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{pmatrix}, \quad \begin{pmatrix} t_T \\ t_B \end{pmatrix} \leftarrow \begin{pmatrix} t_0 \\ \tau_1 \\ t_2 \end{pmatrix},$$

$$\begin{pmatrix} S_{TL} & S_{TR} \\ S_{BL} & S_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} S_{00} & s_{01} & S_{02} \\ s_{10}^T & \sigma_{11} & s_{12}^T \\ S_{20} & s_{21} & S_{22} \end{pmatrix}$$

endwhile

Figure 6.10: Algorithm that computes S from U and the vector t so that $I - USU^H$ equals the compact WY transform. Here U is assumed to be the output of, for example, `HouseQR_unb_var1`. This means that it is a lower trapezoidal matrix, with ones on the diagonal.

Algorithm: $[A, t] := \text{HOUSEQR_BLK_VAR1}(A, t)$

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

where A_{TL} is 0×0 , t_T has 0 rows

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

Determine block size b

Repartition

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

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

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

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

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

$$\left(\begin{array}{c} A_{12} \\ A_{22} \end{array} \right) := \left(\begin{array}{c} A_{12} - U_{11} W_{12} \\ A_{22} - U_{21} W_{12} \end{array} \right)$$

Continue with

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

endwhile

Figure 6.11: Blocked Householder transformation based QR factorization.

- For T_{11} from the Householder vectors using the procedure described in Section 6.6.1:

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

- Now we need to also apply the Householder transformations to the rest of the columns:

$$\begin{aligned}
 \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} &:= \left(I - \left(\frac{U_{11}}{U_{21}} \right) T_{11}^{-1} \left(\frac{U_{11}}{U_{21}} \right)^H \right)^H \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} \\
 &= \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} - \left(\frac{U_{11}}{U_{21}} \right) W_{12} \\
 &= \begin{pmatrix} A_{12} - U_{11}W_{12} \\ A_{22} - U_{21}W_{12} \end{pmatrix},
 \end{aligned}$$

where

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

This motivates the blocked algorithm in Figure 6.11.

6.6.4 Variations on a theme

Merging the unblocked Householder QR factorization and the formation of T

There are many possible algorithms for computing the QR factorization. For example, the unblocked algorithm from Figure 6.4 can be merged with the unblocked algorithm for forming T in Figure 6.8 to yield the algorithm in Figure 6.12.

An alternative unblocked merged algorithm

Let us now again compute the QR factorization of A simultaneous with the forming of T , but now taking advantage of the fact that T is partially computed to change the algorithm into what some would consider a “left-looking” algorithm.

Partition

$$A \rightarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{pmatrix} \quad \text{and} \quad T \rightarrow \begin{pmatrix} T_{00} & t_{01} & T_{02} \\ 0 & \tau_{11} & t_{12}^T \\ 0 & 0 & T_{22} \end{pmatrix}.$$

Assume that $\begin{pmatrix} A_{00} \\ a_{10}^T \\ A_{20} \end{pmatrix}$ has been factored and overwritten with $\begin{pmatrix} U_{00} \\ u_{10}^T \\ U_{20} \end{pmatrix}$ and R_{00} while also computing T_{00} . In the next step, we need to apply previous Householder transformations to the next column of A and then update that column with the next column of R and U . In addition, the next column of T must be computing. This means:

Algorithm: $[A, T] := \text{HOUSEQR_AND_FORMT_UNB_VAR1}(A, T)$

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

where A_{TL} is 0×0 , T_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1 , τ_{11} is 1×1

$$\left[\left(\frac{\alpha_{11}}{a_{21}} \right), \tau_{11} \right] := \left[\left(\frac{\rho_{11}}{u_{21}} \right), \tau_{11} \right] = \text{Housev} \left(\frac{\alpha_{11}}{a_{21}} \right)$$

$$\text{Update } \left(\frac{a_{12}^T}{A_{22}} \right) := \left(I - \frac{1}{\tau_{11}} \left(\frac{1}{u_{21}} \right) \left(\begin{array}{c|c} 1 & u_{21}^H \end{array} \right) \right) \left(\frac{a_{12}^T}{A_{22}} \right)$$

via the steps

- $w_{12}^T := (a_{12}^T + a_{21}^H A_{22}) / \tau_{11}$
 - $\left(\frac{a_{12}^T}{A_{22}} \right) := \left(\frac{a_{12}^T - w_{12}^T}{A_{22} - a_{21} w_{12}^T} \right)$
- $$t_{01} := (a_{10}^T)^H + A_{20}^H a_{21}$$

Continue with

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

endwhile

Figure 6.12: Unblocked Householder transformation based QR factorization merged with the computation of T for the UT transform.

Algorithm: $[A, T] := \text{HOUSEQR_AND_FORMT_UNB_VAR2}(A, T)$

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

where A_{TL} is 0×0 , T_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1 , τ_{11} is 1×1

$$\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{02} \end{array} \right) := \left(I - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array} \right) T_{00}^{-1} \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array} \right)^H \right)^H \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array} \right)$$

via the steps

- $w_{01} := T_{00}^{-H} (U_{00}^H a_{01} + \alpha_{11} (u_{10}^T)^H + U_{20}^H a_{21})$
- $\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{02} \end{array} \right) := \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{02} \end{array} \right) - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array} \right) w_{01}$
- $\left[\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array} \right), \tau_{11} \right] := \left[\left(\begin{array}{c} \rho_{11} \\ \hline u_{21} \end{array} \right), \tau_{11} \right] = \text{Housev} \left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array} \right)$

$t_{01} := (a_{10}^T)^H + A_{20}^H a_{21}$

Continue with

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

endwhile

Figure 6.13: Alternative unblocked Householder transformation based QR factorization merged with the computation of T for the UT transform.

- Update

$$\begin{pmatrix} \frac{a_{01}}{\alpha_{11}} \\ \frac{\alpha_{11}}{a_{21}} \end{pmatrix} := \left(I - \begin{pmatrix} \frac{U_{00}}{u_{10}^T} \\ \frac{u_{10}^T}{U_{20}} \end{pmatrix} T_{00}^{-1} \begin{pmatrix} \frac{U_{00}}{u_{10}^T} \\ \frac{u_{10}^T}{U_{20}} \end{pmatrix}^H \right)^H \begin{pmatrix} \frac{a_{01}}{\alpha_{11}} \\ \frac{\alpha_{11}}{a_{21}} \end{pmatrix}$$

- Compute the next Householder transform:

$$[\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}, \tau_{11}] := \text{Housev}\left(\begin{pmatrix} \alpha_{11} \\ a_{21} \end{pmatrix}\right)$$

- Compute the rest of the next column of T

$$t_{01} := A_{20}^H a_{21}$$

This yields the algorithm in Figure 6.13.

Alternative blocked algorithm (Variant 2)

An alternative blocked variant that uses either of the unblocked factorization routines that merges the formation of T is given in Figure 6.14.

6.7 Enrichments

6.8 Wrapup

6.8.1 Additional exercises

Homework 6.19 In Section 4.4 we discuss how MGS yields a higher quality solution than does CGS in terms of the orthogonality of the computed columns. The classic example, already mentioned in Chapter 4, that illustrates this is

$$A = \left(\begin{array}{c|cc} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{array} \right),$$

where $\epsilon = \sqrt{\epsilon_{\text{mach}}}$. In this exercise, you will compare and contrast the quality of the computed matrix Q for CGS, MGS, and Householder QR factorization.

- Start with the matrix

```
format long      % print out 16 digits
eps = 1.0e-8    % roughly the square root of the machine epsilon
A = [
```

Algorithm: $[A, t] := \text{HOUSEQR_BLK_VAR1}(A, t)$

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

where A_{TL} is 0×0 , t_T has 0 rows

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

Determine block size b

Repartition

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

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

$$\left[\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array} \right), T_{11} \right] := \text{HOUSEQR_FORMT_UNB_VARX}\left(\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array} \right) \right)$$

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

$$\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array} \right) := \left(\begin{array}{c} A_{12} - U_{11} W_{12} \\ \hline A_{22} - U_{21} W_{12} \end{array} \right)$$

Continue with

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

endwhile

Figure 6.14: Alternative blocked Householder transformation based QR factorization.

```

1   1   1
eps  0   0
0   eps  0
0   0   eps
}

```

- With the various routines you implemented for Chapter 4 and the current chapter compute

```

[ Q_CGS, R_CGS ] = CGS_unb_var1( A )
[ Q_MGS, R_MGS ] = MGS_unb_var1( A )
[ A_HQR, t_HQR ] = HQR_unb_var1( A )
Q_HQR = FormQ_unb_var1( A_HQR, t_HQR )

```

- Finally, check whether the columns of the various computed matrices Q are mutually orthonormal:

```

Q_CGS' * Q_CGS
Q_MGS' * Q_MGS
Q_HQR' * Q_HQR

```

What do you notice? When we discuss numerical stability of algorithms we will gain insight into why HQR produces high quality mutually orthogonal columns.

- Check how well QR approximates A :

```

A = Q_CGS * triu( R_CGS )
A = Q_MGS * triu( R_MGS )
A = Q_HQR * triu( A_HQR( 1:3, 1:3 ) )

```

What you notice is that all approximate A well.

Later, we will see how the QR factorization can be used to solve $Ax = b$ and linear least-squares problems. At that time we will examine how accurate the solution is depending on which method for QR factorization is used to compute Q and R .

 SEE ANSWER

Homework 6.20 Consider the matrix $\begin{pmatrix} A \\ B \end{pmatrix}$ where A has linearly independent columns. Let

- $A = Q_A R_A$ be the QR factorization of A .

- $\begin{pmatrix} R_A \\ B \end{pmatrix} = Q_B R_B$ be the QR factorization of $\begin{pmatrix} R_A \\ B \end{pmatrix}$.

- $\begin{pmatrix} A \\ B \end{pmatrix} = QR$ be the QR factorization of $\begin{pmatrix} A \\ B \end{pmatrix}$.

Algorithm: $[R, B, t] := \text{HQR_UPDATE_UNB_VAR1}(R, B, t)$

Partition $R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right), B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right)$

where R_{TL} is 0×0 , B_L has 0 columns, t_T has 0 rows

while $m(R_{TL}) < m(R)$ **do**

Repartition

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

$$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array} \right)$$

where ρ_{11} is 1×1 , b_1 has 1 column, τ_1 has 1 row

Continue with

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

$$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right), \left(\begin{array}{c} t_T \\ \hline t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \hline \tau_1 \\ \hline t_2 \end{array} \right)$$

endwhile

Figure 6.15: Outline for algorithm that compute the QR factorization of a triangular matrix, R , appended with a matrix B . Upon completion, the Householder vectors are stored in B and the scalars associated with the Householder transformations in t .

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

 [SEE ANSWER](#)

Homework 6.21 Consider the matrix $\begin{pmatrix} R \\ \hline B \end{pmatrix}$ where R is an upper triangular matrix. Propose a modification of HQR_UNB_VAR1 that overwrites R and B with Householder vectors and updated matrix R . Importantly, the algorithm should take advantage of the zeros in R (in other words, it should avoid computing with the zeroes below its diagonal). An outline for the algorithm is given in Figure 6.15.

 [SEE ANSWER](#)

Homework 6.22 Implement the algorithm from Homework 6.22.

 [SEE ANSWER](#)

6.8.2 Summary

Notes on Rank Revealing Householder QR Factorization

7.1 Opening Remarks

7.1.1 Launch

It may be good at this point to skip forward to Section 12.4.1 and learn about permutation matrices.

Given $A \in \mathbb{C}^{m \times n}$, with $m \geq n$, the reduced QR factorization $A = QR$ yields $Q \in \mathbb{C}^{m \times n}$, the columns of which form an orthonormal basis for the column space of A . If the matrix has rank r with $r < n$, things get a bit more complicated. Let P be a permutation matrix so that the first r columns of AP^T are linearly independent. Specifically, if we partition

$$AP^T \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right),$$

where A_L has r columns, we can find an orthonormal basis for the column space of A_L by computing its QR factorization $A_L = Q_L R_{TL}$, after which

$$AP^T = Q_L \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right),$$

where $R_{TR} = Q_L^H A_R$.

Now, if A is merely almost of rank r (in other words, $A = B + E$ where B has rank r and $\|E\|_2$ is small), then

$$AP^T = \left(\begin{array}{c|c} A_L & A_R \end{array} \right) = \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right) \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & R_{BR} \end{array} \right) \approx Q_L \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right),$$

where $\|R_{BR}\|_2$ is small. This is known as the *rank revealing QR factorization (RRQR)* and it yields a rank r approximation to A given by

$$A \approx Q_L \left(\begin{array}{c|c} R_{TL} & R_{TR} \end{array} \right).$$

The magnitudes of the diagonal elements of R are typically chosen to be in decreasing order and some criteria is used to declare the matrix of rank r , at which point R_{BR} can be set to zero. Computing the RRQR is commonly called the QR factorization with column pivoting (QRP), for reason that will become apparent.

7.1.2 Outline

7.1	Opening Remarks	147
7.1.1	Launch	147
7.1.2	Outline	148
7.1.3	What you will learn	149
7.2	Modifying MGS to Compute QR Factorization with Column Pivoting	150
7.3	Unblocked Householder QR Factorization with Column Pivoting	151
7.3.1	Basic algorithm	151
7.3.2	Alternative unblocked Householder QR factorization with column pivoting	151
7.4	Blocked HQRP	155
7.5	Computing Q	155
7.6	Enrichments	155
7.6.1	QR factorization with randomization for column pivoting	155
7.7	Wrapup	158
7.7.1	Additional exercises	158
7.7.2	Summary	158

7.1.3 What you will learn

7.2 Modifying MGS to Compute QR Factorization with Column Pivoting

We are going to give a modification of the MGS algorithm for computing QRP and then explain why it works. Let us call this the MGS algorithm with column pivoting (MGSP). In the below discussion, p is a vector of integers that will indicate how columns must be swapped in the course of the computation.

- Partition

$$A \rightarrow \left(\begin{array}{c|c} a_1 & A_2 \end{array} \right), Q \rightarrow \left(\begin{array}{c|c} q_1 & Q_2 \end{array} \right), R \rightarrow \left(\begin{array}{c|c} \rho_{11} & r_{12}^T \\ 0 & R_{22} \end{array} \right), p \rightarrow \left(\begin{array}{c} \pi_1 \\ p_2 \end{array} \right)$$

- Determine the index π_1 of the column of $\left(\begin{array}{c|c} a_1 & A_2 \end{array} \right)$ that is longest.
- Permute $\left(\begin{array}{c|c} a_1 & A_2 \end{array} \right) := \left(\begin{array}{c|c} a_1 & A_2 \end{array} \right) P(\pi_1)^T$, swapping column a_1 with the column that is longest.
- Compute $\rho_{11} := \|a_1\|_2$.
- $q_1 := a_1 / \rho_{11}$.
- Compute $r_{12}^T := q_1^T A_2$.
- Update $A_2 := A_2 - q_1 r_{12}^T$.
- Continue the process with the updated matrix A_2 .

The elements on the diagonal of R will be in non-increasing order (and positive) 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 .

Homework 7.1 Let $A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$, $v = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{pmatrix} = \begin{pmatrix} \|a_0\|_2^2 \\ \|a_1\|_2^2 \\ \vdots \\ \|a_{n-1}\|_2^2 \end{pmatrix}$, $q^T q = 1$ (of 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

$$\begin{pmatrix} \|b_0\|_2^2 \\ \|b_1\|_2^2 \\ \vdots \\ \|b_{n-1}\|_2^2 \end{pmatrix} = \begin{pmatrix} v_0 - \rho_0^2 \\ v_1 - \rho_1^2 \\ \vdots \\ v_{n-1} - \rho_{n-1}^2 \end{pmatrix}.$$

 SEE ANSWER

Building on the last exercise, we make an important observation that greatly reduces the cost of determining the column that is longest. Let us start by computing v as the vector such that the i th entry in v equals the square of the length of the i th column of A . In other words, the i th entry of v equals the dot product of the i column of A with itself. In the above outline for the MGS with column pivoting, we can then also partition

$$v \rightarrow \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}.$$

The question becomes how v_2 before the update $A_2 := A_2 - q_1 r_{12}^T$ compares to v_2 after that update. The answer is that the i th entry of v_2 must be updated by subtracting off the square of the i th entry of r_{12}^T .

Let us introduce the functions $v = \text{COMPUTEWEIGHTS}(A)$ and $v = \text{UPDATEWEIGHTS}_v, r$ to compute the described *weight vector* v and to update a weight vector v by subtracting from its elements the squares of the corresponding entries of r . Also, the function `DETERMINEPIVOT` returns the index of the largest in the vector, and swaps that entry with the first entry. A MGS algorithm with column pivoting, MGSP, is then given in Figure 7.1. In that algorithm, A is overwritten with Q .

7.3 Unblocked Householder QR Factorization with Column Pivoting

7.3.1 Basic algorithm

The insights we gained from discussing the MGSP algorithm can be extended to QR factorization algorithms based on Householder transformations. The unblocked QR factorization discussed in Section 6.3 can be supplemented with column pivoting, yielding HQRP_UNB_VAR1 in Figure 7.2.

7.3.2 Alternative unblocked Householder QR factorization with column pivoting

A moment of reflection tells us that there is no straight forward way for adding column pivoting to HQR_AND_FMT_UNB_VAR2. The reason is that the remaining columns of A must be at least partially updated in order to have enough information to determine how to permute columns in the next step. Instead we introduce yet another variant of HQR that at least partially overcomes this, and can also be used in a blocked HQRP algorithm. Connoisseurs of the LU factorization will notice it has a bit of a flavor of the Crout variant for computing that factorization (see Section 12.9.4).

The insights that underlie the blocked QR factorization discussed in Section 6.6 can be summarized by

$$\underbrace{(I - UT^{-T}U^T)}_{Q^T} \widehat{A} = R,$$

where \widehat{A} represents the original contents of A , U is the matrix of Householder vectors, T is computed from U as described in Section 6.6.1, and R is upper triangular.

Now, partially into the computation we find ourselves in the state where

$$\left(I - \left(\frac{U_{TL}}{U_{BL}} \right) T_{TL}^{-H} \left(\frac{U_{TL}}{U_{BL}} \right)^H \right) \left(\begin{array}{c|c} \widehat{A}_{TL} & \widehat{A}_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right) = \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & \widetilde{A}_{BR} \end{array} \right).$$

Algorithm: $[A, R, p] := \text{MGSP}(A, R, p)$

$v := \text{COMPUTEWEIGHTS}(A)$

$$\text{Partition } A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right), R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right), v \rightarrow \left(\begin{array}{c} v_T \\ \hline v_B \end{array} \right)$$

where A_L has 0 columns, R_{TL} is 0×0 , p_T has 0 rows, v_T has 0 rows

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

Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array} \right),$$

$$\left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ \hline p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ \hline v_B \end{array} \right) \rightarrow \left(\begin{array}{c} v_0 \\ \hline v_1 \\ \hline v_2 \end{array} \right)$$

where a_1 has 1 column, ρ_{11} is 1×1 , π_1 has 1 row, v_1 has 1 row

$$[\left(\begin{array}{c} v_1 \\ \hline v_2 \end{array} \right), \pi_1] = \text{DETERMINEPIVOT}\left(\begin{array}{c} v_1 \\ \hline v_2 \end{array} \right)$$

$$\left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right) := \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right) P(\pi_1)^T$$

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

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

$$r_{12}^T := q_1^T A_2$$

$$A_2 := A_2 - q_1 r_{12}^T$$

$$v_2 := \text{UPDATEWEIGHTS}(v_2, r_{12})$$

Continue with

...

endwhile

Figure 7.1: Modified Gram-Schmidt algorithm with column pivoting.

Algorithm: $[A, t, p] = \text{HQRP_UNB_VAR1 } (A)$

$v := \text{ComputeWeights}(A)$

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

where A_{TL} is 0×0 and t_T has 0 elements

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

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} p_T \\ p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \pi_1 \\ p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ v_B \end{array} \right) \rightarrow \left(\begin{array}{c} v_0 \\ v_1 \\ v_2 \end{array} \right)$$

where α_{11} and τ_1 are scalars

$$\left[\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}, \pi_1 \right] = \text{DETERMINEPIVOT}\left(\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}\right)$$

$$\left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) P(\pi_1)^T$$

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

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

$$\left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right) := \left(\begin{array}{c} a_{12}^T - w_{12}^T \\ A_{22} - a_{21} w_{12}^T \end{array} \right)$$

$$v_2 = \text{UPDATEWEIGHT}(v_2, a_{12})$$

Continue with

...

endwhile

Figure 7.2: Simple unblocked rank revealing QR factorization via Householder transformations (HQRP, unblocked Variant 1).

The matrix A at that point contains

$$\left(\begin{array}{c|c} U \setminus R_{TL} & R_{TR} \\ \hline U_{BL} & \tilde{A}_{BR} \end{array} \right)$$

by which we mean that the Householder vectors computed so far are stored below the diagonal of R_{TL} .

Let us manipulate this a bit:

$$\underbrace{\left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right) - \left(\begin{array}{c} U_{TL} \\ U_{BL} \end{array} \right) T_{TL}^{-H} \left(\begin{array}{c} U_{TL} \\ U_{BL} \end{array} \right)^H \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right)}_{\left(\begin{array}{c|c} W_{TL} & W_{TR} \end{array} \right)} = \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & \tilde{A}_{BR} \end{array} \right).$$

$$\left(\begin{array}{c|c} U_{TL}W_{TL} & U_{TL}W_{TR} \\ \hline U_{BL}W_{TL} & U_{BL}W_{TR} \end{array} \right)$$

from which we find that $\tilde{A}_{BR} = \hat{A}_{BR} - U_{BL}W_{TR}$.

Now, what if we do not bother to update A_{BR} so that instead at this point in the computation A contains

$$\left(\begin{array}{c|c} U \setminus R_{TL} & R_{TR} \\ \hline U_{BL} & \hat{A}_{BR} \end{array} \right),$$

but we also have computed and stored W_{TR} . How would we be able to compute another Householder vector, row of R , and row of W ?

Repartition

$$\underbrace{\left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right) - \left(\begin{array}{c} U_{TL} \\ U_{BL} \end{array} \right) \left(\begin{array}{c|c} W_{TL} & W_{TR} \end{array} \right)}_{\left(\begin{array}{c|c|c} \hat{A}_{00} & \hat{a}_{01} & \hat{A}_{02} \\ \hline \hat{a}_{10}^T & \hat{\alpha}_{11} & \hat{a}_{12}^T \\ \hline \hat{A}_{20} & \hat{a}_{21} & \hat{A}_{22} \end{array} \right)} = \underbrace{\left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline 0 & \tilde{A}_{BR} \end{array} \right)}_{\left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline 0 & \tilde{\alpha}_{11} & \tilde{a}_{12}^T \\ \hline 0 & \tilde{a}_{21} & \tilde{A}_{22} \end{array} \right)}.$$

The important observation is that

$$\left(\begin{array}{c|c} \tilde{\alpha}_{11} & \tilde{a}_{12}^T \\ \hline \tilde{a}_{21} & \tilde{A}_{22} \end{array} \right) = \left(\begin{array}{c|c} \hat{\alpha}_{11} - u_{10}^T w_{01} & \hat{a}_{12}^T - u_{10}^T W_{02} \\ \hline \hat{a}_{21} - U_{20} w_{01} & \hat{A}_{22} - U_{20} W_{02} \end{array} \right)$$

What does this mean? To update the next rows of A and R according to the QR factorization via Householder transformations

- The contents of A must be updated by

- $\alpha_{11} := \alpha_{11} - a_{10}^T w_{01} (= \alpha_{11} - u_{10}^T w_{01})$.

- $a_{21} := a_{21} - A_{20}w_{01}$ ($= a_{21} - U_{20}w_{01}$).
- $a_{12}^T := a_{12}^T - a_{10}^T W_{02}$ ($= a_{12}^T - u_{12}^T W_{02}$).

This updates those parts of A consistent with prior computations;

- The Householder vector can then be computed, updating α_{11} and a_{12} ;
- $w_{12}^T := a_{12}^T - a_{21}^T A_{22} + (a_{21}^T A_{21})W_{02})/\tau$ ($= a_{12}^T - u_{21}^T (A_{22} + U_{21}W_{02})/\tau = a_{12}^T - u_{21}^T \tilde{A}_{22}/\tau$). This computes the next row of W (or, more precisely, the part of that row that will be used in future iterations); and
- $a_{12}^T := a_{12}^T - w_{12}^T$. This computes the remainder of the next row of R , overwriting A .

The resulting algorithm, to which column pivoting is added, is presented in Figure 7.3. In that figure HQRP_UNB_VAR1 is also given for comparison. An extra parameter, r , is added to stop the process after r columns of A have been completely computed. This will allow the algorithm to be used to compute only the first r columns of Q (or rather the Householder vectors from which those columns can be computed) and will also allow it to be incorporated into a blocked algorithm, as we will discuss next.

7.4 Blocked HQRP

Finally, a blocked algorithm that incorporates column pivoting and uses HQRP_UNB_VAR3, is given in Figure 7.4. The idea is that the unblocked algorithm is executed leaving the updating of A_{22} to happen periodically, via a matrix-matrix multiplication. While this casts some of the computation in terms of matrix-matrix multiplication, a careful analysis shows that a comparable amount of computation is in lower performing operations like matrix-vector multiplication.

7.5 Computing Q

Given the Householder vectors computed as part of any of the HQRP algorithms, any number of desired columns of matrix Q can be computed using the techniques described in Section 6.4.

7.6 Enrichments

7.6.1 QR factorization with randomization for column pivoting

The blocked algorithm for rank-revealing QR factorization has the fundamental problem that only about half of the computation can be cast in terms of (high performing) matrix-matrix multiplication. The following paper shows how randomization can be used to overcome this problem:

Per-Gunnar Martinsson, Gregorio Quintana-Orti, Nathan Heavner, Robert van de Geijn. [Householder QR Factorization: Adding Randomization for Column Pivoting](#). FLAME Working Note #78, arXiv:1512.02671. Dec. 2015.

Algorithm: $[A, t, p, v, W] = \text{HQRP_UNB_VAR3} (A, t, p, v, W, r)$

$$\text{Partition } A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ t_B \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ p_B \end{array} \right), v \rightarrow \left(\begin{array}{c} v_T \\ v_B \end{array} \right), W \rightarrow \left(\begin{array}{c|c} W_{TL} & W_{TR} \\ \hline W_{BL} & W_{BR} \end{array} \right)$$

where A_{TL} is 0×0 and t_T has 0 elements

while $n(A_{TL}) < r$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} p_T \\ p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \pi_1 \\ p_2 \end{array} \right),$$

$$\left(\begin{array}{c} v_T \\ v_B \end{array} \right) \rightarrow \left(\begin{array}{c} v_0 \\ v_1 \\ v_2 \end{array} \right), \left(\begin{array}{c|c} W_{TL} & W_{TR} \\ \hline W_{BL} & W_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} W_{00} & w_{01} & W_{02} \\ \hline w_{10}^T & \omega_{11} & w_{12}^T \\ \hline W_{20} & w_{21} & W_{22} \end{array} \right)$$

where α_{11} and τ_1 are scalars

Variant 1 (for reference):

$$[\left(\begin{array}{c} v_1 \\ v_2 \end{array} \right), \pi_1] = \text{DETERMINEPIVOT}\left(\left(\begin{array}{c} v_1 \\ v_2 \end{array} \right)\right)$$

$$\left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) P(\pi_1)^T$$

$$\left[\left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right), \tau_1 \right] := \text{Housev}\left(\frac{\alpha_{11}}{a_{21}}\right)$$

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

$$\left(\begin{array}{c} a_{12}^T \\ A_{22} \end{array} \right) := \left(\begin{array}{c} a_{12}^T - w_{12}^T \\ A_{22} - a_{21} w_{12}^T \end{array} \right)$$

$$v_2 = \text{UPDATEWEIGHT}(v_2, a_{12})$$

Variant 3:

$$[\left(\begin{array}{c} v_1 \\ v_2 \end{array} \right), \pi_1] = \text{DETERMINEPIVOT}\left(\left(\begin{array}{c} v_1 \\ v_2 \end{array} \right)\right)$$

$$\left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) P(\pi_1)^T$$

$$\alpha_{11} := \alpha_{11} - a_{10}^T w_{01}$$

$$a_{21} := a_{21} - A_{20} w_{01}$$

$$a_{12}^T := a_{12}^T - a_{10}^T W_{02}$$

$$\left[\left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right), \tau_1 \right] := \text{Housev}\left(\frac{\alpha_{11}}{a_{21}}\right)$$

$$w_{12}^T := (a_{12}^T + a_{21}^H A_{22} - (a_{21}^H A_{20}) W_{02}) / \tau_1$$

$$a_{12}^T := a_{12}^T - w_{12}^T$$

$$v_2 = \text{UPDATEWEIGHT}(v_2, a_{12})$$

Continue with

...

endwhile

Figure 7.3: HQRP unblocked Variant 3 that updates r rows and columns of A , but leaves the trailing matrix A_{BR} prestine. Here v has already been initialized before calling the routine so that it can be called from a blocked algorithm. The HQRP unblocked Variant 1 from Figure 7.2 is also given for reference.

Algorithm: $[A, t, p] := \text{HQRP_BLK}(A, t, p, r)$

$v := \text{COMPUTEWEIGHTS}(A)$

$$\text{Partition } A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ t_B \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ p_B \end{array} \right), v \rightarrow \left(\begin{array}{c} v_T \\ v_B \end{array} \right), W \rightarrow \left(\begin{array}{c|c} W_L & W_R \end{array} \right)$$

where A_{TL} is 0×0 , t_T has 0 rows, p_T has 0 rows, v_T has 0 rows, W_L has 0 columns

while $n(A_{TL}) < r$ **do**

Determine block size b

Repartition

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

$$\left(\begin{array}{c} p_T \\ p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ p_1 \\ p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ v_B \end{array} \right) \rightarrow \left(\begin{array}{c} v_0 \\ v_1 \\ v_2 \end{array} \right), \left(\begin{array}{c|c} W_L & W_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} W_0 & W_1 & W_2 \end{array} \right)$$

where A_{11} is $b \times b$, t_1 has b rows, p_1 has b rows, v_1 has b rows, W_1 has b columns

$$[\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right), t_1, p_1, \left(\begin{array}{c} v_1 \\ v_2 \end{array} \right), \left(\begin{array}{c|c} W_1 & W_2 \end{array} \right)] :=$$

$$\text{HQRP_UNB_VAR3} \left(\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right), t_1, p_1, \left(\begin{array}{c} v_1 \\ v_2 \end{array} \right), \left(\begin{array}{c|c} W_1 & W_2 \end{array} \right), b \right)$$

$$A_{22} := A_{22} - U_2 W_2$$

Continue with

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

$$\left(\begin{array}{c} p_T \\ p_B \end{array} \right) \leftarrow \left(\begin{array}{c} p_0 \\ p_1 \\ p_2 \end{array} \right), \left(\begin{array}{c} v_T \\ v_B \end{array} \right) \leftarrow \left(\begin{array}{c} v_0 \\ v_1 \\ v_2 \end{array} \right), \left(\begin{array}{c|c} W_L & W_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} W_0 & W_1 & W_2 \end{array} \right)$$

endwhile

Figure 7.4: Blocked HQRP algorithm. Note: W starts as a $b \times n(A)$ matrix. If this is not a uniform block size used during computation, resizing may be necessary.

7.7 Wrapup

7.7.1 Additional exercises

7.7.2 Summary

Chapter **8**

Notes on Solving Linear Least-Squares Problems

For a motivation of the linear least-squares problem, read Week 10 (Sections 10.3-10.5) of
[Linear Algebra: Foundations to Frontiers - Notes to LAFF With \[30\]](#).

Video

Read disclaimer regarding the videos in the preface!

No video... Camera ran out of memory...

8.0.1 Launch

8.0.2 Outline

8.0.1	Launch	159
8.0.2	Outline	160
8.0.3	What you will learn	161
8.1	The Linear Least-Squares Problem	162
8.2	Method of Normal Equations	162
8.3	Solving the LLS Problem Via the QR Factorization	163
8.3.1	Simple derivation of the solution	163
8.3.2	Alternative derivation of the solution	164
8.4	Via Householder QR Factorization	165
8.5	Via the Singular Value Decomposition	166
8.5.1	Simple derivation of the solution	166
8.5.2	Alternative derivation of the solution	167
8.6	What If A Does Not Have Linearly Independent Columns?	167
8.7	Exercise: Using the the <i>LQ</i> factorization to solve underdetermined systems	174
8.8	Wrapup	174
8.8.1	Additional exercises	174
8.8.2	Summary	174

8.0.3 What you will learn

8.1 The Linear Least-Squares Problem

Let $A \in \mathbb{C}^{m \times n}$ and $y \in \mathbb{C}^m$. Then the linear least-square problem (LLS) is given by

$$\text{Find } x \text{ s.t. } \|Ax - y\|_2 = \min_{z \in \mathbb{C}^n} \|Az - y\|_2.$$

In other words, x is the vector that minimizes the expression $\|Ax - y\|_2$. Equivalently, we can solve

$$\text{Find } x \text{ s.t. } \|Ax - y\|_2^2 = \min_{z \in \mathbb{C}^n} \|Az - y\|_2^2.$$

If x solves the linear least-squares problem, then Ax is the vector in $\mathcal{C}(A)$ (the column space of A) closest to the vector y .

8.2 Method of Normal Equations

Let $A \in \mathbb{R}^{m \times n}$ have linearly independent columns (which implies $m \geq n$). Let $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be defined by

$$\begin{aligned} f(x) &= \|Ax - y\|_2^2 = (Ax - y)^T(Ax - y) = x^T A^T A x - x^T A^T y - y^T A x + y^T y \\ &= x^T A^T A x - 2x^T A^T y + y^T y. \end{aligned}$$

This function is minimized when the gradient is zero, $\nabla f(x) = 0$. Now,

$$\nabla f(x) = 2A^T Ax - 2A^T y.$$

If A has linearly independent columns then $A^T A$ is nonsingular. Hence, the x that minimizes $\|Ax - y\|_2$ solves $A^T Ax = A^T y$. This is known as the method of normal equations. Notice that then

$$x = \underbrace{(A^T A)^{-1} A^T}_{A^\dagger} y,$$

where A^\dagger is known as the *pseudo inverse* or *Moore-Penrose pseudo inverse*.

In practice, one performs the following steps:

- Form $B = A^T A$, a symmetric positive-definite (SPD) matrix.
Cost: approximately mn^2 floating point operations (flops), if one takes advantage of symmetry.
- Compute the Cholesky factor L , a lower triangular matrix, so that $B = LL^T$.
This factorization, discussed in Week 8 (Section 8.4.2) of [Linear Algebra: Foundations to Frontiers - Notes to LAFF With](#) and to be revisited later in this course, exists since B is SPD.
Cost: approximately $\frac{1}{3}n^3$ flops.
- Compute $\hat{y} = A^T y$.
Cost: $2mn$ flops.
- Solve $Lz = \hat{y}$ and $L^T x = z$.
Cost: n^2 flops each.

Thus, the total cost of solving the LLS problem via normal equations is approximately $mn^2 + \frac{1}{3}n^3$ flops.

Remark 8.1 We will later discuss that if A is not well-conditioned (its columns are nearly linearly dependent), the Method of Normal Equations is numerically unstable because $A^T A$ is ill-conditioned.

The above discussion can be generalized to the case where $A \in \mathbb{C}^{m \times n}$. In that case, x must solve $A^H A x = A^H y$.

A geometric explanation of the method of normal equations (for the case where A is real valued) can be found in Week 10 (Sections 10.3-10.5) of [Linear Algebra: Foundations to Frontiers - Notes to LAFF With](#).

8.3 Solving the LLS Problem Via the QR Factorization

Assume $A \in \mathbb{C}^{m \times n}$ has linearly independent columns and let $A = Q_L R_{TL}$ be its QR factorization. We wish to compute the solution to the LLS problem: Find $x \in \mathbb{C}^n$ such that

$$\|Ax - y\|_2^2 = \min_{z \in \mathbb{C}^n} \|Az - y\|_2^2.$$

8.3.1 Simple derivation of the solution

Notice that we know that, if A has linearly independent columns, the solution is given by $x = (A^H A)^{-1} A^H y$ (the solution to the normal equations). Now,

$$\begin{aligned} x &= [A^H A]^{-1} A^H y && \text{Solution to the Normal Equations} \\ &= [(Q_L R_{TL})^H (Q_L R_{TL})]^{-1} (Q_L R_{TL})^H y && A = Q_L R_{TL} \\ &= [R_{TL}^H Q_L^H Q_L R_{TL}]^{-1} R_{TL}^H Q_L^H y && (BC)^H = (C^H B^H) \\ &= [R_{TL}^H R_{TL}]^{-1} R_{TL}^H Q_L^H y && Q_L^H Q_L = I \\ &= R_{TL}^{-1} R_{TL}^{-H} R_{TL}^H Q_L^H y && (BC)^{-1} = C^{-1} B^{-1} \\ &= R_{TL}^{-1} Q_L^H y && R_{TL}^{-H} R_{TL}^H = I \end{aligned}$$

Thus, the x that solves $R_{TL}x = Q_L^H y$ solves the LLS problem.

8.3.2 Alternative derivation of the solution

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. Now,

$$\begin{aligned}
& \min_{z \in \mathbb{C}^n} \|Az - y\|_2^2 \\
&= \min_{z \in \mathbb{C}^n} \|Q_L R_{TL} z - y\|_2^2 && \text{(substitute } A = Q_L R_{TL}) \\
&= \min_{z \in \mathbb{C}^n} \|Q^H(Q_L R_{TL} z - y)\|_2^2 && \text{(two-norm is preserved since } Q^H \text{ is unitary)} \\
&= \min_{z \in \mathbb{C}^n} \left\| \left(\begin{array}{c} Q_L^H \\ Q_R^H \end{array} \right) Q_L R_{TL} z - \left(\begin{array}{c} Q_L^H \\ Q_R^H \end{array} \right) y \right\|_2^2 && \text{(partitioning, distributing)} \\
&= \min_{z \in \mathbb{C}^n} \left\| \left(\begin{array}{c} R_{TL} z \\ 0 \end{array} \right) - \left(\begin{array}{c} Q_L^H y \\ Q_R^H y \end{array} \right) \right\|_2^2 && \text{(partitioned matrix-matrix multiplication)} \\
&= \min_{z \in \mathbb{C}^n} \left\| \left(\begin{array}{c} R_{TL} z - Q_L^H y \\ -Q_R^H y \end{array} \right) \right\|_2^2 && \text{(partitioned matrix addition)} \\
&= \min_{z \in \mathbb{C}^n} (\|R_{TL} z - Q_L^H y\|_2^2 + \|Q_R^H y\|_2^2) && \text{(property of the 2-norm:} \\
&&& \left\| \left(\begin{array}{c} x \\ y \end{array} \right) \right\|_2^2 = \|x\|_2^2 + \|y\|_2^2) \\
&= \left(\min_{z \in \mathbb{C}^n} \|R_{TL} z - Q_L^H y\|_2^2 \right) + \|Q_R^H y\|_2^2 && \text{(\textbf{$Q_R^H y$ is independent of z})} \\
&= \|Q_R^H y\|_2^2 && \text{(minimized by x that satisfies $R_{TL}x = Q_L^H y$)}
\end{aligned}$$

Thus, the desired x that minimizes the linear least-squares problem solves $R_{TL}x = Q_L^H y$. The solution is unique because R_{TL} is nonsingular (because A has linearly independent columns).

In practice, one performs the following steps:

- Compute the QR factorization $A = Q_L R_{TL}$.
If Gram-Schmidt or Modified Gram-Schmidt are used, this costs $2mn^2$ flops.
- Form $\hat{y} = Q_L^H y$.
Cost: $2mn$ flops.
- Solve $R_{TL}x = \hat{y}$ (triangular solve).
Cost: n^2 flops.

Thus, the total cost of solving the LLS problem via (Modified) Gram-Schmidt QR factorization is approximately $2mn^2$ flops.

Notice that the solution computed by the Method of Normal Equations (generalized to the complex case) is given by

$$\begin{aligned}
(A^H A)^{-1} A^H y &= ((Q_L R_{TL})^H (Q_L R_{TL}))^{-1} (Q_L R_{TL})^H y = (R_{TL}^H Q_L^H Q_L R_{TL})^{-1} R_{TL}^H Q_L^H y \\
&= (R_{TL}^H R_{TL})^{-1} R_{TL}^H Q_L^H y = R_{TL}^{-1} R_{TL}^{-H} R_{TL}^H Q_L^H y = R_{TL}^{-1} Q_L^H y = R_{TL}^{-1} \hat{y} = x
\end{aligned}$$

where $R_{TL}x = \hat{y}$. This shows that the two approaches compute the same solution, generalizes the Method of Normal Equations to complex valued problems, and shows that the Method of Normal Equations computes the desired result without requiring multivariate calculus.

8.4 Via Householder QR Factorization

Given $A \in \mathbb{C}^{m \times n}$ with linearly independent columns, the Householder QR factorization yields n Householder transformations, H_0, \dots, H_{n-1} , so that

$$\underbrace{H_{n-1} \cdots H_0}_Q A = \begin{pmatrix} R_{TL} \\ 0 \end{pmatrix}.$$

$$Q = \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)^H$$

We wish to solve $R_{TL}x = \underbrace{Q_L^H y}_{\hat{y}}$. But

$$\begin{aligned} \hat{y} = Q_L^H y &= \left[\left(\begin{array}{c|c} I & 0 \end{array} \right) \left(\begin{array}{c} Q_L^H \\ Q_R^H \end{array} \right) \right] y = \left(\begin{array}{c|c} I & 0 \end{array} \right) \left(\begin{array}{c|c} Q_L & Q_R \end{array} \right)^H y = \left(\begin{array}{c|c} I & 0 \end{array} \right) Q^H y \\ &= \left(\begin{array}{c|c} I & 0 \end{array} \right) (H_{n-1} \cdots H_0)y = \left(\begin{array}{c|c} I & 0 \end{array} \right) \left(\underbrace{H_{n-1} \cdots H_0 y}_{w_T} \right) = w_T. \end{aligned}$$

$$w = \begin{pmatrix} w_T \\ w_B \end{pmatrix}$$

This suggests the following approach:

- Compute H_0, \dots, H_{n-1} so that $H_{n-1} \cdots H_0 A = \begin{pmatrix} R_{TL} \\ 0 \end{pmatrix}$, storing the Householder vectors that define H_0, \dots, H_{n-1} over the elements in A that they zero out (see “Notes on Householder QR Factorization”).
Cost: $2mn^2 - \frac{2}{3}n^3$ flops.
- Form $w = (H_{n-1}(\cdots(H_0y)\cdots))$ (see “Notes on Householder QR Factorization”). Partition $w = \begin{pmatrix} w_T \\ w_B \end{pmatrix}$ where $w_T \in \mathbb{C}^n$. Then $\hat{y} = w_T$.
Cost: $4m^2 - 2n^2$ flops. (See “Notes on Householder QR Factorization” regarding this.)
- Solve $R_{TL}x = \hat{y}$.
Cost: n^2 flops.

Thus, the total cost of solving the LLS problem via Householder QR factorization is approximately $2mn^2 - \frac{2}{3}n^3$ flops. This is cheaper than using (Modified) Gram-Schmidt QR factorization, and hence preferred (because it is also numerically more stable, as we will discuss later in the course).

8.5 Via the Singular Value Decomposition

Given $A \in \mathbb{C}^{m \times n}$ with linearly independent columns, let $A = U\Sigma V^H$ be its SVD decomposition. Partition

$$U = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \quad \text{and} \quad \Sigma = \begin{pmatrix} \Sigma_{TL} \\ 0 \end{pmatrix},$$

where $U_L \in \mathbb{C}^{m \times n}$ and $\Sigma_{TL} \in \mathbb{R}^{n \times n}$ so that

$$A = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \begin{pmatrix} \Sigma_{TL} \\ 0 \end{pmatrix} V^H = U_L \Sigma_{TL} V^H.$$

We wish to compute the solution to the LLS problem: Find $x \in \mathbb{C}^n$ such that

$$\|Ax - y\|_2^2 = \min_{z \in \mathbb{C}^n} \|Az - y\|_2^2.$$

8.5.1 Simple derivation of the solution

Notice that we know that, if A has linearly independent columns, the solution is given by $x = (A^H A)^{-1} A^H y$ (the solution to the normal equations). Now,

$$\begin{aligned} x &= [A^H A]^{-1} A^H y && \text{Solution to the Normal Equations} \\ &= [(U_L \Sigma_{TL} V^H)^H (U_L \Sigma_{TL} V^H)]^{-1} (U_L \Sigma_{TL} V^H)^H y && A = U_L \Sigma_{TL} V^H \\ &= [(V \Sigma_{TL} U_L^H) (U_L \Sigma_{TL} V^H)]^{-1} (V \Sigma_{TL} U_L^H) y && (BCD)^H = (D^H C^H B^H) \text{ and } \Sigma_{TL}^H = \Sigma_{TL} \\ &= [V \Sigma_{TL} \Sigma_{TL} V^H]^{-1} V \Sigma_{TL} U_L^H y && U_L^H U_L = I \\ &= V \Sigma_{TL}^{-1} \Sigma_{TL}^{-1} V^H V \Sigma_{TL} U_L^H y && V^{-1} = V^H \text{ and } (BCD)^{-1} = D^{-1} C^{-1} B^{-1} \\ &= V \Sigma_{TL}^{-1} U_L^H y && V^H V = I \text{ and } \Sigma_{TL}^{-1} \Sigma_{TL} = I \end{aligned}$$

8.5.2 Alternative derivation of the solution

We now discuss a derivation of the result that does not depend on the Normal Equations, in preparation for the more general case discussed in the next section.

$$\begin{aligned}
& \min_{z \in \mathbb{C}^n} \|Az - y\|_2^2 \\
&= \min_{z \in \mathbb{C}^n} \|U\Sigma V^H z - y\|_2^2 && (\text{substitute } A = U\Sigma V^H) \\
&= \min_{z \in \mathbb{C}^n} \|U(\Sigma V^H z - U^H y)\|_2^2 && (\text{substitute } UU^H = I \text{ and factor out } U) \\
&= \min_{z \in \mathbb{C}^n} \|\Sigma V^H z - U^H y\|_2^2 && (\text{multiplication by a unitary matrix}) \\
&&& \text{preserves two-norm)} \\
&= \min_{z \in \mathbb{C}^n} \left\| \begin{pmatrix} \Sigma_{TL} \\ 0 \end{pmatrix} V^H z - \begin{pmatrix} U_L^H y \\ U_R^H y \end{pmatrix} \right\|_2^2 && (\text{partition, partitioned matrix-matrix multiplication}) \\
&= \min_{z \in \mathbb{C}^n} \left\| \begin{pmatrix} \Sigma_{TL} V^H z - U_L^H y \\ -U_R^H y \end{pmatrix} \right\|_2^2 && (\text{partitioned matrix-matrix multiplication and addition}) \\
&= \min_{z \in \mathbb{C}^n} \left\| \Sigma_{TL} V^H z - U_L^H y \right\|_2^2 + \left\| U_R^H y \right\|_2^2 && \left(\left\| \begin{pmatrix} v_T \\ v_B \end{pmatrix} \right\|_2^2 = \|v_T\|_2^2 + \|v_B\|_2^2 \right)
\end{aligned}$$

The x that solves $\Sigma_{TL} V^H x = U_L^H y$ minimizes the expression. That x is given by $x = V\Sigma_{TL}^{-1} U_L^H y$.

This suggests the following approach:

- Compute the reduced SVD: $A = U_L \Sigma_{TL} V^H$.
Cost: Greater than computing the QR factorization! We will discuss this in a future note.
- Form $\hat{y} = \Sigma_{TL}^{-1} U_L^H y$.
Cost: approx. $2mn$ flops.
- Compute $z = V\hat{y}$.
Cost: approx. $2mn$ flops.

8.6 What If A Does Not Have Linearly Independent Columns?

In the above discussions we assume that A has linearly independent columns. Things get a bit trickier if A does not have linearly independent columns. There is a variant of the QR factorization known as the QR factorization with column pivoting that can be used to find the solution. We instead focus on using the SVD.

Given $A \in \mathbb{C}^{m \times n}$ with $\text{rank}(A) = r < n$, let $A = U\Sigma V^H$ be its SVD decomposition. Partition

$$U = \left(\begin{array}{c|c} U_L & U_R \end{array} \right), \quad V = \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \quad \text{and} \quad \Sigma = \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right),$$

where $U_L \in \mathbb{C}^{m \times r}$, $V_L \in \mathbb{C}^{n \times r}$ and $\Sigma_{TL} \in \mathbb{R}^{r \times r}$ so that

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

Now,

$$\begin{aligned}
 & \min_{z \in \mathbb{C}^n} \|Az - y\|_2^2 \\
 &= \min_{z \in \mathbb{C}^n} \|U\Sigma V^H z - y\|_2^2 && (\text{substitute } A = U\Sigma V^H) \\
 &= \min_{z \in \mathbb{C}^n} \|U\Sigma V^H z - UU^H y\|_2^2 && (UU^H = I) \\
 &= \min_{\substack{w \in \mathbb{C}^n \\ z = Vw}} \|U\Sigma V^H Vw - UU^H y\|_2^2 && (\text{choosing the max over } w \in \mathbb{C}^n \text{ with } z = Vw \text{ is the same as choosing the max over } z \in \mathbb{C}^n.) \\
 &= \min_{\substack{w \in \mathbb{C}^n \\ z = Vw}} \|U(\Sigma w - U^H y)\|_2^2 && (\text{factor out } U \text{ and } V^H V = I) \\
 &= \min_{\substack{w \in \mathbb{C}^n \\ z = Vw}} \|\Sigma w - U^H y\|_2^2 && (\|Uv\|_2 = \|v\|_2) \\
 &= \min_{\substack{w \in \mathbb{C}^n \\ z = Vw}} \left\| \begin{pmatrix} \Sigma_{TL} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} w_T \\ w_B \end{pmatrix} - \begin{pmatrix} U_L^H \\ U_R^H \end{pmatrix} y \right\|_2^2 && (\text{partition } \Sigma, w, \text{ and } U) \\
 &= \min_{\substack{w \in \mathbb{C}^n \\ z = Vw}} \left\| \begin{pmatrix} \Sigma_{TL} w_T - U_L^H y \\ -U_R^H y \end{pmatrix} \right\|_2^2 && (\text{partitioned matrix-matrix multiplication}) \\
 &= \min_{\substack{w \in \mathbb{C}^n \\ z = Vw}} \|\Sigma_{TL} w_T - U_L^H y\|_2^2 + \|U_R^H y\|_2^2 && \left(\left\| \begin{pmatrix} v_T \\ v_B \end{pmatrix} \right\|_2^2 = \|v_T\|_2^2 + \|v_B\|_2^2 \right)
 \end{aligned}$$

Since Σ_{TL} is a diagonal with no zeroes on its diagonal, we know that Σ_{TL}^{-1} exists. Choosing $w_T = \Sigma_{TL}^{-1} U_L^H y$ means that

$$\min_{\substack{w \in \mathbb{C}^n \\ z = Vw}} \|\Sigma_{TL} w_T - U_L^H y\|_2^2 = 0,$$

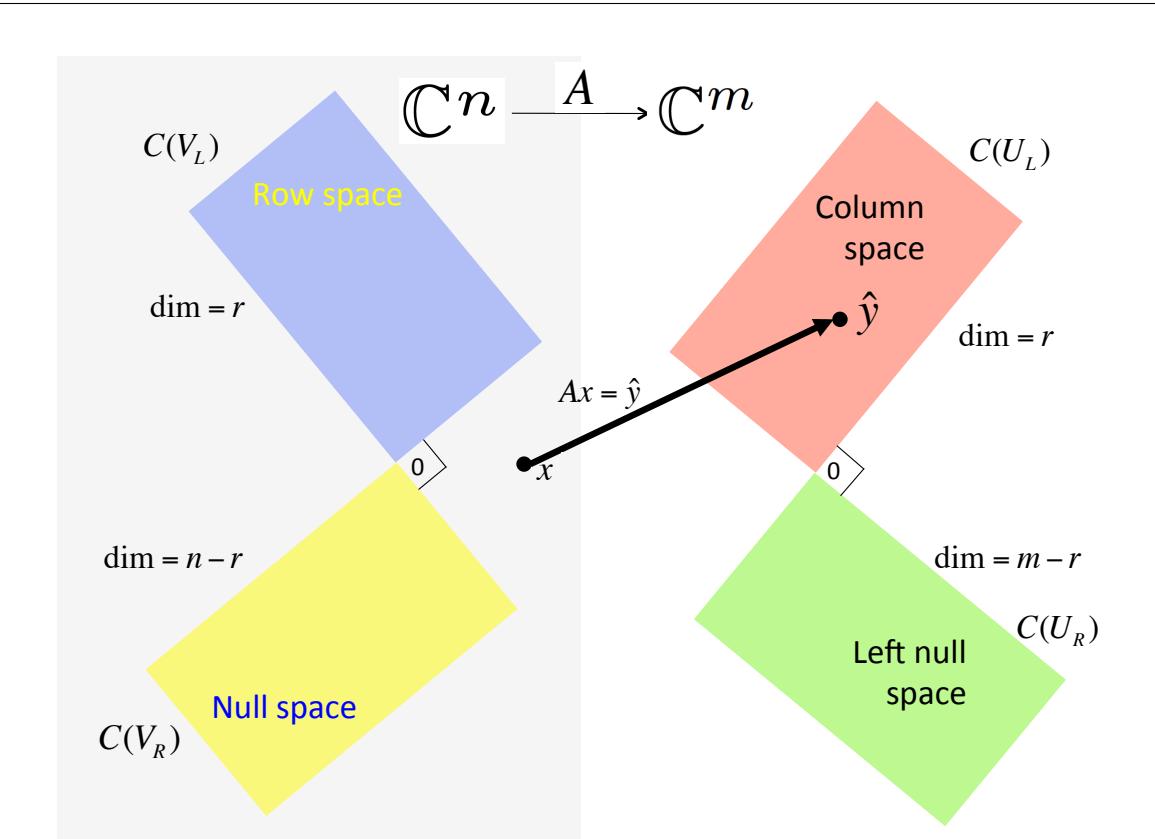
which obviously minimizes the entire expression. We conclude that

$$x = Vw = \left(\begin{array}{c|c} V_L & V_R \end{array} \right) \left(\begin{array}{c} \Sigma_{TL}^{-1} U_L^H y \\ w_B \end{array} \right) = V_L \Sigma_{TL}^{-1} U_L^H y + V_R w_B$$

characterizes all solutions to the linear least-squares problem, where w_B can be chosen to be any vector of size $n - r$. By choosing $w_B = 0$ and hence $x = V_L \Sigma_{TL}^{-1} U_L^H y$ we choose the vector x that itself has minimal 2-norm.

The sequence of pictures on the following pages reasons through the insights that we gained so far (in “Notes on the Singular Value Decomposition” and this note). These pictures can be downloaded as a PowerPoint presentation from

<http://www.cs.utexas.edu/users/flame/Notes/Spaces.pptx>



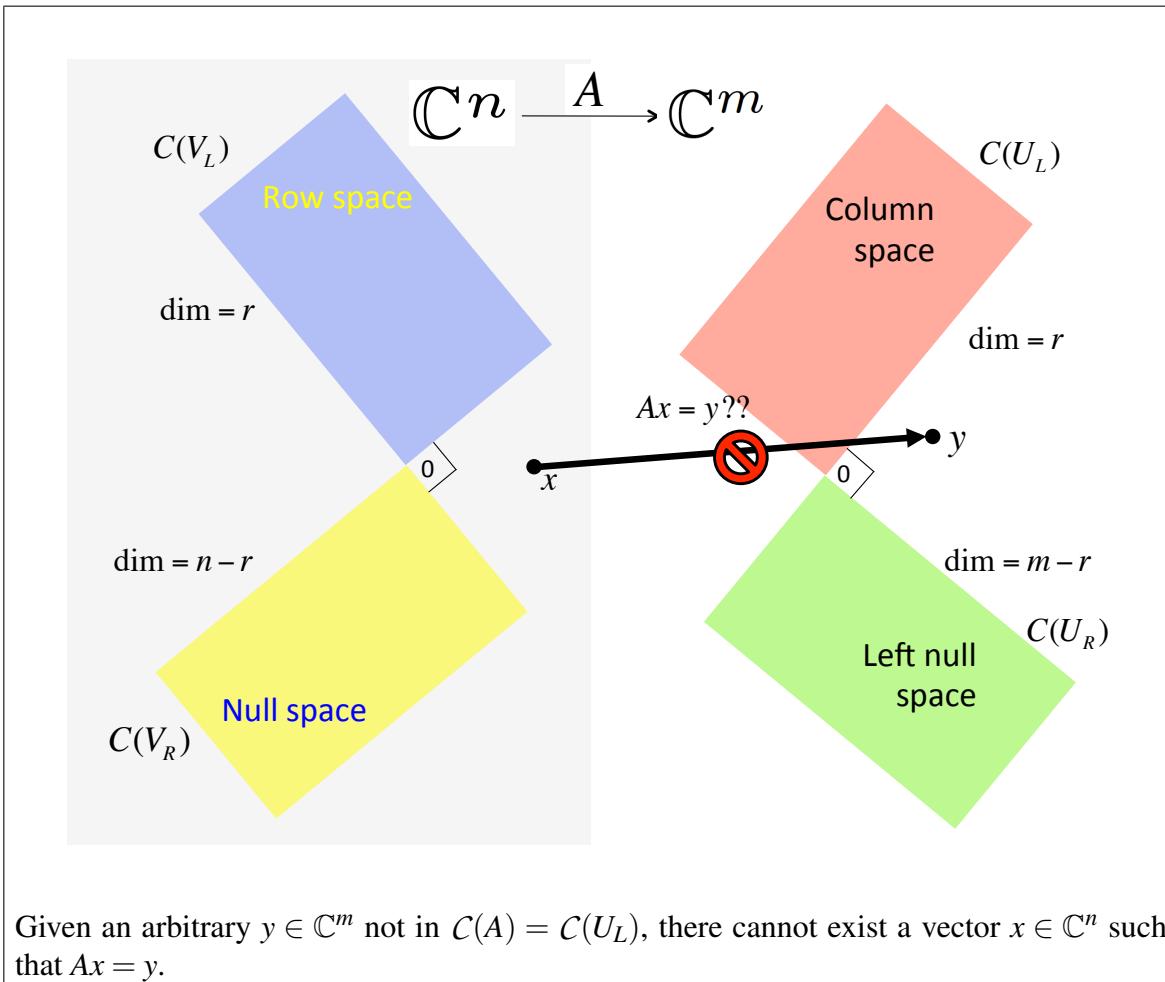
If $A \in \mathbb{C}^{m \times n}$ and

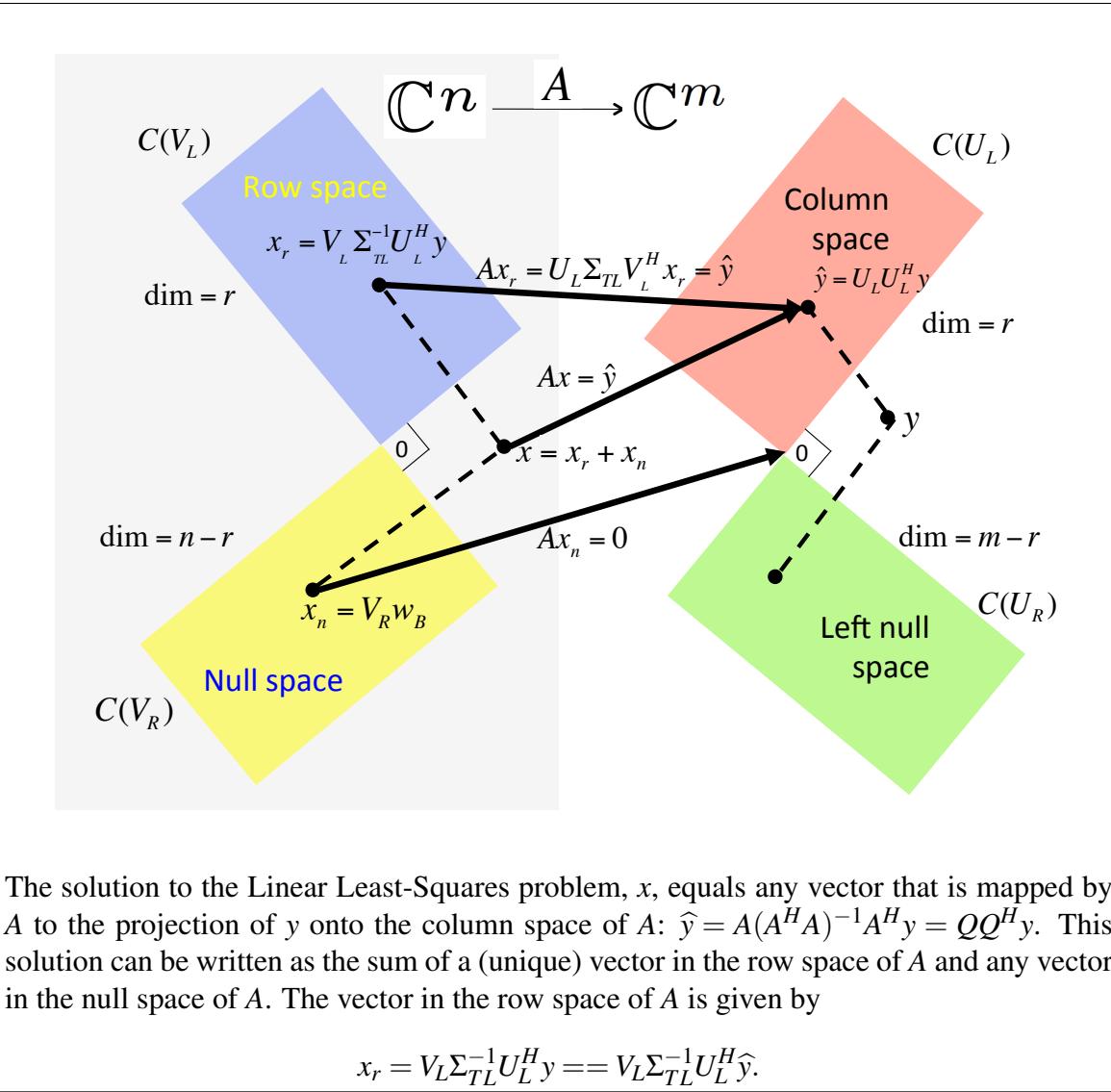
$$A = \left(\begin{array}{c|c} U_L & U_R \end{array} \right) \left(\begin{array}{c|c} \Sigma_{TL} & 0 \\ \hline 0 & 0 \end{array} \right) \left(\begin{array}{c|c} V_L & V_R \end{array} \right)^H = U_L \Sigma_{TL} V_L^H$$

equals the SVD, where $U_L \in \mathbb{C}^{m \times r}$, $V_L \in \mathbb{C}^{n \times r}$, and $\Sigma_{TL} \in \mathbb{C}^{r \times r}$, then

- The row space of A equals $C(V_L)$, the column space of V_L ;
- The null space of A equals $C(V_R)$, the column space of V_R ;
- The column space of A equals $C(U_L)$; and
- The left null space of A equals $C(U_R)$.

Also, given a vector $x \in \mathbb{C}^n$, the matrix A maps x to $\hat{y} = Ax$, which must be in $C(A) = C(U_L)$.





The solution to the Linear Least-Squares problem, x , equals any vector that is mapped by A to the projection of y onto the column space of A : $\hat{y} = A(A^H A)^{-1} A^H y = Q Q^H y$. This solution can be written as the sum of a (unique) vector in the row space of A and any vector in the null space of A . The vector in the row space of A is given by

$$x_r = V_L \Sigma_{TL}^{-1} U_L^H y = V_L \Sigma_{TL}^{-1} U_L^H \hat{y}.$$

The sequence of pictures, and their explanations, suggest a much simply path towards the formula for solving the LLS problem.

- We know that we are looking for the solution x to the equation

$$Ax = U_L U_L^H y.$$

- We know that there must be a solution x_r in the row space of A . It suffices to find w_T such that $x_r = V_L w_T$.
- Hence we search for w_T that satisfies

$$A V_L w_T = U_L U_L^H y.$$

- Since there is a one-to-one mapping by A from the row space of A to the column space of A , we know that w_T is unique. Thus, if we find a solution to the above, we have found *the* solution.
- Multiplying both sides of the equation by U_L^H yields

$$U_L^H A V_L w_T = U_L^H y.$$

- Since $A = U_L \Sigma_{TL}^{-1} V_L^H$, we can rewrite the above equation as

$$\Sigma_{TL} w_T = U_L^H y$$

so that $w_T = \Sigma_{TL}^{-1} U_L^H y$.

- Hence

$$x_r = V_L \Sigma_{TL}^{-1} U_L^H y.$$

- Adding any vector in the null space of A to x_r also yields a solution. Hence all solutions to the LLS problem can be characterized by

$$x = V_L \Sigma_{TL}^{-1} U_L^H y + V_R w_R.$$

Here is yet another important way of looking at the problem:

- We start by considering the LLS problem: Find $x \in \mathbb{C}^n$ such that

$$\|Ax - y\|_2^2 = \max_{z \in \mathbb{C}^n} \|Az - y\|_2^2.$$

- We changed this into the problem of finding w_L that satisfied

$$\Sigma_{TL} w_L = v_T$$

where $x = V_L w_L$ and $\hat{y} = U_L U_L^H y = U_L v_T$.

- Thus, by expressing x in the right basis (the columns of V_L) and the projection of y in the right basis (the columns of U_L), the problem became trivial, since the matrix that related the solution to the right-hand side became diagonal.

8.7 Exercise: Using the the LQ factorization to solve underdetermined systems

We next discuss another special case of the LLS problem: Let $A \in \mathbb{C}^{m \times n}$ where $m < n$ and A has linearly independent rows. A series of exercises will lead you to a practical algorithm for solving the problem of describing all solutions to the LLS problem

$$\|Ax - y\|_2 = \min_z \|Az - y\|_2.$$

You may want to review “Notes on the QR Factorization” as you do this exercise.

Homework 8.2 Let $A \in \mathbb{C}^{m \times n}$ with $m < n$ have linearly independent rows. Show that there exist a lower triangular matrix $L_L \in \mathbb{C}^{m \times m}$ and a matrix $Q_T \in \mathbb{C}^{m \times n}$ with orthonormal rows such that $A = L_L Q_T$, noting that L_L does not have any zeroes on the diagonal. Letting $L = \left(\begin{array}{c|c} L_L & 0 \end{array} \right)$ be $\mathbb{C}^{m \times n}$ and unitary $Q = \left(\begin{array}{c} Q_T \\ Q_B \end{array} \right)$, reason that $A = LQ$.

Don't overthink the problem: use results you have seen before.

[SEE ANSWER](#)

Homework 8.3 Let $A \in \mathbb{C}^{m \times n}$ with $m < n$ have linearly independent rows. Consider

$$\|Ax - y\|_2 = \min_z \|Az - y\|_2.$$

Use the fact that $A = L_L Q_T$, where $L_L \in \mathbb{C}^{m \times m}$ is lower triangular and Q_T has orthonormal rows, to argue that any vector of the form $Q_T^H L_L^{-1} y + Q_B^H w_B$ (where w_B is any vector in \mathbb{C}^{n-m}) is a solution to the LLS problem. Here $Q = \left(\begin{array}{c} Q_T \\ Q_B \end{array} \right)$.

[SEE ANSWER](#)

Homework 8.4 Continuing Exercise 8.2, use Figure 8.1 to give a Classical Gram-Schmidt inspired algorithm for computing L_L and Q_T . (The best way to check you got the algorithm right is to implement it!)

[SEE ANSWER](#)

Homework 8.5 Continuing Exercise 8.2, use Figure 8.2 to give a Householder QR factorization inspired algorithm for computing L and Q , leaving L in the lower triangular part of A and Q stored as Householder vectors above the diagonal of A . (The best way to check you got the algorithm right is to implement it!)

[SEE ANSWER](#)

8.8 Wrapup

8.8.1 Additional exercises

8.8.2 Summary

Algorithm: $[L, Q] := \text{LQ_CGS_UNB}(A, L, Q)$

Partition $A \rightarrow \begin{pmatrix} A_T \\ A_B \end{pmatrix}$, $L \rightarrow \begin{pmatrix} L_{TL} & L_{TR} \\ L_{BL} & L_{BR} \end{pmatrix}$, $Q \rightarrow \begin{pmatrix} Q_T \\ Q_B \end{pmatrix}$

where A_T has 0 rows, L_{TL} is 0×0 , Q_T has 0 rows

while $m(A_T) < m(A)$ **do**

Repartition

$$\begin{pmatrix} A_T \\ A_B \end{pmatrix} \rightarrow \begin{pmatrix} A_0 \\ a_1^T \\ A_2 \end{pmatrix}, \begin{pmatrix} L_{TL} & L_{TR} \\ L_{BL} & L_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{pmatrix}, \begin{pmatrix} Q_T \\ Q_B \end{pmatrix} \rightarrow \begin{pmatrix} Q_0 \\ q_1^T \\ Q_2 \end{pmatrix}$$

where a_1 has 1 row, λ_{11} is 1×1 , q_1 has 1 row

Continue with

$$\begin{pmatrix} A_T \\ A_B \end{pmatrix} \leftarrow \begin{pmatrix} A_0 \\ a_1^T \\ A_2 \end{pmatrix}, \begin{pmatrix} L_{TL} & L_{TR} \\ L_{BL} & L_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{pmatrix}, \begin{pmatrix} Q_T \\ Q_B \end{pmatrix} \leftarrow \begin{pmatrix} Q_0 \\ q_1^T \\ Q_2 \end{pmatrix}$$

endwhile

Figure 8.1: Algorithm skeleton for CGS-like LQ factorization.

Algorithm: $[A, t] := \text{HLQ_UNB}(A, t)$

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

where A_{TL} is 0×0 , t_T has 0 rows

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

Repartition

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

where α_{11} is 1×1 , τ_1 has 1 row

Continue with

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

endwhile

Figure 8.2: Algorithm skeleton for Householder QR factorization inspired LQ factorization.

Notes on the Condition of a Problem

9.1 Opening Remarks

9.1.1 Launch

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, as we will see later, 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.

Naively, we would like to be able to define a program that computes with floating point numbers as being “correct” if it computes an answer that is close to the exact answer. Unfortunately, some problems that are computed this way have the property that a small change in the input yields a large change in the output. Surely we can’t blame the program for not computing an answer close to the exact answer in this case. The mere act of storing the input data as a floating point number may cause a completely different output, even if all computation is exact. We will later define *stability* to be a property of a program. It is what takes the place of correctness. In this note, we instead will focus on when a problem is a “good” problem, meaning that in exact arithmetic a “small” change in the input will always cause at most a “small” change in the output, or a “bad” problem if a “small” change may yield a “large” A good problems will be called *well-conditioned*. A bad problem will be called *ill-conditioned*.

Notice that “small” and “large” are vague. To some degree, norms help us measure size. To some degree, “small” and “large” will be in the eyes of the beholder (in other words, situation dependent).

Video

Read disclaimer regarding the videos in the preface!

Video did not turn out...

9.1.2 Outline

9.1	Opening Remarks	177
9.1.1	Launch	177
Video	177
9.1.2	Outline	178
9.1.3	What you will learn	179
9.2	Notation	180
9.3	The Prototypical Example: Solving a Linear System	180
9.4	Condition Number of a Rectangular Matrix	184
9.5	Why Using the Method of Normal Equations Could be Bad	185
9.6	Why Multiplication with Unitary Matrices is a Good Thing	186
9.7	Balancing a Matrix	186
9.8	Wrapup	188
9.8.1	Additional exercises	188
9.9	Wrapup	189
9.9.1	Additional exercises	189
9.9.2	Summary	189

9.1.3 What you will learn

9.2 Notation

Throughout this note, we will talk about small changes (perturbations) to scalars, vectors, and matrices. To denote these, we attach a “delta” to the symbol for a scalar, vector, or matrix.

- A small change to scalar $\alpha \in \mathbb{C}$ will be denoted by $\delta\alpha \in \mathbb{C}$;
- A small change to vector $x \in \mathbb{C}^n$ will be denoted by $\delta x \in \mathbb{C}^n$; and
- A small change to matrix $A \in \mathbb{C}^{m \times n}$ will be denoted by $\Delta A \in \mathbb{C}^{m \times n}$.

Notice that the “delta” touches the α , x , and A , so that, for example, δx is not mistaken for $\delta \cdot x$.

9.3 The Prototypical Example: Solving a Linear System

Assume that $A \in \mathbb{R}^{n \times n}$ is nonsingular and $x, y \in \mathbb{R}^n$ with $Ax = y$. The problem here is the function that computes x from y and A . Let us assume that no error is introduced in the matrix A when it is stored, but that in the process of storing y a small error is introduced: $\delta y \in \mathbb{R}^n$ so that now $y + \delta y$ is stored. The question becomes by how much the solution x changes as a function of δy . In particular, we would like to quantify how a relative change in the right-hand side y ($\|\delta y\|/\|y\|$ in some norm) translates to a relative change in the solution x ($\|\delta x\|/\|x\|$). It turns out that we will need to compute norms of matrices, using the norm induced by the vector norm that we use.

Since $Ax = y$, if we use a consistent (induced) matrix norm,

$$\|y\| = \|Ax\| \leq \|A\| \|x\| \text{ or, equivalently, } \frac{1}{\|x\|} \leq \|A\| \frac{1}{\|y\|}. \quad (9.1)$$

Also,

$$\left. \begin{array}{rcl} A(x + \delta x) & = & y + \delta y \\ Ax & = & y \end{array} \right\} \text{implies that } A\delta x = \delta y \text{ so that } \delta x = A^{-1}\delta y.$$

Hence

$$\|\delta x\| = \|A^{-1}\delta y\| \leq \|A^{-1}\| \|\delta y\|. \quad (9.2)$$

Combining (9.1) and (9.2) we conclude that

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

What does this mean? It means that the relative error in the solution is at worst the relative error in the right-hand side, amplified by $\|A\| \|A^{-1}\|$. So, if that quantity is “small” *and* the relative error in the right-hand size is “small” *and* exact arithmetic is used, then one is guaranteed a solution with a relatively “small” error.

The quantity $\kappa_{\|\cdot\|}(A) = \|A\| \|A^{-1}\|$ is called the *condition number* of nonsingular matrix A (associated with norm $\|\cdot\|$).

Are we overestimating by how much the relative error can be amplified? The answer to this is **no**. For every nonsingular matrix A , there exists a right-hand side y and perturbation δy such that, if $A(x + \delta x) = y + \delta y$,

$$\frac{\|\delta x\|}{\|x\|} = \|A\| \|A^{-1}\| \frac{\|\delta y\|}{\|y\|}.$$

In order for this equality to hold, we need to find y and δy such that

$$\|y\| = \|Ax\| = \|A\| \|x\| \text{ or, equivalently, } \|A\| = \frac{\|Ax\|}{\|x\|}$$

and

$$\|\delta x\| = \|A^{-1}\delta y\| = \|A^{-1}\| \|\delta y\|. \text{ or, equivalently, } \|A^{-1}\| = \frac{\|A^{-1}\delta y\|}{\|\delta y\|}.$$

In other words, x can be chosen as a vector that maximizes $\|Ax\|/\|x\|$ and δy should maximize $\|A^{-1}\delta y\|/\|\delta y\|$. The vector y is then chosen as $y = Ax$.

What if we use the 2-norm? For this norm, $\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2 = \sigma_0/\sigma_{n-1}$. So, the ratio between the largest and smallest singular value determines whether a matrix is well-conditioned or ill-conditioned.

To show for what vectors the maximal magnification is attained, consider the SVD

$$A = U\Sigma V^T = \left(\begin{array}{c|c|c|c} u_0 & u_1 & \cdots & u_{n-1} \end{array} \right) \left(\begin{array}{ccccc} \sigma_0 & & & & \\ & \sigma_1 & & & \\ & & \ddots & & \\ & & & & \sigma_{n-1} \end{array} \right) \left(\begin{array}{c|c|c|c} v_0 & v_1 & \cdots & v_{n-1} \end{array} \right)^H.$$

Recall that

- $\|A\|_2 = \sigma_0$, v_0 is the vector that maximizes $\max_{\|z\|_2=1} \|Az\|_2$, and $Av_0 = \sigma_0 u_0$;
- $\|A^{-1}\|_2 = 1/\sigma_{n-1}$, u_{n-1} is the vector that maximizes $\max_{\|z\|_2=1} \|A^{-1}z\|_2$, and $Av_{n-1} = \sigma_{n-1} u_{n-1}$.

Now, take $y = \sigma_0 u_0$. Then $Ax = y$ is solved by $x = v_0$. Take $\delta y = \beta \sigma_1 u_1$. Then $A\delta x = \delta y$ is solved by $x = \beta v_1$. Now,

$$\frac{\|\delta y\|_2}{\|y\|_2} = \frac{|\beta| \sigma_1}{\sigma_0} \text{ and } \frac{\|\delta x\|_2}{\|x\|_2} = |\beta|.$$

Hence

$$\frac{\|\delta x\|_2}{\|x\|_2} = \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta y\|_2}{\|y\|_2}$$

This is depicted in Figure 9.1 for $n = 2$.

The SVD can be used to show that A maps the unit ball to an ellipsoid. The singular values are the lengths of the various axes of the ellipsoid. The condition number thus captures the eccentricity of the ellipsoid: the ratio between the lengths of the largest and smallest axes. This is also illustrated in Figure 9.1.

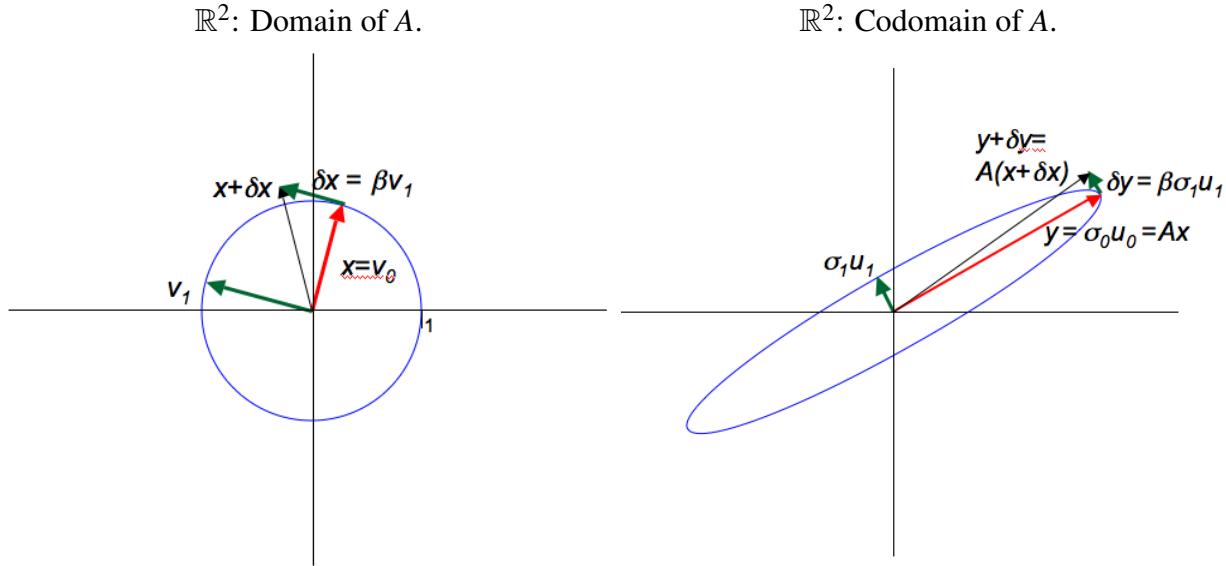


Figure 9.1: Illustration for choices of vectors y and δy that result in $\frac{\|\delta x\|_2}{\|x\|_2} = \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta y\|_2}{\|y\|_2}$. Because of the eccentricity of the ellipse, the relatively small change δy relative to y is amplified into a relatively large change δx relative to x . On the right, we see that $\|\delta y\|_2 / \|y\|_2 = \beta \sigma_1 / \sigma_0$ (since $\|u_0\|_2 = \|u_1\|_2 = 1$). On the left, we see that $\|\delta x\|_2 / \|x\| = \beta$ (since $\|v_0\|_2 = \|v_1\|_2 = 1$).

Number of accurate digits Notice that for scalars $\delta\psi$ and ψ , $\log_{10}\left(\frac{\delta\psi}{\psi}\right) = \log_{10}(\delta\psi) - \log_{10}(\psi)$ roughly equals the number leading decimal digits of $\psi + \delta\psi$ that are accurate, relative to ψ . For example, if $\psi = 32.512$ and $\delta\psi = 0.02$, then $\psi + \delta\psi = 32.532$ which has three accurate digits (highlighted in red). Now, $\log_{10}(32.512) - \log_{10}(0.02) \approx 1.5 - (-1.7) = 3.2$.

Now, if

$$\frac{\|\delta x\|}{\|x\|} = \kappa(A) \frac{\|\delta y\|}{\|y\|}.$$

then

$$\log_{10}(\|\delta x\|) - \log_{10}(\|x\|) = \log_{10}(\kappa(A)) + \log_{10}(\|\delta y\|) - \log_{10}(\|y\|)$$

so that

$$\log_{10}(\|x\|) - \log_{10}(\|\delta x\|) = [\log_{10}(\|y\|) - \log_{10}(\|\delta y\|)] - \log_{10}(\kappa(A)).$$

In other words, if there were k digits of accuracy in the right-hand side, then it is possible that (due only to the condition number of A) there are only $k - \log_{10}(\kappa(A))$ digits of accuracy in the solution. If we start with only 8 digits of accuracy and $\kappa(A) = 10^5$, we may only get 3 digits of accuracy. If $\kappa(A) \geq 10^8$, we may not get *any* digits of accuracy...

Homework 9.1 Show that, if A is a nonsingular matrix, for a consistent matrix norm, $\kappa(A) \geq 1$.

SEE ANSWER

We conclude from this that we can generally only expect as much relative accuracy in the solution as we had in the right-hand side.

Alternative exposition Note: the below links conditioning of matrices to the relative condition number of a more general function. For a more thorough treatment, you may want to read Lecture 12 of “Trefethen and Bau”. That book discusses the subject in much more generality than is needed for our discussion of linear algebra. Thus, if this alternative exposition baffles you, just skip it!

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be a continuous function such that $f(y) = x$. Let $\|\cdot\|$ be a vector norm. Consider for $y \neq 0$

$$\kappa^f(y) = \lim_{\delta \rightarrow 0} \sup_{\substack{\|\delta y\| \leq \delta}} \left(\frac{\|f(y + \delta y) - f(y)\|}{\|f(y)\|} \right) / \left(\frac{\|\delta y\|}{\|y\|} \right).$$

Letting $f(y + \delta y) = x + \delta x$, we find that

$$\kappa^f(y) = \lim_{\delta \rightarrow 0} \sup_{\substack{\|\delta y\| \leq \delta}} \left(\frac{\|x + \delta x\|}{\|x\|} \right) / \left(\frac{\|\delta y\|}{\|y\|} \right).$$

(Obviously, if $\delta y = 0$ or $y = 0$ or $f(y) = 0$, things get a bit hairy, so let's not allow that.)

Roughly speaking, $\kappa^f(y)$ equals the maximum that a(n infinitesimally) small relative error in y is magnified into a relative error in $f(y)$. This can be considered the **relative condition number** of function f . A large relative condition number means a small relative error in the input (y) can be magnified into a large relative error in the output ($x = f(y)$). This is bad, since small errors will invariably occur.

Now, if $f(y) = x$ is the function that returns x where $Ax = y$ for a nonsingular matrix $A \in \mathbb{C}^{n \times n}$, then via an argument similar to what we did earlier in this section we find that $\kappa^f(y) \leq \kappa(A) = \|A\| \|A^{-1}\|$, the condition number of matrix A :

$$\begin{aligned} & \lim_{\delta \rightarrow 0} \sup_{\substack{\|\delta y\| \leq \delta}} \left(\frac{\|f(y + \delta y) - f(y)\|}{\|f(y)\|} \right) / \left(\frac{\|\delta y\|}{\|y\|} \right) \\ &= \lim_{\delta \rightarrow 0} \sup_{\substack{\|\delta y\| \leq \delta}} \left(\frac{\|A^{-1}(y + \delta y) - A^{-1}(y)\|}{\|A^{-1}y\|} \right) / \left(\frac{\|\delta y\|}{\|y\|} \right) \\ &= \lim_{\delta \rightarrow 0} \max_{\substack{\|z\| = 1}} \left(\frac{\|A^{-1}(y + \delta y) - A^{-1}(y)\|}{\|A^{-1}y\|} \right) / \left(\frac{\|\delta y\|}{\|y\|} \right) \\ &\quad \delta y = \delta \cdot z \\ &= \lim_{\delta \rightarrow 0} \max_{\substack{\|z\| = 1}} \left(\frac{\|A^{-1}\delta y\|}{\|\delta y\|} \right) / \left(\frac{\|A^{-1}y\|}{\|y\|} \right) \\ &\quad \delta y = \delta \cdot z \\ &= \lim_{\delta \rightarrow 0} \max_{\substack{\|z\| = 1}} \left(\frac{\|A^{-1}\delta y\|}{\|\delta y\|} \right) \left(\frac{\|y\|}{\|A^{-1}y\|} \right) \\ &\quad \delta y = \delta \cdot z \end{aligned}$$

$$\begin{aligned}
&= \left[\lim_{\delta \rightarrow 0} \max_{\substack{\|z\|=1 \\ \delta y = \delta \cdot z}} \left(\frac{\|A^{-1}\delta y\|}{\|\delta y\|} \right) \right] \left[\left(\frac{\|y\|}{\|A^{-1}y\|} \right) \right] \\
&= \lim_{\delta \rightarrow 0} \max_{\substack{\|z\|=1 \\ \delta y = \delta \cdot z}} \left(\frac{\|A^{-1}(\delta \cdot z)\|}{\|\delta \cdot z\|} \right) \left(\frac{\|y\|}{\|A^{-1}y\|} \right) \\
&\quad \delta y = \delta \cdot z \\
&= \max_{\|z\|=1} \left(\frac{\|A^{-1}z\|}{\|z\|} \right) \left(\frac{\|y\|}{\|A^{-1}y\|} \right) \\
&= \max_{\|z\|=1} \left(\frac{\|A^{-1}z\|}{\|z\|} \right) \left(\frac{\|Ax\|}{\|x\|} \right) \\
&\leq \left[\max_{\|z\|=1} \left(\frac{\|A^{-1}z\|}{\|z\|} \right) \right] \left[\max_{x \neq 0} \left(\frac{\|Ax\|}{\|x\|} \right) \right] \\
&= \|A\| \|A^{-1}\|,
\end{aligned}$$

where $\|\cdot\|$ is the matrix norm induced by vector norm $\|\cdot\|$.

9.4 Condition Number of a Rectangular Matrix

Given $A \in \mathbb{C}^{m \times n}$ with linearly independent columns and $y \in \mathbb{C}^m$, consider the linear least-squares (LLS) problem

$$\|Ax - y\|_2 = \min_w \|Aw - y\|_2 \quad (9.3)$$

and the perturbed problem

$$\|A(x + \delta x) - y\|_2 = \min_{w+\delta w} \|A(w + \delta w) - (y + \delta y)\|_2. \quad (9.4)$$

We will again bound by how much the relative error in y is amplified.

Notice that the solutions to (9.3) and (9.4) respectively satisfy

$$\begin{aligned}
A^H Ax &= A^H y \\
A^H A(x + \delta x) &= A^H(y + \delta y)
\end{aligned}$$

so that $A^H A \delta x = A^H \delta y$ (subtracting the first equation from the second) and hence

$$\|\delta x\|_2 = \|(A^H A)^{-1} A^H \delta y\|_2 \leq \|(A^H A)^{-1} A^H\|_2 \|\delta y\|_2.$$

Now, let $z = A(A^H A)^{-1} A^H y$ be the projection of y onto $C(A)$ and let θ be the angle between z and y . Let us assume that y is not orthogonal to $C(A)$ so that $z \neq 0$. Then $\cos \theta = \|z\|_2 / \|y\|_2$ so that

$$\cos \theta \|y\|_2 = \|z\|_2 = \|Ax\|_2 \leq \|A\|_2 \|x\|_2$$

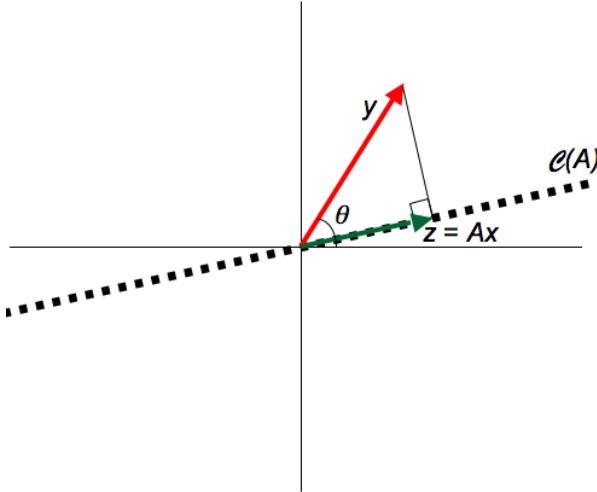


Figure 9.2: Linear least-squares problem $\|Ax - y\|_2 = \min_v \|Av - y\|_2$. Vector z is the projection of y onto $C(A)$.

and hence

$$\frac{1}{\|x\|_2} \leq \frac{\|A\|_2}{\cos \theta \|y\|_2}$$

We conclude that

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq \frac{\|A\|_2 \|(A^H A)^{-1} A^H\|_2}{\cos \theta} \frac{\|\delta y\|_2}{\|y\|_2} = \frac{1}{\cos \theta} \frac{\sigma_0}{\sigma_{n-1}} \frac{\|\delta y\|_2}{\|y\|_2}$$

where σ_0 and σ_{n-1} are (respectively) the largest and smallest singular values of A , because of the following result:

Homework 9.2 If A has linearly independent columns, show that $\|(A^H A)^{-1} A^H\|_2 = 1/\sigma_{n-1}$, where σ_{n-1} equals the smallest singular value of A . Hint: Use the SVD of A .

SEE ANSWER

The condition number of $A \in \mathbb{C}^{m \times n}$ with linearly independent columns is $\kappa_2(A) = \sigma_0/\sigma_{n-1}$.

Notice the effect of the $\cos \theta$. When y is almost perpendicular to $C(A)$, then its projection z is small and $\cos \theta$ is small. Hence a small relative change in y can be greatly amplified. This makes sense: if y is almost perpendicular to $C(A)$, then $x \approx 0$, and any small $\delta y \in C(A)$ can yield a relatively large change δx .

9.5 Why Using the Method of Normal Equations Could be Bad

Homework 9.3 Let A have linearly independent columns. Show that $\kappa_2(A^H A) = \kappa_2(A)^2$.

SEE ANSWER

Homework 9.4 Let $A \in \mathbb{C}^{n \times n}$ have linearly independent columns.

- Show that $Ax = y$ if and only if $A^H A x = A^H y$.
- Reason that using the method of normal equations to solve $Ax = y$ has a condition number of $\kappa_2(A)^2$.

 [SEE ANSWER](#)

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 $\min_x \|Ax - y\|_2$, one ends up solving the square linear system $A^H A x = A^H y$. Now, $\kappa_2(A^H A) = \kappa_2(A)^2$. Hence, using this method squares the condition number of the matrix being used.

9.6 Why Multiplication with Unitary Matrices is a Good Thing

Next, consider the computation $C = AB$ where $A \in \mathbb{C}^{m \times m}$ is nonsingular and $B, \Delta B, C, \Delta C \in \mathbb{C}^{m \times n}$. Then

$$\begin{aligned}(C + \Delta C) &= A(B + \Delta B) \\ C &= AB \\ \Delta C &= A\Delta B\end{aligned}$$

Thus,

$$\|\Delta C\|_2 = \|A\Delta B\|_2 \leq \|A\|_2 \|\Delta B\|_2.$$

Also, $B = A^{-1}C$ so that

$$\|B\|_2 = \|A^{-1}C\|_2 \leq \|A^{-1}\|_2 \|C\|_2$$

and hence

$$\frac{1}{\|C\|_2} \leq \|A^{-1}\|_2 \frac{1}{\|B\|_2}.$$

Thus,

$$\frac{\|\Delta C\|_2}{\|C\|_2} \leq \|A\|_2 \|A^{-1}\|_2 \frac{\|\Delta B\|_2}{\|B\|_2} = \kappa_2(A) \frac{\|\Delta B\|_2}{\|B\|_2}.$$

This means that the relative error in matrix $C = AB$ is at most $\kappa_2(A)$ greater than the relative error in B .

The following exercise gives us a hint as to why algorithms that cast computation in terms of multiplication by unitary matrices avoid the buildup of error:

Homework 9.5 Let $U \in \mathbb{C}^{n \times n}$ be unitary. Show that $\kappa_2(U) = 1$.

 [SEE ANSWER](#)

This means is that the relative error in matrix $C = UB$ is no greater than the relative error in B when U is unitary.

Homework 9.6 Characterize the set of all square matrices A with $\kappa_2(A) = 1$.

 [SEE ANSWER](#)

9.7 Balancing a Matrix

Consider the following problem: You buy two items, apples and oranges, at a price of χ_0 and χ_1 , respectively, but you forgot how much each was. But what you do remember is that the first time you

$$\begin{aligned}2 \text{ kg. of apples} \times \frac{\text{dollars}}{\text{kg. of apples}} + 3 \text{ kg. of oranges} \times \frac{\text{dollars}}{\text{kg. of oranges}} &= 8 \text{ dollars} \\ 3 \text{ kg. of apples} \times \frac{\text{dollars}}{\text{kg. of apples}} + 2 \text{ kg. of oranges} \times \frac{\text{dollars}}{\text{kg. of oranges}} &= 5 \text{ dollars}\end{aligned}$$

In matrix notation this becomes

$$\begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} 8 \\ 5 \end{pmatrix}.$$

The condition number of the matrix $A = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}$ is

$$\kappa_2(A) = 5.$$

Now, let us change the problem to

$$2 \text{ kg.s of apples} \times \zeta_0 \frac{\text{dollars}}{\text{kg. of apples}} + 3000 \text{ g. of oranges} \times \zeta_1 \frac{\text{dollars}}{\text{g. of oranges}} = 8 \text{ dollars}$$

$$3 \text{ kg.s of apples} \times \zeta_0 \frac{\text{dollars}}{\text{kg. of apples}} + 2000 \text{ g. of oranges} \times \zeta_1 \frac{\text{dollars}}{\text{g. of oranges}} = 5 \text{ dollars}$$

Clearly, this is an equivalent problem, except that χ_1 is computed as the cost per gram of oranges. In matrix notation this becomes

$$\begin{pmatrix} 2 & 3000 \\ 3 & 2000 \end{pmatrix} \begin{pmatrix} \zeta_0 \\ \zeta_1 \end{pmatrix} = \begin{pmatrix} 8 \\ 5 \end{pmatrix}.$$

The condition number of the matrix $B = \begin{pmatrix} 2 & 3000 \\ 3 & 2000 \end{pmatrix}$ is

$$\kappa_2(B) = 2600.$$

So what is going on?

One way to look at this is that matrix B is close to singular: If only three significant digits are stored, then compared to 3000 and 2000 the values 2 and 3 are not that different from 0. So in that case

$$\begin{pmatrix} 2 & 3000 \\ 3 & 2000 \end{pmatrix} \approx \begin{pmatrix} 0 & 3000 \\ 0 & 2000 \end{pmatrix},$$

which is a singular matrix, with condition number equal to infinity.

Notice that the vectors x and z are related by

$$\begin{pmatrix} \zeta_0 \\ \zeta_1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1000 \end{pmatrix}}_{D_R} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}.$$

Also

$$Ax = b$$

can be changed to

$$\underbrace{AD_R}_{B} D_R^{-1} z = b.$$

What this in turn shows is that *sometimes* a poorly conditioned matrix can be transformed into a well-conditioned matrix (or, at least, a better conditioned matrix). When presented with $Ax = b$, a procedure for solving for x may want to start by balancing the norms of rows and columns of A :

$$\underbrace{D_L A D_R^{-1}}_B \underbrace{D_R x}_z = \underbrace{D_L b}_{\hat{b}},$$

where D_L and D_R are diagonal matrices chosen so that $\kappa(B)$ is smaller than $\kappa(A)$. This is known as balancing the matrix. The example also shows how this can be thought of as picking the units in terms of which the entries of x and b are expressed carefully so that large values are not artificially introduced into the matrix.

9.8 Wrapup

9.8.1 Additional exercises

In our discussions of the conditioning of $Ax = y$ where A is nonsingular, we assumed that error only occurred in y : $A(x + \delta x) = y + \delta y$. Now, A is also input to the problem and hence can also have error in it. The following questions relate to this.

Homework 9.7 Let $\|\cdot\|$ be a vector norm with corresponding induced matrix norm. If $A \in \mathbb{C}^{n \times n}$ is nonsingular and

$$\frac{\|\Delta A\|}{\|A\|} < \frac{1}{\kappa(A)}. \quad (9.5)$$

then $A + \Delta A$ is nonsingular.

[SEE ANSWER](#)

Homework 9.8 Let $A \in \mathbb{C}^{n \times n}$ be nonsingular, $Ax = y$, and $(A + \Delta A)(x + \delta x) = y$. Show that (for induced norm)

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leq \kappa(A) \frac{\|\Delta A\|}{\|A\|}.$$

[SEE ANSWER](#)

Homework 9.9 Let $A \in \mathbb{C}^{n \times n}$ be nonsingular, $\|\Delta A\|/\|A\| < 1/\kappa(A)$, $y \neq 0$, $Ax = y$, and $(A + \Delta A)(x + \delta x) = y$. Show that (for induced norm)

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A) \frac{\|\Delta A\|}{\|A\|}}{1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}}.$$

[SEE ANSWER](#)

Homework 9.10 Let $A \in \mathbb{C}^{n \times n}$ be nonsingular, $\|\Delta A\|/\|A\| < 1$, $y \neq 0$, $Ax = y$, and $(A + \Delta A)(x + \delta x) = y + \delta y$. Show that (for induced norm)

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A) \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\delta y\|}{\|y\|} \right)}{1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}}.$$

[SEE ANSWER](#)

9.9 Wrapup

9.9.1 Additional exercises

9.9.2 Summary

Chapter **10**

Notes on the Stability of an Algorithm

Based on “Goal-Oriented and Modular Stability Analysis” [7, 8] by Paolo Bientinesi and Robert van de Geijn.

Video

Read disclaimer regarding the videos in the preface!

 [Lecture on the Stability of an Algorithm](#)

-  [YouTube](#)
-  [Download from UT Box](#)
-  [View After Local Download](#)

(For help on viewing, see Appendix A.)

10.0.1 Launch

10.0.2 Outline

10.0.1	Launch	191
10.0.2	Outline	192
10.0.3	What you will learn	193
10.1	Motivation	194
10.2	Floating Point Numbers	195
10.3	Notation	197
10.4	Floating Point Computation	197
10.4.1	Model of floating point computation	197
10.4.2	Stability of a numerical algorithm	198
10.4.3	Absolute value of vectors and matrices	198
10.5	Stability of the Dot Product Operation	199
10.5.1	An algorithm for computing DOT	199
10.5.2	A simple start	199
10.5.3	Preparation	201
10.5.4	Target result	203
10.5.5	A proof in traditional format	204
10.5.6	A weapon of math induction for the war on (backward) error (optional)	204
10.5.7	Results	206
10.6	Stability of a Matrix-Vector Multiplication Algorithm	207
10.6.1	An algorithm for computing GEMV	207
10.6.2	Analysis	207
10.7	Stability of a Matrix-Matrix Multiplication Algorithm	209
10.7.1	An algorithm for computing GEMM	209
10.7.2	Analysis	209
10.7.3	An application	210
10.8	Wrapup	210
10.8.1	Additional exercises	210
10.8.2	Summary	210

10.0.3 What you will learn

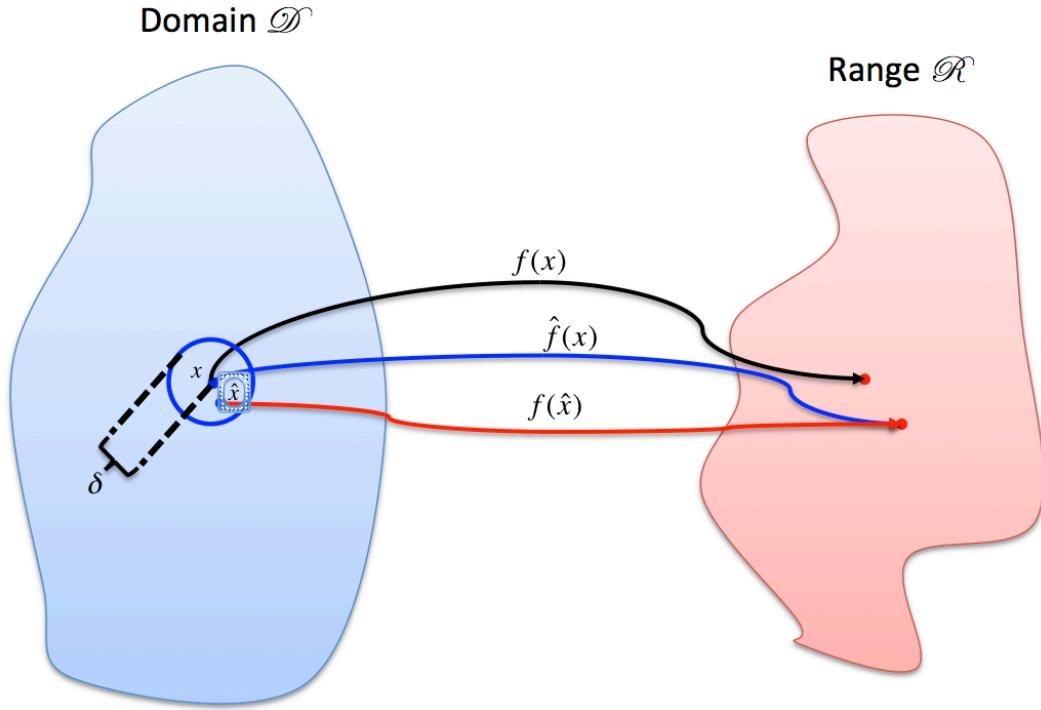


Figure 10.1: In this illustration, $f : \mathcal{D} \rightarrow \mathcal{R}$ is a function to be evaluated. The function \check{f} represents the implementation of the function that uses floating point arithmetic, thus incurring errors. The fact that for a nearby value \check{x} the computed value equals the exact function applied to the slightly perturbed x , $f(\check{x}) = \check{f}(x)$, 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 .

10.1 Motivation

Correctness in the presence of error (e.g., when floating point computations are performed) takes on a different meaning. For many problems for which computers are used, there is one correct answer and we expect that answer to be computed by our program. The problem is that most real numbers cannot be stored exactly in a computer memory. They are stored as approximations, floating point numbers, instead. Hence storing them and/or computing with them inherently incurs error. The question thus becomes “When is a program correct in the presence of such errors?”

Let us assume that we wish to evaluate the mapping $f : \mathcal{D} \rightarrow \mathcal{R}$ where $\mathcal{D} \subset \mathbb{R}^n$ is the domain and $\mathcal{R} \subset \mathbb{R}^m$ is the range (codomain). Now, we will let $\hat{f} : \mathcal{D} \rightarrow \mathcal{R}$ denote a computer implementation of this function. Generally, for $x \in \mathcal{D}$ it is the case that $f(x) \neq \hat{f}(x)$. Thus, the computed value is not “correct”. From the Notes on Conditioning, we know that it may not be the case that $\hat{f}(x)$ is “close to” $f(x)$. After all, even if \hat{f} is an exact implementation of f , the mere act of storing x may introduce a small error δ_x and $f(x + \delta_x)$ may be far from $f(x)$ if f is ill-conditioned.

The following defines a property that captures correctness in the presence of the kinds of errors that are introduced by computer arithmetic:

Definition 10.1 Let given the mapping $f : D \rightarrow R$, where $D \subset \mathbb{R}^n$ is the domain and $R \subset \mathbb{R}^m$ is the range (codomain), let $\hat{f} : D \rightarrow R$ be a computer implementation of this function. We will call \hat{f} a (numerically) stable implementation of f on domain \mathcal{D} if for all $x \in D$ there exists a \hat{x} “close” to x such that $\hat{f}(x) = f(\hat{x})$.

In other words, \hat{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 10.1 for a specific input x . If an implementation is not stable, it is numerically unstable.

10.2 Floating Point Numbers

Only a finite number of (binary) digits can be used to store a real number number. For so-called single-precision and double-precision floating point numbers 32 bits and 64 bits are typically employed, respectively. Let us focus on double precision numbers.

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 F as the set of all numbers $\chi = \mu\beta^e$ such that $\beta = 2$, $\mu = \pm.\delta_0\delta_1 \dots \delta_{t-1}$ has only t (binary) digits ($\delta_j \in \{0, 1\}$), $\delta_0 = 0$ iff $\mu = 0$ (the mantissa is normalized), and $-L \leq e \leq U$. Elements in F can be stored with a finite number of (binary) digits.

- There is a largest number (in absolute value) that can be stored. Any number that is larger “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 “underflows”. Typically, this causes a zero to be stored.

Example 10.2 For $x \in \mathbb{R}^n$, consider computing

$$\|x\|_2 = \sqrt{\sum_{i=0}^{n-1} \chi_i^2}. \quad (10.1)$$

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 (10.1) 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 further discussions, we will ignore overflow and underflow issues.

What is important is that any time a real number is stored in our computer, it is stored as the nearest floating point number (element in F). We first assume that it is truncated (which makes our explanation slightly simpler).

Let positive χ be represented by

$$\chi = .\delta_0\delta_1 \dots \times 2^e,$$

where $\delta_0 = 1$ (the mantissa is normalized). If t digits are stored by our floating point system, then $\hat{\chi} = .\delta_0\delta_1 \dots \delta_{t-1} \times 2^e$ is stored. Let $\delta\chi = \chi - \hat{\chi}$. Then

$$\delta\chi = \underbrace{.\delta_0\delta_1 \dots \delta_{t-1}\delta_t \dots \times 2^e}_{\chi} - \underbrace{.\delta_0\delta_1 \dots \delta_{t-1} \times 2^e}_{\hat{\chi}} = \underbrace{.0 \dots 00}_{t} \delta_t \dots \times 2^e < \underbrace{.0 \dots 01}_{t} \times 2^e = 2^{e-t}.$$

Also, since χ is positive,

$$\chi = .\delta_0\delta_1 \dots \times 2^e \geq .1 \times 2^e = 2^{e-1}.$$

Thus,

$$\frac{\delta\chi}{\chi} \leq \frac{2^{e-t}}{2^{e-1}} = 2^{1-t}$$

which can also be written as

$$\delta\chi \leq 2^{1-t}\chi.$$

A careful analysis of what happens when χ might equal zero or be negative yields

$$|\delta\chi| \leq 2^{1-t}|\chi|.$$

Now, in practice any base β can be used and floating point computation uses rounding rather than truncating. A similar analysis can be used to show that then

$$|\delta\chi| \leq u|\chi|$$

where $u = \frac{1}{2}\beta^{1-t}$ is known as the **machine epsilon** or **unit roundoff**. When using single precision or double precision real arithmetic, $u \approx 10^{-8}$ or 10^{-16} , respectively. The quantity u is machine dependent; it is a function of the parameters characterizing the machine arithmetic.

The unit roundoff is often alternatively defined as the maximum positive floating point number which can be added to the number stored as 1 without changing the number stored as 1. In the notation introduced below, $\text{fl}(1+u) = 1$.

Homework 10.3 Assume a floating point number system with $\beta = 2$ and a mantissa with t digits so that a typical positive number is written as $.d_0d_1 \dots d_{t-1} \times 2^e$, with $d_i \in \{0, 1\}$.

- Write the number 1 as a floating point number.
- What is the largest positive real number u (represented as a binary fraction) such that the floating point representation of $1+u$ equals the floating point representation of 1? (Assume rounded arithmetic.)
- Show that $u = \frac{1}{2}2^{1-t}$.

 SEE ANSWER

10.3 Notation

When discussing error analyses, 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, assuming that the expressions are evaluated from left to right, $\text{fl}(\chi + \psi + \zeta/\omega)$ is equivalent to $\text{fl}(\text{fl}(\text{fl}(\chi) + \text{fl}(\psi)) + \text{fl}(\text{fl}(\zeta)/\text{fl}(\omega)))$. Equality between the quantities *lhs* and *rhs* is denoted by $\text{lhs} = \text{rhs}$. Assignment of *rhs* to *lhs* is denoted by $\text{lhs} := \text{rhs}$ (*lhs* becomes *rhs*). In the context of a program, the statements $\text{lhs} := \text{rhs}$ and $\text{lhs} := \text{fl(rhs)}$ 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 .

10.4 Floating Point Computation

We introduce definitions and results regarding floating point arithmetic. In this note, we focus on real valued arithmetic only. Extensions to complex arithmetic are straightforward.

10.4.1 Model of floating point computation

The **Standard Computational Model (SCM)** assumes that, for any two floating point numbers χ and ψ , the basic arithmetic operations satisfy the equality

$$\text{fl}(\chi \text{ op } \psi) = (\chi \text{ op } \psi)(1 + \varepsilon), \quad |\varepsilon| \leq \mathbf{u}, \text{ and } \text{op} \in \{+, -, *, /\}.$$

The quantity ε is a function of χ, ψ and op . Sometimes we add a subscript ($\varepsilon_+, \varepsilon_*, \dots$) to indicate what operation generated the $(1 + \varepsilon)$ error factor. We always assume that all the input variables to an operation are floating point numbers. **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 (equal to that introduced when a real number is stored as a floating point number).**

Remark 10.4 *Given $\chi, \psi \in F$, performing any operation $\text{op} \in \{+, -, *, /\}$ with χ and ψ in floating point arithmetic, $[\chi \text{ op } \psi]$, is a stable operation: Let $\zeta = \chi \text{ op } \psi$ and $\hat{\zeta} = \zeta + \delta\zeta = [\chi \text{ (op) } \psi]$. Then $|\delta\zeta| \leq \mathbf{u}|\zeta|$ and hence $\hat{\zeta}$ is close to ζ (it has k correct binary digits).*

For certain problems it is convenient to use the **Alternative Computational Model (ACM)** [26], which also assumes for the basic arithmetic operations that

$$\text{fl}(\chi \text{ op } \psi) = \frac{\chi \text{ op } \psi}{1 + \varepsilon}, \quad |\varepsilon| \leq \mathbf{u}, \text{ and } \text{op} \in \{+, -, *, /\}.$$

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.

Remark 10.5 *The Taylor series expansion of $1/(1 + \varepsilon)$ is given by*

$$\frac{1}{1 + \varepsilon} = 1 + (-\varepsilon) + O(\varepsilon^2),$$

which explains how the SCM and ACM are related.

Remark 10.6 *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.*

10.4.2 Stability of a numerical algorithm

In the presence of round-off error, an algorithm involving numerical computations cannot be expected to yield the exact result. Thus, the conventional notion of “correctness” applies only to the execution of algorithms in exact arithmetic. Here we briefly introduce the notion of “stability” of algorithms.

Let $f : \mathcal{D} \rightarrow \mathcal{R}$ be a mapping from the domain \mathcal{D} to the range \mathcal{R} and let $\check{f} : \mathcal{D} \rightarrow \mathcal{R}$ represent the mapping that captures the execution in floating point arithmetic of a given algorithm which computes f .

The algorithm is said to be **backward stable** if for all $x \in \mathcal{D}$ there exists a perturbed input $\check{x} \in \mathcal{D}$, close to x , such that $\check{f}(x) = f(\check{x})$. In other words, the computed result equals the result obtained when the exact function is applied to slightly perturbed data. The difference between \check{x} and x , $\delta x = \check{x} - x$, is the perturbation to the original input x .

The reasoning behind backward stability is as follows. The input to a function typically has some errors associated with it. Uncertainty may be due to measurement errors when obtaining the input and/or may be the result of converting real numbers to floating point numbers when storing the input on a computer. If it can be shown that an implementation is backward stable, then it has been proved that the result could have been obtained through exact computations performed on slightly corrupted input. Thus, one can think of the error introduced by the implementation as being comparable to the error introduced when obtaining the input data in the first place.

When discussing error analyses, δx , the difference between x and \check{x} , is the backward error and the difference $\check{f}(x) - f(x)$ is the forward error. Throughout the remainder of this note we will be concerned with bounding the backward and/or forward errors introduced by the algorithms executed with floating point arithmetic.

The algorithm is said to be **forward stable** on domain \mathcal{D} if for all $x \in \mathcal{D}$ it is the case that $\check{f}(x) \approx f(x)$. In other words, the computed result equals a slight perturbation of the exact result.

10.4.3 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. Instead, for the operations analyzed in this note, all bounds are given 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 10.7 Given $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$,

$$|x| = \begin{pmatrix} |\chi_0| \\ |\chi_1| \\ \vdots \\ |\chi_{n-1}| \end{pmatrix} \quad \text{and} \quad |A| = \begin{pmatrix} |\alpha_{0,0}| & |\alpha_{0,1}| & \dots & |\alpha_{0,n-1}| \\ |\alpha_{1,0}| & |\alpha_{1,1}| & \dots & |\alpha_{1,n-1}| \\ \vdots & \vdots & \ddots & \vdots \\ |\alpha_{m-1,0}| & |\alpha_{m-1,1}| & \dots & |\alpha_{m-1,n-1}| \end{pmatrix}.$$

Definition 10.8 Let $\Delta \in \{<, \leq, =, \geq, >\}$ and $x, y \in \mathbb{R}^n$. Then

$$|x| \Delta |y| \quad \text{iff} \quad |\chi_i| \Delta |\psi_i|,$$

with $i = 0, \dots, n-1$. Similarly, given A and $B \in \mathbb{R}^{m \times n}$,

$$|A| \Delta |B| \quad \text{iff} \quad |\alpha_{ij}| \Delta |\beta_{ij}|,$$

with $i = 0, \dots, m-1$ and $j = 0, \dots, n-1$.

The next Lemma is exploited in later sections:

Lemma 10.9 *Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$. Then $|AB| \leq |A||B|$.*

Homework 10.10 *Prove Lemma 10.9.*

☞ SEE ANSWER

The fact that the bounds that we establish can be easily converted into bounds involving norms is a consequence of the following theorem, where $\|\cdot\|_F$ indicates the Frobenius matrix norm.

Theorem 10.11 *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 $\|A\|_F \leq \|B\|_F$.*

Homework 10.12 *Prove Theorem 10.11.*

☞ SEE ANSWER

10.5 Stability of the Dot Product Operation

The matrix-vector multiplication algorithm discussed in the next section requires the computation of the dot (inner) product (DOT) of vectors $x, y \in \mathbb{R}^n$: $\kappa := x^T y$. In this section, we give an algorithm for this operation and the related error results.

10.5.1 An algorithm for computing DOT

We will consider the algorithm given in Figure 10.2. It uses the FLAME notation [25, 4] to express the computation

$$\kappa := \left(((\chi_0 \psi_0 + \chi_1 \psi_1) + \dots) + \chi_{n-2} \psi_{n-2} \right) + \chi_{n-1} \psi_{n-1} \quad (10.2)$$

in the indicated order.

10.5.2 A simple start

Before giving a general result, let us focus on the case where $n = 2$:

$$\kappa := \chi_0 \psi_0 + \chi_1 \psi_1.$$

Then, under the computational model given in Section 10.4, if $\kappa := \chi_0 \psi_0 + \chi_1 \psi_1$ is executed, the computed result, $\check{\kappa}$, satisfies

$$\begin{aligned} \check{\kappa} &= \left[\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right] \\ &= [\chi_0 \psi_0 + \chi_1 \psi_1] \\ &= [[\chi_0 \psi_0] + [\chi_1 \psi_1]] \\ &= [\chi_0 \psi_0 (1 + \epsilon_*^{(0)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)})] \\ &= (\chi_0 \psi_0 (1 + \epsilon_*^{(0)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)})) (1 + \epsilon_+^{(1)}) \\ &= \chi_0 \psi_0 (1 + \epsilon_*^{(0)}) (1 + \epsilon_+^{(1)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)}) (1 + \epsilon_+^{(1)}) \end{aligned}$$

Algorithm: DOT: $\kappa := x^T y$

$$x \rightarrow \begin{pmatrix} x_T \\ x_B \end{pmatrix}, y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix}$$

while $m(x_T) < m(x)$ **do**

$$\begin{pmatrix} x_T \\ x_B \end{pmatrix} \rightarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \psi_1 \\ y_2 \end{pmatrix}$$

$$\kappa := \kappa + \chi_1 \psi_1$$

$$\begin{pmatrix} x_T \\ x_B \end{pmatrix} \leftarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \psi_1 \\ y_2 \end{pmatrix}$$

endwhile

Figure 10.2: Algorithm for computing $\kappa := x^T y$.

$$\begin{aligned} &= \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0(1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) \\ \psi_1(1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix} \\ &= \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) & 0 \\ 0 & (1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \\ &= \begin{pmatrix} \chi_0(1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) \\ \chi_1(1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix}, \end{aligned}$$

where $|\epsilon_*^{(0)}|, |\epsilon_*^{(1)}|, |\epsilon_+^{(1)}| \leq u$.

Homework 10.13 Repeat the above steps for the computation

$$\kappa := ((\chi_0 \psi_0 + \chi_1 \psi_1) + \chi_2 \psi_2),$$

computing in the indicated order

 SEE ANSWER

10.5.3 Preparation

Under the computational model given in Section 10.4, the computed result of (10.2), $\check{\kappa}$, satisfies

$$\begin{aligned}\check{\kappa} &= \left(\left((\chi_0 \Psi_0 (1 + \varepsilon_*^{(0)}) + \chi_1 \Psi_1 (1 + \varepsilon_*^{(1)})) (1 + \varepsilon_+^{(1)}) + \dots \right) (1 + \varepsilon_+^{(n-2)}) + \chi_{n-1} \Psi_{n-1} (1 + \varepsilon_*^{(n-1)}) \right) (1 + \varepsilon_+^{(n-1)}) \\ &= \sum_{i=0}^{n-1} \left(\chi_i \Psi_i (1 + \varepsilon_*^{(i)}) \prod_{j=i}^{n-1} (1 + \varepsilon_+^{(j)}) \right),\end{aligned}\quad (10.3)$$

where $\varepsilon_+^{(0)} = 0$ and $|\varepsilon_*^{(0)}|, |\varepsilon_*^{(j)}|, |\varepsilon_+^{(j)}| \leq \mathbf{u}$ for $j = 1, \dots, n-1$.

Clearly, a notation to keep expressions from becoming unreadable is desirable. For this reason we introduce the symbol θ_j :

Lemma 10.14 *Let $\varepsilon_i \in \mathbb{R}$, $0 \leq i \leq n-1$, $n\mathbf{u} < 1$, and $|\varepsilon_i| \leq \mathbf{u}$. Then $\exists \theta_n \in \mathbb{R}$ such that*

$$\prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1} = 1 + \theta_n,$$

with $|\theta_n| \leq n\mathbf{u}/(1 - n\mathbf{u})$.

Proof: By Mathematical Induction.

Base case. $n = 1$. Trivial.

Inductive Step. The Inductive Hypothesis (I.H.) tells us that for all $\varepsilon_i \in \mathbb{R}$, $0 \leq i \leq n-1$, $n\mathbf{u} < 1$, and $|\varepsilon_i| \leq \mathbf{u}$, there exists a $\theta_n \in \mathbb{R}$ such that

$$\prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1} = 1 + \theta_n, \text{ with } |\theta_n| \leq n\mathbf{u}/(1 - n\mathbf{u}).$$

We will show that if $\varepsilon_i \in \mathbb{R}$, $0 \leq i \leq n$, $(n+1)\mathbf{u} < 1$, and $|\varepsilon_i| \leq \mathbf{u}$, then there exists a $\theta_{n+1} \in \mathbb{R}$ such that

$$\prod_{i=0}^n (1 + \varepsilon_i)^{\pm 1} = 1 + \theta_{n+1}, \text{ with } |\theta_{n+1}| \leq (n+1)\mathbf{u}/(1 - (n+1)\mathbf{u}).$$

Case 1: $\prod_{i=0}^n (1 + \varepsilon_i)^{\pm 1} = \prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1} (1 + \varepsilon_n)$. See Exercise 10.15.

Case 2: $\prod_{i=0}^n (1 + \varepsilon_i)^{\pm 1} = (\prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1}) / (1 + \varepsilon_n)$. By the I.H. there exists a θ_n such that $(1 + \theta_n) = \prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1}$ and $|\theta_n| \leq n\mathbf{u}/(1 - n\mathbf{u})$. Then

$$\frac{\prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1}}{1 + \varepsilon_n} = \frac{1 + \theta_n}{1 + \varepsilon_n} = 1 + \underbrace{\frac{\theta_n - \varepsilon_n}{1 + \varepsilon_n}}_{\theta_{n+1}},$$

which tells us how to pick θ_{n+1} . Now

$$\begin{aligned}|\theta_{n+1}| &= \left| \frac{\theta_n - \varepsilon_n}{1 + \varepsilon_n} \right| \leq \frac{|\theta_n| + \mathbf{u}}{1 - \mathbf{u}} \leq \frac{\frac{n\mathbf{u}}{1 - n\mathbf{u}} + \mathbf{u}}{1 - \mathbf{u}} = \frac{n\mathbf{u} + (1 - n\mathbf{u})\mathbf{u}}{(1 - n\mathbf{u})(1 - \mathbf{u})} \\ &= \frac{(n+1)\mathbf{u} - n\mathbf{u}^2}{1 - (n+1)\mathbf{u} + n\mathbf{u}^2} \leq \frac{(n+1)\mathbf{u}}{1 - (n+1)\mathbf{u}}.\end{aligned}$$

By the Principle of Mathematical Induction, the result holds.

Homework 10.15 Complete the proof of Lemma 10.14.

☞ SEE ANSWER

The quantity θ_n will be used throughout this note. **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 $(1 + \varepsilon_i)$ and/or $(1 + \varepsilon_i)^{-1}$ that are grouped together to form $(1 + \theta_n)$.

Since the bound on $|\theta_n|$ occurs often, we assign it a symbol as follows:

Definition 10.16 For all $n \geq 1$ and $n\mathbf{u} < 1$, define $\gamma_n := n\mathbf{u}/(1 - n\mathbf{u})$.

With this notation, (10.3) simplifies to

$$\check{\kappa} = \chi_0\psi_0(1 + \theta_n) + \chi_1\psi_1(1 + \theta_n) + \cdots + \chi_{n-1}\psi_{n-1}(1 + \theta_2) \quad (10.4)$$

$$= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} (1 + \theta_n) & 0 & 0 & \cdots & 0 \\ 0 & (1 + \theta_n) & 0 & \cdots & 0 \\ 0 & 0 & (1 + \theta_{n-1}) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (1 + \theta_2) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix} \quad (10.5)$$

$$= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \left(I + \begin{pmatrix} \theta_n & 0 & 0 & \cdots & 0 \\ 0 & \theta_n & 0 & \cdots & 0 \\ 0 & 0 & \theta_{n-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \theta_2 \end{pmatrix} \right) \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix},$$

where $|\theta_j| \leq \gamma_j$, $j = 2, \dots, n$.

Two instances of the symbol θ_n , appearing even in the same expression, typically do not represent the same number. For example, in (10.4) 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.

As part of the analyses the following bounds will be useful to bound error that accumulates:

Lemma 10.17 If $n, b \geq 1$ then $\gamma_n \leq \gamma_{n+b}$ and $\gamma_n + \gamma_b + \gamma_n\gamma_b \leq \gamma_{n+b}$.

Homework 10.18 Prove Lemma 10.17.

☞ SEE ANSWER

10.5.4 Target result

It is of interest to accumulate the roundoff error encountered during computation as a perturbation of input and/or output parameters:

- $\check{\kappa} = (x + \delta x)^T y$; ($\check{\kappa}$ is the exact output for a slightly perturbed x)
- $\check{\kappa} = x^T (y + \delta y)$; ($\check{\kappa}$ is the exact output for a slightly perturbed y)
- $\check{\kappa} = x^T y + \delta \kappa$. ($\check{\kappa}$ equals the exact result plus an error)

The first two are backward error results (error is accumulated onto input parameters, showing that the algorithm is numerically stable since it yields the exact output for a slightly perturbed input) while the last one is a forward error result (error is accumulated onto the answer). We will see that in different situations, a different error result may be needed by analyses of operations that require a dot product.

Let us focus on the second result. Ideally one would show that each of the entries of y is slightly perturbed relative to that entry:

$$\delta y = \begin{pmatrix} \sigma_0 \psi_0 \\ \vdots \\ \sigma_{n-1} \psi_{n-1} \end{pmatrix} = \begin{pmatrix} \sigma_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{n-1} \end{pmatrix} \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \Sigma y,$$

where each σ_i is “small” and $\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{n-1})$. The following special structure of Σ , inspired by (10.5) will be used in the remainder of this note:

$$\Sigma^{(n)} = \begin{cases} 0 \times 0 \text{ matrix} & \text{if } n = 0 \\ \theta_1 & \text{if } n = 1 \\ \text{diag}(\theta_n, \theta_n, \theta_{n-1}, \dots, \theta_2) & \text{otherwise.} \end{cases} \quad (10.6)$$

Recall that θ_j is an order of magnitude variable with $|\theta_j| \leq \gamma_j$.

Homework 10.19 Let $k \geq 0$ and assume that $|\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}$, with $\varepsilon_1 = 0$ if $k = 0$. Show that

$$\left(\begin{array}{c|c} I + \Sigma^{(k)} & 0 \\ \hline 0 & (1 + \varepsilon_1) \end{array} \right) (1 + \varepsilon_2) = (I + \Sigma^{(k+1)}).$$

Hint: reason the case where $k = 0$ separately from the case where $k > 0$.

 [SEE ANSWER](#)

We state a theorem that captures how error is accumulated by the algorithm.

Theorem 10.20 Let $x, y \in \mathbb{R}^n$ and let $\kappa := x^T y$ be computed by executing the algorithm in Figure 10.2. Then

$$\check{\kappa} = [x^T y] = x^T (I + \Sigma^{(n)}) y.$$

10.5.5 A proof in traditional format

In the below proof, we will pick symbols for various (sub)vectors so that the proof can be easily related to the alternative framework to be presented in Section 10.5.6.

Proof: By Mathematical Induction on n , the length of vectors x and y .

Base case. $m(x) = m(y) = 0$. Trivial.

Inductive Step. I.H.: Assume that if $x_T, y_T \in \mathbb{R}^k$, $k > 0$, then

$$\text{fl}(x_T^T y_T) = x_T^T (I + \Sigma_T) y_T, \text{ where } \Sigma_T = \Sigma^{(k)}.$$

We will show that when $x_T, y_T \in \mathbb{R}^{k+1}$, the equality $\text{fl}(x_T^T y_T) = x_T^T (I + \Sigma_T) y_T$ holds *true* again.

Assume that $x_T, y_T \in \mathbb{R}^{k+1}$, and partition $x_T \rightarrow \begin{pmatrix} x_0 \\ \chi_1 \end{pmatrix}$ and $y_T \rightarrow \begin{pmatrix} y_0 \\ \Psi_1 \end{pmatrix}$. Then

$$\begin{aligned} \text{fl}\left(\begin{pmatrix} x_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} y_0 \\ \Psi_1 \end{pmatrix}\right) &= \text{fl}(\text{fl}(x_0^T y_0) + \text{fl}(\chi_1 \Psi_1)) && (\text{definition}) \\ &= \text{fl}(x_0^T (I + \Sigma_0) y_0 + \text{fl}(\chi_1 \Psi_1)) && (\text{I.H. with } x_T = x_0, \\ &&& y_T = y_0, \text{ and } \Sigma_0 = \Sigma^{(k)}) \\ &= (x_0^T (I + \Sigma_0) y_0 + \chi_1 \Psi_1 (1 + \varepsilon_*)) (1 + \varepsilon_+) && (\text{SCM, twice}) \\ &= \begin{pmatrix} x_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (I + \Sigma_0) & 0 \\ 0 & (1 + \varepsilon_*) \end{pmatrix} (1 + \varepsilon_+) \begin{pmatrix} y_0 \\ \Psi_1 \end{pmatrix} && (\text{rearrangement}) \\ &= x_T^T (I + \Sigma_T) y_T && (\text{renaming}), \end{aligned}$$

where $|\varepsilon_*|, |\varepsilon_+| \leq u$, $\varepsilon_+ = 0$ if $k = 0$, and $(I + \Sigma_T) = \begin{pmatrix} (I + \Sigma_0) & 0 \\ 0 & (1 + \varepsilon_*) \end{pmatrix} (1 + \varepsilon_+)$ so that $\Sigma_T = \Sigma^{(k+1)}$.

By the Principle of Mathematical Induction, the result holds.

10.5.6 A weapon of math induction for the war on (backward) error (optional)

We focus the reader's attention on Figure 10.3 in which we present a framework, which we will call the **error worksheet**, for presenting the inductive proof of Theorem 10.20 side-by-side with the algorithm for DOT. This framework, in a slightly different form, was first introduced in [3]. The expressions enclosed by {} (in the grey boxes) are predicates describing the state of the variables used in the algorithms and in their analysis. In the worksheet, we use superscripts to indicate the iteration number, thus, the symbols v^i and v^{i+1} do not denote two different variables, but two different states of variable v .

The proof presented in Figure 10.3 goes hand in hand with the algorithm, as it shows that before and after each iteration of the loop that computes $\kappa := x^T y$, the variables $\kappa, x_T, y_T, \Sigma_T$ are such that the predicate

$$\{\check{\kappa} = x_T^T (I + \Sigma_T) y_T \wedge k = m(x_T) \wedge \Sigma_T = \Sigma^{(k)}\} \quad (10.7)$$

	Error side	Step
$\kappa := 0$	$\{ \Sigma = 0 \}$	1a
Partition $x \rightarrow \begin{pmatrix} x_T \\ x_B \end{pmatrix}, y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix},$ where x_T and y_T are empty, and Σ_T is 0×0	$\Sigma \rightarrow \begin{pmatrix} \Sigma_T & 0 \\ 0 & \Sigma_B \end{pmatrix}$	4
	$\{ \check{\kappa} = x_T^T(I + \Sigma_T)y_T \wedge \Sigma_T = \Sigma^{(k)} \wedge m(x_T) = k \}$	2a
while $m(x_T) < m(x)$ do		3
	$\{ \check{\kappa} = x_T^T(I + \Sigma_T)y_T \wedge \Sigma_T = \Sigma^{(k)} \wedge m(x_T) = k \}$	2b
Repartition $\begin{pmatrix} x_T \\ x_B \end{pmatrix} \rightarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \psi_1 \\ y_2 \end{pmatrix}, \quad \begin{pmatrix} \Sigma_T & 0 \\ 0 & \Sigma_B \end{pmatrix} \rightarrow \begin{pmatrix} \Sigma_0^i & 0 & 0 \\ 0 & \sigma_1^i & 0 \\ 0 & 0 & \Sigma_2 \end{pmatrix}$ where χ_1, ψ_1 , and σ_1^i are scalars		
	$\{ \check{\kappa}^i = x_0^T(I + \Sigma_0^i)y_0 \wedge \Sigma_0^i = \Sigma^{(k)} \wedge m(x_0) = k \}$	5a
		6
$\kappa := \kappa + \chi_1 \psi_1$	$\begin{aligned} \check{\kappa}^{i+1} &= (\check{\kappa}^i + \chi_1 \psi_1 (1 + \varepsilon_*))(1 + \varepsilon_+) && \text{SCM, twice} \\ &= (x_0^T(I + \Sigma_0^{(k)})y_0 + \chi_1 \psi_1 (1 + \varepsilon_*))(1 + \varepsilon_+) && (\varepsilon_+ = 0 \text{ if } k = 0) \\ &= \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right)^T \left(\begin{array}{c c} I + \Sigma_0^{(k)} & 0 \\ \hline 0 & 1 + \varepsilon_* \end{array} \right) (1 + \varepsilon_+) \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right) && \text{Step 6: I.H.} \\ &= \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right)^T (I + \Sigma^{(k+1)}) \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right) && \text{Rearrange} \end{aligned}$	8
	$\left\{ \begin{aligned} \check{\kappa}^{i+1} &= \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right)^T \left(I + \left(\begin{array}{c c} \Sigma_0^{i+1} & 0 \\ \hline 0 & \sigma_1^{i+1} \end{array} \right) \right) \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right) \\ &\wedge \left(\begin{array}{c c} \Sigma_0^{i+1} & 0 \\ \hline 0 & \sigma_1^{i+1} \end{array} \right) = \Sigma^{(k+1)} \wedge m\left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right) = (k+1) \end{aligned} \right\}$	7
Continue with $\begin{pmatrix} x_T \\ x_B \end{pmatrix} \leftarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \psi_1 \\ y_2 \end{pmatrix}, \quad \begin{pmatrix} \Sigma_T & 0 \\ 0 & \Sigma_B \end{pmatrix} \leftarrow \begin{pmatrix} \Sigma_0^{i+1} & 0 & 0 \\ 0 & \sigma_1^{i+1} & 0 \\ 0 & 0 & \Sigma_2 \end{pmatrix}$		5b
	$\{ \check{\kappa} = x_T^T(I + \Sigma_T)y_T \wedge \Sigma_T = \Sigma^{(k)} \wedge m(x_T) = k \}$	2c
endwhile		
	$\{ \check{\kappa} = x_T^T(I + \Sigma_T)y_T \wedge \Sigma_T = \Sigma^{(k)} \wedge m(x_T) = k \wedge m(x_T) = m(x) \}$	2d
	$\{ \check{\kappa} = x^T(I + \Sigma^{(n)})y \wedge m(x) = n \}$	1b

Figure 10.3: Error worksheet completed to establish the backward error result for the given algorithm that computes the DOT operation.

holds *true*. This relation is satisfied at each iteration of the loop, so it is also satisfied when the loop completes. Upon completion, the loop guard is $m(x_T) = m(x) = n$, which implies that $\check{\kappa} = x^T(I + \Sigma^{(n)})y$, i.e., the thesis of the theorem, is satisfied too.

In details, the inductive proof of Theorem 10.20 is captured by the error worksheet as follows:

Base case. In Step 2a, i.e. before the execution of the loop, predicate (10.7) is satisfied, as $k = m(x_T) = 0$.

Inductive step. Assume that the predicate (10.7) holds *true* at Step 2b, i.e., at the top of the loop. Then Steps 6, 7, and 8 in Figure 10.3 prove that the predicate is satisfied again at Step 2c, i.e., the bottom of the loop. Specifically,

- Step 6 holds by virtue of the equalities $x_0 = x_T, y_0 = y_T$, and $\Sigma_0^i = \Sigma_T$.
- The update in Step 8-left introduces the error indicated in Step 8-right (SCM, twice), yielding the results for Σ_0^{i+1} and σ_1^{i+1} , leaving the variables in the state indicated in Step 7.
- Finally, the redefinition of Σ_T in Step 5b transforms the predicate in Step 7 into that of Step 2c, completing the inductive step.

By the **Principle of Mathematical Induction**, the predicate (10.7) holds for all iterations. In particular, when the loop terminates, the predicate becomes

$$\check{\kappa} = x^T(I + \Sigma^{(n)})y \wedge n = m(x_T).$$

This completes the discussion of the proof as captured by Figure 10.3.

In the derivation of algorithms, the concept of **loop-invariant** plays a central role. Let \mathcal{L} be a loop and P a predicate. If P is *true* before the execution of \mathcal{L} , at the beginning and at the end of each iteration of \mathcal{L} , and after the completion of \mathcal{L} , then predicate P is a *loop-invariant* with respect to \mathcal{L} . Similarly, we give the definition of **error-invariant**.

Definition 10.21 We call the predicate involving the operands and error operands in Steps 2a–d the **error-invariant** for the analysis. This predicate is *true* before and after each iteration of the loop.

For any algorithm, the loop-invariant and the error-invariant are related in that the former describes the status of the computation at the beginning and the end of each iteration, while the latter captures an error result for the computation indicated by the loop-invariant.

The reader will likely think that the error worksheet is an overkill when proving the error result for the dot product. We agree. However, it links a proof by induction to the execution of a loop, which we believe is useful. Elsewhere, as more complex operations are analyzed, the benefits of the structure that the error worksheet provides will become more obvious. (We will analyze more complex algorithms as the course proceeds.)

10.5.7 Results

A number of useful consequences of Theorem 10.20 follow. These will be used later as an inventory (library) of error results from which to draw when analyzing operations and algorithms that utilize DOT.

Corollary 10.22 Under the assumptions of Theorem 10.20 the following relations hold:

R1-B: (Backward analysis) $\check{\kappa} = (x + \delta x)^T y$, where $|\delta x| \leq \gamma_n |x|$, and $\check{\kappa} = x^T(y + \delta y)$, where $|\delta y| \leq \gamma_n |y|$;

R1-F: (Forward analysis) $\kappa = x^T y + \delta\kappa$, where $|\delta\kappa| \leq \gamma_n |x|^T |y|$.

Proof: We leave the proof of R1-B as an exercise. For R1-F, let $\delta\kappa = x^T \Sigma^{(n)} y$, where $\Sigma^{(n)}$ is as in Theorem 10.20. Then

$$\begin{aligned} |\delta\kappa| &= |x^T \Sigma^{(n)} y| \\ &\leq |\chi_0| |\theta_n| |\psi_0| + |\chi_1| |\theta_n| |\psi_1| + \cdots + |\chi_{n-1}| |\theta_2| |\psi_{n-1}| \\ &\leq \gamma_n |\chi_0| |\psi_0| + \gamma_n |\chi_1| |\psi_1| + \cdots + \gamma_2 |\chi_{n-1}| |\psi_{n-1}| \\ &\leq \gamma_n |x|^T |y|. \end{aligned}$$

Homework 10.23 Prove R1-B.

☞ SEE ANSWER

10.6 Stability of a Matrix-Vector Multiplication Algorithm

In this section, we discuss the numerical stability of the specific matrix-vector multiplication algorithm that computes $y := Ax$ via dot products. This allows us to show how results for the dot product can be used in the setting of a more complicated algorithm.

10.6.1 An algorithm for computing GEMV

We will consider the algorithm given in Figure 10.4 for computing $y := Ax$, which computes y via dot products.

10.6.2 Analysis

Assume $A \in \mathbb{R}^{m \times n}$ and partition

$$A = \begin{pmatrix} a_0^T \\ a_1^T \\ \vdots \\ a_{m-1}^T \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix}.$$

Then

$$\begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} := \begin{pmatrix} a_0^T x \\ a_1^T x \\ \vdots \\ a_{m-1}^T x \end{pmatrix}.$$

Algorithm: GEMV: $y := Ax$ <hr/> $A \rightarrow \begin{pmatrix} A_T \\ A_B \end{pmatrix}, y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix}$ <p>while $m(A_T) < m(A)$ do</p> $\begin{pmatrix} A_T \\ A_B \end{pmatrix} \rightarrow \begin{pmatrix} A_0 \\ a_1^T \\ A_2 \end{pmatrix}, \begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ y_2 \end{pmatrix}$ <hr/> $\Psi_1 := a_1^T x$ <hr/> $\begin{pmatrix} x_T \\ x_B \end{pmatrix} \leftarrow \begin{pmatrix} x_0 \\ \chi_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ y_2 \end{pmatrix}$ <p>endwhile</p>
--

Figure 10.4: Algorithm for computing $y := Ax$.

From Corollary 10.22 R1-B regarding the dot product we know that

$$\check{y} = \begin{pmatrix} \check{\Psi}_0 \\ \check{\Psi}_1 \\ \vdots \\ \check{\Psi}_{m-1} \end{pmatrix} = \begin{pmatrix} (a_0 + \delta a_0)^T x \\ (a_1 + \delta a_1)^T x \\ \vdots \\ (a_{m-1} + \delta a_{m-1})^T x \end{pmatrix} = \left(\begin{pmatrix} a_0^T \\ a_1^T \\ \vdots \\ a_{m-1}^T \end{pmatrix} + \begin{pmatrix} \delta a_0^T \\ \delta a_1^T \\ \vdots \\ \delta a_{m-1}^T \end{pmatrix} \right) x = (A + \Delta A)x,$$

where $|\delta a_i| \leq \gamma_n |a_i|$, $i = 0, \dots, m-1$, and hence $|\Delta A| \leq \gamma_n |A|$.

Also, from Corollary 10.22 R1-F regarding the dot product we know that

$$\check{y} = \begin{pmatrix} \check{\Psi}_0 \\ \check{\Psi}_1 \\ \vdots \\ \check{\Psi}_{m-1} \end{pmatrix} = \begin{pmatrix} a_0^T x + \delta \psi_0 \\ a_1^T x + \delta \psi_1 \\ \vdots \\ a_{m-1}^T x + \delta \psi_{m-1} \end{pmatrix} = \begin{pmatrix} a_0^T \\ a_1^T \\ \vdots \\ a_{m-1}^T \end{pmatrix} x + \begin{pmatrix} \delta \psi_0 \\ \delta \psi_1 \\ \vdots \\ \delta \psi_{m-1} \end{pmatrix} = Ax + \delta y.$$

where $|\delta \psi_i| \leq \gamma_n |a_i|^T |x|$ and hence $|\delta y| \leq \gamma_n |A| |x|$.

The above observations can be summarized in the following theorem:

Theorem 10.24 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 the algorithm in Figure 10.4. Then these equalities hold:*

Algorithm: GEMM: $C := AB$
$C \rightarrow \left(\begin{array}{c c} C_L & C_R \end{array} \right), B \rightarrow \left(\begin{array}{c c} B_L & B_R \end{array} \right)$ while $n(C_L) < n(C)$ do $\left(\begin{array}{c c} C_L & C_R \end{array} \right) \rightarrow \left(\begin{array}{c c c} C_0 & c_1 & C_2 \end{array} \right), \left(\begin{array}{c c} B_L & B_R \end{array} \right) \rightarrow \left(\begin{array}{c c c} B_0 & b_1 & B_2 \end{array} \right)$
$c_1 := Ab_1$
$\left(\begin{array}{c c} C_L & C_R \end{array} \right) \leftarrow \left(\begin{array}{c c c} C_0 & c_1 & C_2 \end{array} \right), \left(\begin{array}{c c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c c c} B_0 & b_1 & B_2 \end{array} \right)$ endwhile

Figure 10.5: Algorithm for computing $C := AB$ one column at a time.

R1-B: $\check{y} = (A + \Delta A)x$, where $|\Delta A| \leq \gamma_n |A|$.

R2-F: $\check{y} = Ax + \delta y$, where $|\delta y| \leq \gamma_n |A| |x|$.

Homework 10.25 In the above theorem, could one instead prove the result

$$\check{y} = A(x + \delta x),$$

where δx is “small”?

☞ SEE ANSWER

10.7 Stability of a Matrix-Matrix Multiplication Algorithm

In this section, we discuss the numerical stability of the specific matrix-matrix multiplication algorithm that computes $C := AB$ via the matrix-vector multiplication algorithm from the last section, where $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$.

10.7.1 An algorithm for computing GEMM

We will consider the algorithm given in Figure 10.5 for computing $C := AC$, which computes one column at a time so that the matrix-vector multiplication algorithm from the last section can be used.

10.7.2 Analysis

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 Corollary 10.22 R1-B regarding the dot product we know that

$$\begin{aligned} \left(\begin{array}{c|c|c|c} \check{c}_0 & \check{c}_1 & \cdots & \check{c}_{n-1} \end{array} \right) &= \left(\begin{array}{c|c|c|c} Ab_0 + \delta c_0 & Ab_1 + \delta c_1 & \cdots & Ab_{n-1} + \delta c_{n-1} \end{array} \right) \\ &= \left(\begin{array}{c|c|c|c} Ab_0 & Ab_1 & \cdots & Ab_{n-1} \end{array} \right) + \left(\begin{array}{c|c|c|c} \delta c_0 & \delta c_1 & \cdots & \delta c_{n-1} \end{array} \right) \\ &= AB + \Delta C. \end{aligned}$$

where $|\delta c_j| \leq \gamma_k |A| |b_j|$, $j = 0, \dots, n-1$, and hence $|\Delta C| \leq \gamma_k |A| |B|$.

The above observations can be summarized in the following theorem:

Theorem 10.26 (Forward) error results 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 the algorithm in Figure 10.5. Then the following equality holds:

R1-F: $\check{C} = AB + \Delta C$, where $|\Delta C| \leq \gamma_k |A| |B|$.

Homework 10.27 In the above theorem, could one instead prove the result

$$\check{C} = (A + \Delta A)(B + \Delta B),$$

where ΔA and ΔB are “small”?

☞ SEE ANSWER

10.7.3 An application

A collaborator of ours recently implemented a matrix-matrix multiplication algorithm and wanted to check if it gave the correct answer. To do so, he followed the following steps:

- He created random matrices $A \in \mathbb{R}^{m \times k}$, and $C \in \mathbb{R}^{m \times n}$, with positive entries in the range $(0, 1)$.
- He computed $C = AB$ with an implementation that was known to be “correct” and assumed it yields the exact solution. (Of course, it has error in it as well. We discuss how he compensated for that, below.)
- He computed $\check{C} = AB$ with his new implementation.
- He computed $\Delta C = \check{C} - C$ and checked that each of its entries satisfied $\delta y_{i,j} \leq 2ku\gamma_{i,j}$.
- In the above, he took advantage of the fact that A and B had positive entries so that $|A||B| = AB = C$. He also approximated $\gamma_k = \frac{ku}{1-ku}$ with ku , and introduced the factor 2 to compensate for the fact that C itself was inexactly computed.

10.8 Wrapup

10.8.1 Additional exercises

10.8.2 Summary

Notes on Performance

How to attain high performance on modern architectures is of importance: linear algebra is fundamental to scientific computing. Scientific computing often involves very large problems that require the fastest computers in the world to be employed. One wants to use such computers efficiently.

For now, we suggest the reader become familiar with the following resources:

- Week 5 of [Linear Algebra: Foundations to Frontiers - Notes to LAFF With \[30\]](#).
Focus on Section 5.4 Enrichment.
- Kazushige Goto and Robert van de Geijn.
[Anatomy of high-performance matrix multiplication \[23\]](#).
[ACM Transactions on Mathematical Software, 34 \(3\), 2008](#).
- Field G. Van Zee and Robert van de Geijn.
[BLIS: A Framework for Rapid Instantiation of BLAS Functionality \[43\]](#).
[ACM Transactions on Mathematical Software, to appear](#).
- Robert van de Geijn.
[How to Optimize Gemm](#).
wiki.cs.utexas.edu/rvdg/HowToOptimizeGemm.
(An exercise on how to write a high-performance matrix-matrix multiplication in C.)

A similar exercise:

Michael Lehn

GEMM: From Pure C to SSE Optimized Micro Kernels

<http://apfel.mathematik.uni-ulm.de/~lehnsghpc/gemm/index.html>.

Chapter 12

Notes on Gaussian Elimination and LU Factorization

12.1 Opening Remarks

12.1.1 Launch

The LU factorization is also known as the LU decomposition and the operations it performs are equivalent to those performed by Gaussian elimination. For details, we recommend that the reader consult Weeks 6 and 7 of

“Linear Algebra: Foundations to Frontiers - Notes to LAFF With” [30].

Video

Read disclaimer regarding the videos in the preface!

Lecture on Gaussian Elimination and LU factorization:

- 👉 [YouTube](#)
- 👉 [Download from UT Box](#)
- 👉 [View After Local Download](#)

Lecture on deriving dense linear algebra algorithms:

- 👉 [YouTube](#)
- 👉 [Download from UT Box](#)
- 👉 [View After Local Download](#)
- 👉 [Slides](#)

(For help on viewing, see Appendix A.)

12.1.2 Outline

12.1	Opening Remarks	213
12.1.1	Launch	213
12.1.2	Outline	214
12.1.3	What you will learn	216
12.2	Definition and Existence	217
12.3	LU Factorization	217
12.3.1	First derivation	217
12.3.2	Gauss transforms	218
12.3.3	Cost of LU factorization	220
12.4	LU Factorization with Partial Pivoting	221
12.4.1	Permutation matrices	221
12.4.2	The algorithm	223
12.5	Proof of Theorem 12.3	228
12.6	LU with Complete Pivoting	230
12.7	Solving $Ax = y$ Via the LU Factorization with Pivoting	231
12.8	Solving Triangular Systems of Equations	231
12.8.1	$Lz = y$	231
12.8.2	$Ux = z$	234
12.9	Other LU Factorization Algorithms	234
12.9.1	Variant 1: Bordered algorithm	239
12.9.2	Variant 2: Left-looking algorithm	239
12.9.3	Variant 3: Up-looking variant	240
12.9.4	Variant 4: Crout variant	240
12.9.5	Variant 5: Classical LU factorization	241
12.9.6	All algorithms	241
12.9.7	Formal derivation of algorithms	241
12.10	Numerical Stability Results	243
12.11	Is LU with Partial Pivoting Stable?	244
12.12	Blocked Algorithms	244
12.12.1	Blocked classical LU factorization (Variant 5)	244
12.12.2	Blocked classical LU factorization with pivoting (Variant 5)	247
12.13	Variations on a Triple-Nested Loop	248
12.14	Inverting a Matrix	249
12.14.1	Basic observations	249
12.14.2	Via the LU factorization with pivoting	249

12.14.3	Gauss-Jordan inversion	250
12.14.4	(Almost) never, ever invert a matrix	251
12.15	Efficient Condition Number Estimation	252
12.15.1	The problem	252
12.15.2	Insights	252
12.15.3	A simple approach	253
12.15.4	Discussion	255
12.16	Wrapup	256
12.16.1	Additional exercises	256
12.16.2	Summary	256

12.1.3 What you will learn

12.2 Definition and Existence

Definition 12.1 LU factorization (decomposition) Given a matrix $A \in \mathbb{C}^{m \times n}$ with $m \leq n$ 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.

The first question we will ask is when the LU factorization exists. For this, we need a definition.

Definition 12.2 The $k \times k$ principle leading submatrix of a matrix A is defined to be the square matrix $A_{TL} \in \mathbb{C}^{k \times k}$ such that $A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$.

This definition allows us to indicate when a matrix has an LU factorization:

Theorem 12.3 Existence Let $A \in \mathbb{C}^{m \times n}$ and $m \leq n$ have linearly independent columns. Then A has a unique LU factorization if and only if all its principle leading submatrices are nonsingular.

The proof of this theorem is a bit involved and can be found in Section 12.5.

12.3 LU Factorization

We are going to present two different ways of deriving the most commonly known algorithm. The first is a straight forward derivation. The second presents the operation as the application of a sequence of Gauss transforms.

12.3.1 First derivation

Partition A , L , and U as follows:

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right), \quad L \rightarrow \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right), \quad \text{and} \quad U \rightarrow \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right).$$

Then $A = LU$ means that

$$\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right) \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right) = \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline l_{21}v_{11} & l_{21}u_{12}^T + L_{22}U_{22} \end{array} \right).$$

This means that

$$\begin{array}{c|c} \alpha_{11} = v_{11} & a_{12}^T = u_{12}^T \\ \hline a_{21} = v_{11}l_{21} & A_{22} = l_{21}u_{12}^T + L_{22}U_{22} \end{array}$$

or, equivalently,

$$\begin{array}{c|c} \alpha_{11} = v_{11} & a_{12}^T = u_{12}^T \\ \hline a_{21} = v_{11}l_{21} & A_{22} - l_{21}u_{12}^T = L_{22}U_{22} \end{array}.$$

If we let U overwrite the original matrix A this suggests the algorithm

- $l_{21} = a_{21}/\alpha_{11}$.
- $a_{21} = 0$.
- $A_{22} := A_{22} - \textcolor{blue}{l}_{21}a_{12}^T$.
- Continue by overwriting the updated A_{22} with its LU factorization.

This is captured in the algorithm in Figure 12.1.

12.3.2 Gauss transforms

Definition 12.4 A matrix L_k of the form $L_k = \left(\begin{array}{c|cc|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & l_{21} & 0 \end{array} \right)$ where I_k is $k \times k$ is called a Gauss transform.

Example 12.5 Gauss transforms can be used to take multiples of a row and subtract these multiples from other rows:

$$\left(\begin{array}{c|cc|cc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & -\lambda_{21} & 1 & 0 \\ 0 & -\lambda_{31} & 0 & 1 \end{array} \right) \left(\begin{array}{c} \hat{a}_0^T \\ \hat{a}_1^T \\ \hat{a}_2^T \\ \hat{a}_3^T \end{array} \right) = \left(\begin{array}{c} \hat{a}_0^T \\ \hline \hat{a}_1^T \\ \hline \left(\begin{array}{c} \hat{a}_2^T \\ \hat{a}_3^T \end{array} \right) - \left(\begin{array}{c} \lambda_{21} \\ \lambda_{31} \end{array} \right) \hat{a}_1^T \end{array} \right) = \left(\begin{array}{c} \hat{a}_0^T \\ \hline \hat{a}_1^T \\ \hline \hat{a}_2^T - \lambda_{21}\hat{a}_1^T \\ \hat{a}_3^T - \lambda_{31}\hat{a}_1^T \end{array} \right).$$

Notice the similarity with what one does in Gaussian Elimination: take a multiples of one row and subtract these from other rows.

Now assume that the LU factorization in the previous subsection has proceeded to where A contains

$$\left(\begin{array}{c|cc|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right)$$

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: $\left(\begin{array}{c|c} a_{21} & A_{22} \end{array} \right)$. The vehicle is a Gauss transform: we must determine l_{21} so that

$$\left(\begin{array}{c|cc|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|cc|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|cc|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right).$$

Algorithm: Compute LU factorization of A , overwriting L with factor L and A with factor U

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$, $L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right)$

where A_{TL} and L_{TL} are 0×0

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

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \quad \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right)$$

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

$$\begin{cases} l_{21} := a_{21}/\alpha_{11} \\ A_{22} := A_{22} - l_{21}a_{12}^T \\ (a_{21} := 0) \end{cases}$$

or, alternatively,

$$\begin{cases} l_{21} := a_{21}/\alpha_{11} \\ \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & 0 \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) \\ = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right) \end{cases}$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \quad \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right)$$

endwhile

Figure 12.1: Most commonly known algorithm for overwriting a matrix with its LU factorization.

This means we must pick $l_{21} = a_{21}/\alpha_{11}$ since

$$\left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} - \alpha_{11}l_{21} & A_{22} - l_{21}a_{12}^T \end{array} \right).$$

The resulting algorithm is summarized in Figure 12.1 under “or, alternatively.”. Notice that this algorithm is identical to the algorithm for computing LU factorization discussed before!

How can this be? The following set of exercises explains it.

Homework 12.6 Show that

$$\left(\begin{array}{c|c|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array} \right)^{-1} = \left(\begin{array}{c|c|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & l_{21} & I \end{array} \right)$$

[SEE ANSWER](#)

Now, clearly, what the algorithm does is to compute a sequence of n Gauss transforms $\widehat{L}_0, \dots, \widehat{L}_{n-1}$ such that $\widehat{L}_{n-1}\widehat{L}_{n-2}\cdots\widehat{L}_1\widehat{L}_0A = U$. Or, equivalently, $A = L_0L_1\cdots L_{n-2}L_{n-1}U$, where $L_k = \widehat{L}_k^{-1}$. What will show next is that $L = L_0L_1\cdots L_{n-2}L_{n-1}$ is the unit lower triangular matrix computed by the LU factorization.

Homework 12.7 Let $\tilde{L}_k = L_0L_1\dots L_k$. Assume that \tilde{L}_k has the form $\tilde{L}_{k-1} = \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & 0 & I \end{array} \right)$, where \tilde{L}_{00} is $k \times k$. Show that \tilde{L}_k is given by $\tilde{L}_k = \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & l_{21} & I \end{array} \right)$.. (Recall: $\widehat{L}_k = \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array} \right)$.)

[SEE ANSWER](#)

What this exercise shows is that $L = L_0L_1\cdots L_{n-2}L_{n-1}$ is the triangular matrix that is created by simply placing the computed vectors l_{21} below the diagonal of a unit lower triangular matrix.

12.3.3 Cost of LU factorization

The cost of the LU factorization algorithm given in Figure 12.1 can be analyzed as follows:

- Assume A is $n \times n$.
- During the k th iteration, A_{TL} is initially $k \times k$.

- Computing $l_{21} := a_{21}/\alpha_{11}$ is typically implemented as $\beta := 1/\alpha_{11}$ and then the scaling $l_{21} := \beta a_{21}$. The reason is that divisions are expensive relative to multiplications. We will ignore the cost of the division (which will be insignificant if n is large). Thus, we count this as $n - k - 1$ multiplies.
- The rank-1 update of A_{22} requires $(n - k - 1)^2$ multiplications and $(n - k - 1)^2$ additions.
- Thus, the total cost (in flops) can be approximated by

$$\begin{aligned}
 \sum_{k=0}^{n-1} [(n-k-1) + 2(n-k-1)^2] &= \sum_{j=0}^{n-1} [j + 2j^2] \quad (\text{Change of variable: } j = n - k - 1) \\
 &= \sum_{j=0}^{n-1} j + 2 \sum_{j=0}^{n-1} j^2 \\
 &\approx \frac{n(n-1)}{2} + 2 \int_0^n x^2 dx \\
 &= \frac{n(n-1)}{2} + \frac{2}{3}n^3 \\
 &\approx \frac{2}{3}n^3
 \end{aligned}$$

Notice that this involves roughly half the number of floating point operations as are required for a Householder transformation based QR factorization.

12.4 LU Factorization with Partial Pivoting

It is well-known that the LU factorization is numerically unstable under general circumstances. In particular, a backward stability analysis, given for example in [3, 9, 7] and summarized in Section 12.10, shows that the computed matrices \check{L} and \check{U} satisfy

$$(A + \Delta A) = \check{L} \check{U} \quad \text{where } |\Delta A| \leq \gamma_n |\check{L}| |\check{U}|.$$

(This is the backward error result for the Crout variant for LU factorization, discussed later in this note. Some of the other variants have an error result of $(A + \Delta A) = \check{L} \check{U}$ where $|\Delta A| \leq \gamma_n (|A| + |\check{L}| |\check{U}|)$.) Now, if α_{11} is small in magnitude compared to the entries of a_{21} then not only will l_{21} have large entries, but the update $A_{22} - l_{21}a_{12}^T$ will potentially introduce large entries in the updated A_{22} (in other words, the part of matrix A from which the future matrix U will be computed), a phenomenon referred to as *element growth*. To overcome this, we take will swap rows in A as the factorization proceeds, resulting in an algorithm known as LU factorization with partial pivoting.

12.4.1 Permutation matrices

Definition 12.8 An $n \times n$ matrix P is said to be a *permutation matrix*, or *permutation*, if, when applied to a vector $x = (\chi_0, \chi_1, \dots, \chi_{n-1})^T$, it merely rearranges the order of the elements in that vector. Such a permutation can be represented by the vector of integers, $(\pi_0, \pi_1, \dots, \pi_{n-1})^T$, where $\{\pi_0, \pi_1, \dots, \pi_{n-1}\}$ is a permutation of the integers $\{0, 1, \dots, n-1\}$ and the permuted vector Px is given by $(\chi_{\pi_0}, \chi_{\pi_1}, \dots, \chi_{\pi_{n-1}})^T$.

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

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

$$L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right), p \rightarrow \left(\begin{array}{c} p_T \\ p_B \end{array} \right).$$

where A_{TL} and L_{TL} are 0×0 and p_T is 0×1

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

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \pi_1 \\ p_2 \end{array} \right)$$

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

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

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

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

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

$$(a_{21} := 0)$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ p_B \end{array} \right) \leftarrow \left(\begin{array}{c} p_0 \\ \pi_1 \\ p_2 \end{array} \right)$$

endwhile

Figure 12.2: LU factorization with partial pivoting.

If P is a permutation matrix then PA rearranges the rows of A exactly as the elements of x are rearranged by Px .

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. We will denote that matrix by

$$P(\pi) = \begin{cases} I_n & \text{if } \pi = 0 \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & I_{\pi-1} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{n-\pi-1} \end{pmatrix} & \text{otherwise,} \end{cases}$$

where n is the dimension of the permutation matrix. In the following we will use the notation P_n to indicate that the matrix P is of size n . Let p be a vector of integers satisfying the conditions

$$p = (\pi_0, \dots, \pi_{k-1})^T, \text{ where } 1 \leq k \leq n \text{ and } 0 \leq \pi_i < n - i, \quad (12.1)$$

then $P_n(p)$ will denote the permutation:

$$P_n(p) = \begin{pmatrix} I_{k-1} & 0 \\ 0 & P_{n-k+1}(\pi_{k-1}) \end{pmatrix} \begin{pmatrix} I_{k-2} & 0 \\ 0 & P_{n-k+2}(\pi_{k-2}) \end{pmatrix} \dots \begin{pmatrix} 1 & 0 \\ 0 & P_{n-1}(\pi_1) \end{pmatrix} P_n(\pi_0).$$

Remark 12.9 In the algorithms, the subscript that indicates the matrix dimensions is omitted.

Example 12.10 Let $a_0^T, a_1^T, \dots, a_{n-1}^T$ be the rows of a matrix A . The application of $P(p)$ to A yields a matrix that results from swapping row a_0^T with $a_{\pi_0}^T$, then swapping a_1^T with $a_{\pi_1+1}^T$, a_2^T with $a_{\pi_2+2}^T$, until finally a_{k-1}^T is swapped with $a_{\pi_{k-1}+k-1}^T$.

Remark 12.11 For those familiar with how pivot information is stored in LINPACK and LAPACK, notice that those packages store the vector of pivot information $(\pi_0 + 1, \pi_1 + 2, \dots, \pi_{k-1} + k)^T$.

12.4.2 The algorithm

Having introduced our notation for permutation matrices, we can now define the LU factorization with partial pivoting: Given an $n \times n$ matrix A , we wish to compute a) a vector p of n integers which satisfies the conditions (12.1), b) a unit lower trapezoidal matrix L , and c) an upper triangular matrix U so that $P(p)A = LU$. An algorithm for computing this operation is typically represented by

$$[A, p] := \text{LUpivA},$$

where upon completion A has been overwritten by $\{L \setminus U\}$.

Let us start with revisiting the first derivation of the LU factorization. The first step is to find a first permutation matrix $P(\pi_1)$ such that the element on the diagonal in the first column is maximal in value. 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:

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right), \quad \text{and} \quad L \rightarrow \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right).$$

- Compute $\pi_1 = \max_i \left(\frac{\alpha_{11}}{a_{21}} \right)$.
- Permute the rows: $\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) := P(\pi_1) \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right)$.
- Compute $l_{21} := a_{21}/\alpha_{11}$.
- Update $A_{22} := A_{22} - l_{21}a_{12}^T$.

Now, in general, assume that the computation has proceeded to the point where matrix A has been overwritten by

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right)$$

where A_{00} is upper triangular. If no pivoting was added one would compute $l_{21} := a_{21}/\alpha_{11}$ followed by the update

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right).$$

Now, instead one performs the steps

- Compute $\pi_1 = \max_i \left(\frac{\alpha_{11}}{a_{21}} \right)$.
- Permute the rows: $\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right)$
- Compute $l_{21} := a_{21}/\alpha_{11}$.
- Update

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right).$$

This algorithm is summarized in Figure 12.2.

Now, what this algorithm computes is a sequence of Gauss transforms $\widehat{L}_0, \dots, \widehat{L}_{n-1}$ and permutations P_0, \dots, P_{n-1} such that

$$\widehat{L}_{n-1}P_{n-1} \cdots \widehat{L}_0P_0A = U$$

or, equivalently,

$$A = P_0^T L_0 \cdots \widehat{P}_{n-1}^T \widehat{L}_{n-1} U,$$

where $L_k = \widehat{L}_k^{-1}$. What we will finally show is that there are Gauss transforms $\bar{L}_0, \dots, \bar{L}_{n-1}$ (here the “bar” does NOT mean conjugation. It is just a symbol) such that

$$A = P_0^T \cdots P_{n-1}^T \underbrace{\bar{L}_0 \cdots \bar{L}_{n-1}}_L U$$

or, equivalently,

$$P(p)A = P_{n-1} \cdots P_0 A = \underbrace{\bar{L}_0 \cdots \bar{L}_{n-1}}_L U,$$

which is what we set out to compute.

Here is the insight. Assume that after k steps of LU factorization we have computed p_T, L_{TL}, L_{BL} , etc. so that

$$P(p_T)A = \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & I \end{array} \right) \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right),$$

where A_{TL} is upper triangular and $k \times k$.

Now compute the next step of LU factorization with partial pivoting with $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right)$:

- Partition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{01} & A_{02} \end{array} \right)$$

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

- Permute $\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right)$

- Compute $l_{21} := a_{21}/\alpha_{11}$.

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

Partition $A \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix}$, $L \rightarrow \begin{pmatrix} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{pmatrix}$, $p \rightarrow \begin{pmatrix} p_T \\ p_B \end{pmatrix}$.

where A_{TL} and L_{TL} are 0×0 and p_T is 0×1

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

Repartition

$$\begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{pmatrix}, \begin{pmatrix} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{pmatrix}, \begin{pmatrix} p_T \\ p_B \end{pmatrix} \rightarrow \begin{pmatrix} p_0 \\ \pi_1 \\ p_2 \end{pmatrix}$$

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

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

$$\begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline l_{10}^T & \alpha_{11} & a_{12}^T \\ \hline L_{20} & a_{21} & A_{22} \end{pmatrix} := \begin{pmatrix} I & 0 \\ \hline 0 & P(\pi_1) \end{pmatrix} \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline l_{10}^T & \alpha_{11} & a_{12}^T \\ \hline L_{20} & a_{21} & A_{22} \end{pmatrix}$$

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

$$\begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{pmatrix} := \begin{pmatrix} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & 0 \end{pmatrix} \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{pmatrix}$$

Continue with

$$\begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{pmatrix}, \begin{pmatrix} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{pmatrix}, \begin{pmatrix} p_T \\ p_B \end{pmatrix} \leftarrow \begin{pmatrix} p_0 \\ \pi_1 \\ p_2 \end{pmatrix}$$

endwhile

Figure 12.3: LU factorization with partial pivoting.

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

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

where A_{TL} is 0×0 and p_T is 0×1

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

Repartition

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

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

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

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

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

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

Continue with

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

endwhile

Figure 12.4: LU factorization with partial pivoting, overwriting A with the factors.

- Update

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right)$$

After this,

$$P(p_T)A = \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & I \end{array} \right) \left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right) \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & l_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right) \quad (12.2)$$

But

$$\begin{aligned} & \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & I \end{array} \right) \left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right) \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & l_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right) \\ &= \left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right) \left(\begin{array}{c|c|c} L_{TL} & 0 \\ \hline P(\pi_1)L_{BL} & I \end{array} \right) \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & l_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right) \\ &= \left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right) \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline \bar{L}_{BL} & I \end{array} \right) \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & l_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right), \end{aligned}$$

NOTE: There is a “bar” above some of L ’s and l ’s. Very hard to see!!! For this reason, these show up in red as well. Here we use the fact that $P(\pi_1) = P(\pi_1)^T$ because of its very special structure.

Bringing the permutation to the left of (12.2) and “repartitioning” we get

$$\underbrace{\left(\begin{array}{c|c} I & 0 \\ \hline 0 & P(\pi_1) \end{array} \right)}_{P\left(\frac{p_0}{\pi_1}\right)} P(\textcolor{blue}{p}_0) A = \underbrace{\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \bar{L}_{10}^T & 1 & 0 \\ \hline \bar{L}_{20} & 0 & I \end{array} \right)}_{\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \bar{L}_{10}^T & 1 & 0 \\ \hline \bar{L}_{20} & l_{21} & I \end{array} \right)} \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & l_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline 0 & \alpha_{11} & a_{12}^T \\ \hline 0 & 0 & A_{22} - l_{21}a_{12}^T \end{array} \right).$$

This explains how the algorithm in Figure 12.3 compute p , L , and U (overwriting A with U) so that $P(p)A = LU$.

Finally, we recognize that L can overwrite the entries of A below its diagonal, yielding the algorithm in Figure 12.4.

12.5 Proof of Theorem 12.3

Proof:

(\Rightarrow) Let nonsingular A have a (unique) LU factorization. We will show that its principle leading subma-

trices are nonsingular. Let

$$\underbrace{\begin{pmatrix} A_{TL} & | & A_{TR} \\ \hline A_{BL} & | & A_{BR} \end{pmatrix}}_A = \underbrace{\begin{pmatrix} L_{TL} & | & 0 \\ \hline L_{BL} & | & L_{BR} \end{pmatrix}}_L \underbrace{\begin{pmatrix} U_{TL} & | & U_{TR} \\ \hline 0 & | & U_{BR} \end{pmatrix}}_U$$

be the LU factorization of A where A_{TL} , L_{TL} , and U_{TL} are $k \times k$. Notice that U cannot have a zero on the diagonal since then A would not have linearly independent columns. Now, the $k \times k$ principle 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 principle 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 principle 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 .

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 principle leading submatrices. Now, if an LU factorization of A exists, $A = LU$, then it would have to form

$$\underbrace{\begin{pmatrix} A_{00} & | & a_{01} \\ \hline a_{10}^T & | & \alpha_{11} \\ \hline A_{20} & | & a_{21} \end{pmatrix}}_A = \underbrace{\begin{pmatrix} L_{00} & | & 0 \\ \hline l_{10}^T & | & 1 \\ \hline L_{20} & | & l_{21} \end{pmatrix}}_L \underbrace{\begin{pmatrix} U_{00} & | & u_{01} \\ \hline 0 & | & v_{11} \end{pmatrix}}_U. \quad (12.3)$$

If we can show that the different parts of L and U exist and are unique, we are done. Equation (12.3) can be rewritten as

$$\begin{pmatrix} A_{00} \\ a_{10}^T \\ A_{20} \end{pmatrix} = \begin{pmatrix} L_{00} \\ l_{10}^T \\ L_{20} \end{pmatrix} U_{00} \quad \text{and} \quad \begin{pmatrix} a_{01} \\ \alpha_{11} \\ a_{21} \end{pmatrix} = \begin{pmatrix} L_{00}u_{01} \\ l_{10}^Tu_{01} + v_{11} \\ L_{20}u_{01} + l_{21}v_{11} \end{pmatrix}.$$

Now, by the Induction Hypothesis L_{11} , l_{10}^T , and L_{20} exist and are unique. So the question is whether u_{01} , v_{11} , and l_{21} exist and are unique:

- u_{01} exists and is unique. Since L_{00} is nonsingular (it has ones on its diagonal) $L_{00}u_{01} = a_{01}$ has a solution that is unique.
- v_{11} exists, is unique, and is nonzero. Since l_{10}^T and u_{01} exist and are unique, $v_{11} = \alpha_{11} - l_{10}^T u_{01}$ exists and is unique. It is also nonzero since the principle leading submatrix of A given by

$$\left(\begin{array}{c|c} A_{00} & a_{01} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & v_{11} \end{array} \right),$$

is nonsingular by assumption and therefore v_{11} must be nonzero.

- l_{21} exists and is unique. Since v_{11} exists and is nonzero, $l_{21} = a_{21}/v_{11}$ exists and is uniquely determined.

Thus the $m \times (k+1)$ matrix A has a unique LU factorization.

By the Principle of Mathematical Induction the result holds.

Homework 12.12 Implement LU factorization with partial pivoting with the FLAME@lab API, in M-script.

SEE ANSWER

12.6 LU 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

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

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_0, ρ_0) . Then, one permutes

$$\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) := P(\pi_0) \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) P(\rho_0)^T,$$

making α_{11} the largest element in magnitude. This then reduces the magnitude of multipliers and element growth.

It can be shown that the maximal element growth experienced when employing LU with complete pivoting indeed reduces element growth. The problem is that it requires $O(n^2)$ comparisons per iteration. Worse, it completely destroys the ability to utilize blocked algorithms, which attain much greater performance.

In practice LU with complete pivoting is not used.

12.7 Solving $Ax = y$ Via the LU Factorization with Pivoting

Given nonsingular matrix $A \in \mathbb{C}^{m \times m}$, 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

$$Ax = y$$

we multiply both sides of the equation by permutation matrix P

$$PAx = \underbrace{Py}_{\hat{y}}$$

and substitute LU for PA

$$L \underbrace{Ux}_{z} \hat{y}.$$

We now notice that we can solve the lower triangular system

$$Lz = \hat{y}$$

after which x can be computed by solving the upper triangular system

$$Ux = z.$$

12.8 Solving Triangular Systems of Equations

12.8.1 $Lz = y$

First, we discuss solving $Lz = y$ where L is a unit lower triangular matrix.

Variant 1

Consider $Lz = y$ where L is unit lower triangular. Partition

$$L \rightarrow \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right), \quad z \rightarrow \left(\begin{array}{c} \zeta_1 \\ z_2 \end{array} \right) \quad \text{and} \quad y \rightarrow \left(\begin{array}{c} \psi_1 \\ y_2 \end{array} \right).$$

Then

$$\underbrace{\left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right)}_L \underbrace{\left(\begin{array}{c} \zeta_1 \\ z_2 \end{array} \right)}_z = \underbrace{\left(\begin{array}{c} \psi_1 \\ y_2 \end{array} \right)}_y.$$

Algorithm: Solve $Lz = y$, overwriting y (Var. 1)

Partition $L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$

where L_{TL} is 0×0 , y_T has 0 rows

while $m(L_{TL}) < m(L)$ **do**

Repartition

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

$$\left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \Psi_1 \\ y_2 \end{array} \right)$$

$$y_2 := y_2 - \Psi_1 l_{21}$$

Continue with

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

$$\left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \Psi_1 \\ y_2 \end{array} \right)$$

endwhile

Algorithm: Solve $Lz = y$, overwriting y (Var. 2)

Partition $L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$

where L_{TL} is 0×0 , y_T has 0 rows

while $m(L_{TL}) < m(L)$ **do**

Repartition

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

$$\left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \Psi_1 \\ y_2 \end{array} \right)$$

$$\Psi_1 := \Psi_1 - l_{10}^T y_0$$

Continue with

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

$$\left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \Psi_1 \\ y_2 \end{array} \right)$$

endwhile

Figure 12.5: Algorithms for the solution of a unit lower triangular system $Lz = y$ that overwrite y with z .

Multiplying out the left-hand side yields

$$\begin{pmatrix} \zeta_1 \\ \zeta_1 l_{21} + L_{22} z_2 \end{pmatrix} = \begin{pmatrix} \psi_1 \\ y_2 \end{pmatrix}$$

and the equalities

$$\begin{aligned} \zeta_1 &= \psi_1 \\ \zeta_1 l_{21} + L_{22} z_2 &= y_2, \end{aligned}$$

which can be rearranged as

$$\begin{aligned} \zeta_1 &= \psi_1 \\ L_{22} z_2 &= y_2 - \zeta_1 l_{21}. \end{aligned}$$

These insights justify the algorithm in Figure 12.5 (left), which overwrites y with the solution to $Lz = y$.

Variant 2

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

$$\begin{aligned} L_{00} z_0 &= y_0 \\ l_{10}^T z_0 + \zeta_1 &= \psi_1. \end{aligned}$$

The idea now is as follows: Assume that the elements of z_0 were computed in previous iterations in the algorithm in Figure 12.5 (left), overwriting y_0 . Then in the current iteration we can compute $\zeta_1 := \psi_1 - l_{10}^T z_0$, overwriting ψ_1 .

Discussion

Notice that Variant 1 casts the computation in terms of an AXPY operation while Variant 2 casts it in terms of DOT products.

12.8.2 $Ux = z$

Next, we discuss solving $Ux = y$ where U is an upper triangular matrix (with no assumptions about its diagonal entries).

Homework 12.13 Derive an algorithm for solving $Ux = y$, overwriting y with the solution, that casts most computation in terms of DOT products. Hint: Partition

$$U \rightarrow \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right).$$

Call this Variant 1 and use Figure 12.6 to state the algorithm.

[SEE ANSWER](#)

Homework 12.14 Derive an algorithm for solving $Ux = y$, overwriting y with the solution, that casts most computation in terms of AXPY operations. Call this Variant 2 and use Figure 12.6 to state the algorithm.

[SEE ANSWER](#)

12.9 Other LU Factorization Algorithms

There are actually five different (unblocked) algorithms for computing the LU factorization that were discovered over the course of the centuries¹. The LU factorization in Figure 12.1 is sometimes called *classical LU factorization* or the *right-looking* algorithm. We now briefly describe how to derive the other algorithms.

Finding the algorithms starts with the following observations.

- Our algorithms will overwrite the matrix A , and hence we introduce \hat{A} to denote the original contents of A . We will say that the *precondition* for the algorithm is that

$$A = \hat{A}$$

(A starts by containing the original contents of \hat{A} .)

- We wish to overwrite A with L and U . Thus, the postcondition for the algorithm (the state in which we wish to exit the algorithm) is that

$$A = L \setminus U \wedge LU = \hat{A}$$

(A is overwritten by L below the diagonal and U on and above the diagonal, where multiplying L and U yields the original matrix \hat{A} .)

- All the algorithms will march through the matrices from top-left to bottom-right. Thus, at a representative point in the algorithm, the matrices are viewed as quadrants:

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), \quad L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right), \quad \text{and} \quad U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right).$$

where A_{TL} , L_{TL} , and U_{TL} are all square and equally sized.

¹For a thorough discussion of the different LU factorization algorithms that also gives a historic perspective, we recommend “Matrix Algorithms Volume 1” by G.W. Stewart [36]

<p>Algorithm: Solve $Uz = y$, overwriting y (Variant 1)</p> <p>Partition $U \rightarrow \begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix}$, $y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix}$</p> <p>where U_{BR} is 0×0, y_B has 0 rows</p> <p>while $m(U_{BR}) < m(U)$ do</p> <p>Repartition</p> $\begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{pmatrix},$ $\begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ \hline y_2 \end{pmatrix}$ <hr/> <p>Continue with</p> $\begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{pmatrix},$ $\begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ \hline y_2 \end{pmatrix}$ <p>endwhile</p>	<p>Algorithm: Solve $Uz = y$, overwriting y (Variant 2)</p> <p>Partition $U \rightarrow \begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix}$, $y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix}$</p> <p>where U_{BR} is 0×0, y_B has 0 rows</p> <p>while $m(U_{BR}) < m(U)$ do</p> <p>Repartition</p> $\begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{pmatrix},$ $\begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ \hline y_2 \end{pmatrix}$ <hr/> <p>Continue with</p> $\begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{pmatrix},$ $\begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \Psi_1 \\ \hline y_2 \end{pmatrix}$ <p>endwhile</p>
--	--

Figure 12.6: Algorithms for the solution of an upper triangular system $Ux = y$ that overwrite y with x .

- In terms of these exposed quadrants, in the end we wish for matrix A to contain

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & L \setminus U_{BR} \end{array} \right)$$

where $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right) \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right) = \left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right)$

- Manipulating this yields what we call the Partitioned Matrix Expression (PME), which can be viewed as a recursive definition of the LU factorization:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & L \setminus U_{BR} \end{array} \right)$$

$\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}}$

Now, consider the code skeleton for the LU factorization in Figure 12.7. 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 have 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 subexpression. For example, consider the following condition on the contents of A :

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & \hat{A}_{BR} - L_{BL}U_{TR} \end{array} \right)$$

$\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{\cancel{L_{BL}U_{TR} + L_{BR}U_{BR}} = \hat{A}_{BR}}$

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 algorithm that we discussed previously in this document maintains!** What is left is to factor A_{BR} , since it contains $\hat{A}_{BR} - L_{BL}U_{TR}$, and $\hat{A}_{BR} - L_{BL}U_{TR} = L_{BR}U_{BR}$.

- By carefully analyzing the order in which computation must occur (in compiler lingo: by performing a dependence analysis), we can identify five states that can be maintained at the top of the loop, by deleting subexpressions from the PME. These are called *loop invariants* and are listed in Figure 12.8.
- 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.

Algorithm: $A := \text{LU}(A)$

Partition $A \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix}, L \rightarrow \begin{pmatrix} L_{TL} & L_{TR} \\ L_{BL} & L_{BR} \end{pmatrix}, U \rightarrow \begin{pmatrix} U_{TL} & U_{TR} \\ U_{BL} & U_{BR} \end{pmatrix}$

where A_{TL} is 0×0 , L_{TL} is 0×0 , U_{TL} is 0×0

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

Repartition

$$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{pmatrix}, \begin{pmatrix} L_{TL} & L_{TR} \\ L_{BL} & L_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{pmatrix},$$

$$\begin{pmatrix} U_{TL} & U_{TR} \\ U_{BL} & U_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \nu_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{pmatrix}$$

where α_{11} is 1×1 , λ_{11} is 1×1 , ν_{11} is 1×1

Continue with

$$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ a_{10}^T & \alpha_{11} & a_{12}^T \\ A_{20} & a_{21} & A_{22} \end{pmatrix}, \begin{pmatrix} L_{TL} & L_{TR} \\ L_{BL} & L_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} L_{00} & l_{01} & L_{02} \\ l_{10}^T & \lambda_{11} & l_{12}^T \\ L_{20} & l_{21} & L_{22} \end{pmatrix},$$

$$\begin{pmatrix} U_{TL} & U_{TR} \\ U_{BL} & U_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ u_{10}^T & \nu_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{pmatrix}$$

endwhile

Figure 12.7: Code skeleton for LU factorization.

Variant	Algorithm	State (loop invariant)
1	Bordered	$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} L \setminus U_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right)$ $\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}}$
2	Left-looking	$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} L \setminus U_{TL} & \hat{A}_{TR} \\ \hline L_{BL} & \hat{A}_{BR} \end{array} \right)$ $\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}}$
3	Up-looking	$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} L \setminus U_{TL} & U_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right)$ $\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}}$
4	Crout variant	$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & \hat{A}_{BR} \end{array} \right)$ $\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}}$
5	Classical LU	$\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & \hat{A}_{BR} - L_{BL}U_{TR} \end{array} \right)$ $\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}}$

Figure 12.8: Loop invariants for various LU factorization algorithms.

12.9.1 Variant 1: Bordered algorithm

Consider the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array} \right)$$

$$\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}}$$

At the top of the loop, after repartitioning, A contains

$$\left(\begin{array}{c|c|c} L \setminus U_{00} & \hat{a}_{01} & \hat{A}_{02} \\ \hline \hat{a}_{10}^T & \hat{\alpha}_{11} & \hat{a}_{12}^T \\ \hline \hat{A}_{20} & \hat{a}_{21} & \hat{A}_{22} \end{array} \right)$$

while at the bottom it must contain

$$\left(\begin{array}{c|c|c} L \setminus U_{00} & u_{01} & \hat{A}_{02} \\ \hline l_{10}^T & v_{11} & \hat{a}_{12}^T \\ \hline \hat{A}_{20} & \hat{a}_{21} & \hat{A}_{22} \end{array} \right)$$

where the entries in blue are to be computed. Now, considering $LU = \hat{A}$ we notice that

$$\begin{array}{ccc|cc|c} L_{00}U_{00} = \hat{A}_{00} & L_{00}u_{01} = \hat{a}_{01} & L_{00}U_{02} = \hat{A}_{02} \\ \hline l_{10}^T U_{00} = \hat{a}_{10}^T & l_{10}^T u_{01} + v_{11} = \hat{\alpha}_{11} & l_{10}^T U_{02} + u_{12}^T = \hat{a}_{12}^T \\ \hline L_{20}U_{00} = \hat{A}_{20} & L_{20}u_{01} + v_{11}l_{21} = \hat{a}_{21} & L_{20}U_{02} + l_{21}u_{12}^T + L_{22}U_{22} = \hat{A}_{22} \end{array}$$

where the entries in red are already known. The equalities in yellow can be used to compute the desired parts of L and U :

- Solve $L_{00}u_{01} = a_{01}$ for u_{01} , overwriting a_{01} with the result.
- 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.
- Compute $v_{11} = \alpha_{11} - l_{10}^T u_{01}$, overwriting α_{11} with the result.

Homework 12.15 If A is an $n \times n$ matrix, show that the cost of Variant 1 is approximately $\frac{2}{3}n^3$ flops.

SEE ANSWER

12.9.2 Variant 2: Left-looking algorithm

Consider the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & \hat{A}_{TR} \\ \hline L_{BL} & \hat{A}_{BR} \end{array} \right)$$

$$\wedge \frac{L_{TL}U_{TL} = \hat{A}_{TL}}{L_{BL}U_{TL} = \hat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \hat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}}$$

At the top of the loop, after repartitioning, A contains

$L \setminus U_{00}$	\hat{a}_{01}	\hat{A}_{02}
l_{10}^T	$\hat{\alpha}_{11}$	\hat{a}_{12}^T
L_{20}	\hat{a}_{21}	\hat{A}_{22}

while at the bottom it must contain

$L \setminus U_{00}$	u_{01}	\hat{A}_{02}
l_{10}^T	v_{11}	\hat{a}_{12}^T
L_{20}	l_{21}	\hat{A}_{22}

where the entries in blue are to be computed. Now, considering $LU = \hat{A}$ we notice that

$$\begin{array}{c|c|c} L_{00}U_{00} = \hat{A}_{00} & L_{00}u_{01} = \hat{a}_{01} & L_{00}U_{02} = \hat{A}_{02} \\ \hline l_{10}^T U_{00} = \hat{a}_{10}^T & l_{10}^T u_{01} + v_{11} = \hat{\alpha}_{11} & l_{10}^T U_{02} + u_{12}^T = \hat{a}_{12}^T \\ \hline L_{20}U_{00} = \hat{A}_{20} & L_{20}u_{01} + v_{11}l_{21} = \hat{a}_{21} & L_{20}U_{02} + l_{21}u_{12}^T + L_{22}U_{22} = \hat{A}_{22} \end{array}$$

The equalities in yellow can be used to compute the desired parts of L and U :

- Solve $L_{00}u_{01} = a_{01}$ for u_{01} , overwriting a_{01} with the result.
- Compute $v_{11} = \alpha_{11} - l_{10}^T u_{01}$, overwriting α_{11} with the result.
- Compute $l_{21} := (a_{21} - L_{20}u_{01})/v_{11}$, overwriting a_{21} with the result.

12.9.3 Variant 3: Up-looking variant

Homework 12.16 Derive the up-looking variant for computing the LU factorization.

 [SEE ANSWER](#)

12.9.4 Variant 4: Crout variant

Consider the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline L_{BL} & \hat{A}_{BR} \end{array} \right)$$

$$\wedge \left(\begin{array}{c|c} L_{TL}U_{TL} = \hat{A}_{TL} & L_{TL}U_{TR} = \hat{A}_{TR} \\ \hline L_{BL}U_{TL} = \hat{A}_{BL} & \cancel{L_{BL}U_{TR} + L_{BR}U_{BR} = \hat{A}_{BR}} \end{array} \right)$$

At the top of the loop, after repartitioning, A contains

$L \setminus U_{00}$	u_{01}	U_{02}
l_{10}^T	$\hat{\alpha}_{11}$	\hat{a}_{12}^T
L_{20}	\hat{a}_{21}	\hat{A}_{22}

while at the bottom it must contain

$L \setminus U_{00}$	u_{01}	U_{02}
l_{10}^T	v_{11}	u_{12}^T
L_{20}	l_{21}	\hat{A}_{22}

where the entries in blue are to be computed. Now, considering $LU = \hat{A}$ we notice that

$$\begin{array}{|c|c|c|} \hline L_{00}U_{00} = \hat{A}_{00} & L_{00}u_{01} = \hat{a}_{01} & L_{00}U_{02} = \hat{A}_{02} \\ \hline l_{10}^T U_{00} = \hat{a}_{10}^T & l_{10}^T u_{01} + v_{11} = \hat{\alpha}_{11} & l_{10}^T U_{02} + u_{12}^T = \hat{a}_{12}^T \\ \hline L_{20}U_{00} = \hat{A}_{20} & L_{20}u_{01} + v_{11}l_{21} = \hat{a}_{21} & L_{20}U_{02} + l_{21}u_{12}^T + L_{22}U_{22} = \hat{A}_{22} \\ \hline \end{array}$$

The equalities in yellow can be used to compute the desired parts of L and U :

- Compute $v_{11} = \alpha_{11} - l_{10}^T u_{01}$, overwriting α_{11} with the result.
- Compute $l_{21} := (\alpha_{21} - L_{20}u_{01})/v_{11}$, overwriting a_{21} with the result.
- Compute $u_{12}^T := a_{12}^T - l_{10}^T U_{02}$, overwriting a_{12}^T with the result.

12.9.5 Variant 5: Classical LU factorization

We have already derived this algorithm. You may want to try rederiving it using the techniques discussed in this section.

12.9.6 All algorithms

All five algorithms for LU factorization are summarized in Figure 12.9.

Homework 12.17 Implement all five LU factorization algorithms with the FLAME@lab API, in M-script.

☞ SEE ANSWER

Homework 12.18 Which of the five variants can be modified to incorporate partial pivoting?

☞ SEE ANSWER

12.9.7 Formal derivation of algorithms

The described approach to deriving algorithms, linking the process to the *a priori* identification of loop invariants, was first proposed in [25]. It was refined into what we call the “worksheet” for deriving algorithms hand-in-hand with their proofs of correctness, in [4]. A book that describes the process at a level also appropriate for the novice is “The Science of Programming Matrix Computations” [40].

Algorithm: $A := L \setminus U = LUA$

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

where A_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1

A_{00} contains L_{00} and U_{00} in its strictly lower and upper triangular part, respectively.

Variant 1 Bordered	Variant 2 Left-looking	Variant 3 Up-looking	Variant 4 Crout variant	Variant 5 Classical LU
$a_{01} := L_{00}^{-1} a_{01}$ $a_{10}^T := a_{10}^T U_{00}^{-1}$ $\alpha_{11} := \alpha_{11} - a_{21}^T a_{01}$	$a_{01} := L_{00}^{-1} a_{01}$ $\alpha_{11} := \alpha_{11} - a_{21}^T a_{01}$ $a_{21} := a_{21} - A_{20} a_{01}$ $a_{21} := a_{21} / \alpha_{11}$	Exercise in 12.16	$\alpha_{11} := \alpha_{11} - a_{21}^T a_{01}$ $a_{12}^T := a_{12}^T - a_{10}^T A_{02}$ $a_{21} := a_{21} - A_{20} a_{01}$ $a_{21} := a_{21} / \alpha_{11}$	$a_{21} := a_{21} / \alpha_{11}$ $A_{22} := A_{22} - a_{21} a_{12}^T$

Continue with

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

endwhile

Figure 12.9: All five LU factorization algorithms.

12.10 Numerical Stability Results

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 and references [22, 26, 36]. Of particular interest may be the analysis of the Crout variant (Variant 4) in [9], since it uses our notation as well as the results in “Notes on Numerical Stability”. (We recommend the technical report version [7] of the paper, since it has more details as well as exercises to help the reader understand.) In that paper, a systematic approach towards the derivation of backward error results is given that mirrors the systematic approach to deriving the algorithms given in [?, 4, 40].

Here are pertinent results from that paper, assuming floating point arithmetic obeys the model of computation given in “Notes on Numerical Stability” (as well as [9, 7, 26]). It is assumed that the reader is familiar with those notes.

Theorem 12.19 *Let $A \in \mathbb{R}^{n \times n}$ and let the LU factorization of A be computed via the Crout variant (Variant 4), 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 12.20 *Let $L \in \mathbb{R}^{n \times n}$ be lower triangular and $y, z \in \mathbb{R}^n$ with $Lz = y$. Let \check{z} be the approximate solution that is computed. Then*

$$(L + \Delta L)\check{z} = y \quad \text{with} \quad |\Delta L| \leq \gamma_n |L|.$$

Theorem 12.21 *Let $U \in \mathbb{R}^{n \times n}$ be upper triangular and $x, z \in \mathbb{R}^n$ with $Ux = z$. Let \check{x} be the approximate solution that is computed. Then*

$$(U + \Delta U)\check{x} = z \quad \text{with} \quad |\Delta U| \leq \gamma_n |U|.$$

Theorem 12.22 *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 $\check{L}\check{z} = y$, yielding approximate solution \check{z} .
- Solve $\check{U}\check{x} = \check{z}$, yielding approximate solution \check{x} .

Then $(A + \Delta A)\check{x} = y$ with $|\Delta A| \leq (3\gamma_n + \gamma_n^2)|\check{L}| |\check{U}|$.

The analysis of LU factorization without partial pivoting is related that of LU factorization with partial pivoting. We have shown that LU 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 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

12.11 Is LU with Partial Pivoting Stable?

Homework 12.23 Apply LU with partial pivoting to

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & & \cdots & 1 & 1 \\ -1 & -1 & & \cdots & -1 & 1 \end{pmatrix}.$$

Pivot only when necessary.

[SEE ANSWER](#)

From this exercise we conclude that even LU factorization with partial pivoting can yield large (exponential) element growth in U . You may enjoy the collection of problems for which Gaussian elimination with partial pivoting is unstable by Stephen Wright [48].

In practice, this does not seem to happen and LU factorization is considered to be stable.

12.12 Blocked Algorithms

It is well-known that matrix-matrix multiplication can achieve high performance on most computer architectures [2, 23, 20]. As a result, many dense matrix algorithms are reformulated to be rich in matrix-matrix multiplication operations. An interface to a library of such operations is known as the level-3 Basic Linear Algebra Subprograms (BLAS) [18]. In this section, we show how LU factorization can be rearranged so that most computation is in matrix-matrix multiplications.

12.12.1 Blocked classical LU factorization (Variant 5)

Partition A , L , and U as follows:

$$A \rightarrow \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right), \quad L \rightarrow \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right), \quad \text{and} \quad U \rightarrow \left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array} \right),$$

where A_{11} , L_{11} , and U_{11} are $b \times b$. Then $A = LU$ means that

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right) \left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array} \right) = \left(\begin{array}{c|c} L_{11}U_{11} & L_{11}U_{12} \\ \hline L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{array} \right).$$

This means that

$$\begin{array}{c|c} A_{11} = L_{11}U_{11} & A_{12} = L_{11}U_{12} \\ \hline A_{21} = L_{21}U_{11} & A_{22} = L_{21}U_{12} + L_{22}U_{22} \end{array}$$

Algorithm: $A := LU(A)$

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

where A_{TL} is 0×0

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

Determine block size b

Repartition

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

where A_{11} is $b \times b$

A_{00} contains L_{00} and U_{00} in its strictly lower and upper triangular part, respectively.

Variant 1:

$$A_{01} := L_{00}^{-1} A_{01}$$

$$A_{10} := A_{10} U_{00}^{-1}$$

$$A_{11} := A_{11} - A_{10} A_{01}$$

$$A_{11} := LUA_{11}$$

Variant 2:

$$A_{01} := L_{00}^{-1} A_{01}$$

$$A_{11} := A_{11} - A_{10} A_{01}$$

$$A_{11} := LUA_{11}$$

$$A_{21} := (A_{21} - A_{20} A_{01}) U_{11}^{-1}$$

Variant 3:

$$A_{10} := A_{10} U_{00}^{-1}$$

$$A_{11} := A_{11} - A_{10} A_{01}$$

$$A_{11} := LUA_{11}$$

$$A_{12} := A_{12} - A_{10} A_{02}$$

Variant 4:

$$A_{11} := A_{11} - A_{10} A_{01}$$

$$A_{11} := LUA_{11}$$

$$A_{21} := (A_{21} - A_{20} A_{01}) U_{11}^{-1}$$

$$A_{12} := L_{11}^{-1} (A_{12} - A_{10} A_{02}) A_{22} := A_{22} - A_{21} A_{12}$$

Variant 5:

$$A_{11} := LUA_{11}$$

$$A_{21} := A_{21} U_{11}^{-1}$$

$$A_{12} := L_{11}^{-1} A_{12}$$

Continue with

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

endwhile

Figure 12.10: Blocked algorithms for computing the LU factorization.

Algorithm: $[A, p] := \text{LUPIV_BLK}(A, p)$

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

where A_{TL} is 0×0 , p_T has 0 elements.

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

Determine block size b

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline p_1 \\ \hline p_2 \end{array} \right)$$

where A_{11} is $b \times b$, p_1 has b elements

Variant 2:

$$A_{01} := L_{00}^{-1} A_{01}$$

$$A_{11} := A_{11} - A_{10}A_{01}$$

$$A_{21} := A_{21} - A_{20}A_{01}$$

$$\left[\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, p_1 \right] :=$$

$$\text{LUpiv} \left(\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, p_1 \right)$$

$$\left(\begin{array}{c|c} A_{10} & | A_{12} \\ \hline A_{20} & | A_{22} \end{array} \right) :=$$

$$P(p_1) \left(\begin{array}{c|c} A_{10} & | A_{12} \\ \hline A_{20} & | A_{22} \end{array} \right)$$

Variant 4:

$$A_{11} := A_{11} - A_{10}A_{01}$$

$$A_{21} := A_{21} - A_{20}A_{01}$$

$$\left[\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, p_1 \right] :=$$

$$\text{LUpiv} \left(\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, p_1 \right)$$

$$\left(\begin{array}{c|c} A_{10} & | A_{12} \\ \hline A_{20} & | A_{22} \end{array} \right) :=$$

$$P(p_1) \left(\begin{array}{c|c} A_{10} & | A_{12} \\ \hline A_{20} & | A_{22} \end{array} \right)$$

$$A_{12} := A_{12} - A_{10}A_{02}$$

$$A_{12} := L_{11}^{-1} A_{12}$$

Variant 5:

$$\left[\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, p_1 \right] :=$$

$$\text{LUpiv} \left(\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}, p_1 \right)$$

$$\left(\begin{array}{c|c} A_{10} & | A_{12} \\ \hline A_{20} & | A_{22} \end{array} \right) :=$$

$$P(p_1) \left(\begin{array}{c|c} A_{10} & | A_{12} \\ \hline A_{20} & | A_{22} \end{array} \right)$$

$$A_{12} := L_{11}^{-1} A_{12}$$

$$A_{22} := A_{22} - A_{21}A_{12}$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} p_T \\ \hline p_B \end{array} \right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline p_1 \\ \hline p_2 \end{array} \right)$$

endwhile

Figure 12.11: Blocked algorithms for computing the LU factorization with partial pivoting.

or, equivalently,

$$\begin{array}{c|c} A_{11} = L_{11}U_{11} & A_{12} = L_{11}U_{12} \\ \hline A_{21} = L_{21}U_{11} & A_{22} - L_{21}U_{12} = L_{22}U_{22} \end{array}.$$

If we let L and U overwrite the original matrix A this suggests the algorithm

- Compute the LU factorization $A_{11} = L_{11}U_{11}$, overwriting A_{11} with $L \setminus U_{11}$. Notice that any of the “unblocked” algorithms previously discussed in this note can be used for this factorization.
- Solve $L_{11}U_{12} = A_{12}$, overwriting A_{12} with U_{12} . (This can also be expressed as $A_{12} := L_{11}^{-1}A_{12}$.)
- Solve $L_{21}U_{11} = A_{21}$, overwiting A_{21} with U_{21} . (This can also be expressed as $A_{21} := A_{21}U_{11}^{-1}$.)
- Update $A_{22} := A_{22} - A_{21}A_{12}$.
- Continue by overwriting the updated A_{22} with its LU factorization.

If b is small relative to n , then most computation is in the last step, which is a matrix-matrix multiplication. Similarly, blocked algorithms for the other variants can be derived. All are given in Figure 12.10.

12.12.2 Blocked classical LU factorization with pivoting (Variant 5)

Pivoting can be added to some of the blocked algorithms. Let us focus once again on Variant 5.

Partition A , L , and U as follows:

$$A \rightarrow \left(\begin{array}{c|cc|c} A_{00} & A_{01} & A_{02} & \\ \hline A_{10} & A_{11} & A_{12} & \\ A_{20} & A_{21} & A_{22} & \end{array} \right), \quad L \rightarrow \left(\begin{array}{c|cc|c} L_{00} & 0 & 0 & \\ \hline L_{10} & L_{11} & 0 & \\ L_{20} & L_{21} & L_{22} & \end{array} \right), \quad \text{and} \quad U \rightarrow \left(\begin{array}{c|cc|c} U_{00} & U_{01} & U_{02} & \\ \hline 0 & U_{11} & U_{12} & \\ 0 & 0 & U_{22} & \end{array} \right),$$

where A_{00} , L_{00} , and U_{00} are $k \times k$, and A_{11} , L_{11} , and U_{11} are $b \times b$.

Assume that the computation has proceeded to the point where A contains

$$\left(\begin{array}{c|cc|c} A_{00} & A_{01} & A_{02} & \\ \hline A_{10} & A_{11} & A_{12} & \\ A_{20} & A_{21} & A_{22} & \end{array} \right) = \left(\begin{array}{c|cc|c} L \setminus U_{00} & U_{01} & U_{02} & \\ \hline L_{10} & \hat{A}_{11} - L_{10}U_{01} & A_{12} - L_{10}U_{02} & \\ L_{20} & \hat{A}_{21} - L_{20}U_{01} & A_{22} - L_{20}U_{02} & \end{array} \right),$$

where, as before, \hat{A} denotes the original contents of A and

$$P(p_0) \left(\begin{array}{c|cc|c} \hat{A}_{00} & \hat{A}_{01} & \hat{A}_{02} & \\ \hline \hat{A}_{10} & \hat{A}_{11} & \hat{A}_{12} & \\ \hat{A}_{20} & \hat{A}_{21} & \hat{A}_{22} & \end{array} \right) = \left(\begin{array}{c|cc|c} L_{00} & 0 & 0 & \\ \hline L_{10} & I & 0 & \\ L_{20} & 0 & I & \end{array} \right) \left(\begin{array}{c|cc|c} U_{00} & U_{01} & U_{02} & \\ \hline 0 & A_{11} & A_{12} & \\ 0 & A_{21} & A_{22} & \end{array} \right).$$

In the current blocked step, we now perform the following computations

- Compute the LU factorization with pivoting of the “current panel” $\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$:

$$P(p_1) \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} U_{11},$$

overwriting A_{11} with $L \setminus U_{11}$ and A_{21} with L_{21} .

- Correspondingly, swap rows in the remainder of the matrix

$$\left(\begin{array}{c|c} A_{10} & A_{12} \\ \hline A_{20} & A_{22} \end{array} \right) := P(p_1) \left(\begin{array}{c|c} A_{10} & A_{12} \\ \hline A_{20} & A_{22} \end{array} \right).$$

- Solve $L_{11}U_{12} = A_{12}$, overwriting A_{12} with U_{12} . (This can also be more concisely written as $A_{12} := L_{11}^{-1}A_{12}$.)
- Update $A_{22} := A_{22} - A_{21}A_{12}$.

Careful consideration shows that this puts the matrix A in the state

$$\left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c|c} L \setminus U_{00} & U_{01} & U_{02} \\ \hline L_{10} & L \setminus U_{11} & U_{12} \\ \hline L_{20} & L_{21} & \widehat{A}_{22} - L_{20}U_{02} - L_{21}U_{12} \end{array} \right),$$

where

$$P\left(\begin{pmatrix} p_0 \\ p_1 \end{pmatrix}\right) \left(\begin{array}{c|c|c} \widehat{A}_{00} & \widehat{A}_{01} & \widehat{A}_{02} \\ \hline \widehat{A}_{10} & \widehat{A}_{11} & \widehat{A}_{12} \\ \hline \widehat{A}_{20} & \widehat{A}_{21} & \widehat{A}_{22} \end{array} \right) = \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & I \end{array} \right) \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & A_{22} \end{array} \right).$$

Similarly, blocked algorithms with pivoting for some of the other variants can be derived. All are given in Figure 12.10.

12.13 Variations on a Triple-Nested Loop

All LU factorization algorithms presented in this note perform exactly the same floating point operations (with some rearrangement of data thrown in for the algorithms that perform pivoting) as does the triple-nested loop that implements Gaussian elimination:

```

for  $j = 0, \dots, n - 1$  (zero the elements below  $(j, j)$  element)
  for  $i = j + 1, \dots, n - 1$ 
     $\alpha_{i,j} := \alpha_{i,j} / \alpha_{j,j}$  (compute multiplier  $\lambda_{i,j}$ , overwriting  $\alpha_{i,j}$ )
    for  $k = j + 1, \dots, n - 1$  (subtract  $\lambda_{i,j}$  times the  $j$ th row from  $i$ th row)
       $\alpha_{i,k} := \alpha_{i,k} - \alpha_{i,j} \alpha_{j,k}$ 
    endfor
  endfor
endfor

```

12.14 Inverting a Matrix

12.14.1 Basic observations

If $n \times n$ matrix A is invertable (nonsingular, has linearly independent columns, etc), then its inverse, $A^{-1} = X$ satisfies

$$AX = I$$

where I is the identity matrix. Partitioning X and I by columns, we find that

$$A \left(\begin{array}{c|c|c|c} x_0 & \cdots & x_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} e_0 & \cdots & e_{n-1} \end{array} \right),$$

where e_j equals the j th standard basis vector. Thus $Ax_j = e_j$, and hence any method for solving $Ax = y$ can be used to solve each of these problems, of which there are n .

12.14.2 Via the LU factorization with pivoting

We have learned how to compute the LU factorization with partial pivoting, yielding P , L , and U such that

$$PA = LU$$

at an approximate cost of $\frac{2}{3}n^3$ flops.

Obviously, for each column of X can then solve $Ax_j = e_j$ or, equivalently,

$$L(Ux_j) = Pe_j.$$

The cost of this would be approximately $2n^2$ per column x_j for a total of $2n^3$ flops. This is in addition to the cost of factoring A in the first place, for a grand total of $\frac{8}{3}n^3$ operations.

Alternatively, one can manipulate $AX = I$ into

$$LUX = P$$

or, equivalently,

$$UX = L^{-1}P$$

and finally

$$U(XP^T) = L^{-1}.$$

this suggests the steps

- Compute P , L , and U such that $PA = LU$.
This costs, approximately $\frac{2}{3}n^3$.
- Invert L , yielding unit lower triangular matrix L^{-1} .
Homework 12.26 will show that this can be done at a cost of, approximately, $\frac{1}{3}n^3$ flops.
- Solve $UY = L^{-1}$.
Homework exercises will show that this can be done at a cost of, approximately, $\frac{3}{3}n^3$ flops. (I need to check this!)
- Permute the columns of the result: $X = YP$.
This incurs a $O(n^2)$ cost.

The total cost now is $2n^3$ flops.

Homework 12.24 Let L be a unit lower triangular matrix partitioned as

$$L = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \end{array} \right).$$

Show that

$$L^{-1} = \left(\begin{array}{c|c} L_{00}^{-1} & \\ \hline -l_{10}^T L_{00}^{-1} & 1 \end{array} \right).$$

☞ SEE ANSWER

Homework 12.25 Use Homework 12.24 and Figure 12.12 to propose an algorithm for overwriting a unit lower triangular matrix L with its inverse.

☞ SEE ANSWER

Homework 12.26 Show that the cost of the algorithm in Homework 12.25 is, approximately, $\frac{1}{3}n^3$, where L is $n \times n$.

☞ SEE ANSWER

Homework 12.27 Given $n \times n$ upper triangular matrix U and $n \times n$ unit lower triangular matrix L , propose an algorithm for computing Y where $UY = L$. Be sure to take advantage of zeros in U and L . The cost of the resulting algorithm should be, approximately, n^3 flops.

There is a way of, given that L and U have overwritten matrix A , overwriting this matrix with Y . Try to find that algorithm. It is not easy...

☞ SEE ANSWER

12.14.3 Gauss-Jordan inversion

The above discussion breaks the computation of A^{-1} down into several steps. Alternatively, one can use variants of the Gauss-Jordan algorithm to invert a matrix, also at a cost of $2n^3$ flops. For details, see Chapter 8 of “Linear Algebra: Foundations to Frontiers” [30] and/or [33]. A thorough discussion of algorithms for inverting symmetric positive definite (SPD) matrices, which also discusses algorithms for inverting triangular matrices, can be found in [5].

Algorithm: $[L] := \text{LINV_UNB_VAR1}(L)$

$$\text{Partition } L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right)$$

where L_{TL} is 0×0

while $m(L_{TL}) < m(L)$ **do**

Repartition

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right)$$

where λ_{11} is 1×1

Continue with

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right)$$

endwhile

Figure 12.12: Algorithm skeleton for inverting a unit lower triangular matrix.

12.14.4 (Almost) never, ever invert a matrix

There are very few applications for which explicit inversion of a matrix is needed. When talking about solving $Ax = y$, the term “inverting the matrix” is often used, since $y = A^{-1}x$. However, notice that inverting the matrix and then multiplying it times vector y requires $2n^3$ flops for the inversion and then $2n^2$ flops for the multiplication. In contrast, computing the LU factorization requires $\frac{2}{3}n^3$ flops, after which the solves with L and U requires an additional $2n^2$ flops. Thus, it is essentially always beneficial to instead compute the LU factorization and to use this to solve for x .

There are also numerical stability benefits to this, which we will not discuss now.

12.15 Efficient Condition Number Estimation

Given that the condition number of a matrix tells one how accurately one can expect the solution to $Ax = y$ to be when y is slightly perturbed, a routine that solves this problem should ideally also return the condition number of the matrix so that a user can assess the quality of the solution (or, rather, the difficulty of the posed problem).

Here, we briefly discuss simple techniques that underlie the more sophisticated methods used by LAPACK, focusing on $A \in \mathbb{R}^{n \times n}$.

12.15.1 The problem

The problem is to estimate $\kappa(A) = \|A\| \|A^{-1}\|$ for some matrix norm induced by a vector norm. If A is a dense matrix, then inverting the matrix requires approximately $2n^3$ computations, which is more expensive than solving $Ax = y$ via, for example, LU factorization with partial pivoting. A second problem is that computing the 2-norms $\|A\|_2$ and $\|A^{-1}\|_2$ is less than straight-forward and computationally expensive.

12.15.2 Insights

The second problem can be solved by choosing either the 1-norm or the ∞ -norm. The first problem can be solved by settling for an estimation of the condition number. For this we will try to find a lower bound close to the actual condition number, since then the user gains insight into how many digits of accuracy might be lost and will know this is a poorly conditioned problem if the estimate is large. If the estimate is small, the jury is out whether the matrix is actually well-conditioned.

Notice that

$$\|A^{-1}\|_\infty = \max_{\|y\|_\infty=1} \|A^{-1}y\|_\infty \geq \|A^{-1}d\|_\infty = \|x\|_\infty$$

where $Ax = d$ and $\|d\|_\infty = 1$. So, we want to find an approximation for the vector (direction) d with $\|d\|_\infty = 1$ that has the property that $\|x\|_\infty$ is nearly maximally magnified. Remember from the proof of the fact that

$$\|A\|_\infty = \max_j \|\hat{a}_j\|_1$$

(where \hat{a}_j^T equals the j th row of A) that the maximum

$$\|A\|_\infty = \max_{\|y\|_\infty=1} \|Ay\|_\infty$$

is attained for a vector

$$y = \begin{pmatrix} \pm 1 \\ \pm 1 \\ \vdots \\ \pm 1 \end{pmatrix},$$

which has property that $\|y\|_\infty = 1$. So, it would be good to look for a vector d with the special property

that

$$d = \begin{pmatrix} \pm 1 \\ \pm 1 \\ \vdots \\ \pm 1 \end{pmatrix},$$

12.15.3 A simple approach

It helps to read up on how to solve triangular linear systems, in Chapter ??.

Condition number estimation is usually required in context of finding the solution to $Ax = y$, where one also wants to know how well-conditioned the problem is. This means that typically the LU factorization with partial pivoting is already being computed as part of the solution method. In other words, permutation matrix P , unit lower triangular matrix L , and upper triangular matrix U are known so that $PA = LU$. Now

$$\begin{aligned} \|A^{-1}\|_1 &= \|A^{-T}\|_\infty = \|(PLU)^{-T}\|_\infty = \|((PLU)^T)^{-1}\|_\infty = \|(U^T L^T P^T)^{-1}\|_\infty = \|PL^{-T} U^{-T}\|_\infty \\ &= \|L^{-T} U^{-T}\|_\infty. \end{aligned}$$

So, we are looking for x that satisfies $U^T L^T x = d$ where d has ± 1 for each of its elements. Importantly, we get to choose d .

The way a simple approach proceeds is to pick d in such a way in an effort to make $\|z\|_\infty$ large, where $U^T z = d$. After this $L^T x = z$ is solved and the estimate for $\|A^{-T}\|_\infty$ is $\|x\|_\infty$.

Partition

$$U \rightarrow \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right), z \rightarrow \begin{pmatrix} \zeta_1 \\ z_2 \end{pmatrix}, \text{ and } d \rightarrow \begin{pmatrix} \delta_1 \\ d_2 \end{pmatrix}.$$

Then $U^T z = d$ means

$$\left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right)^T \begin{pmatrix} \zeta_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \delta_1 \\ d_2 \end{pmatrix}$$

and hence

$$\begin{aligned} v_{11}\zeta_1 &= \delta_1 \\ \zeta_1 u_{12} + U_{22} z_2 &= d_2 \end{aligned}$$

Now, δ_1 can be chosen as ± 1 . (The choice does not make a difference. Might as well make it $\delta_1 = 1$.) Then $\zeta_1 = 1/v_{11}$.

Now assume that the process has proceeded to the point where

$$U \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & v_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array} \right), z \rightarrow \begin{pmatrix} z_0 \\ \zeta_1 \\ z_2 \end{pmatrix}, \text{ and } d \rightarrow \begin{pmatrix} d_0 \\ \delta_1 \\ d_2 \end{pmatrix}$$

Algorithm:	Algorithm:
<p>Partition $U \rightarrow \left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right)$, $z \rightarrow \begin{pmatrix} z_T \\ z_B \end{pmatrix}$</p> <p>where U_{TL} is 0×0, z_T has 0 rows</p> <p>while $m(U_{TL}) < m(U)$ do</p> <p style="color: blue;">Repartition</p> $\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c c c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \begin{pmatrix} z_T \\ z_B \end{pmatrix} \rightarrow \begin{pmatrix} z_0 \\ \zeta_1 \\ z_2 \end{pmatrix}$ <hr/> <p>$\zeta_1 := -u_{01}^T z_0$</p> <p>$\zeta_1 := (\text{sign}(\zeta_1) + \zeta_1)/v_{11}$</p> <hr/> <p>Continue with</p> $\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c c c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \begin{pmatrix} z_T \\ z_B \end{pmatrix} \leftarrow \begin{pmatrix} z_0 \\ \zeta_1 \\ z_2 \end{pmatrix}$ <p>endwhile</p>	<p>Partition $U \rightarrow \left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right)$, $z \rightarrow \begin{pmatrix} z_T \\ z_B \end{pmatrix}$</p> <p>where U_{TL} is 0×0, z_T has 0 rows</p> <p>while $m(U_{TL}) < m(U)$ do</p> <p style="color: blue;">Repartition</p> $\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c c c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \begin{pmatrix} z_T \\ z_B \end{pmatrix} \rightarrow \begin{pmatrix} z_0 \\ \zeta_1 \\ z_2 \end{pmatrix}$ <hr/> <p>$\zeta_1 := (\text{sign}(\zeta_1) + \zeta_1)/v_{11}$</p> <p>$z_2 := z_2 - \zeta_1 u_{12}$</p> <hr/> <p>Continue with</p> $\left(\begin{array}{c c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c c c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right), \begin{pmatrix} z_T \\ z_B \end{pmatrix} \leftarrow \begin{pmatrix} z_0 \\ \zeta_1 \\ z_2 \end{pmatrix}$ <p>endwhile</p>

Figure 12.13: Two equivalent algorithms for computing z so that $\|z\|_\infty \leq \|U^{-T}\|_\infty$.

so that $U^T z = d$ means

$$\begin{aligned} U_{00}^T z_0 &= d_0 \\ u_{01}^T z_0 + v_{11} \zeta_1 &= \delta_1 \\ U_{02}^T z_0 + \zeta_1 u_{12} + U_{22}^T z_2 &= d_2. \end{aligned}$$

Assume that d_0 and z_0 have already been computed and now δ_1 and ζ_1 are to be computed. Then

$$\zeta_1 = (\delta_1 - u_{01}^T z_0)/v_{11}.$$

Constrained by the fact that $\delta_1 = \pm 1$, the magnitude of ζ_1 can be (locally) maximized by choosing

$$\delta_1 := -\text{sign}(u_{01}^T z_0)$$

and then

$$\zeta_1 := (\delta_1 - u_{01}^T z_0)/v_{11}.$$

This suggests the algorithm in Figure 12.13(left). By then taking the output vector z and solving the unit upper triangular system $L^T x = z$ we compute $\|x\|_\infty \leq \|A^{-T}\|_\infty = \|A^{-1}\|_1$ so that

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \geq \|A\|_1 \|x\|_\infty.$$

An alternative algorithm that casts computation in terms of AXPY instead of DOT, but computes the exact same z as the last algorithm, can be derived as follows: The first (top) entry of d can again be chosen to equal 1. Assume we have proceeded to the point where

$$U \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & v_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array} \right), z \rightarrow \begin{pmatrix} z_0 \\ \zeta_1 \\ z_2 \end{pmatrix}, \text{ and } d \rightarrow \begin{pmatrix} d_0 \\ \delta_1 \\ d_2 \end{pmatrix}$$

so that $U^T z = d$ again means

$$\begin{aligned} U_{00}^T z_0 &= d_0 \\ u_{01}^T z_0 + v_{11} \zeta_1 &= \delta_1 \\ U_{02}^T z_0 + \zeta_1 u_{12} + U_{22}^T z_2 &= d_2. \end{aligned}$$

Assume that d_0 and z_0 have already been computed and that ζ_1 contains $-u_{01}^T z_0$ while z_2 contains $-U_{02}^T z_0$. Then

$$\zeta_1 := (\delta_1 - u_{01}^T z_0) / v_{11} = (\delta_1 + \zeta_1) / v_{11}.$$

Constrained by the fact that $\delta_1 = \pm 1$, the magnitude of ζ_1 can be (locally) maximized by choosing

$$\delta_1 := -\text{sign}(u_{01}^T z_0) = \text{sign}(\zeta_1)$$

and then

$$\zeta_1 := (\delta_1 + \zeta_1) / v_{11}.$$

After this, z_2 is updated with

$$z_2 := z_2 - \zeta_1 u_{12}.$$

This suggests the algorithm in Figure 12.13(right). It computes the same z as does the algorithm in Figure 12.13(left).

By then taking the output vector z and solving the unit upper triangular system $L^T x = z$ we again compute $\|x\|_\infty \leq \|A^{-T}\|_\infty = \|A^{-1}\|_1$ so that

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \geq \|A\|_1 \|x\|_\infty.$$

12.15.4 Discussion

There is a subtle reason for why we estimate $\|A\|_1$ by estimating $\|A^{-T}\|_\infty$ instead. The reason is that when the LU factorization with pivoting, $PA = LU$, is computed, poor conditioning of the problem tends to translate more into poor conditioning of matrix U than L , since L is chosen to be unit lower triangular and the magnitudes of its strictly lower triangular elements are bounded by one. So, but choosing d based on U , we use information about the triangular matrix that is more important.

A secondary reason is that if element growth happens during the factorization, that element growth exhibits itself in artificially large elements of U , which then translates to a less well-conditioned matrix U . In this case the condition number of A may not be all that bad, but when its factorization is used to solve $Ax = y$ the effective conditioning is worse and that effective conditioning is due to U .

12.16 Wrapup

12.16.1 Additional exercises

12.16.2 Summary

Chapter **13**

Notes on Cholesky Factorization

13.1 Opening Remarks

Video

Read disclaimer regarding the videos in the preface!

- 👉 [YouTube](#)
- 👉 [Download from UT Box](#)
- 👉 [View After Local Download](#)

(For help on viewing, see Appendix A.)

13.1.1 Launch

13.1.2 Outline

13.1	Opening Remarks	257
13.1.1	Launch	257
13.1.2	Outline	258
13.1.3	What you will learn	259
13.2	Definition and Existence	260
13.3	Application	260
13.4	An Algorithm	261
13.5	Proof of the Cholesky Factorization Theorem	262
13.6	Blocked Algorithm	263
13.7	Alternative Representation	263
13.8	Cost	266
13.9	Solving the Linear Least-Squares Problem via the Cholesky Factorization	267
13.10	Other Cholesky Factorization Algorithms	267
13.11	Implementing the Cholesky Factorization with the (Traditional) BLAS	269
13.11.1	What are the BLAS?	269
13.11.2	A simple implementation in Fortran	272
13.11.3	Implementation with calls to level-1 BLAS	272
13.11.4	Matrix-vector operations (level-2 BLAS)	272
13.11.5	Matrix-matrix operations (level-3 BLAS)	276
13.11.6	Impact on performance	276
13.12	Alternatives to the BLAS	277
13.12.1	The FLAME/C API	277
13.12.2	BLIS	277
13.13	Wrapup	278
13.13.1	Additional exercises	278
13.13.2	Summary	278

13.1.3 What you will learn

13.2 Definition and Existence

This operation is only defined for Hermitian positive definite matrices:

Definition 13.1 A matrix $A \in \mathbb{C}^{m \times m}$ is Hermitian positive definite (HPD) if and only if it is Hermitian ($A^H = A$) and for all nonzero vectors $x \in \mathbb{C}^m$ it is the case that $x^H A x > 0$. If in addition $A \in \mathbb{R}^{m \times m}$ 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 13.2 Consider the case where $m = 1$ 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 $\alpha\chi^2 > 0$.

First some exercises:

Homework 13.3 Let $B \in \mathbb{C}^{m \times n}$ have linearly independent columns. Prove that $A = B^H B$ is HPD.

 [SEE ANSWER](#)

Homework 13.4 Let $A \in \mathbb{C}^{m \times m}$ be HPD. Show that its diagonal elements are real and positive.

 [SEE ANSWER](#)

We will prove the following theorem in Section 13.5:

Theorem 13.5 (Cholesky Factorization Theorem) Given a HPD matrix A there exists a lower triangular matrix L such that $A = LL^H$.

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.

The operation that overwrites the lower triangular part of matrix A with its Cholesky factor will be denoted by $A := \text{Chol}A$, which should be read as “ A becomes its Cholesky factor.” 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 this discussion, we will assume that the lower triangular part of A is stored and overwritten.

13.3 Application

The Cholesky factorization is used to solve the linear system $Ax = y$ when A is HPD: Substituting the factors into the equation yields $LL^H x = y$. Letting $z = L^H x$,

$$Ax = L \underbrace{(L^H x)}_z = Lz = y.$$

Thus, z can be computed by solving the triangular system of equations $Lz = y$ and subsequently the desired solution x can be computed by solving the triangular linear system $L^H x = z$.

13.4 An Algorithm

The most common algorithm for computing $A := \text{Chol}A$ can be derived as follows: Consider $A = LL^H$. Partition

$$A = \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array} \right) \quad \text{and} \quad L = \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array} \right). \quad (13.1)$$

Remark 13.6 We adopt the commonly used notation where Greek lower case letters refer to scalars, lower case letters refer to (column) vectors, and upper case letters refer to matrices. (This is convention is often attributed to Alston Householder.) The \star refers to a part of A that is neither stored nor updated.

By substituting these partitioned matrices into $A = LL^H$ we find that

$$\begin{aligned} \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array} \right) &= \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array} \right) \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array} \right)^H = \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array} \right) \left(\begin{array}{c|c} \bar{\lambda}_{11} & l_{21}^H \\ \hline 0 & L_{22}^H \end{array} \right) \\ &= \left(\begin{array}{c|c} |\lambda_{11}|^2 & \star \\ \hline \bar{\lambda}_{11}l_{21} & l_{21}l_{21}^H + L_{22}L_{22}^H \end{array} \right), \end{aligned}$$

from which we conclude that

$$\frac{|\lambda_{11}| = \sqrt{\alpha_{11}}}{l_{21} = a_{21}/\bar{\lambda}_{11}} \left| \begin{array}{c} \star \\ L_{22} = \text{Chol}A_{22} - l_{21}l_{21}^H \end{array} \right..$$

These equalities motivate the algorithm

1. Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array} \right)$.
2. Overwrite $\alpha_{11} := \lambda_{11} = \sqrt{\alpha_{11}}$. (Picking $\lambda_{11} = \sqrt{\alpha_{11}}$ makes it positive and real, and ensures uniqueness.)
3. Overwrite $a_{21} := l_{21} = a_{21}/\lambda_{11}$.
4. 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*.
5. Continue with $A = A_{22}$. (Back to Step 1.)

Remark 13.7 Similar to the `tril` function in Matlab, we use `tril(B)` to denote the lower triangular part of matrix B .

13.5 Proof of the Cholesky Factorization Theorem

In this section, we partition A as in (13.1):

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{21}^H \\ \hline a_{21} & A_{22} \end{array} \right).$$

The following lemmas are key to the proof:

Lemma 13.8 *Let $A \in \mathbb{C}^{n \times n}$ be HPD. Then α_{11} is real and positive.*

Proof: This is special case of Exercise 13.4.

Lemma 13.9 *Let $A \in \mathbb{C}^{m \times m}$ 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 $x_2 \in \mathbb{C}^{(n-1) \times (n-1)}$ 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 $x \neq 0$,

$$\begin{aligned} 0 < x^H A x &= \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix}^H \left(\begin{array}{c|c} \alpha_{11} & a_{21}^H \\ \hline a_{21} & A_{22} \end{array} \right) \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} \\ &= \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix}^H \begin{pmatrix} \alpha_{11}\chi_1 + a_{21}^H x_2 \\ a_{21}\chi_1 + A_{22}x_2 \end{pmatrix} \\ &= \alpha_{11}|\chi_1|^2 + \bar{\chi}_1 a_{21}^H x_2 + x_2^H a_{21} \chi_1 + x_2^H A_{22} x_2 \\ &= \alpha_{11} \frac{a_{21}^H x_2}{\alpha_{11}} \frac{x_2^H a_{21}}{\alpha_{11}} - \frac{x_2^H a_{21}}{\alpha_{11}} a_{21}^H x_2 - x_2^H a_{21} \frac{a_{21}^H x_2}{\alpha_{11}} + x_2^H A_{22} x_2 \\ &= x_2^H (A_{22} - \frac{a_{21}a_{21}^H}{\alpha_{11}}) x_2 \quad (\text{since } x_2^H a_{21} a_{21}^H x_2 \text{ is real and hence equals } a_{21}^H x_2 x_2^H a_{21}) \\ &= x_2^H (A_{22} - l_{21}l_{21}^H) x_2. \end{aligned}$$

We conclude that $A_{22} - l_{21}l_{21}^H$ is HPD.

Proof: of the Cholesky Factorization Theorem

Proof by induction.

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.

Inductive step: Assume the result is true for HPD matrix $A \in \mathbb{C}^{(n-1) \times (n-1)}$. We will show that it holds for $A \in \mathbb{C}^{n \times n}$. Let $A \in \mathbb{C}^{n \times n}$ be HPD. Partition A and L as in (13.1) and let $\lambda_{11} = \sqrt{\alpha_{11}}$ (which is well-defined by Lemma 13.8), $l_{21} = a_{21}/\lambda_{11}$, and $L_{22} = \text{Chol}A_{22} - l_{21}l_{21}^H$ (which exists as a consequence of the Inductive Hypothesis and Lemma 13.9). Then L is the desired Cholesky factor of A .

By the principle of mathematical induction, the theorem holds.

13.6 Blocked Algorithm

In order to attain high performance, the computation is cast in terms of matrix-matrix multiplication by so-called blocked algorithms. For the Cholesky factorization a blocked version of the algorithm can be derived by partitioning

$$A \rightarrow \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array} \right) \quad \text{and} \quad L \rightarrow \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right),$$

where A_{11} and L_{11} are $b \times b$. By substituting these partitioned matrices into $A = LL^H$ we find that

$$\left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right) \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array} \right)^H = \left(\begin{array}{c|c} L_{11}L_{11}^H & \star \\ \hline L_{21}L_{11}^H & L_{21}L_{21}^H + L_{22}L_{22}^H \end{array} \right).$$

From this we conclude that

$$\begin{array}{c|c} L_{11} = \text{Chol}A_{11} & \star \\ \hline L_{21} = A_{21}L_{11}^{-H} & L_{22} = \text{Chol}A_{22} - L_{21}L_{21}^H \end{array}.$$

An algorithm is then described by the steps

1. Partition $A \rightarrow \left(\begin{array}{c|c} A_{11} & \star \\ \hline A_{21} & A_{22} \end{array} \right)$, where A_{11} is $b \times b$.
2. Overwrite $A_{11} := L_{11} = \text{Chol}A_{11}$.
3. Overwrite $A_{21} := L_{21} = A_{21}L_{11}^{-H}$.
4. Overwrite $A_{22} := A_{22} - L_{21}L_{21}^H$ (updating only the lower triangular part).
5. Continue with $A = A_{22}$. (Back to Step 1.)

Remark 13.10 The Cholesky factorization $A_{11} := L_{11} = \text{Chol}A_{11}$ can be computed with the unblocked algorithm or by calling the blocked Cholesky factorization algorithm recursively.

Remark 13.11 Operations like $L_{21} = A_{21}L_{11}^{-H}$ are computed by solving the equivalent linear system with multiple right-hand sides $L_{11}L_{21}^H = A_{21}^H$.

13.7 Alternative Representation

When explaining the above algorithm in a classroom setting, invariably it is accompanied by a picture sequence like the one in Figure 13.1(left)¹ and the (verbal) explanation:

¹ Picture modified from a similar one in [25].

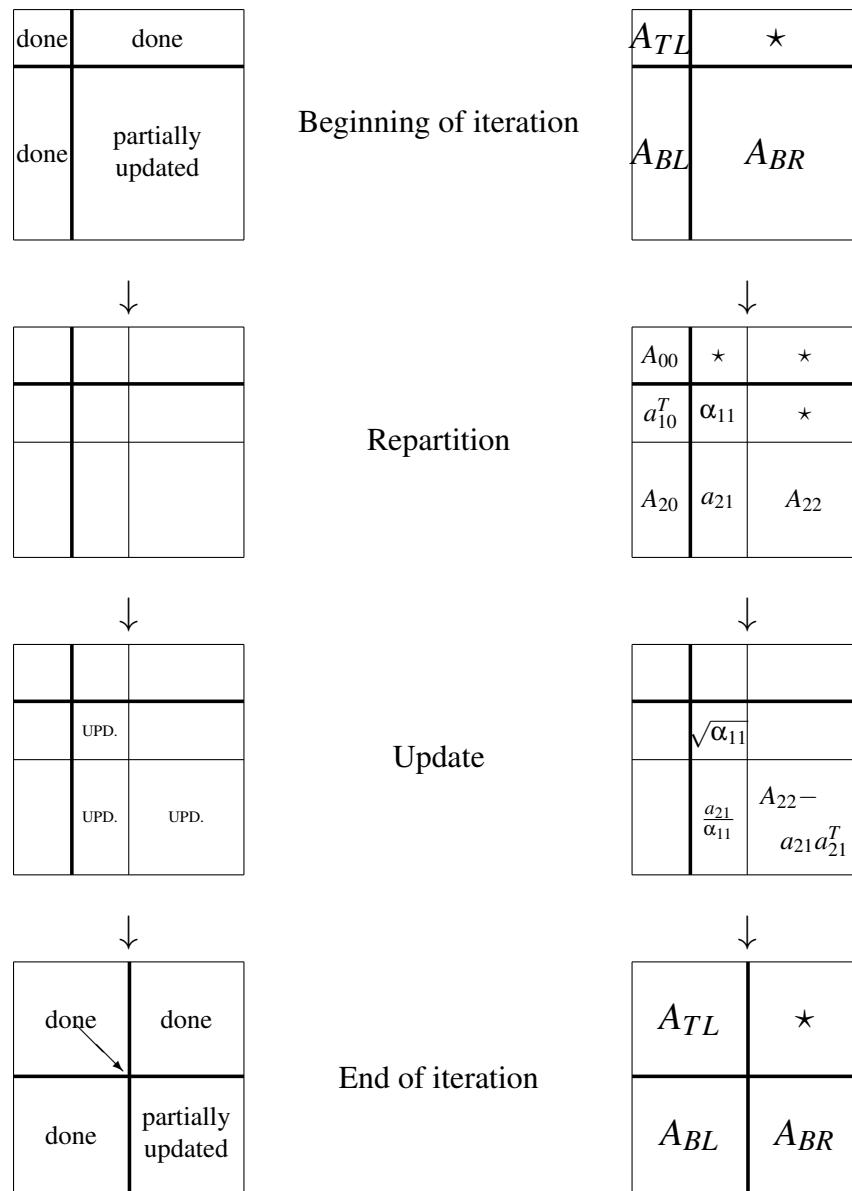


Figure 13.1: Left: Progression of pictures that explain Cholesky factorization algorithm. Right: Same pictures, annotated with labels and updates.

<p>Algorithm: $A := \text{CHOL_UNB}(A)$</p> <p>Partition $A \rightarrow \begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix}$ where A_{TL} is 0×0</p> <p>while $m(A_{TL}) < m(A)$ do</p> <p style="padding-left: 20px;">Repartition</p> $\begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & * & * \\ \hline a_{10}^T & \alpha_{11} & * \\ \hline A_{20} & a_{21} & A_{22} \end{pmatrix}$ <hr/> <p>$\alpha_{11} := \sqrt{\alpha_{11}}$ $a_{21} := a_{21}/\alpha_{11}$ $A_{22} := A_{22} - \text{tril}(a_{21}a_{21}^H)$</p> <p>Continue with</p> $\begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & * & * \\ \hline a_{10}^T & \alpha_{11} & * \\ \hline A_{20} & a_{21} & A_{22} \end{pmatrix}$ <p>endwhile</p>	<p>Algorithm: $A := \text{CHOL_BLK}(A)$</p> <p>Partition $A \rightarrow \begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix}$ where A_{TL} is 0×0</p> <p>while $m(A_{TL}) < m(A)$ do</p> <p style="padding-left: 20px;">Repartition</p> $\begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & * & * \\ \hline A_{10} & A_{11} & * \\ \hline A_{20} & A_{21} & A_{22} \end{pmatrix}$ <hr/> <p>$A_{11} := \text{Chol}A_{11}$ $A_{21} := A_{21} \text{tril}(A_{11})^{-H}$ $A_{22} := A_{22} - \text{tril}(A_{21}A_{21}^H)$</p> <p>Continue with</p> $\begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & * & * \\ \hline A_{10} & A_{11} & * \\ \hline A_{20} & A_{21} & A_{22} \end{pmatrix}$ <p>endwhile</p>
--	---

Figure 13.2: Unblocked and blocked algorithms for computing the Cholesky factorization in FLAME notation.

Beginning of iteration: At some stage of the algorithm (Top of the loop), the computation has moved through the matrix to the point indicated by the thick lines. Notice that we have finished with the parts of the matrix that are in the top-left, top-right (which is not to be touched), and bottom-left quadrants. The bottom-right quadrant has been updated to the point where we only need to perform a Cholesky factorization of it.

Repartition: We now repartition the bottom-right submatrix to expose α_{11} , a_{21} , and A_{22} .

Update: α_{11} , a_{21} , and A_{22} are updated as discussed before.

End of iteration: The thick lines are moved, since we now have completed more of the computation, and only a factorization of A_{22} (which becomes the new bottom-right quadrant) remains to be performed.

Continue: The above steps are repeated until the submatrix A_{BR} is empty.

To motivate our notation, we annotate this progression of pictures as in Figure 13.1 (right). In those pictures, “T”, “B”, “L”, and “R” stand for “Top”, “Bottom”, “Left”, and “Right”, respectively. This then motivates the format of the algorithm in Figure 13.2 (left). It uses what we call the FLAME notation for representing algorithms [25, 24, 40]. A similar explanation can be given for the blocked algorithm, which is given in Figure 13.2 (right). In the algorithms, $m(A)$ indicates the number of rows of matrix A .

Remark 13.12 *The indices in our more stylized presentation of the algorithms are subscripts rather than indices in the conventional sense.*

Remark 13.13 *The notation in Figs. 13.1 and 13.2 allows the contents of matrix A at the beginning of the iteration to be formally stated:*

$$A = \left(\begin{array}{c|c} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L_{TL} & * \\ \hline L_{BL} & \hat{A}_{BR} - \text{tril}(L_{BL}L_{BL}^H) \end{array} \right),$$

where $L_{TL} = \text{Chol}\hat{A}_{TL}$, $L_{BL} = \hat{A}_{BL}L_{TL}^{-H}$, and $\hat{A}_{TL}, \hat{A}_{BL}$ and \hat{A}_{BR} denote the original contents of the quadrants A_{TL}, A_{BL} and A_{BR} , respectively.

Homework 13.14 Implement the Cholesky factorization with M-script.

☞ SEE ANSWER

13.8 Cost

The cost of the Cholesky factorization of $A \in \mathbb{C}^{m \times m}$ can be analyzed as follows: In Figure 13.2 (left) during the k th iteration (starting k at zero) A_{00} is $k \times k$. Thus, the operations in that iteration cost

- $\alpha_{11} := \sqrt{\alpha_{11}}$: negligible when k is large.
- $a_{21} := a_{21}/\alpha_{11}$: approximately $(m - k - 1)$ flops.
- $A_{22} := A_{22} - \text{tril}(a_{21}a_{21}^H)$: approximately $(m - k - 1)^2$ flops. (A rank-1 update of all of A_{22} would have cost $2(m - k - 1)^2$ flops. Approximately half the entries of A_{22} are updated.)

Thus, the total cost in flops is given by

$$\begin{aligned}
 C_{\text{Chol}}(m) &\approx \underbrace{\sum_{k=0}^{m-1} (m-k-1)^2}_{\text{(Due to update of } A_{22})} + \underbrace{\sum_{k=0}^{m-1} (m-k-1)}_{\text{(Due to update of } a_{21})} \\
 &= \sum_{j=0}^{m-1} j^2 + \sum_{j=0}^{m-1} j \approx \frac{1}{3}m^3 + \frac{1}{2}m^2 \approx \frac{1}{3}m^3
 \end{aligned}$$

which allows us to state that (obvious) most computation is in the update of A_{22} . It can be shown that the blocked Cholesky factorization algorithm performs exactly the same number of floating point operations.

Comparing the cost of the Cholesky factorization to that of the LU factorization from “Notes on LU Factorization” we see that taking advantage of symmetry cuts the cost approximately in half.

13.9 Solving the Linear Least-Squares Problem via the Cholesky Factorization

Recall that if $B \in \mathbb{C}^{m \times n}$ has linearly independent columns, then $A = B^H B$ is HPD. Also, recall from “Notes on Linear Least-Squares” that the solution, $x \in \mathbb{C}^n$ to the linear least-squares (LLS) problem

$$\|Bx - y\|_2 = \min_{z \in \mathbb{C}^n} \|Bz - y\|_2$$

equals the solution to the normal equations

$$\underbrace{B^H B}_A x = \underbrace{B^H y}_{\hat{y}} .$$

This makes it obvious how the Cholesky factorization can (and often is) used to solve the LLS problem.

Homework 13.15 Consider $B \in \mathbb{C}^{m \times n}$ with linearly independent columns. Recall that B has a QR factorization, $B = QR$ where Q has orthonormal columns and R is an upper triangular matrix with positive diagonal elements. How are the Cholesky factorization of $B^H B$ and the QR factorization of B related?

 SEE ANSWER

13.10 Other Cholesky Factorization Algorithms

There are actually three different unblocked and three different blocked algorithms for computing the Cholesky factorization. The algorithms we discussed so far in this note are sometimes called *right-looking* algorithms. Systematic derivation of all these algorithms, as well as their blocked counterparts, are given in Chapter 6 of [40]. In this section, a sequence of exercises leads to what is often referred to as the *bordered* Cholesky factorization algorithm.

Algorithm: $A := \text{CHOL_UNB}(A)$ Bordered algorithm)

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{array} \right)$

where A_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1

Continue with

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

endwhile

Figure 13.3: Unblocked Cholesky factorization, bordered variant, for Homework 13.17.

Homework 13.16 Let A be SPD and partition

$$A \rightarrow \left(\begin{array}{c|c} A_{00} & a_{10} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right)$$

(Hint: For this exercise, use techniques similar to those in Section 13.5.)

1. Show that A_{00} is SPD.
2. Assuming that $A_{00} = L_{00}L_{00}^T$, where L_{00} is lower triangular and nonsingular, argue that the assignment $l_{10}^T := a_{10}^T L_{00}^{-T}$ is well-defined.
3. Assuming that A_{00} is SPD, $A_{00} = L_{00}L_{00}^T$ where L_{00} is lower triangular and nonsingular, and $l_{10}^T = a_{10}^T L_{00}^{-T}$, show that $\alpha_{11} - l_{10}^T l_{10} > 0$ so that $\lambda_{11} := \sqrt{\alpha_{11} - l_{10}^T l_{10}}$ is well-defined.
4. Show that

$$\left(\begin{array}{c|c} A_{00} & a_{10} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array} \right) \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array} \right)^T$$

☞ SEE ANSWER

Homework 13.17 Use the results in the last exercise to give an alternative proof by induction of the Cholesky Factorization Theorem and to give an algorithm for computing it by filling in Figure 13.3. This algorithm is often referred to as the bordered Cholesky factorization algorithm.

☞ SEE ANSWER

Homework 13.18 Show that the cost of the bordered algorithm is, approximately, $\frac{1}{3}n^3$ flops.

☞ SEE ANSWER

13.11 Implementing the Cholesky Factorization with the (Traditional) 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 [39].

13.11.1 What are the BLAS?

The BLAS interface [29, 19, 18] 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.

	Algorithm	Code
Simple	<pre> for $j = 1 : n$ $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$ for $i = j + 1 : n$ $\alpha_{i,j} := \alpha_{i,j}/\alpha_{j,j}$ endfor for $k = j + 1 : n$ for $i = k : n$ $\alpha_{i,k} := \alpha_{i,k} - \alpha_{i,j}\alpha_{k,j}$ endfor endfor endfor </pre>	<pre> 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 </pre>
Vector-vector	<pre> for $j = 1 : n$ $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$ $\alpha_{j+1:n,j} := \alpha_{j+1:n,j}/\alpha_{j,j}$ for $k = j + 1 : n$ $\alpha_{k:n,k} := -\alpha_{k,j}\alpha_{k:n,j} + \alpha_{k:n,k}$ endfor endfor </pre>	<pre> 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 </pre>

Figure 13.4: Simple and vector-vector (level-1 BLAS) based representations of the right-looking algorithm.

Algorithm	Code
<p>Matrix-vector</p> <pre> for $j = 1 : n$ $\alpha_{j,j} := \sqrt{\alpha_{j,j}}$ $\alpha_{j+1:n,j} := \alpha_{j+1:n,j} / \alpha_{j,j}$ $\alpha_{j+1:n,j+1:n} := -\text{tril}(\alpha_{j+1:n,j} \alpha_{j+1:n,j}^T) + \alpha_{j+1:n,j+1:n}$ endfor </pre> <p>Partition $A \rightarrow \begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix}$ where A_{TL} is 0×0</p> <p>while $m(A_{TL}) < m(A)$ do</p> <p style="color: blue;">Repartition</p> <p>$\begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & * & * \\ \hline a_{10}^T & \alpha_{11} & * \\ \hline A_{20} & a_{21} & A_{22} \end{pmatrix}$ where α_{11} is 1×1</p> <hr/> <p>$\alpha_{11} := \sqrt{\alpha_{11}}$ $a_{21} := a_{21} / \alpha_{11}$ $A_{22} := A_{22} - \text{tril}(a_{21} a_{21}^H)$</p> <hr/> <p>Continue with</p> <p>$\begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & * & * \\ \hline a_{10}^T & \alpha_{11} & * \\ \hline A_{20} & a_{21} & A_{22} \end{pmatrix}$</p> <p>endwhile</p>	<pre> 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, -1.0, A(j+1,j), 1, A(j+1,j+1), lda) enddo int Chol_unb_var3(FLA_Obj A) { FLA_Obj ATL, ATR, A00, a01, A02, ABL, ABR, a10t, alphall, a12t, a20, a21, A22; FLA_Part_2x2(A, &ATL, &ATR, &ABL, &ABR, 0, 0, FLA_TL); while (FLA_Obj_length(ATL) < FLA_Obj_length(A)){ FLA_Repart_2x2_to_3x3(ATL, /**/ ATR, &A00, /**/ &a01, &A02, /* **** */ /* */ /* **** */ /* **** */ &a10t,/* */ &alphall, &a12t, ABL, /* */ ABR, &A20, /* */ &a21, &A22, 1, 1, FLA_BR); /*-----*/ FLA_Sqrt(alphall); FLA_Invscale(alphall, a21); FLA_Syr(FLA_LOWER_TRIANGULAR, FLA_MINUS_ONE, A21, A22); /*-----*/ FLA_Cont_with_3x3_to_2x2(&ATL, /* */ &ATR, A00, a01, /* */ A02, a10t, alphall, /* */ a12t, /* **** */ /* */ /* **** */ /* **** */ &ABL, /* */ &ABR, A20, a21, /* */ A22, FLA_TL); } return FLA_SUCCESS; } </pre>
FLAME Notation	

Figure 13.5: Matrix-vector (level-2 BLAS) based representations of the right-looking algorithm.

13.11.2 A simple implementation in Fortran

We start with a simple implementation in Fortran. A simple algorithm that does not use BLAS and the corresponding code is given in the row labeled “Simple” in Figure 13.4. This sets the stage for our explanation of how the algorithm and code can be represented with vector-vector, matrix-vector, and matrix-matrix operations, and the corresponding calls to BLAS routines.

13.11.3 Implementation with calls to level-1 BLAS

The first BLAS interface [29] was proposed in the 1970s when vector supercomputers like the early Cray architectures reigned. On such computers, it sufficed to cast computation in terms of vector operations. As long as memory was accessed mostly contiguously, near-peak performance could be achieved. This interface is now referred to as the Level-1 BLAS. It was used for the implementation of the first widely used dense linear algebra package, LINPACK [17].

Let x and y be vectors of appropriate length and α be scalar. In this and other notes we have *vector-vector operations* such as scaling of a vector ($x := \alpha x$), inner (dot) product ($\alpha := x^T y$), and scaled vector addition ($y := \alpha x + y$). This last operation is known as an `axpy`, which stands for `alpha times x plus y`.

Our Cholesky factorization algorithm expressed in terms of such vector-vector operations and the corresponding code are given in Figure 13.4 in the row labeled “Vector-vector”. If the operations supported by `dscal` and `daxpy` achieve high performance on a target architecture (as it was in the days of vector supercomputers) then so will the implementation of the Cholesky factorization, since it casts most computation in terms of those operations. Unfortunately, vector-vector operations perform $O(n)$ computation on $O(n)$ data, meaning that these days the bandwidth to memory typically limits performance, since retrieving a data item from memory is often more than an order of magnitude more costly than a floating point operation with that data item.

We summarize information about level-1 BLAS in Figure 13.6.

13.11.4 Matrix-vector operations (level-2 BLAS)

The next level of BLAS supports operations with matrices and vectors. The simplest example of such an operation is the matrix-vector product: $y := Ax$ where x and y are vectors and A is a matrix. Another example is the computation $A_{22} = -a_{21}a_{21}^T + A_{22}$ (symmetric rank-1 update) in the Cholesky factorization. Here only the lower (or upper) triangular part of the matrix is updated, taking advantage of symmetry.

The use of symmetric rank-1 update is illustrated in Figure 13.5, in the row labeled “Matrix-vector”. There `dsyr` is the routine that implements a double precision symmetric rank-1 update. Readability of the code is improved by casting computation in terms of routines that implement the operations that appear in the algorithm: `dscal` for $a_{21} = a_{21}/\alpha_{11}$ and `dsyr` for $A_{22} = -a_{21}a_{21}^T + A_{22}$.

If the operation supported by `dsyr` achieves high performance on a target architecture (as it was in the days of vector supercomputers) then so will this implementation of the Cholesky factorization, since it casts most computation in terms of that operation. Unfortunately, matrix-vector operations perform $O(n^2)$ computation on $O(n^2)$ data, meaning that these days the bandwidth to memory typically limits performance.

We summarize information about level-2 BLAS in Figure 13.7.

A proto-typical calling sequence for a level-1 BLAS routine is

$\square\text{axpy}(n, \alpha, x, \text{incx}, y, \text{incy}),$

which implements the scaled vector addition operation $y = \alpha x + y$. Here

- The “ \square ” indicates the data type. The choices for this first letter are

s	single precision
d	double precision
c	single precision complex
z	double precision complex

- The operation is identified as axpy: alpha times x plus y.
- n indicates the number of elements in the vectors x and y.
- alpha is the scalar α .
- x and y indicate the memory locations where the first elements of x and y are stored, respectively.
- incx and incy equal the increment by which one has to stride through memory for the elements of vectors x and y, respectively

The following are the most frequently used level-1 BLAS:

routine/ function	operation
$\square\text{swap}$	$x \leftrightarrow y$
$\square\text{scal}$	$x \leftarrow \alpha x$
$\square\text{copy}$	$y \leftarrow x$
$\square\text{axpy}$	$y \leftarrow \alpha x + y$
$\square\text{dot}$	$x^T y$
$\square\text{nrm2}$	$\ x\ _2$
$\square\text{asum}$	$\ \text{re}(x)\ _1 + \ \text{im}(x)\ _1$
$i\square\text{max}$	$\min(k) : \text{re}(x_k) + \text{im}(x_k) = \max(\text{re}(x_i) + \text{im}(x_i))$

Figure 13.6: Summary of the most commonly used level-1 BLAS.

The naming convention for level-2 BLAS routines is given by

$$\square \text{XXYY},$$

where

- “ \square ” can take on the values s, d, c, z.
- XX indicates the shape of the matrix.
- YY indicates the operation to be performed:

XX	matrix shape	YY	matrix shape
ge	general (rectangular)	mv	matrix <u>v</u> ector multiplication
sy	symmetric	sv	solve <u>v</u> ector
he	Hermitian	r	rank-1 update
tr	triangular	r2	rank-2 update

- In addition, operations with banded matrices are supported, which we do not discuss here.

A representative call to a level-2 BLAS operation is given by

```
dsyr( uplo, n, alpha, x, incx, A, lda )
```

which implements the operation $A = \alpha xx^T + A$, updating the lower or upper triangular part of A by choosing uplo as ‘Lower triangular’ or ‘Upper triangular’, respectively. The parameter lda (the leading dimension of matrix A) indicates the increment by which memory has to be traversed in order to address successive elements in a row of matrix A.

The following table gives the most commonly used level-2 BLAS operations:

routine/ function	operation
\square gemv	general <u>m</u> atrix- <u>v</u> ector multiplication
\square symv	symmetric <u>m</u> atrix- <u>v</u> ector multiplication
\square trmv	triangular <u>m</u> atrix- <u>v</u> ector multiplication
\square trsv	triangular <u>s</u> olve <u>v</u> ector
\square ger	general <u>r</u> ank-1 update
\square syr	symmetric <u>r</u> ank-1 update
\square syr2	symmetric <u>r</u> ank-2 update

Figure 13.7: Summary of the most commonly used level-2 BLAS.

	Algorithm	Code
Matrix-matrix	<pre> for $j = 1 : n$ in steps of n_b $b := \min(n - j + 1, n_b)$ $A_{j:j+b-1, j:j+b-1} := \text{Chol} A_{j:j+b-1, j:j+b-1}$ $A_{j+b:n, j:j+b-1} :=$ $A_{j+b:n, j:j+b-1} A_{j:j+b-1}^{-H}$ $A_{j+b:n, j:b:n} := A_{j+b:n, j:b:n}$ $- \text{tril} \left(A_{j+b:n, j:j+b-1} A_{j+b:n, j:j+b-1}^H \right)$ endfor </pre>	<pre> do j=1, n, nb jb = min(nb, n-j+1) call chol(jb, A(j, j), lda) call dtrsm('Right', 'Lower triangular', 'Transpose', 'Nonunit diag', J-JB+1, JB, 1.0d00, A(j, j), lda, A(j+jb, j), lda) call dsyrk('Lower triangular', 'No transpose', J-JB+1, JB, -1.0d00, A(j+jb, j), lda, 1.0d00, A(j+jb, j+jb), lda) enddo </pre>
FLAME Notation	<p>Partition $A \rightarrow \begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix}$ where A_{TL} is 0×0</p> <p>while $m(A_{TL}) < m(A)$ do</p> <p style="color: blue;">Determine block size b</p> <p style="color: blue;">Repartition</p> <p style="text-align: center;"> $\begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & * & * \\ \hline A_{10} & A_{11} & * \\ \hline A_{20} & A_{21} & A_{22} \end{pmatrix}$ where A_{11} is $b \times b$ </p> <hr/> <p>$A_{11} := \text{Chol} A_{11}$ $A_{21} := A_{21} \text{tril}(A_{11})^{-H}$ $A_{22} := A_{22} - \text{tril}(A_{21} A_{21}^H)$</p> <hr/> <p>Continue with</p> <p style="text-align: center;"> $\begin{pmatrix} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & * & * \\ \hline A_{10} & A_{11} & * \\ \hline A_{20} & A_{21} & A_{22} \end{pmatrix}$ </p> <p>endwhile</p>	<pre> int Chol_unb_var3(FLA_Obj A) { FLA_Obj ATL, ATR, A00, a01, A02, ABL, ABR, a10t, alphai1, a12t, A20, a21, A22; FLA_Part_2x2(A, &ATL, &ATR, &ABL, &ABR, 0, 0, FLA_TL); while (FLA_Obj_length(ATL) < FLA_Obj_length(A)){ FLA_Repart_2x2_to_3x3(ATL, /* */ ATR, &A00, /* */ &a01, /* */ &A02, /* **** */ /* */ /* **** */ /* */ &a10t, /* */ &alphai1, &a12t, ABL, /* */ ABR, &A20, /* */ &a21, /* */ &A22, 1, 1, FLA_BR); /*-----*/ FLA_Sqrt(alphai1); FLA_Invscal(alphai1, a21); FLA_Syr(FLA_LOWER_TRIANGULAR, FLA_MINUS_ONE, A21, A22); /*-----*/ FLA_Cont_with_3x3_to_2x2(&ATL, /* */ &ATR, A00, a01, /* */ A02, a10t, alphai1, /* */ a12t, /* **** */ /* */ /* **** */ /* */ &ABL, /* */ &ABR, A20, a21, /* */ A22, FLA_TL); } return FLA_SUCCESS; } </pre>

Figure 13.8: Blocked algorithm and implementation with level-3 BLAS.

The naming convention for level-3 BLAS routines are similar to those for the level-2 BLAS. A representative call to a level-3 BLAS operation is given by

```
dsyrk( uplo, trans, n, k, alpha, A, lda, beta, C, ldc )
```

which implements the operation $C := \alpha AA^T + \beta C$ or $C := \alpha A^T A + \beta C$ depending on whether `trans` is chosen as 'No transpose' or 'Transpose', respectively. It updates the lower or upper triangular part of C depending on whether `uplo` equal 'Lower triangular' or 'Upper triangular', respectively. The parameters `lda` and `ldc` are the leading dimensions of arrays A and C , respectively.

The following table gives the most commonly used Level-3 BLAS operations

routine/ function	operation
<code>gemm</code>	<u>general</u> <u>matrix-matrix</u> multiplication
<code>symm</code>	<u>symmetric</u> <u>matrix-matrix</u> multiplication
<code>trmm</code>	<u>triangular</u> <u>matrix-matrix</u> multiplication
<code>trsm</code>	<u>triangular</u> <u>solve with</u> <u>multiple right-hand sides</u>
<code>dsyrk</code>	<u>symmetric</u> <u>rank-k</u> update
<code>dsyr2k</code>	<u>symmetric</u> <u>rank-2k</u> update

Figure 13.9: Summary of the most commonly used level-3 BLAS.

13.11.5 Matrix-matrix operations (level-3 BLAS)

Finally, we turn to the implementation of the blocked Cholesky factorization algorithm from Section 13.6. The algorithm is expressed with FLAME notation and Matlab-like notation in Figure 13.8.

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 $A_{22} := -L_{21}L_{21}^T + A_{22}$.

The bulk of the computation is now cast in terms of matrix-matrix operations which can achieve high performance.

We summarize information about level-3 BLAS in Figure 13.9.

13.11.6 Impact on performance

Figure 13.10 illustrates the performance benefits that come from using the different levels of BLAS on a typical architecture.

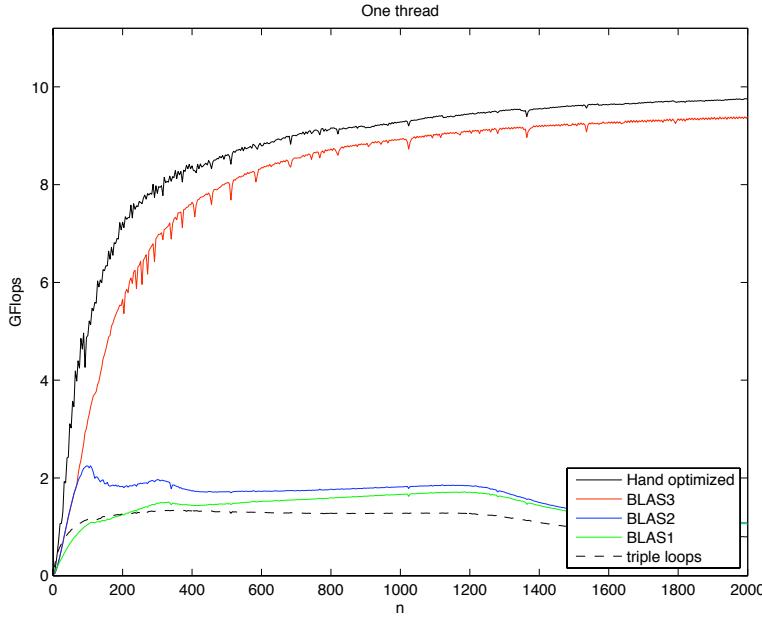


Figure 13.10: Performance of the different implementations of Cholesky factorization that use different levels of BLAS. The target processor has a peak of 11.2 Gflops (billions of floating point operations per second). BLAS1, BLAS2, and BLAS3 indicate that the bulk of computation was cast in terms of level-1, -2, or -3 BLAS, respectively.

13.12 Alternatives to the BLAS

13.12.1 The FLAME/C API

In a number of places in these notes we presented algorithms in FLAME notation. Clearly, there is a disconnect between this notation and how the algorithms are then encoded with the BLAS interface. In Figures 13.4, 13.5, and 13.8 we also show how the FLAME API for the C programming language [6] allows the algorithms to be more naturally translated into code. While the traditional BLAS interface underlies the implementation of Cholesky factorization and other algorithms in the widely used LAPACK library [1], the FLAME/C API is used in our `libflame` library [25, 41, 42].

13.12.2 BLIS

The implementations that call BLAS in this paper are coded in Fortran. More recently, the languages of choice for scientific computing have become C and C++. While there is a C interface to the traditional BLAS called the CBLAS [11], we believe a more elegant such interface is the BLAS-like Library Instantiation Software (BLIS) interface [43]. BLIS is not only a framework for rapid implementation of the traditional BLAS, but also presents an alternative interface for C and C++ users.

13.13 Wrapup

13.13.1 Additional exercises

13.13.2 Summary

Chapter **14**

Notes on Eigenvalues and Eigenvectors

If you have forgotten how to find the eigenvalues and eigenvectors of 2×2 and 3×3 matrices, you may want to review

[Linear Algebra: Foundations to Frontiers - Notes to LAFF With \[30\].](#)

Video

Read disclaimer regarding the videos in the preface!

- 👉 [YouTube](#)
- 👉 [Download from UT Box](#)
- 👉 [View After Local Download](#)

(For help on viewing, see Appendix [A](#).)

14.0.1 Outline

Video	279
14.0.1 Outline	280
14.1 Definition	280
14.2 The Schur and Spectral Factorizations	283
14.3 Relation Between the SVD and the Spectral Decomposition	285

14.1 Definition

Definition 14.1 Let $A \in \mathbb{C}^{m \times m}$. Then $\lambda \in \mathbb{C}$ and nonzero $x \in \mathbb{C}^m$ are said to be an eigenvalue and corresponding eigenvector if $Ax = \lambda x$. The tuple (λ, x) is said to be an eigenpair. The set of all eigenvalues of A is denoted by $\Lambda(A)$ and is called the spectrum of A . The spectral radius of A , $\rho(A)$, equals the magnitude of the largest eigenvalue, in magnitude:

$$\rho(A) = \max_{\lambda \in \Lambda(A)} |\lambda|.$$

The action of A on an eigenvector x is as if it were multiplied by a scalar. The direction does not change, only its length is scaled:

$$Ax = \lambda x.$$

Theorem 14.2 Scalar λ is an eigenvalue of A if and only if

$$(\lambda I - A) \left\{ \begin{array}{l} \text{is singular} \\ \text{has a nontrivial null-space} \\ \text{has linearly dependent columns} \\ \det(\lambda I - A) = 0 \\ (\lambda I - A)x = 0 \text{ has a nontrivial solution} \\ \text{etc.} \end{array} \right.$$

The following exercises expose some other basic properties of eigenvalues and eigenvectors:

Homework 14.3 Eigenvectors are not unique.

 [SEE ANSWER](#)

Homework 14.4 Let λ be an eigenvalue of A and let $E_\lambda(A) = \{x \in \mathbb{C}^m | Ax = \lambda x\}$ denote the set of all eigenvectors of A associated with λ (including the zero vector, which is not really considered an eigenvector). Show that this set is a (nontrivial) subspace of \mathbb{C}^m .

 [SEE ANSWER](#)

Definition 14.5 Given $A \in \mathbb{C}^{m \times m}$, the function $p_m(\lambda) = \det(\lambda I - A)$ is a polynomial of degree at most m . This polynomial is called the characteristic polynomial of A .

The definition of $p_m(\lambda)$ and the fact that it is a polynomial of degree at most m is a consequence of the definition of the determinant of an arbitrary square matrix. This definition is not particularly enlightening other than that it allows one to succinctly relate eigenvalues to the roots of the characteristic polynomial.

Remark 14.6 *The relation between eigenvalues and the roots of the characteristic polynomial yield a disconcerting insight: A general formula for the eigenvalues of a $m \times m$ matrix with $m > 4$ does not exist.*

The reason is that there is no general formula for the roots of a polynomial of degree $m > 4$. Given any polynomial $p_m(\chi)$ of degree m , an $m \times m$ matrix can be constructed such that its characteristic polynomial is $p_m(\lambda)$. If

$$p_m(\chi) = \alpha_0 + \alpha_1\chi + \cdots + \alpha_{m-1}\chi^{m-1} + \chi^m$$

and

$$A = \begin{pmatrix} -\alpha_{n-1} & -\alpha_{n-2} & -\alpha_{n-3} & \cdots & -\alpha_1 & -\alpha_0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

then

$$p_m(\lambda) = \det(\lambda I - A)$$

Hence, we conclude that no general formula can be found for the eigenvalues for $m \times m$ matrices when $m > 4$. What we will see in future “Notes on ...” is that we will instead create algorithms that *converge* to the eigenvalues and/or eigenvectors of matrices.

Theorem 14.7 *Let $A \in \mathbb{C}^{m \times m}$ and $p_m(\lambda)$ be its characteristic polynomial. Then $\lambda \in \Lambda(A)$ if and only if $p_m(\lambda) = 0$.*

Proof: This is an immediate consequence of Theorem 14.2.

In other words, λ is an eigenvalue of A if and only if it is a root of $p_m(\lambda)$. This has the immediate consequence that A has at most m eigenvalues and, if one counts multiple roots by their multiplicity, it has exactly m eigenvalues. (One says “Matrix $A \in \mathbb{C}^{m \times m}$ has m eigenvalues, multiplicity counted.”)

Homework 14.8 *The eigenvalues of a diagonal matrix equal the values on its diagonal. The eigenvalues of a triangular matrix equal the values on its diagonal.*

SEE ANSWER

Corollary 14.9 *If $A \in \mathbb{R}^{m \times m}$ is real valued then some or all of its eigenvalues may be complex valued. In this case, if $\lambda \in \Lambda(A)$ then so is its conjugate, $\bar{\lambda}$.*

Proof: It can be shown that if A is real valued, then the coefficients of its characteristic polynomial are all real valued. Complex roots of a polynomial with real coefficients come in conjugate pairs.

It is not hard to see that an eigenvalue that is a root of multiplicity k has at most k eigenvectors. It is, however, not necessarily the case that an eigenvalue that is a root of multiplicity k also has k linearly

independent eigenvectors. In other words, the null space of $\lambda I - A$ may have dimension less than the algebraic multiplicity of λ . The prototypical counter example is the $k \times k$ matrix

$$J(\mu) = \begin{pmatrix} \mu & 1 & 0 & \cdots & 0 & 0 \\ 0 & \mu & 1 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & \mu & 1 \\ 0 & 0 & 0 & \cdots & 0 & \mu \end{pmatrix}$$

where $k > 1$. Observe that $\lambda I - J(\mu)$ is singular if and only if $\lambda = \mu$. Since $\mu I - J(\mu)$ has $k - 1$ linearly independent columns its null-space has dimension one: all eigenvectors are scalar multiples of each other. This matrix is known as a *Jordan block*.

Definition 14.10 A matrix $A \in \mathbb{C}^{m \times m}$ that has fewer than m linearly independent eigenvectors is said to be *defective*. A matrix that does have m linearly independent eigenvectors is said to be *nondefective*.

Theorem 14.11 Let $A \in \mathbb{C}^{m \times m}$. There exist nonsingular matrix X and diagonal matrix Λ such that $A = X\Lambda X^{-1}$ if and only if A is nondefective.

Proof:

(\Rightarrow). Assume there exist nonsingular matrix X and diagonal matrix Λ so that $A = X\Lambda X^{-1}$. Then, equivalently, $AX = X\Lambda$. Partition X by columns so that

$$\begin{aligned} A \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right) &= \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right) \left(\begin{array}{c|c|c|c} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{m-1} \end{array} \right) \\ &= \left(\begin{array}{c|c|c|c} \lambda_0 x_0 & \lambda_1 x_1 & \cdots & \lambda_{m-1} x_{m-1} \end{array} \right). \end{aligned}$$

Then, clearly, $Ax_j = \lambda_j x_j$ so that A has m linearly independent eigenvectors and is thus nondefective.

(\Leftarrow). Assume that A is nondefective. Let $\{x_0, \dots, x_{m-1}\}$ equal m linearly independent eigenvectors corresponding to eigenvalues $\{\lambda_0, \dots, \lambda_{m-1}\}$. If $X = \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right)$ then $AX = X\Lambda$ where $\Lambda = \text{diag}(\lambda_0, \dots, \lambda_{m-1})$. Hence $A = X\Lambda X^{-1}$.

Definition 14.12 Let $\mu \in \Lambda(A)$ and $p_m(\lambda)$ be the characteristic polynomial of A . Then the algebraic multiplicity of μ is defined as the multiplicity of μ as a root of $p_m(\lambda)$.

Definition 14.13 Let $\mu \in \Lambda(A)$. Then the geometric multiplicity of μ is defined to be the dimension of $E_\mu(A)$. In other words, the geometric multiplicity of μ equals the number of linearly independent eigenvectors that are associated with μ .

Theorem 14.14 Let $A \in \mathbb{C}^{m \times m}$. Let the eigenvalues of A be given by $\lambda_0, \lambda_1, \dots, \lambda_{k-1}$, where an eigenvalue is listed exactly n times if it has geometric multiplicity n . There exists a nonsingular matrix X such that

$$A = X \begin{pmatrix} J(\lambda_0) & & & & \\ \hline & 0 & & \cdots & 0 \\ \hline & & J(\lambda_1) & \cdots & 0 \\ \hline & \vdots & \vdots & \ddots & \vdots \\ \hline & 0 & 0 & \cdots & J(\lambda_{k-1}) \end{pmatrix}.$$

For our discussion, the sizes of the Jordan blocks $J(\lambda_i)$ are not particularly important. Indeed, this decomposition, known as the Jordan Canonical Form of matrix A , is not particularly interesting in practice. For this reason, we don't discuss it further and do not give its proof.

14.2 The Schur and Spectral Factorizations

Theorem 14.15 Let $A, Y, B \in \mathbb{C}^{m \times m}$, assume Y is nonsingular, and let $B = Y^{-1}AY$. Then $\Lambda(A) = \Lambda(B)$.

Proof: Let $\lambda \in \Lambda(A)$ and x be an associated eigenvector. Then $Ax = \lambda x$ if and only if $Y^{-1}AYY^{-1}x = Y^{-1}\lambda x$ if and only if $B(Y^{-1}x) = \lambda(Y^{-1}x)$.

Definition 14.16 Matrices A and B are said to be similar if there exists a nonsingular matrix Y such that $B = Y^{-1}AY$.

Given a nonsingular matrix Y the transformation $Y^{-1}AY$ is called a similarity transformation of A .

It is not hard to expand the last proof to show that if A is similar to B and $\lambda \in \Lambda(A)$ has algebraic/geometric multiplicity k then $\lambda \in \Lambda(B)$ has algebraic/geometric multiplicity k .

The following is the fundamental theorem for the algebraic eigenvalue problem:

Theorem 14.17 Schur Decomposition Theorem Let $A \in \mathbb{C}^{m \times m}$. Then there exist a unitary matrix Q and upper triangular matrix U such that $A = QUQ^H$. This decomposition is called the Schur decomposition of matrix A .

In the above theorem, $\Lambda(A) = \Lambda(U)$ and hence the eigenvalues of A can be found on the diagonal of U .

Proof: We will outline how to construct Q so that $Q^H A Q = U$, an upper triangular matrix.

Since a polynomial of degree m has at least one root, matrix A has at least one eigenvalue, λ_1 , and corresponding eigenvector q_1 , where we normalize this eigenvector to have length one. Thus $Aq_1 = \lambda_1 q_1$. Choose Q_2 so that $Q = \begin{pmatrix} q_1 & | & Q_2 \end{pmatrix}$ is unitary. Then

$$\begin{aligned} Q^H A Q &= \left(\begin{array}{c|c} q_1 & Q_2 \end{array} \right)^H A \left(\begin{array}{c|c} q_1 & Q_2 \end{array} \right) \\ &= \left(\begin{array}{c|c} q_1^H A q_1 & q_1^H A Q_2 \\ \hline Q_2^H A q_1 & Q_2^H A Q_2 \end{array} \right) = \left(\begin{array}{c|c} \lambda_1 & q_1^H A Q_2 \\ \hline \lambda Q_2^H q_1 & Q_2^H A Q_2 \end{array} \right) = \left(\begin{array}{c|c} \lambda_1 & w^T \\ \hline 0 & B \end{array} \right), \end{aligned}$$

where $w^T = q_1^H A Q_2$ and $B = Q_2^H A Q_2$. This insight can be used to construct an inductive proof.

One should not mistake the above theorem and its proof as a constructive way to compute the Schur decomposition: finding an eigenvalue and/or the eigenvalue associated with it is difficult.

Lemma 14.18 Let $A \in \mathbb{C}^{m \times m}$ be of form $A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right)$. Assume that Q_{TL} and Q_{BR} are unitary “of appropriate size”. Then

$$A = \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right)^H \left(\begin{array}{c|c} Q_{TL}A_{TL}Q_{TL}^H & Q_{TL}A_{TR}Q_{BR}^H \\ \hline 0 & Q_{BR}A_{BR}Q_{BR}^H \end{array} \right) \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right).$$

Homework 14.19 Prove Lemma 14.18. Then generalize it to a result for block upper triangular matrices:

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

SEE ANSWER

Corollary 14.20 Let $A \in \mathbb{C}^{m \times m}$ be of for $A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right)$. Then $\Lambda(A) = \Lambda(A_{TL}) \cup \Lambda(A_{BR})$.

Homework 14.21 Prove Corollary 14.20. Then generalize it to a result for block upper triangular matrices.

SEE ANSWER

A theorem that will later allow the eigenvalues and vectors of a real matrix to be computed (mostly) without requiring complex arithmetic is given by

Theorem 14.22 Let $A \in \mathbb{R}^{m \times m}$. Then there exist a unitary matrix $Q \in \mathbb{R}^{m \times m}$ and quasi upper triangular matrix $U \in \mathbb{R}^{m \times m}$ such that $A = QUQ^T$.

A quasi upper triangular matrix is a block upper triangular matrix where the blocks on the diagonal are 1×1 or 2×2 . Complex eigenvalues of A are found as the complex eigenvalues of those 2×2 blocks on the diagonal.

Theorem 14.23 Spectral Decomposition Theorem Let $A \in \mathbb{C}^{m \times m}$ be Hermitian. Then there exist a unitary matrix Q and diagonal matrix $\Lambda \in \mathbb{R}^{m \times m}$ such that $A = Q\Lambda Q^H$. This decomposition is called the Spectral decomposition of matrix A .

Proof: From the Schur Decomposition Theorem we know that there exist a matrix Q and upper triangular matrix U such that $A = QUQ^H$. Since $A = A^H$ we know that $QUQ^H = QU^HQ^H$ and hence $U = U^H$. But a Hermitian triangular matrix is diagonal with real valued diagonal entries.

What we conclude is that a Hermitian matrix is nondefective and its eigenvectors can be chosen to form an orthogonal basis.

Homework 14.24 Let A be Hermitian and λ and μ be distinct eigenvalues with eigenvectors x_λ and x_μ , respectively. Then $x_\lambda^H x_\mu = 0$. (In other words, the eigenvectors of a Hermitian matrix corresponding to distinct eigenvalues are orthogonal.)

SEE ANSWER

14.3 Relation Between the SVD and the Spectral Decomposition

Homework 14.25 Let $A \in \mathbb{C}^{m \times m}$ be a Hermitian matrix, $A = Q\Lambda Q^H$ its Spectral Decomposition, and $A = U\Sigma V^H$ its SVD. Relate Q , U , V , Λ , and Σ .

 [SEE ANSWER](#)

Homework 14.26 Let $A \in \mathbb{C}^{m \times m}$ and $A = U\Sigma V^H$ its SVD. Relate the Spectral decompositions of $A^H A$ and AA^H to U , V , and Σ .

 [SEE ANSWER](#)

Chapter **15**

Notes on the Power Method and Related Methods

You may want to review Chapter 12 of

[Linear Algebra: Foundations to Frontiers - Notes to LAFF With \[30\]](#)

in which the Power Method and Inverse Power Methods are discussed at a more rudimentary level.

Video

Read disclaimer regarding the videos in the preface!

Tragically, I forgot to turn on the camera... This was a great lecture!

15.0.1 Outline

Video	287
15.0.1 Outline	288
15.1 The Power Method	288
15.1.1 First attempt	289
15.1.2 Second attempt	289
15.1.3 Convergence	290
15.1.4 Practical Power Method	293
15.1.5 The Rayleigh quotient	294
15.1.6 What if $\lambda_0 \geq \lambda_1$?	294
15.2 The Inverse Power Method	294
15.3 Rayleigh-quotient Iteration	295

15.1 The Power Method

The Power Method is a simple method that under mild conditions yields a vector corresponding to the eigenvalue that is largest in magnitude.

Throughout this section we will assume that a given matrix $A \in \mathbb{C}^{m \times m}$ is *nondeficient*: there exists a nonsingular matrix X and diagonal matrix Λ such that $A = X\Lambda X^{-1}$. (Sometimes this is called a diagonalizable matrix since there exists a matrix X so that

$$X^{-1}AX = \Lambda \text{ or, equivalently, } A = X\Lambda X^{-1}.$$

From “Notes on Eigenvalues and Eigenvectors” we know then that the columns of X equal eigenvectors of A and the elements on the diagonal of Λ equal the eigenvalues::

$$X = \left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{m-1} \end{array} \right) \quad \text{and} \quad \Lambda = \left(\begin{array}{c|c|c|c} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{m-1} \end{array} \right)$$

so that

$$Ax_i = \lambda_i x_i \quad \text{for } i = 0, \dots, m-1.$$

For most of this section we will assume that

$$|\lambda_0| > |\lambda_1| \geq \cdots \geq |\lambda_{m-1}|.$$

In particular, λ_0 is the eigenvalue with maximal absolute value.

15.1.1 First attempt

Now, let $v^{(0)} \in \mathbb{C}^{m \times m}$ be an “initial guess”. Our (first attempt at the) Power Method iterates as follows:

```
for k = 0, ...
    v^{(k+1)} = Av^{(k)}
endfor
```

Clearly $v^{(k)} = A^k v^{(0)}$. Let

$$v^{(0)} = Xy = \psi_0 x_0 + \psi_1 x_1 + \cdots + \psi_{m-1} x_{m-1}.$$

What does this mean? We view the columns of X as forming a basis for \mathbb{C}^m and then the elements in vector $y = X^{-1}v^{(0)}$ equal the coefficients for describing $v^{(0)}$ in that basis. Then

$$\begin{aligned} v^{(1)} = Av^{(0)} &= A(\psi_0 x_0 + \psi_1 x_1 + \cdots + \psi_{m-1} x_{m-1}) \\ &= \psi_0 \lambda_0 x_0 + \psi_1 \lambda_0 x_1 + \cdots + \psi_{m-1} \lambda_{m-1} x_{m-1}, \\ v^{(2)} = Av^{(1)} &= \psi_0 \lambda_0^2 x_0 + \psi_1 \lambda_1^2 x_1 + \cdots + \psi_{m-1} \lambda_{m-1}^2 x_{m-1}, \\ &\vdots \\ v^{(k)} = Av^{(k-1)} &= \psi_0 \lambda_0^k x_0 + \psi_1 \lambda_1^k x_1 + \cdots + \psi_{m-1} \lambda_{m-1}^k x_{m-1}. \end{aligned}$$

Now, as long as $\psi_0 \neq 0$ clearly $\psi_0 \lambda_0^k x_0$ will eventually dominate which means that $v^{(k)}$ will start pointing in the direction of x_0 . In other words, it will start pointing in the direction of an eigenvector corresponding to λ_0 . The problem is that it will become infinitely long if $|\lambda_0| > 1$ or infinitesimally short if $|\lambda_0| < 1$. All is good if $|\lambda_0| = 1$.

15.1.2 Second attempt

Again, let $v^{(0)} \in \mathbb{C}^{m \times m}$ be an “initial guess”. The second attempt at the Power Method iterates as follows:

```
for k = 0, ...
    v^{(k+1)} = Av^{(k)} / \lambda_0
endfor
```

It is not hard to see that then

$$\begin{aligned} v^{(k)} &= Av^{(k-1)} / \lambda_0 = A^k v^{(0)} / \lambda_0^k \\ &= \psi_0 \left(\frac{\lambda_0}{\lambda_0} \right)^k x_0 + \psi_1 \left(\frac{\lambda_1}{\lambda_0} \right)^k x_1 + \cdots + \psi_{m-1} \left(\frac{\lambda_{m-1}}{\lambda_0} \right)^k x_{m-1} \\ &= \psi_0 x_0 + \psi_1 \left(\frac{\lambda_1}{\lambda_0} \right)^k x_1 + \cdots + \psi_{m-1} \left(\frac{\lambda_{m-1}}{\lambda_0} \right)^k x_{m-1}. \end{aligned}$$

Clearly $\lim_{k \rightarrow \infty} v^{(k)} = \psi_0 x_0$, as long as $\psi_0 \neq 0$, since $\left| \frac{\lambda_k}{\lambda_0} \right| < 1$ for $k > 0$.

Another way of stating this is to notice that

$$A^k = \underbrace{(AA \cdots A)}_{k \text{ times}} = \underbrace{(X \Lambda X^{-1})(X \Lambda X^{-1}) \cdots (X \Lambda X^{-1})}_{\Lambda^k} = X \Lambda^k X^{-1}.$$

so that

$$\begin{aligned}
 v^{(k)} &= A^k v^{(0)} / \lambda_0^k \\
 &= A^k X y / \lambda_0^k \\
 &= X \Lambda^k X^{-1} X y / \lambda_0^k \\
 &= X \Lambda^k y / \lambda_0^k \\
 &= X \left(\Lambda^k / \lambda_0^k \right) y = X \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \left(\frac{\lambda_1}{\lambda_0} \right)^k & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0} \right)^k \end{pmatrix} y.
 \end{aligned}$$

Now, since $\left| \frac{\lambda_k}{\lambda_0} \right| < 1$ for $k > 1$ we can argue that

$$\begin{aligned}
 \lim_{k \rightarrow \infty} v^{(k)} &= \lim_{k \rightarrow \infty} X \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \left(\frac{\lambda_1}{\lambda_0} \right)^k & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0} \right)^k \end{pmatrix} y = X \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} y \\
 &= X \psi_0 e_0 = \psi_0 X e_0 = \psi_0 x_0.
 \end{aligned}$$

Thus, as long as $\psi_0 \neq 0$ (which means v must have a component in the direction of x_0) this method will eventually yield a vector in the direction of x_0 . However, this time the problem is that we don't know λ_0 when we start.

15.1.3 Convergence

Before we make the algorithm practical, let us examine how fast the iteration converges. This requires a few definitions regarding rates of convergence.

Definition 15.1 Let $\alpha_0, \alpha_1, \alpha_2, \dots \in \mathbb{C}$ be an infinite sequence of scalars. Then α_k is said to converge to α if

$$\lim_{k \rightarrow \infty} |\alpha_k - \alpha| = 0.$$

Let $x_0, x_1, x_2, \dots \in \mathbb{C}^m$ be an infinite sequence of vectors. Then x_k is said to converge to x in the $\|\cdot\|$ norm if

$$\lim_{k \rightarrow \infty} \|x_k - x\| = 0.$$

Notice that because of the equivalence of norms, if the sequence converges in one norm, it converges in all norms.

Definition 15.2 Let $\alpha_0, \alpha_1, \alpha_2, \dots \in \mathbb{C}$ be an infinite sequence of scalars that converges to α . Then

- α_k is said to converge linearly to α if for large enough k

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|$$

for some constant $C < 1$.

- α_k is said to converge super-linearly to α if

$$|\alpha_{k+1} - \alpha| \leq C_k |\alpha_k - \alpha|$$

with $C_k \rightarrow 0$.

- α_k is said to converge quadratically to α if for large enough k

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|^2$$

for some constant C .

- α_k is said to converge super-quadratically to α if

$$|\alpha_{k+1} - \alpha| \leq C_k |\alpha_k - \alpha|^2$$

with $C_k \rightarrow 0$.

- α_k is said to converge cubically to α if for large enough k

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|^3$$

for some constant C .

Linear convergence can be slow. Let's say that for $k \geq K$ we observe that

$$|\alpha_{k+1} - \alpha| \leq C|\alpha_k - \alpha|.$$

Then, clearly, $|\alpha_{k+n} - \alpha| \leq C^n |\alpha_k - \alpha|$. If $C = 0.99$, progress may be very, very slow. If $|\alpha_k - \alpha| = 1$, then

$$\begin{aligned} |\alpha_{k+1} - \alpha| &\leq 0.99000 \\ |\alpha_{k+2} - \alpha| &\leq 0.98010 \\ |\alpha_{k+3} - \alpha| &\leq 0.97030 \\ |\alpha_{k+4} - \alpha| &\leq 0.96060 \\ |\alpha_{k+5} - \alpha| &\leq 0.95099 \\ |\alpha_{k+6} - \alpha| &\leq 0.94148 \\ |\alpha_{k+7} - \alpha| &\leq 0.93206 \\ |\alpha_{k+8} - \alpha| &\leq 0.92274 \\ |\alpha_{k+9} - \alpha| &\leq 0.91351 \end{aligned}$$

Quadratic convergence is fast. Now

$$\begin{aligned} |\alpha_{k+1} - \alpha| &\leq C|\alpha_k - \alpha|^2 \\ |\alpha_{k+2} - \alpha| &\leq C|\alpha_{k+1} - \alpha|^2 \leq C(C|\alpha_k - \alpha|^2)^2 = C^3|\alpha_k - \alpha|^4 \\ |\alpha_{k+3} - \alpha| &\leq C|\alpha_{k+2} - \alpha|^2 \leq C(C^3|\alpha_k - \alpha|^4)^2 = C^7|\alpha_k - \alpha|^8 \\ &\vdots \\ |\alpha_{k+n} - \alpha| &\leq C^{2^n-1}|\alpha_k - \alpha|^{2^n} \end{aligned}$$

Even $C = 0.99$ and $|\alpha_k - \alpha| = 1$, then

$$\begin{aligned} |\alpha_{k+1} - \alpha| &\leq 0.99000 \\ |\alpha_{k+2} - \alpha| &\leq 0.970299 \\ |\alpha_{k+3} - \alpha| &\leq 0.932065 \\ |\alpha_{k+4} - \alpha| &\leq 0.860058 \\ |\alpha_{k+5} - \alpha| &\leq 0.732303 \\ |\alpha_{k+6} - \alpha| &\leq 0.530905 \\ |\alpha_{k+7} - \alpha| &\leq 0.279042 \\ |\alpha_{k+8} - \alpha| &\leq 0.077085 \\ |\alpha_{k+9} - \alpha| &\leq 0.005882 \\ |\alpha_{k+10} - \alpha| &\leq 0.000034 \end{aligned}$$

If we consider α the correct result then, eventually, the number of correct digits roughly doubles in each iteration. This can be explained as follows: If $|\alpha_k - \alpha| < 1$, then the number of correct decimal digits is given by

$$-\log_{10} |\alpha_k - \alpha|.$$

Since \log_{10} is a monotonically increasing function,

$$\log_{10} |\alpha_{k+1} - \alpha| \leq \log_{10} C |\alpha_k - \alpha|^2 = \log_{10}(C) + 2 \log_{10} |\alpha_k - \alpha| \leq 2 \log_{10}(|\alpha_k - \alpha|)$$

and hence

$$\underbrace{-\log_{10} |\alpha_{k+1} - \alpha|}_{\substack{\text{number of correct} \\ \text{digits in } \alpha_{k+1}}} \geq 2 \left(\underbrace{-\log_{10}(|\alpha_k - \alpha|)}_{\substack{\text{number of correct} \\ \text{digits in } \alpha_k}} \right).$$

Cubic convergence is dizzyingly fast: Eventually the number of correct digits triples from one iteration to the next.

We now define a convenient norm.

Lemma 15.3 *Let $X \in \mathbb{C}^{m \times m}$ be nonsingular. Define $\|\cdot\|_X : \mathbb{C}^m \rightarrow \mathbb{R}$ by $\|y\|_X = \|Xy\|$ for some given norm $\|\cdot\| : \mathbb{C}^m \rightarrow \mathbb{R}$. Then $\|\cdot\|_X$ is a norm.*

Homework 15.4 *Prove Lemma 15.3.*

☞ SEE ANSWER

With this new norm, we can do our convergence analysis:

$$v^{(k)} - \psi_0 x_0 = A^k v^{(0)} / \lambda_0^k - \psi_0 x_0 = X \left(\begin{array}{c|cc|c} 1 & 0 & \cdots & 0 \\ \hline 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{array} \right) X^{-1} v^{(0)} - \psi_0 x_0$$

$$= X \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{pmatrix} y - \psi_0 x_0 = X \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{pmatrix} y$$

Hence

$$X^{-1}(v^{(k)} - \psi_0 x_0) = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & \left(\frac{\lambda_1}{\lambda_0}\right)^k & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\lambda_{m-1}}{\lambda_0}\right)^k \end{pmatrix} y$$

and

$$X^{-1}(v^{(k+1)} - \psi_0 x_0) = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & \frac{\lambda_1}{\lambda_0} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\lambda_{m-1}}{\lambda_0} \end{pmatrix} X^{-1}(v^{(k)} - \psi_0 x_0).$$

Now, let $\|\cdot\|$ be a p-norm¹ and its induced matrix norm and $\|\cdot\|_{X^{-1}}$ as defined in Lemma 15.3. Then

$$\begin{aligned} \|v^{(k+1)} - \psi_0 x_0\|_{X^{-1}} &= \|X^{-1}(v^{(k+1)} - \psi_0 x_0)\| \\ &= \left\| \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & \frac{\lambda_1}{\lambda_0} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\lambda_{m-1}}{\lambda_0} \end{pmatrix} X^{-1}(v^{(k)} - \psi_0 x_0) \right\| \\ &\leq \left| \frac{\lambda_1}{\lambda_0} \right| \|X^{-1}(v^{(k)} - \psi_0 x_0)\| = \left| \frac{\lambda_1}{\lambda_0} \right| \|v^{(k)} - \psi_0 x_0\|_{X^{-1}}. \end{aligned}$$

This shows that, in this norm, the convergence of $v^{(k)}$ to $\psi_0 x_0$ is linear: The difference between current approximation, $v^{(k)}$, and the solution, $\psi_0 x_0$, is reduced by at least a constant factor in each iteration.

15.1.4 Practical Power Method

The following algorithm, known as the Power Method, avoids the problem of $v^{(k)}$ growing or shrinking in length, without requiring λ_0 to be known, by scaling it to be of unit length at each step:

```

for k = 0, ...
    v^{(k+1)} = Av^{(k)}
    v^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|
endfor

```

¹We choose a p-norm to make sure that the norm of a diagonal matrix equals the absolute value of the largest element (in magnitude) on its diagonal.

15.1.5 The Rayleigh quotient

A question is how to extract an approximation of λ_0 given an approximation of x_0 . The following theorem provides the answer:

Theorem 15.5 *If x is an eigenvector of A then $\lambda = x^H A x / (x^H x)$ is the associated eigenvalue of A . This ratio is known as the Rayleigh quotient.*

Proof: Let x be an eigenvector of A and λ the associated eigenvalue. Then $Ax = \lambda x$. Multiplying on the left by x^H yields $x^H A x = \lambda x^H x$ which, since $x \neq 0$ means that $\lambda = x^H A x / (x^H x)$.

Clearly this ratio as a function of x is continuous and hence an approximation to x_0 when plugged into this formula would yield an approximation to λ_0 .

15.1.6 What if $|\lambda_0| \geq |\lambda_1|$?

Now, what if

$$|\lambda_0| = \dots = |\lambda_{k-1}| > |\lambda_k| \geq \dots \geq |\lambda_{m-1}|?$$

By extending the above analysis one can easily show that $v^{(k)}$ will converge to a vector in the subspace spanned by the eigenvectors associated with $\lambda_0, \dots, \lambda_{k-1}$.

An important special case is when $k = 2$: if A is real valued then λ_0 still may be complex valued in which case $\bar{\lambda}_0$ is also an eigenvalue and it has the same magnitude as λ_0 . We deduce that $v^{(k)}$ will always be in the space spanned by the eigenvectors corresponding to λ_0 and $\bar{\lambda}_0$.

15.2 The Inverse Power Method

The Power Method homes in on an eigenvector associated with the largest (in magnitude) eigenvalue. The Inverse Power Method homes in on an eigenvector associated with the smallest eigenvalue (in magnitude).

Throughout this section we will assume that a given matrix $A \in \mathbb{C}^{m \times m}$ is *nondeficient* and nonsingular so that there exist matrix X and diagonal matrix Λ such that $A = X\Lambda X^{-1}$. We further assume that $\Lambda = \text{diag}(\lambda_0, \dots, \lambda_{m-1})$ and

$$|\lambda_0| \geq |\lambda_1| \geq \dots \geq |\lambda_{m-2}| > |\lambda_{m-1}|.$$

Theorem 15.6 *Let $A \in \mathbb{C}^{m \times m}$ be nonsingular. Then λ and x are an eigenvalue and associated eigenvector of A if and only if $1/\lambda$ and x are an eigenvalue and associated eigenvector of A^{-1} .*

Homework 15.7 Assume that

$$|\lambda_0| \geq |\lambda_1| \geq \dots \geq |\lambda_{m-2}| > |\lambda_{m-1}| > 0.$$

Show that

$$\left| \frac{1}{\lambda_{m-1}} \right| > \left| \frac{1}{\lambda_{m-2}} \right| \geq \left| \frac{1}{\lambda_{m-3}} \right| \geq \dots \geq \left| \frac{1}{\lambda_0} \right|.$$

SEE ANSWER

Thus, an eigenvector associated with the smallest (in magnitude) eigenvalue of A is an eigenvector associated with the largest (in magnitude) eigenvalue of A^{-1} . This suggest the following naive iteration:

```

for k = 0, ...
  v(k+1) = A-1v(k)
  v(k+1) = λm-1v(k+1)
endfor

```

Of course, we would want to factor $A = LU$ once and solve $L(Uv^{(k+1)}) = v^{(k)}$ rather than multiplying with A^{-1} . From the analysis of the convergence of the “second attempt” for a Power Method algorithm we conclude that now

$$\|v^{(k+1)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}} \leq \left| \frac{\lambda_{m-1}}{\lambda_{m-2}} \right| \|v^{(k)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}}.$$

A practical Inverse Power Method algorithm is given by

```

for k = 0, ...
  v(k+1) = A-1v(k)
  v(k+1) = v(k+1) / \|v(k+1)\|
endfor

```

Often, we would expect the Invert Power Method to converge faster than the Power Method. For example, take the case where $|\lambda_k|$ are equally spaced between 0 and m : $|\lambda_k| = (k+1)$. Then

$$\left| \frac{\lambda_1}{\lambda_0} \right| = \frac{m-1}{m} \quad \text{and} \quad \left| \frac{\lambda_{m-1}}{\lambda_{m-2}} \right| = \frac{1}{2}.$$

which means that the Power Method converges much more slowly than the Inverse Power Method.

15.3 Rayleigh-quotient Iteration

The next observation is captured in the following lemma:

Lemma 15.8 *Let $A \in \mathbb{C}^{m \times m}$ and $\mu \in \mathbb{C}$. Then (λ, x) is an eigenpair of A if and only if $(\lambda - \mu, x)$ is an eigenpair of $(A - \mu I)$.*

Homework 15.9 *Prove Lemma 15.8.*

 [SEE ANSWER](#)

The matrix $A - \mu I$ is referred to as the matrix A that has been “shifted” by μ . What the lemma says is that shifting A by μ shifts the spectrum of A by μ :

Lemma 15.10 *Let $A \in \mathbb{C}^{m \times m}$, $A = X\Lambda X^{-1}$ and $\mu \in \mathbb{C}$. Then $A - \mu I = X(\Lambda - \mu I)X^{-1}$.*

Homework 15.11 *Prove Lemma 15.10.*

 [SEE ANSWER](#)

This suggests the following (naive) iteration: Pick a value μ close to λ_{m-1} . Iterate

for $k = 0, \dots$

$$\begin{aligned} v^{(k+1)} &= (A - \mu I)^{-1} v^{(k)} \\ v^{(k+1)} &= (\lambda_{m-1} - \mu) v^{(k+1)} \end{aligned}$$

endfor

Of course one would solve $(A - \mu I)v^{(k+1)} = v^{(k)}$ rather than computing and applying the inverse of A .

If we index the eigenvalues so that $|\lambda_0 - \mu| \leq \dots \leq |\lambda_{m-2} - \mu| < |\lambda_{m-1} - \mu|$ then

$$\|v^{(k+1)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}} \leq \left| \frac{\lambda_{m-1} - \mu}{\lambda_{m-2} - \mu} \right| \|v^{(k)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}}.$$

The closer to λ_{m-1} the “shift” (so named because it shifts the spectrum of A) is chosen, the more favorable the ratio that dictates convergence.

A more practical algorithm is given by

for $k = 0, \dots$

$$\begin{aligned} v^{(k+1)} &= (A - \mu I)^{-1} v^{(k)} \\ v^{(k+1)} &= v^{(k+1)} / \|v^{(k+1)}\| \end{aligned}$$

endfor

The question now becomes how to chose μ so that it is a good guess for λ_{m-1} . Often an application inherently supplies a reasonable approximation for the smallest eigenvalue or an eigenvalue of particular interest. However, we know that eventually $v^{(k)}$ becomes a good approximation for x_{m-1} and therefore the Rayleigh quotient gives us a way to find a good approximation for λ_{m-1} . This suggests the (naive) Rayleigh-quotient iteration:

for $k = 0, \dots$

$$\begin{aligned} \mu_k &= v^{(k)H} A v^{(k)} / (v^{(k)H} v^{(k)}) \\ v^{(k+1)} &= (A - \mu_k I)^{-1} v^{(k)} \\ v^{(k+1)} &= (\lambda_{m-1} - \mu_k) v^{(k+1)} \end{aligned}$$

endfor

Now²

$$\|v^{(k+1)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}} \leq \left| \frac{\lambda_{m-1} - \mu_k}{\lambda_{m-2} - \mu_k} \right| \|v^{(k)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}}$$

with

$$\lim_{k \rightarrow \infty} (\lambda_{m-1} - \mu_k) = 0$$

which means *super linear* convergence is observed. In fact, it can be shown that once k is large enough

$$\|v^{(k+1)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}} \leq C \|v^{(k)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}}^2,$$

which is known as quadratic convergence. Roughly speaking this means that every iteration doubles the number of correct digits in the current approximation. To prove this, one shows that $|\lambda_{m-1} - \mu_k| \leq C \|v^{(k)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}}$.

² I think... I have not checked this thoroughly. But the general idea holds. λ_{m-1} has to be defined as the eigenvalue to which the method eventually converges.

Better yet, it can be shown that if A is Hermitian, then, once k is large enough,

$$\|v^{(k+1)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}} \leq C\|v^{(k)} - \Psi_{m-1}x_{m-1}\|_{X^{-1}}^3,$$

which is known as cubic convergence. Roughly speaking this means that every iteration triples the number of correct digits in the current approximation. This is mind-boggling fast convergence!

A practical Rayleigh quotient iteration is given by

```

 $v^{(0)} = v^{(0)}/\|v^{(0)}\|_2$ 
for  $k = 0, \dots$ 
 $\mu_k = v^{(k)H}Av^{(k)}$  (Now  $\|v^{(k)}\|_2 = 1$ )
 $v^{(k+1)} = (A - \mu_k I)^{-1}v^{(k)}$ 
 $v^{(k+1)} = v^{(k+1)}/\|v^{(k+1)}\|$ 
endfor

```


Chapter **16**

Notes on the QR Algorithm and other Dense Eigensolvers

Video

Read disclaimer regarding the videos in the preface!

Tragically, the camera ran out of memory for the first lecture... Here is the second lecture, which discusses the implicit QR algorithm

- 👉 [YouTube](#)
- 👉 [Download from UT Box](#)
- 👉 [View After Local Download](#)

(For help on viewing, see Appendix A.)

In most of this note, we focus on the case where A is symmetric and real valued. The reason for this is that many of the techniques can be more easily understood in that setting.

16.0.1 Outline

Video	299
16.0.1 Outline	300
16.1 Preliminaries	301
16.2 Subspace Iteration	301
16.3 The QR Algorithm	306
16.3.1 A basic (unshifted) QR algorithm	306
16.3.2 A basic shifted QR algorithm	306
16.4 Reduction to Tridiagonal Form	308
16.4.1 Householder transformations (reflectors)	308
16.4.2 Algorithm	309
16.5 The QR algorithm with a Tridiagonal Matrix	311
16.5.1 Givens' rotations	311
16.6 QR Factorization of a Tridiagonal Matrix	312
16.7 The Implicitly Shifted QR Algorithm	314
16.7.1 Upper Hessenberg and tridiagonal matrices	314
16.7.2 The Implicit Q Theorem	315
16.7.3 The Francis QR Step	316
16.7.4 A complete algorithm	318
16.8 Further Reading	322
16.8.1 More on reduction to tridiagonal form	322
16.8.2 Optimizing the tridiagonal QR algorithm	322
16.9 Other Algorithms	322
16.9.1 Jacobi's method for the symmetric eigenvalue problem	322
16.9.2 Cuppen's Algorithm	325
16.9.3 The Method of Multiple Relatively Robust Representations (MRRR)	325
16.10 The Nonsymmetric QR Algorithm	325
16.10.1 A variant of the Schur decomposition	325
16.10.2 Reduction to upperHessenberg form	327
16.10.3 The implicitly double-shifted QR algorithm	329

16.1 Preliminaries

The QR algorithm is a standard method for computing all eigenvalues and eigenvectors of a matrix. In this note, we focus on the real valued symmetric eigenvalue problem (the case where $A \in \mathbb{R}^{n \times n}$). For this case, recall the Spectral Decomposition Theorem:

Theorem 16.1 *If $A \in \mathbb{R}^{n \times n}$ then there exists unitary matrix Q and diagonal matrix Λ such that $A = Q\Lambda Q^T$.*

We will partition $Q = \left(\begin{array}{c|c|c} q_0 & \cdots & q_{n-1} \end{array} \right)$ and assume that $\Lambda = \text{diag}((\lambda_0, \dots, \lambda_{n-1}))$ so that throughout this note, q_i and λ_i refer to the i th column of Q and the i th diagonal element of Λ , which means that each tuple (λ, q_i) is an eigenpair.

16.2 Subspace Iteration

We start with a matrix $V \in \mathbb{R}^{n \times r}$ with normalized columns and iterate something like

$$V^{(0)} = V$$

for $k = 0, \dots$ convergence

$$V^{(k+1)} = AV^{(k)}$$

Normalize the columns to be of unit length.

end for

The problem with this approach is that all columns will (likely) converge to an eigenvector associated with the dominant eigenvalue, since the Power Method is being applied to all columns simultaneously. We will now lead the reader through a succession of insights towards a practical algorithm.

Let us examine what $\widehat{V} = AV$ looks like, for the simple case where $V = \left(\begin{array}{c|c|c} v_0 & v_1 & v_2 \end{array} \right)$ (three columns). We know that

$$v_j = Q \underbrace{Q^T v_j}_{y_j} .$$

Hence

$$\begin{aligned} v_0 &= \sum_{j=0}^{n-1} \psi_{0,j} q_j, \\ v_1 &= \sum_{j=0}^{n-1} \psi_{1,j} q_j, \text{ and} \\ v_2 &= \sum_{j=0}^{n-1} \psi_{2,j} q_j, \end{aligned}$$

where $\psi_{i,j}$ equals the i th element of y_j . Then

$$\begin{aligned} AV &= A \left(\begin{array}{c|c|c} v_0 & v_1 & v_2 \end{array} \right) \\ &= A \left(\begin{array}{c|c|c} \sum_{j=0}^{n-1} \psi_{0,j} q_j & \sum_{j=0}^{n-1} \psi_{1,j} q_j & \sum_{j=0}^{n-1} \psi_{2,j} q_j \end{array} \right) \\ &= \left(\begin{array}{c|c|c} \sum_{j=0}^{n-1} \psi_{0,j} A q_j & \sum_{j=0}^{n-1} \psi_{1,j} A q_j & \sum_{j=0}^{n-1} \psi_{2,j} A q_j \end{array} \right) \\ &= \left(\begin{array}{c|c|c} \sum_{j=0}^{n-1} \psi_{0,j} \lambda_j q_j & \sum_{j=0}^{n-1} \psi_{1,j} \lambda_j q_j & \sum_{j=0}^{n-1} \psi_{2,j} \lambda_j q_j \end{array} \right) \end{aligned}$$

- If we happened to know λ_0 , λ_1 , and λ_2 then we could divide the columns by these, respectively, and get new vectors

$$\begin{aligned} \left(\begin{array}{c|c|c} \hat{v}_0 & \hat{v}_1 & \hat{v}_2 \end{array} \right) &= \left(\begin{array}{c|c|c} \sum_{j=0}^{n-1} \psi_{0,j} \left(\frac{\lambda_j}{\lambda_0} \right) q_j & \sum_{j=0}^{n-1} \psi_{1,j} \left(\frac{\lambda_j}{\lambda_1} \right) q_j & \sum_{j=0}^{n-1} \psi_{2,j} \left(\frac{\lambda_j}{\lambda_2} \right) q_j \end{array} \right) \\ &= \left(\begin{array}{c|c|c} \psi_{0,0} q_0 + & \psi_{1,0} \left(\frac{\lambda_0}{\lambda_1} \right) q_0 + & \psi_{2,0} \left(\frac{\lambda_0}{\lambda_2} \right) q_0 + \psi_{2,1} \left(\frac{\lambda_1}{\lambda_2} \right) q_1 + \\ \psi_{0,1} q_1 + & \psi_{1,1} q_1 + & \psi_{2,2} q_2 + \\ \sum_{j=1}^{n-1} \psi_{0,j} \left(\frac{\lambda_j}{\lambda_0} \right) q_j & \sum_{j=2}^{n-1} \psi_{1,j} \left(\frac{\lambda_j}{\lambda_1} \right) q_j & \sum_{j=3}^{n-1} \psi_{2,j} \left(\frac{\lambda_j}{\lambda_2} \right) q_j \end{array} \right) \quad (16.1) \end{aligned}$$

- Assume that $|\lambda_0| > |\lambda_1| > |\lambda_2| > |\lambda_3| \geq \dots \geq |\lambda_{n-1}|$. Then, similar as for the power method,
 - The first column will see components in the direction of $\{q_1, \dots, q_{n-1}\}$ shrink relative to the component in the direction of q_0 .
 - The second column will see components in the direction of $\{q_2, \dots, q_{n-1}\}$ shrink relative to the component in the direction of q_1 , but the component in the direction of q_0 increases, relatively, since $|\lambda_0/\lambda_1| > 1$.
 - The third column will see components in the direction of $\{q_3, \dots, q_{n-1}\}$ shrink relative to the component in the direction of q_2 , but the components in the directions of q_0 and q_1 increase, relatively, since $|\lambda_0/\lambda_2| > 1$ and $|\lambda_1/\lambda_2| > 1$.

How can we make it so that v_j converges to a vector in the direction of q_j ?

- If we happen to know q_0 , then we can subtract out the component of

$$\hat{v}_1 = \psi_{1,0} \frac{\lambda_0}{\lambda_1} q_0 + \psi_{1,1} q_1 + \sum_{j=2}^{n-1} \psi_{1,j} \frac{\lambda_j}{\lambda_1} q_j$$

in the direction of q_0 :

$$\hat{v}_1 - q_0^T \hat{v}_1 q_0 = \psi_{1,1} q_1 + \sum_{j=2}^{n-1} \psi_{1,j} \frac{\lambda_j}{\lambda_1} q_j$$

so that we are left with the component in the direction of q_1 and components in directions of q_2, \dots, q_{n-1} that are suppressed every time through the loop.

- Similarly, if we also know q_1 , the components of \hat{v}_2 in the direction of q_0 and q_1 can be subtracted from that vector.

- We do not know λ_0 , λ_1 , and λ_2 but from the discussion about the Power Method we remember that we can just normalize the so updated \hat{v}_0 , \hat{v}_1 , and \hat{v}_2 to have unit length.

How can we make these insights practical?

- We do not know q_0 , q_1 , and q_2 , but we can informally argue that if we keep iterating,
 - The vector \hat{v}_0 , normalized in each step, will eventually point in the direction of q_0 .
 - $S(\hat{v}_0, \hat{v}_1)$ will eventually equal $S(q_0, q_1)$.
 - In each iteration, we can subtract the component of \hat{v}_1 in the direction of \hat{v}_0 from \hat{v}_1 , and then normalize \hat{v}_1 so that eventually result in a the vector that points in the direction of q_1 .
 - $S(\hat{v}_0, \hat{v}_1, \hat{v}_2)$ will eventually equal $S(q_0, q_1, q_2)$.
 - In each iteration, we can subtract the component of \hat{v}_2 in the directions of \hat{v}_0 and \hat{v}_1 from \hat{v}_2 , and then normalize the result, to make \hat{v}_2 eventually point in the direction of q_2 .

What we recognize is that normalizing \hat{v}_0 , subtracting out the component of \hat{v}_1 in the direction of \hat{v}_0 , and then normalizing \hat{v}_1 , etc., is exactly what the Gram-Schmidt process does. And thus, we can use any convenient (and stable) QR factorization method. This also shows how the method can be generalized to work with more than three columns and even all columns simultaneously.

The algorithm now becomes:

$$\begin{aligned} V^{(0)} &= I^{n \times p} \quad (I^{n \times p} \text{ represents the first } p \text{ columns of } I) \\ \text{for } k = 0, \dots \text{ convergence} \\ AV^{(k)} &\rightarrow V^{(k+1)}R^{(k+1)} \quad (\text{QR factorization with } R^{(k+1)} \in \mathbb{R}^{p \times p}) \\ \text{end for} \end{aligned}$$

Now consider again (16.1), focusing on the third column:

$$\begin{pmatrix} \Psi_{2,0}\left(\frac{\lambda_0}{\lambda_2}\right)q_0 + \Psi_{2,1}\left(\frac{\lambda_1}{\lambda_2}\right)q_1 + \\ \Psi_{2,2}q_2 + \\ \sum_{j=3}^{n-1} \Psi_j\left(\frac{\lambda_j}{\lambda_2}\right)q_j \end{pmatrix} = \begin{pmatrix} \Psi_{2,0}\left(\frac{\lambda_0}{\lambda_2}\right)q_0 + \Psi_{2,1}\left(\frac{\lambda_1}{\lambda_2}\right)q_1 + \\ \Psi_{2,2}q_2 + \\ \Psi_j\left[\left(\frac{\lambda_3}{\lambda_2}\right)\right]q_3 + \sum_{j=4}^{n-1} \Psi_{2,j}\left(\frac{\lambda_j}{\lambda_2}\right)q_j \end{pmatrix}.$$

This shows that, if the components in the direction of q_0 and q_1 are subtracted out, it is the component in the direction of q_3 that is diminished in length the most slowly, dictated by the ratio $\left|\frac{\lambda_3}{\lambda_2}\right|$. This, of course, generalizes: the j th column of $V^{(k)}$, $v_i^{(k)}$ will have a component in the direction of q_{j+1} , of length $|q_{j+1}^T v_j^{(k)}|$, that can be expected to shrink most slowly.

We demonstrate this in Figure 16.1, which shows the execution of the algorithm with $p = n$ for a 5×5 matrix, and shows how $|q_{j+1}^T v_j^{(k)}|$ converge to zero as function of k .

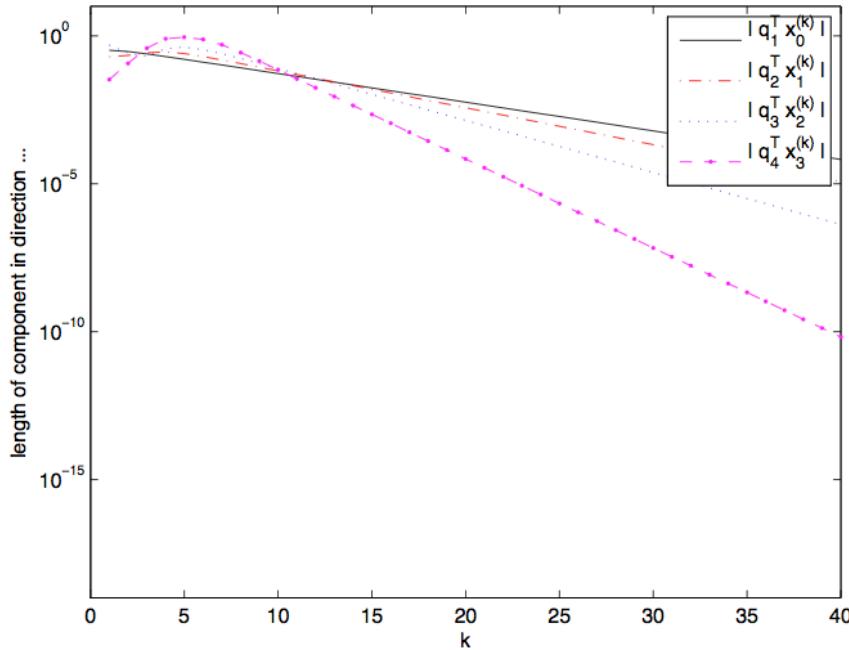


Figure 16.1: Convergence of the subspace iteration for a 5×5 matrix. This graph is mislabeled: x should be labeled with v . The (linear) convergence of v_j to a vector in the direction of q_j is dictated by how quickly the component in the direction q_{j+1} converges to zero. The line labeled $|q_{j+1}^T x_j|$ plots the length of the component in the direction q_{j+1} as a function of the iteration number.

Next, we observe that if $V \in \mathbb{R}^{n \times n}$ in the above iteration (which means we are iterating with n vectors at a time), then AV yields a next-to last column of the form

$$\begin{pmatrix} \sum_{j=0}^{n-3} \gamma_{n-2,j} \left(\frac{\lambda_j}{\lambda_{n-2}} \right) q_j + \\ \Psi_{n-2,n-2} q_{n-2} + \\ \Psi_{n-2,n-1} \boxed{\left(\frac{\lambda_{n-1}}{\lambda_{n-2}} \right)} q_{n-1} \end{pmatrix},$$

where $\Psi_{i,j} = q_j^T v_i$. Thus, given that the components in the direction of q_j , $j = 0, \dots, n-2$ can be expected in later iterations to be greatly reduced by the QR factorization that subsequently happens with AV , we notice that it is $\left| \frac{\lambda_{n-1}}{\lambda_{n-2}} \right|$ that dictates how fast the component in the direction of q_{n-1} disappears from $v_{n-2}^{(k)}$. This is a ratio we also saw in the Inverse Power Method and that we noticed we could accelerate in the Rayleigh Quotient Iteration: At each iteration we should shift the matrix to $(A - \mu_k I)$ where $\mu_k \approx \lambda_{n-1}$. Since the last column of $V^{(k)}$ is supposed to be converging to q_{n-1} , it seems reasonable to use $\mu_k = v_{n-1}^{(k)T} A v_{n-1}^{(k)}$ (recall that $v_{n-1}^{(k)}$ has unit length, so this is the Rayleigh quotient.)

The above discussion motivates the iteration

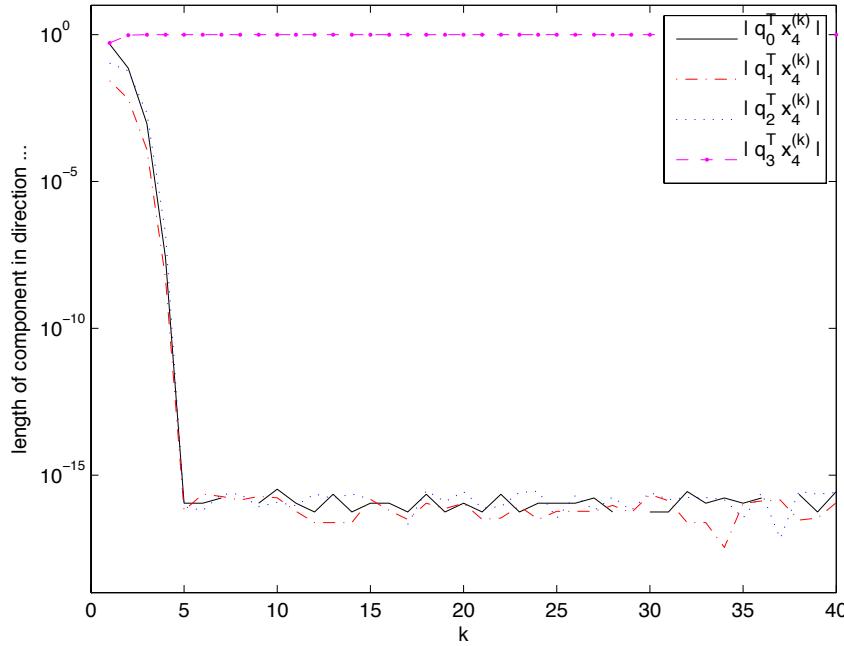


Figure 16.2: Convergence of the shifted subspace iteration for a 5×5 matrix. This graph is mislabeled: x should be labeled with v . What this graph shows is that the components of v_4 in the directions q_0 through q_3 disappear very quickly. The vector v_4 quickly points in the direction of the eigenvector associated with the smallest (in magnitude) eigenvalue. Just like the Rayleigh-quotient iteration is not guaranteed to converge to the eigenvector associated with the smallest (in magnitude) eigenvalue, the shifted subspace iteration may home in on a different eigenvector than the one associated with the smallest (in magnitude) eigenvalue. **Something is wrong in this graph: All curves should quickly drop to (near) zero!**

$$V^{(0)} := I \quad (V^{(0)} \in \mathbb{R}^{n \times n})$$

for $k := 0, \dots$ convergence

$$\mu_k := v_{n-1}^{(k)T} A v_{n-1}^{(k)} \quad (\text{Rayleigh quotient})$$

$$(A - \mu_k I) V^{(k)} \rightarrow V^{(k+1)} R^{(k+1)} \quad (\text{QR factorization})$$

end for

Notice that this does *not* require one to solve with $(A - \mu_k I)$, unlike in the Rayleigh Quotient Iteration. However, it does require a QR factorization, which requires more computation than the LU factorization (approximately $\frac{4}{3}n^3$ flops).

We demonstrate the convergence in Figure 16.2, which shows the execution of the algorithm with a 5×5 matrix and illustrates how $|q_j^T v_{n-1}^{(k)}|$ converge to zero as a function of k .

Subspace iteration	QR algorithm
$\widehat{A}^{(0)} := A$	$A^{(0)} := A$
$\widehat{V}^{(0)} := I$	$V^{(0)} := I$
for $k := 0, \dots$ until convergence	for $k := 0, \dots$ until convergence
$A\widehat{V}^{(k)} \rightarrow \widehat{V}^{(k+1)}\widehat{R}^{(k+1)}$ (QR factorization)	$A^{(k)} \rightarrow Q^{(k+1)}R^{(k+1)}$ (QR factorization)
$\widehat{A}^{(k+1)} := \widehat{V}^{(k+1)T}A\widehat{V}^{(k+1)}$	$A^{(k+1)} := R^{(k+1)}Q^{(k+1)}$
end for	end for

Figure 16.3: Basic subspace iteration and basic QR algorithm.

16.3 The QR Algorithm

The QR algorithm is a classic algorithm for computing all eigenvalues and eigenvectors of a matrix. While we explain it for the symmetric eigenvalue problem, it generalizes to the nonsymmetric eigenvalue problem as well.

16.3.1 A basic (unshifted) QR algorithm

We have informally argued that the columns of the orthogonal matrices $V^{(k)} \in \mathbb{R}^{n \times n}$ generated by the (unshifted) subspace iteration converge to eigenvectors of matrix A . (The exact conditions under which this happens have not been fully discussed.) In Figure 16.3 (left), we restate the subspace iteration. In it, we denote matrices $V^{(k)}$ and $R^{(k)}$ from the subspace iteration by $\widehat{V}^{(k)}$ and \widehat{R} to distinguish them from the ones computed by the algorithm on the right. The algorithm on the left also computes the matrix $\widehat{A}^{(k)} = V^{(k)T}AV^{(k)}$, a matrix that hopefully converges to Λ , the diagonal matrix with the eigenvalues of A on its diagonal. To the right is the QR algorithm. The claim is that the two algorithms compute the same quantities.

Homework 16.2 Prove that in Figure 16.3, $\widehat{V}^{(k)} = V^{(k)}$, and $\widehat{A}^{(k)} = A^{(k)}$, $k = 0, \dots$

☞ SEE ANSWER

We conclude that if $\widehat{V}^{(k)}$ converges to the matrix of orthonormal eigenvectors when the subspace iteration is applied to $V^{(0)} = I$, then $A^{(k)}$ converges to the diagonal matrix with eigenvalues along the diagonal.

16.3.2 A basic shifted QR algorithm

In Figure 16.4 (left), we restate the subspace iteration with shifting. In it, we denote matrices $V^{(k)}$ and $R^{(k)}$ from the subspace iteration by $\widehat{V}^{(k)}$ and \widehat{R} to distinguish them from the ones computed by the algorithm on the right. The algorithm on the left also computes the matrix $\widehat{A}^{(k)} = V^{(k)T}AV^{(k)}$, a matrix that hopefully converges to Λ , the diagonal matrix with the eigenvalues of A on its diagonal. To the right is the shifted QR algorithm. The claim is that the two algorithms compute the same quantities.

Subspace iteration	QR algorithm
$\widehat{A}^{(0)} := A$	$A^{(0)} := A$
$\widehat{V}^{(0)} := I$	$V^{(0)} := I$
for $k := 0, \dots$ until convergence	for $k := 0, \dots$ until convergence
$\widehat{\mu}_k := \widehat{v}_{n-1}^{(k)T} \widehat{A} \widehat{v}_{n-1}^{(k)}$	$\mu_k = \alpha_{n-1,n-1}^{(k)}$
$(A - \widehat{\mu}_k I) \widehat{V}^{(k)} \rightarrow \widehat{V}^{(k+1)} \widehat{R}^{(k+1)}$ (QR factorization)	$A^{(k)} - \mu_k I \rightarrow Q^{(k+1)} R^{(k+1)}$ (QR factorization)
$\widehat{A}^{(k+1)} := \widehat{V}^{(k+1)T} A \widehat{V}^{(k+1)}$	$A^{(k+1)} := R^{(k+1)} Q^{(k+1)} + \mu_k I$
	$V^{(k+1)} := V^{(k)} Q^{(k+1)}$
end for	end for

Figure 16.4: Basic shifted subspace iteration and basic shifted QR algorithm.

Homework 16.3 Prove that in Figure 16.4, $\widehat{V}^{(k)} = V^{(k)}$, and $\widehat{A}^{(k)} = A^{(k)}$, $k = 0, \dots$

SEE ANSWER

We conclude that if $\widehat{V}^{(k)}$ converges to the matrix of orthonormal eigenvectors when the shifted subspace iteration is applied to $V^{(0)} = I$, then $A^{(k)}$ converges to the diagonal matrix with eigenvalues along the diagonal.

The convergence of the basic shifted QR algorithm is illustrated below. Pay particular attention to the convergence of the last row and column.

$$\begin{aligned}
 A^{(0)} &= \begin{pmatrix} 2.01131953448 & 0.05992695085 & 0.14820940917 \\ 0.05992695085 & 2.30708673171 & 0.93623515213 \\ 0.14820940917 & 0.93623515213 & 1.68159373379 \end{pmatrix} & A^{(1)} &= \begin{pmatrix} 2.21466116574 & 0.34213192482 & 0.31816754245 \\ 0.34213192482 & 2.54202325042 & 0.57052186467 \\ 0.31816754245 & 0.57052186467 & 1.24331558383 \end{pmatrix} \\
 A^{(2)} &= \begin{pmatrix} 2.63492207667 & 0.47798481637 & 0.07654607908 \\ 0.47798481637 & 2.35970859985 & 0.06905042811 \\ 0.07654607908 & 0.06905042811 & 1.00536932347 \end{pmatrix} & A^{(3)} &= \begin{pmatrix} 2.87588550968 & 0.32971207176 & 0.00024210487 \\ 0.32971207176 & 2.12411444949 & 0.00014361630 \\ 0.00024210487 & 0.00014361630 & 1.00000004082 \end{pmatrix} \\
 A^{(4)} &= \begin{pmatrix} 2.96578660126 & 0.18177690194 & 0.00000000000 \\ 0.18177690194 & 2.03421339873 & 0.00000000000 \\ 0.00000000000 & 0.00000000000 & 1.00000000000 \end{pmatrix} & A^{(5)} &= \begin{pmatrix} 2.9912213907 & 0.093282073553 & 0.00000000000 \\ 0.0932820735 & 2.008778609226 & 0.00000000000 \\ 0.00000000000 & 0.00000000000 & 1.00000000000 \end{pmatrix}
 \end{aligned}$$

Once the off-diagonal elements of the last row and column have converged (are sufficiently small), the problem can be *deflated* by applying the following theorem:

Theorem 16.4 Let

$$A = \left(\begin{array}{c|c|c|c}
 A_{0,0} & A_{01} & \cdots & A_{0N-1} \\ \hline
 0 & A_{1,1} & \cdots & A_{1,N-1} \\ \hline
 \vdots & \vdots & \ddots & \vdots \\ \hline
 0 & 0 & \cdots & A_{N-1,N-1}
 \end{array} \right),$$

where $A_{k,k}$ are all square. Then $\Lambda(A) = \bigcup_{k=0}^{N-1} \Lambda(A_{k,k})$.

Homework 16.5 Prove the above theorem.

 SEE ANSWER

In other words, once the last row and column have converged, the algorithm can continue with the submatrix that consists of the first $n - 1$ rows and columns.

The problem with the QR algorithm, as stated, is that each iteration requires $O(n^3)$ operations, which is too expensive given that many iterations are required to find all eigenvalues and eigenvectors.

16.4 Reduction to Tridiagonal Form

In the next section, we will see that if $A^{(0)}$ is a tridiagonal matrix, then so are all $A^{(k)}$. This reduces the cost of each iteration from $O(n^3)$ to $O(n)$. We first show how unitary similarity transformations can be used to reduce a matrix to tridiagonal form.

16.4.1 Householder transformations (reflectors)

We briefly review the main tool employed to reduce a matrix to tridiagonal form: the Householder transform, also known as a reflector. Full details were given in Chapter 6.

Definition 16.6 Let $u \in \mathbb{R}^n$, $\tau \in \mathbb{R}$. Then $H = H(u) = I - uu^T/\tau$, where $\tau = \frac{1}{2}u^T u$, is said to be a reflector or Householder transformation.

We observe:

- Let z be any vector that is perpendicular to u . Applying a Householder transform $H(u)$ to z leaves the vector unchanged: $H(u)z = z$.
- Let any vector x be written as $x = z + u^T xu$, where z is perpendicular to u and $u^T xu$ is the component of x in the direction of u . Then $H(u)x = z - u^T xu$.

This can be interpreted as follows: The space perpendicular to u acts as a “mirror”: any vector in that space (along the mirror) is not reflected, while any other vector has the component that is orthogonal to the space (the component outside and orthogonal to the mirror) reversed in direction. Notice that a reflection preserves the length of the vector. Also, it is easy to verify that:

1. $HH = I$ (reflecting the reflection of a vector results in the original vector);
2. $H = H^T$, and so $H^T H = HH^T = I$ (a reflection is an orthogonal matrix and thus preserves the norm); and
3. if H_0, \dots, H_{k-1} are Householder transformations and $Q = H_0 H_1 \cdots H_{k-1}$, then $Q^T Q = QQ^T = I$ (an accumulation of reflectors is an orthogonal matrix).

As part of the reduction to condensed form operations, given a vector x we will wish to find a Householder transformation, $H(u)$, such that $H(u)x$ equals a vector with zeroes below the first element: $H(u)x = \mp\|x\|_2 e_0$ where e_0 equals the first column of the identity matrix. It can be easily checked that choosing $u = x \pm \|x\|_2 e_0$ yields the desired $H(u)$. Notice that any nonzero scaling of u has the same property, and the convention is to scale u so that the first element equals one. Let us define $[u, \tau, h] = \text{HouseV}(x)$ to be the function that returns u with first element equal to one, $\tau = \frac{1}{2}u^T u$, and $h = H(u)x$.

16.4.2 Algorithm

The first step towards computing the eigenvalue decomposition of a symmetric matrix is to reduce the matrix to tridiagonal form.

The basic algorithm for reducing a symmetric matrix to tridiagonal form, overwriting the original matrix with the result, can be explained as follows. We assume that symmetric A is stored only in the lower triangular part of the matrix and that only the diagonal and subdiagonal of the symmetric tridiagonal matrix is computed, overwriting those parts of A . Finally, the Householder vectors used to zero out parts of A overwrite the entries that they annihilate (set to zero).

- Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{21}^T \\ \hline a_{21} & A_{22} \end{array} \right)$.

- Let $[u_{21}, \tau, a_{21}] := \text{HouseV}(a_{21})$.¹

- Update

$$\left(\begin{array}{c|c} \alpha_{11} & a_{21}^T \\ \hline a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & H \end{array} \right) \left(\begin{array}{c|c} \alpha_{11} & a_{21}^T \\ \hline a_{21} & A_{22} \end{array} \right) \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & H \end{array} \right) = \left(\begin{array}{c|c} \alpha_{11} & a_{21}^T H \\ \hline Ha_{21} & HA_{22}H \end{array} \right)$$

where $H = H(u_{21})$. Note that $a_{21} := Ha_{21}$ need not be executed since this update was performed by the instance of HouseV above.² Also, a_{12}^T is not stored nor updated due to symmetry. Finally, only the lower triangular part of $HA_{22}H$ is computed, overwriting A_{22} . The update of A_{22} warrants closer scrutiny:

$$\begin{aligned} A_{22} &:= (I - \frac{1}{\tau} u_{21} u_{21}^T) A_{22} (I - \frac{1}{\tau} u_{21} u_{21}^T) \\ &= (A_{22} - \frac{1}{\tau} u_{21} \underbrace{u_{21}^T A_{22}}_{y_{21}^T}) (I - \frac{1}{\tau} u_{21} u_{21}^T) \\ &= A_{22} - \frac{1}{\tau} u_{21} y_{21}^T - \frac{1}{\tau} \underbrace{A u_{21}}_{y_{21}} u_{21}^T + \frac{1}{\tau^2} u_{21} \underbrace{y_{21}^T u_{21}}_{2\beta} u_{21}^T \\ &= A_{22} - \left(\frac{1}{\tau} u_{21} y_{21}^T - \frac{\beta}{\tau^2} u_{21} u_{21}^T \right) - \left(\frac{1}{\tau} y_{21} u_{21}^T - \frac{\beta}{\tau^2} u_{21} u_{21}^T \right) \\ &= A_{22} - u_{21} \underbrace{\frac{1}{\tau} \left(y_{21}^T - \frac{\beta}{\tau} u_{21}^T \right)}_{w_{21}^T} - \underbrace{\frac{1}{\tau} \left(y_{21} - \frac{\beta}{\tau} u_{21} \right)}_{w_{21}} u_{21}^T \\ &= \underbrace{A_{22} - u_{21} w_{21}^T - w_{21} u_{21}^T}_{\text{symmetric}}. \end{aligned}$$

rank-2 update

¹ Note that the semantics here indicate that a_{21} is overwritten by Ha_{21} .

² In practice, the zeros below the first element of Ha_{21} are not actually written. Instead, the implementation overwrites these elements with the corresponding elements of the vector u_{21} .

Algorithm: $[A, t] := \text{TRIRED_UNB}(A)$

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

where A_{TL} is 0×0 and t_b has 0 elements

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

Repartition

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

where α_{11} and τ_1 are scalars

$$[u_{21}, \tau_1, a_{21}] := \text{HouseV}(a_{21})$$

$$y_{21} := A_{22}u_{21}$$

$$\beta := u_{21}^T y_{21} / 2$$

$$w_{21} := (y_{21} - \beta u_{21} / \tau_1) / \tau_1$$

$$A_{22} := A_{22} - \text{tril}(u_{21} w_{21}^T + w_{21} u_{21}^T) \quad (\text{symmetric rank-2 update})$$

Continue with

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

endwhile

Figure 16.5: Basic algorithm for reduction of a symmetric matrix to tridiagonal form.

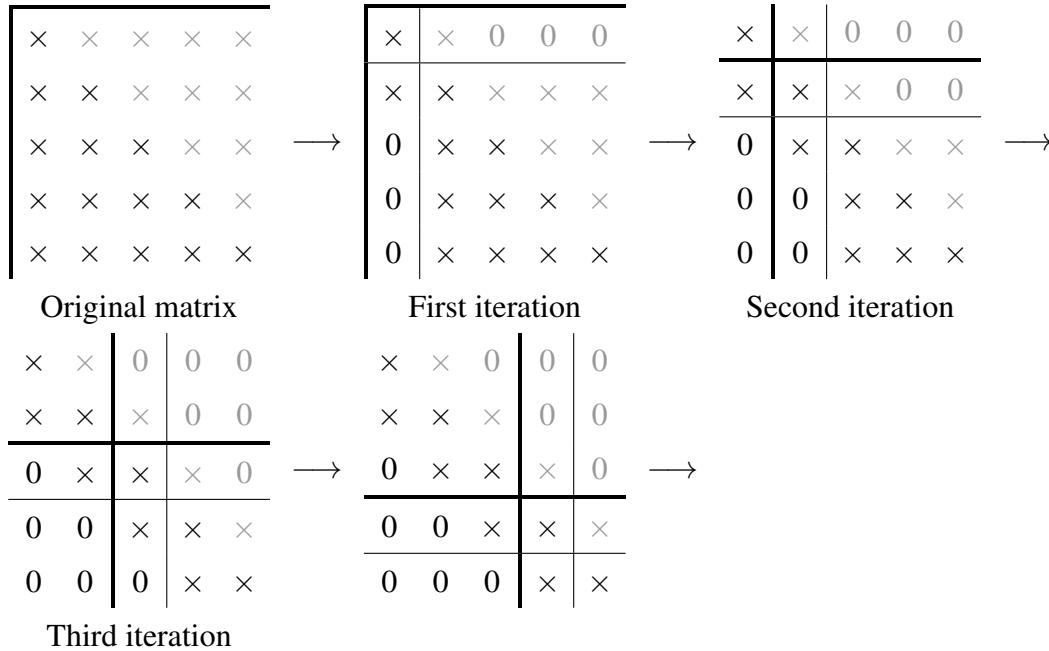


Figure 16.6: Illustration of reduction of a symmetric matrix to tridiagonal form. The \times s denote nonzero elements in the matrix. The gray entries above the diagonal are not actually updated.

- Continue this process with the updated A_{22} .

This is captured in the algorithm in Figure 16.5. It is also illustrated in Figure 16.6.

The total cost for reducing $A \in \mathbb{R}^{n \times n}$ is approximately

$$\sum_{k=0}^{n-1} (4(n-k-1)^2) \text{ flops} \approx \frac{4}{3}n^3 \text{ flops.}$$

This equals, approximately, the cost of one QR factorization of matrix A .

16.5 The QR algorithm with a Tridiagonal Matrix

We are now ready to describe an algorithm for the QR algorithm with a tridiagonal matrix.

16.5.1 Givens' rotations

First, we introduce another important class of unitary matrices known as Givens' rotations. Given a vector $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2$, there exists an orthogonal matrix G such that $G^T x = \begin{pmatrix} \pm \|x\|_2 \\ 0 \end{pmatrix}$. The Householder transformation is one example of such a matrix G . An alternative is the Givens' rotation: $G = \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}$

where $\gamma^2 + \sigma^2 = 1$. (Notice that γ and σ can be thought of as the cosine and sine of an angle.) Then

$$\begin{aligned} G^T G &= \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}^T \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix} = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix} \\ &= \begin{pmatrix} \gamma^2 + \sigma^2 & -\gamma\sigma + \gamma\sigma \\ \gamma\sigma - \gamma\sigma & \gamma^2 + \sigma^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \end{aligned}$$

which means that a Givens' rotation is a unitary matrix.

Now, if $\gamma = \chi_1/\|x\|_2$ and $\sigma = \chi_2/\|x\|_2$, then $\gamma^2 + \sigma^2 = (\chi_1^2 + \chi_2^2)/\|x\|_2^2 = 1$ and

$$\begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}^T \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \begin{pmatrix} \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} (\chi_1^2 + \chi_2^2)/\|x\|_2 \\ (\chi_1\chi_2 - \chi_1\chi_2)/\|x\|_2 \end{pmatrix} = \begin{pmatrix} \|x\|_2 \\ 0 \end{pmatrix}.$$

16.6 QR Factorization of a Tridiagonal Matrix

Now, consider the 4×4 tridiagonal matrix

$$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & 0 \\ 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{pmatrix}$$

From $\begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \end{pmatrix}$ one can compute $\gamma_{1,0}$ and $\sigma_{1,0}$ so that

$$\begin{pmatrix} \gamma_{1,0} & -\sigma_{1,0} \\ \sigma_{1,0} & \gamma_{1,0} \end{pmatrix}^T \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \end{pmatrix} = \begin{pmatrix} \hat{\alpha}_{0,0} \\ 0 \end{pmatrix}.$$

Then

$$\left(\begin{array}{c|cc|c} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ \hline 0 & \hat{\alpha}_{1,1} & \hat{\alpha}_{1,2} & 0 \\ \hline 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \hline 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) = \left(\begin{array}{c|cc|c} \gamma_{1,0} & \sigma_{1,0} & 0 & 0 \\ \hline -\sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} \alpha_{0,0} & \alpha_{0,1} & 0 & 0 \\ \hline \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & 0 \\ \hline 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \hline 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right)$$

Next, from $\begin{pmatrix} \hat{\alpha}_{1,1} \\ \alpha_{2,1} \end{pmatrix}$ one can compute $\gamma_{2,1}$ and $\sigma_{2,1}$ so that

$$\begin{pmatrix} \gamma_{2,1} & -\sigma_{2,1} \\ \sigma_{2,1} & \gamma_{2,1} \end{pmatrix}^T \begin{pmatrix} \hat{\alpha}_{1,1} \\ \alpha_{2,1} \end{pmatrix} = \begin{pmatrix} \hat{\hat{\alpha}}_{1,1} \\ 0 \end{pmatrix}.$$

Then

$$\left(\begin{array}{c|cc|c} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ \hline 0 & \hat{\hat{\alpha}}_{1,1} & \hat{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ 0 & 0 & \hat{\alpha}_{2,2} & \hat{\alpha}_{2,3} \\ \hline 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) = \left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & \gamma_{2,1} & \sigma_{2,1} & 0 \\ 0 & -\sigma_{2,1} & \gamma_{2,1} & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ \hline 0 & \hat{\alpha}_{1,1} & \hat{\alpha}_{1,2} & 0 \\ 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \hline 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right)$$

Finally, from $\begin{pmatrix} \hat{\alpha}_{2,2} \\ \alpha_{3,2} \end{pmatrix}$ one can compute $\gamma_{3,2}$ and $\sigma_{3,2}$ so that $\begin{pmatrix} \gamma_{3,2} & -\sigma_{3,2} \\ \sigma_{3,2} & \gamma_{3,2} \end{pmatrix}^T \begin{pmatrix} \hat{\alpha}_{2,2} \\ \alpha_{3,2} \end{pmatrix} = \begin{pmatrix} \hat{\hat{\alpha}}_{2,2} \\ 0 \end{pmatrix}$.

Then

$$\left(\begin{array}{c|cc|c} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ \hline 0 & \hat{\hat{\alpha}}_{1,1} & \hat{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ 0 & 0 & \hat{\hat{\alpha}}_{2,2} & \hat{\hat{\alpha}}_{2,3} \\ \hline 0 & 0 & 0 & \hat{\alpha}_{3,3} \end{array} \right) = \left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 1 & 0 & \gamma_{3,2} & \sigma_{3,2} \\ 0 & 1 & -\sigma_{3,2} & \gamma_{3,2} \end{array} \right) \left(\begin{array}{c|cc|c} \hat{\alpha}_{0,0} & \hat{\alpha}_{0,1} & \hat{\alpha}_{0,2} & 0 \\ \hline 0 & \hat{\hat{\alpha}}_{1,1} & \hat{\hat{\alpha}}_{1,2} & \hat{\hat{\alpha}}_{1,3} \\ 0 & 0 & \hat{\alpha}_{2,2} & \hat{\alpha}_{2,3} \\ \hline 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right)$$

The matrix Q is the orthogonal matrix that results from multiplying the different Givens' rotations together:

$$Q = \left(\begin{array}{c|cc|c} \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \hline \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & \gamma_{2,1} & -\sigma_{2,1} & 0 \\ 0 & \sigma_{2,1} & \gamma_{2,1} & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & \gamma_{3,2} & -\sigma_{3,2} \\ 0 & 0 & \sigma_{3,2} & \gamma_{3,2} \end{array} \right). \quad (16.2)$$

However, it is typically not explicitly formed.

The next question is how to compute RQ given the QR factorization of the tridiagonal matrix:

$$\left(\begin{array}{cc|cc} \widehat{\alpha}_{0,0} & \widehat{\alpha}_{0,1} & \widehat{\alpha}_{0,2} & 0 \\ 0 & \widehat{\alpha}_{1,1} & \widehat{\alpha}_{1,2} & \widehat{\alpha}_{1,3} \\ \hline 0 & 0 & \widehat{\alpha}_{2,2} & \widehat{\alpha}_{2,3} \\ 0 & 0 & 0 & \widehat{\alpha}_{3,3} \end{array} \right) \left(\begin{array}{cc|cc} \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} 1 & 0 & 0 & 0 \\ \hline 0 & \gamma_{2,1} & -\sigma_{2,1} & 0 \\ 0 & \sigma_{2,1} & \gamma_{2,1} & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & \gamma_{3,2} & -\sigma_{3,2} \\ 0 & 0 & \sigma_{3,2} & \gamma_{3,2} \end{array} \right) \\
 \left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{0,1} & \widehat{\alpha}_{0,2} & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \widehat{\alpha}_{1,2} & \widehat{\alpha}_{1,3} \\ \hline 0 & 0 & \widehat{\alpha}_{2,2} & \widehat{\alpha}_{2,3} \\ 0 & 0 & 0 & \widehat{\alpha}_{3,3} \end{array} \right) \\
 \left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{0,1} & \tilde{\alpha}_{0,2} & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \tilde{\alpha}_{1,2} & \widehat{\alpha}_{1,3} \\ 0 & \tilde{\alpha}_{2,1} & \tilde{\alpha}_{2,2} & \widehat{\alpha}_{2,3} \\ \hline 0 & 0 & 0 & \widehat{\alpha}_{3,3} \end{array} \right) \\
 \left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{0,1} & \tilde{\alpha}_{0,2} & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \tilde{\alpha}_{1,2} & \tilde{\alpha}_{1,3} \\ 0 & \tilde{\alpha}_{2,1} & \tilde{\alpha}_{2,2} & \tilde{\alpha}_{2,3} \\ \hline 0 & 0 & \tilde{\alpha}_{3,2} & \tilde{\alpha}_{3,3} \end{array} \right).$$

A symmetry argument can be used to motivate that $\tilde{\alpha}_{0,2} = \tilde{\alpha}_{1,3} = 0$.

16.7 The Implicitly Shifted QR Algorithm

16.7.1 Upper Hessenberg and tridiagonal matrices

Definition 16.7 A matrix is said to be upper Hessenberg if all entries below its first subdiagonal equal zero.

In other words, if matrix $A \in \mathbb{R}^{n \times n}$ is upper Hessenberg, it looks like

$$A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \cdots & \alpha_{0,n-1} & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n-1} & \alpha_{1,n-1} \\ 0 & \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n-1} & \alpha_{2,n-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & \alpha_{n-2,n-2} & \alpha_{n-2,n-2} \\ 0 & 0 & 0 & \cdots & \alpha_{n-1,n-2} & \alpha_{n-1,n-2} \end{pmatrix}.$$

Obviously, a tridiagonal matrix is a special case of an upper Hessenberg matrix.

16.7.2 The Implicit Q Theorem

The following theorem sets up one of the most remarkable algorithms in numerical linear algebra, which allows us to greatly simplify the implementation of the shifted QR algorithm when A is tridiagonal.

Theorem 16.8 (Implicit Q Theorem) *Let $A, B \in \mathbb{R}^{n \times n}$ where B is upper Hessenberg and has only positive elements on its first subdiagonal and assume there exists a unitary matrix Q such that $Q^T A Q = B$. Then Q and B are uniquely determined by A and the first column of Q .*

Proof: Partition

$$Q = \left(\begin{array}{c|c|c|c|c|c} q_0 & q_1 & q_2 & \cdots & q_{n-2} & q_{n-1} \end{array} \right) \text{ and } B = \left(\begin{array}{cccccc} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} & \cdots & \beta_{0,n-2} & \beta_{0,n-1} \\ \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & \cdots & \beta_{1,n-2} & \beta_{1,n-1} \\ 0 & \beta_{2,1} & \beta_{2,2} & \cdots & \beta_{2,n-2} & \beta_{2,n-1} \\ 0 & 0 & \beta_{3,2} & \cdots & \beta_{3,n-2} & \beta_{3,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \beta_{n-1,n-2} & \beta_{n-1,n-1} \end{array} \right).$$

Notice that $AQ = QB$ and hence

$$A \left(\begin{array}{c|c|c|c|c|c} q_0 & q_1 & q_2 & \cdots & q_{n-2} & q_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c|c|c} q_0 & q_1 & q_2 & \cdots & q_{n-2} & q_{n-1} \end{array} \right) \left(\begin{array}{cccccc} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} & \cdots & \beta_{0,n-2} & \beta_{0,n-1} \\ \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & \cdots & \beta_{1,n-2} & \beta_{1,n-1} \\ 0 & \beta_{2,1} & \beta_{2,2} & \cdots & \beta_{2,n-1} & \\ 0 & 0 & \beta_{3,2} & \cdots & \beta_{3,n-2} & \beta_{3,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \beta_{n-1,n-2} & \beta_{n-1,n-1} \end{array} \right).$$

Equating the first column on the left and right, we notice that

$$Aq_0 = \beta_{0,0}q_0 + \beta_{1,0}q_1.$$

Now, q_0 is given and $\|q_0\|_2$ since Q is unitary. Hence

$$q_0^T A q_0 = \beta_{0,0} q_0^T q_0 + \beta_{1,0} q_0^T q_1 = \beta_{0,0}.$$

Next,

$$\beta_{1,0}q_1 = Aq_0 - \beta_{0,0}q_0 = \tilde{q}_1.$$

Since $\|q_1\|_2 = 1$ (it is a column of a unitary matrix) and $\beta_{1,0}$ is assumed to be positive, then we know that

$$\beta_{1,0} = \|\tilde{q}_1\|_2.$$

Finally,

$$q_1 = \tilde{q}_1 / \beta_{1,0}.$$

The point is that the first column of B and second column of Q are prescribed by the first column of Q and the fact that B has positive elements on the first subdiagonal.

In this way, each column of Q and each column of B can be determined, one by one.

Homework 16.9 Give all the details of the above proof.

 SEE ANSWER

Notice the similarity between the above proof and the proof of the existence and uniqueness of the QR factorization!

To take advantage of the special structure of A being symmetric, the theorem can be expanded to

Theorem 16.10 (Implicit Q Theorem) Let $A, B \in \mathbb{R}^{n \times n}$ where B is upper Hessenberg and has only positive elements on its first subdiagonal and assume there exists a unitary matrix Q such that $Q^T A Q = B$. Then Q and B are uniquely determined by A and the first column of Q . If A is symmetric, then B is also symmetric and hence tridiagonal.

16.7.3 The Francis QR Step

The Francis QR Step combines the steps $(A^{(k-1)} - \mu_k I) \rightarrow Q^{(k)} R^{(k)}$ and $A^{(k+1)} := R^{(k)} Q^{(k)} + \mu_k I$ into a single step.

Now, consider the 4×4 tridiagonal matrix

$$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & 0 \\ 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{pmatrix} - \mu I$$

The first Givens' rotation is computed from $\begin{pmatrix} \alpha_{0,0} - \mu \\ \alpha_{1,0} \end{pmatrix}$, yielding $\gamma_{1,0}$ and $\sigma_{1,0}$ so that

$$\begin{pmatrix} \gamma_{1,0} & -\sigma_{1,0} \\ \sigma_{1,0} & \gamma_{1,0} \end{pmatrix}^T \begin{pmatrix} \alpha_{0,0} - \mu I \\ \alpha_{1,0} \end{pmatrix}$$

has a zero second entry. Now, to preserve eigenvalues, any orthogonal matrix that is applied from the left must also have its transpose applied from the right. Let us compute

$$\left(\begin{array}{c|cc|c} \tilde{\alpha}_{0,0} & \hat{\alpha}_{1,0} & \hat{\alpha}_{2,0} & 0 \\ \hat{\alpha}_{1,0} & \hat{\alpha}_{1,1} & \hat{\alpha}_{1,2} & 0 \\ \hline \hat{\alpha}_{2,0} & \hat{\alpha}_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) = \left(\begin{array}{c|cc|c} \gamma_{1,0} & \sigma_{1,0} & 0 & 0 \\ -\sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} \alpha_{0,0} & \alpha_{0,1} & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & 0 \\ \hline 0 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) \left(\begin{array}{c|cc|c} \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right).$$

Next, from $\begin{pmatrix} \hat{\alpha}_{1,0} \\ \hat{\alpha}_{2,0} \end{pmatrix}$ one can compute $\gamma_{2,0}$ and $\sigma_{2,0}$ so that $\begin{pmatrix} \gamma_{2,0} & -\sigma_{2,0} \\ \sigma_{2,0} & \gamma_{2,0} \end{pmatrix}^T \begin{pmatrix} \hat{\alpha}_{1,0} \\ \hat{\alpha}_{2,0} \end{pmatrix} = \begin{pmatrix} \tilde{\alpha}_{1,0} \\ 0 \end{pmatrix}$.

Then

$$\left(\begin{array}{c|cc|c} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{1,0} & 0 & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} \\ \hline 0 & \hat{\alpha}_{2,1} & \hat{\alpha}_{2,2} & \hat{\alpha}_{2,3} \\ 0 & \hat{\alpha}_{3,1} & \hat{\alpha}_{3,2} & \alpha_{3,3} \end{array} \right) = \left(\begin{array}{c|cc|c} 1 & 0 & 0 & 0 \\ 0 & \gamma_{2,0} & \sigma_{2,0} & 0 \\ \hline 0 & -\sigma_{2,0} & \gamma_{2,0} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{c|cc|c} \tilde{\alpha}_{0,0} & \hat{\alpha}_{1,0} & \hat{\alpha}_{2,0} & 0 \\ \hat{\alpha}_{1,0} & \hat{\alpha}_{1,1} & \hat{\alpha}_{1,2} & 0 \\ \hline \hat{\alpha}_{2,0} & \hat{\alpha}_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ 0 & 0 & \alpha_{3,2} & \alpha_{3,3} \end{array} \right) \left(\begin{array}{c|cc|c} 1 & 0 & 0 & 0 \\ 0 & \gamma_{2,0} & -\sigma_{2,0} & 0 \\ \hline 0 & \sigma_{2,0} & \gamma_{2,0} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right)$$

From: Gene H Golub <golub@stanford.edu>
Date: Sun, 19 Aug 2007 13:54:47 -0700 (PDT)
Subject: John Francis, Co-Inventor of QR

Dear Colleagues,

For many years, I have been interested in meeting J G F Francis, one of the co-inventors of the QR algorithm for computing eigenvalues of general matrices. Through a lead provided by the late Erin Brent and with the aid of Google, I finally made contact with him.

John Francis was born in 1934 in London and currently lives in Hove, near Brighton. His residence is about a quarter mile from the sea; he is a widower. In 1954, he worked at the National Research Development Corp (NRDC) and attended some lectures given by Christopher Strachey. In 1955,'56 he was a student at Cambridge but did not complete a degree. He then went back to NRDC as an assistant to Strachey where he got involved in flutter computations and this led to his work on QR.

After leaving NRDC in 1961, he worked at the Ferranti Corp and then at the University of Sussex. Subsequently, he had positions with various industrial organizations and consultancies. He is now retired. His interests were quite general and included Artificial Intelligence, computer languages, systems engineering. He has not returned to numerical computation.

He was surprised to learn there are many references to his work and that the QR method is considered one of the ten most important algorithms of the 20th century. He was unaware of such developments as TeX and Math Lab. Currently he is working on a degree at the Open University.

John Francis did remarkable work and we are all in his debt. Along with the conjugate gradient method, it provided us with one of the basic tools of numerical analysis.

Gene Golub

Figure 16.7: Posting by the late Gene Golub in NA Digest Sunday, August 19, 2007 Volume 07 : Issue 34. An article on the ten most important algorithms of the 20th century, published in SIAM News, can be found at <http://www.uta.edu/faculty/rcli/TopTen/topten.pdf>.

again preserves eigenvalues. Finally, from $\begin{pmatrix} \widehat{\alpha}_{2,1} \\ \widehat{\alpha}_{3,1} \end{pmatrix}$ one can compute $\gamma_{3,1}$ and $\sigma_{3,1}$ so that

$$\begin{pmatrix} \gamma_{3,1} & -\sigma_{3,1} \\ \sigma_{3,1} & \gamma_{3,1} \end{pmatrix}^T \begin{pmatrix} \widehat{\alpha}_{2,1} \\ \widehat{\alpha}_{3,1} \end{pmatrix} = \begin{pmatrix} \tilde{\alpha}_{2,1} \\ 0 \end{pmatrix}.$$

Then

$$\left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{1,0} & 0 & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \tilde{\alpha}_{2,1} & 0 \\ \hline 0 & \tilde{\alpha}_{2,1} & \tilde{\alpha}_{2,2} & \tilde{\alpha}_{2,3} \\ 0 & 0 & \tilde{\alpha}_{3,2} & \tilde{\alpha}_{3,3} \end{array} \right) = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 1 & 0 & \gamma_{3,2} & \sigma_{3,2} \\ 0 & 1 & -\sigma_{3,2} & \gamma_{3,2} \end{array} \right) \left(\begin{array}{cc|cc} \tilde{\alpha}_{0,0} & \tilde{\alpha}_{1,0} & 0 & 0 \\ \tilde{\alpha}_{1,0} & \tilde{\alpha}_{1,1} & \widehat{\tilde{\alpha}}_{2,1} & \widehat{\tilde{\alpha}}_{3,1} \\ \hline 0 & \widehat{\tilde{\alpha}}_{2,1} & \widehat{\tilde{\alpha}}_{2,2} & \widehat{\tilde{\alpha}}_{2,3} \\ 0 & \widehat{\tilde{\alpha}}_{3,1} & \widehat{\tilde{\alpha}}_{3,2} & \alpha_{3,3} \end{array} \right) \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 1 & 0 & \gamma_{3,1} & -\sigma_{3,1} \\ 0 & 1 & \sigma_{3,1} & \gamma_{3,1} \end{array} \right)$$

The matrix Q is the orthogonal matrix that results from multiplying the different Givens' rotations together:

$$Q = \left(\begin{array}{cc|cc} \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & \gamma_{2,0} & -\sigma_{2,0} & 0 \\ \hline 0 & \sigma_{2,0} & \gamma_{2,0} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & \gamma_{3,1} & -\sigma_{3,1} \\ 0 & 0 & \sigma_{3,1} & \gamma_{3,1} \end{array} \right).$$

It is important to note that the first columns of Q is given by

$$\begin{pmatrix} \gamma_{1,0} \\ \sigma_{1,0} \\ 0 \\ 0 \end{pmatrix}$$

, which is exactly the same first column had Q been computed as in Section 16.6 (Equation 16.2). Thus, by the Implicit Q Theorem, the tridiagonal matrix that results from this approach is equal to the tridiagonal matrix that would be computed by applying the QR factorization from Section 16.6 with $A - \mu I, A - \mu I \rightarrow QR$ followed by the formation of $RQ + \mu I$ using the algorithm for computing RQ in Section 16.6.

The successive elimination of elements $\widehat{\alpha}_{i+1,i}$ is often referred to as *chasing the bulge* while the entire process that introduces the bulge and then chases it is known as a Francis Implicit QR Step. Obviously, the method generalizes to matrices of arbitrary size, as illustrated in Figure 16.8. An algorithm for the chasing of the bulge is given in Figure 18.4. (Note that in those figures T is used for A , something that needs to be made consistent in these notes, eventually.) In practice, the tridiagonal matrix is not stored as a matrix. Instead, its diagonal and subdiagonal are stored as vectors.

16.7.4 A complete algorithm

This last section shows how one iteration of the QR algorithm can be performed on a tridiagonal matrix by implicitly shifting and then “chasing the bulge”. All that is left to complete the algorithm is to note that

- The shift μ_k can be chosen to equal $\alpha_{n-1,n-1}$ (the last element on the diagonal, which tends to converge to the eigenvalue smallest in magnitude). In practice, choosing the shift to be an eigenvalue of the bottom-right 2×2 matrix works better. This is known as the *Wilkinson Shift*.

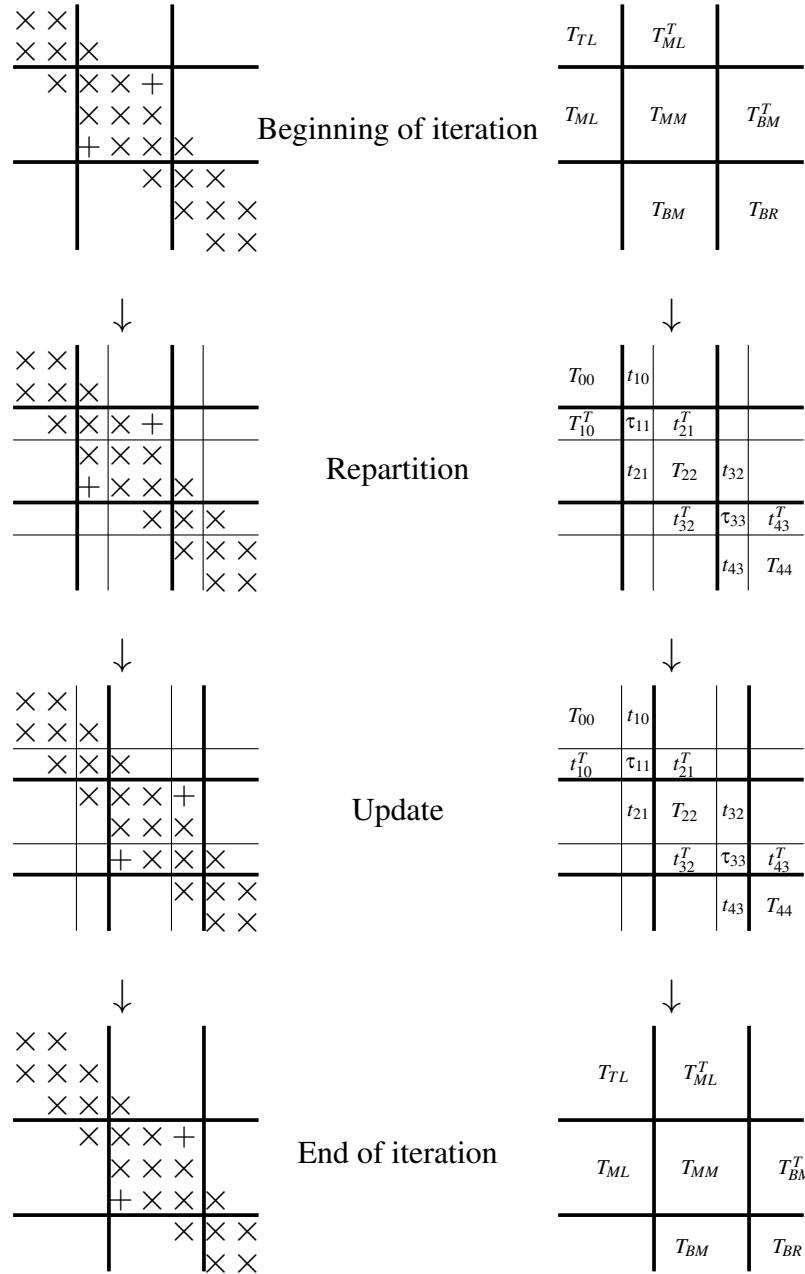


Figure 16.8: One step of “chasing the bulge” in the implicitly shifted symmetric QR algorithm.

- If an element of the subdiagonal (and corresponding element on the superdiagonal) becomes small enough, it can be considered to be zero and the problem deflates (decouples) into two smaller tridiagonal matrices. Small is often taken to mean that $|\alpha_{i+1,i}| \leq \epsilon(|\alpha_{i,i}| + |\alpha_{i+1,i+1}|)$ where ϵ is some quantity close to the machine epsilon (unit roundoff).
- If $A = QTQ^T$ reduced A to the tridiagonal matrix T before the QR algorithm commenced, then the Givens’ rotations encountered as part of the implicitly shifted QR algorithm can be applied from the right to the appropriate columns of Q so that upon completion Q is overwritten with the eigenvectors of A . Notice that applying a Givens’ rotation to a pair of columns of Q requires $O(n)$ computation

Algorithm: $T := \text{CHASEBULGE}(T)$

Partition $T \rightarrow \begin{pmatrix} T_{TL} & * & * \\ \hline T_{ML} & T_{MM} & * \\ \hline 0 & T_{BM} & T_{BR} \end{pmatrix}$

where T_{TL} is 0×0 and T_{MM} is 3×3

while $m(T_{BR}) > 0$ **do**

Repartition

$$\begin{pmatrix} T_{TL} & * & 0 \\ \hline T_{ML} & T_{MM} & * \\ \hline 0 & T_{BM} & T_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} T_{00} & * & 0 & 0 & 0 \\ \hline t_{10}^T & \tau_{11} & * & 0 & 0 \\ \hline 0 & t_{21} & T_{22} & * & 0 \\ \hline 0 & 0 & t_{32}^T & \tau_{33} & * \\ \hline 0 & 0 & 0 & t_{43} & T_{44} \end{pmatrix}$$

where τ_{11} and τ_{33} are scalars
(during final step, τ_{33} is 0×0)

Compute (γ, σ) s.t. $G_{\gamma, \sigma}^T t_{21} = \begin{pmatrix} \tau_{21} \\ 0 \end{pmatrix}$, and **assign** $t_{21} = \begin{pmatrix} \tau_{21} \\ 0 \end{pmatrix}$

$$T_{22} = G_{\gamma, \sigma}^T T_{22} G_{\gamma, \sigma}$$

$$t_{32}^T = t_{32}^T G_{\gamma, \sigma} \quad (\text{not performed during final step})$$

Continue with

$$\begin{pmatrix} T_{TL} & * & 0 \\ \hline T_{ML} & T_{MM} & * \\ \hline 0 & T_{BM} & T_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} T_{00} & * & 0 & 0 & 0 \\ \hline t_{10}^T & \tau_{11} & * & 0 & 0 \\ \hline 0 & t_{21} & T_{22} & * & 0 \\ \hline 0 & 0 & t_{32}^T & \tau_{33} & * \\ \hline 0 & 0 & 0 & t_{43} & T_{44} \end{pmatrix}$$

endwhile

Figure 16.9: Chasing the bulge.

per Givens rotation. For each Francis implicit QR step $O(n)$ Givens' rotations are computed, making the application of Givens' rotations to Q of cost $O(n^2)$ per iteration of the implicitly shifted QR algorithm. Typically a few (2-3) iterations are needed per eigenvalue that is uncovered (by deflation) meaning that $O(n)$ iterations are needed. Thus, the QR algorithm is roughly of cost $O(n^3)$ if the eigenvalues are accumulated (in addition to the cost of forming the Q from the reduction to tridiagonal form, which takes another $O(n^3)$ operations.)

- If an element on the subdiagonal becomes zero (or very small), and hence the corresponding element of the superdiagonal, then the problem can be *deflated*: If

$$T = \left(\begin{array}{c|c} T_{00} & 0 \\ \hline 0 & T_{11} \end{array} \right)$$

then

- The computation can continue separately with T_{00} and T_{11} .
- One can pick the shift from the bottom-right of T_{00} as one continues finding the eigenvalues of T_{00} , thus accelerating the computation.
- One can pick the shift from the bottom-right of T_{11} as one continues finding the eigenvalues of T_{11} , thus accelerating the computation.
- One must continue to accumulate the eigenvectors by applying the rotations to the appropriate columns of Q .

Because of the connection between the QR algorithm and the Inverse Power Method, subdiagonal entries near the bottom-right of T are more likely to converge to a zero, so most deflation will happen there.

- A question becomes when an element on the subdiagonal, $\tau_{i+1,i}$ can be considered to be zero. The answer is when $|\tau_{i+1,i}|$ is small relative to $|\tau_i|$ and $|\tau_{i+1,i+1}|$. A typical condition that is used is

$$|\tau_{i+1,i}| \leq \mathbf{u} \sqrt{|\tau_{i,i}\tau_{i+1,i+1}|}.$$

- If $A \in \mathbb{C}^{n \times n}$ is Hermitian, then its Spectral Decomposition is also computed via the following steps, which mirror those for the real symmetric case:

- Reduce to tridiagonal form. Householder transformation based similarity transformations can again be used for this. This leaves one with a tridiagonal matrix, T , with real values along the diagonal (because the matrix is Hermitian) and values on the subdiagonal and superdiagonal that may be complex valued.
- The matrix Q_T such $A = Q_T T Q_T^H$ can then be formed from the Householder transformations.
- A simple step can be used to then change this tridiagonal form to have real values even on the subdiagonal and superdiagonal. The matrix Q_T can be updated accordingly.
- The tridiagonal QR algorithm that we described can then be used to diagonalize the matrix, accumulating the eigenvectors by applying the encountered Givens' rotations to Q_T . This is where the real expense is: Apply the Givens' rotations to matrix T requires $O(n)$ per sweep. Applying the Givens' rotation to Q_T requires $O(n^2)$ per sweep.

For details, see some of our papers mentioned in the next section.

16.8 Further Reading

16.8.1 More on reduction to tridiagonal form

The reduction to tridiagonal form can only be partially cast in terms of matrix-matrix multiplication [21]. This is a severe hindrance to high performance for that first step towards computing all eigenvalues and eigenvector of a symmetric matrix. Worse, a considerable fraction of the total cost of the computation is in that first step.

For a detailed discussion on the blocked algorithm that uses FLAME notation, we recommend [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. 39, No. 1, 2012

(Reduction to tridiagonal form is one case of what is more generally referred to as “condensed form”.)

16.8.2 Optimizing the tridiagonal QR algorithm

As the Givens’ rotations are applied to the tridiagonal matrix, they are also applied to a matrix in which eigenvectors are accumulated. While one Implicit Francis Step requires $O(n)$ computation, this accumulation of the eigenvectors requires $O(n^2)$ computation with $O(n^2)$ data. We have learned before that this means the cost of accessing data dominates on current architectures.

In a recent paper, we showed how accumulating the Givens’ rotations for several Francis Steps allows one to attain performance similar to that attained by a matrix-matrix multiplication. Details can be found in [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), Vol. 40, No. 3, 2014.

16.9 Other Algorithms

16.9.1 Jacobi’s method for the symmetric eigenvalue problem

(Not to be mistaken for the Jacobi iteration for solving linear systems.)

The oldest algorithm for computing the eigenvalues and eigenvectors of a matrix is due to Jacobi and dates back to 1846 [27]. This is a method that keeps resurfacing, since it parallelizes easily. The operation count tends to be higher (by a constant factor) than that of reduction to tridiagonal form followed by the tridiagonal QR algorithm.

The idea is as follows: Given a symmetric 2×2 matrix

$$A_{31} = \begin{pmatrix} \alpha_{11} & \alpha_{13} \\ \alpha_{31} & \alpha_{33} \end{pmatrix}$$

Sweep 1											
\times	0	\times	\times	\times	\times	0	\times	\times	\times	0	
0	\times	\times	\times		\rightarrow	\times	\times	\times	\times		\rightarrow
\times	\times	\times	\times			0	\times	\times	\times		
\times	\times	\times	\times				\times	\times	\times		
zero (1,0)				zero (2,0)				zero (3,0)			
\times	\times	\times	0	\times	\times	\times	\times	\times	\times	\times	
\times	\times	0	\times		\rightarrow	\times	\times	\times	0		\rightarrow
\times	0	\times	\times			\times	\times	\times	\times	0	
0	\times	\times	\times			\times	0	\times	\times	\times	
zero (2,1)				zero (3,1)				zero (3,2)			
Sweep 2											
\times	0	\times	\times	\times	\times	0	\times	\times	\times	0	
0	\times	\times	\times		\rightarrow	\times	\times	\times	\times		\rightarrow
\times	\times	\times	0			0	\times	\times	\times		
\times	\times	0	\times				\times	\times	\times		
zero (1,0)				zero (2,0)				zero (3,0)			
\times	\times	\times	0	\times	\times	\times	\times	\times	\times	\times	
\times	\times	0	\times		\rightarrow	\times	\times	\times	0		\rightarrow
\times	0	\times	\times			\times	\times	\times	\times	0	
0	\times	\times	\times			\times	0	\times	\times	\times	
zero (2,1)				zero (3,1)				zero (3,2)			

Figure 16.10: Column-cyclic Jacobi algorithm.

There exists a rotation (which is of course unitary)

$$J_{31} = \begin{pmatrix} \gamma_{11} & -\sigma_{13} \\ \sigma_{31} & \gamma_{33} \end{pmatrix}$$

such that

$$J_{31} A_{31} J_{31}^T = \begin{pmatrix} \gamma_{11} & -\sigma_{31} \\ \sigma_{31} & \gamma_{33} \end{pmatrix} \begin{pmatrix} \alpha_{11} & \alpha_{31} \\ \alpha_{31} & \alpha_{33} \end{pmatrix} \begin{pmatrix} \gamma_{11} & -\sigma_{31} \\ \sigma_{31} & \gamma_{33} \end{pmatrix}^T = \begin{pmatrix} \hat{\alpha}_{11} & 0 \\ 0 & \hat{\alpha}_{33} \end{pmatrix}.$$

We know this exists since the Spectral Decomposition of the 2×2 matrix exists. Such a rotation is called a Jacobi rotation. (Notice that it is different from a Givens' rotation because it diagonalizes a 2×2 matrix

when used as a unitary similarity transformation. By contrast, a Givens' rotation zeroes an element when applied from one side of a matrix.)

Homework 16.11 In the above discussion, show that $\alpha_{11}^2 + 2\alpha_{31}^2 + \alpha_{33}^2 = \hat{\alpha}_{11}^2 + \hat{\alpha}_{33}^2$.

SEE ANSWER

Jacobi rotation rotations can be used to selectively zero off-diagonal elements by observing the following:

$$JAJ^T = \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ 0 & \gamma_{11} & 0 & -\sigma_{31} & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & \sigma_{31} & 0 & \gamma_{33} & 0 \\ 0 & 0 & 0 & 0 & I \end{pmatrix} \begin{pmatrix} A_{00} & \mathbf{a}_{10} & A_{20}^T & \mathbf{a}_{30} & A_{40}^T \\ \mathbf{a}_{10}^T & \alpha_{11} & \mathbf{a}_{21}^T & \alpha_{31} & \mathbf{a}_{41}^T \\ A_{20} & \mathbf{a}_{21} & A_{22} & \mathbf{a}_{32} & A_{42}^T \\ \mathbf{a}_{30}^T & \alpha_{31} & \mathbf{a}_{32}^T & \alpha_{33} & \mathbf{a}_{43}^T \\ A_{40} & \mathbf{a}_{41} & A_{42} & \mathbf{a}_{43} & A_{44} \end{pmatrix} \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ 0 & \gamma_{11} & 0 & -\sigma_{13} & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & \sigma_{31} & 0 & \gamma_{33} & 0 \\ 0 & 0 & 0 & 0 & I \end{pmatrix}^T$$

$$= \begin{pmatrix} A_{00} & \hat{\mathbf{a}}_{10} & A_{20}^T & \hat{\mathbf{a}}_{30} & A_{40}^T \\ \hat{\mathbf{a}}_{10}^T & \hat{\alpha}_{11} & \hat{\mathbf{a}}_{21}^T & 0 & \hat{\mathbf{a}}_{41}^T \\ A_{20} & \hat{\mathbf{a}}_{21} & A_{22} & \hat{\mathbf{a}}_{32} & A_{42}^T \\ \hat{\mathbf{a}}_{30}^T & 0 & \hat{\mathbf{a}}_{32}^T & \hat{\alpha}_{33} & \hat{\mathbf{a}}_{43}^T \\ A_{40} & \hat{\mathbf{a}}_{41} & A_{42} & \hat{\mathbf{a}}_{43} & A_{44} \end{pmatrix} = \hat{A},$$

where

$$\begin{pmatrix} \gamma_{11} & -\sigma_{31} \\ \sigma_{31} & \gamma_{33} \end{pmatrix} \begin{pmatrix} \mathbf{a}_{10}^T \\ \mathbf{a}_{30}^T \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{a}}_{10}^T \\ \hat{\mathbf{a}}_{30}^T \end{pmatrix}.$$

Importantly,

$$\begin{aligned} a_{10}^T a_{10} + a_{30}^T a_{30} &= \hat{a}_{10}^T \hat{a}_{10} + \hat{a}_{30}^T \hat{a}_{30} \\ a_{21}^T a_{21} + a_{32}^T a_{32} &= \hat{a}_{21}^T \hat{a}_{21} + \hat{a}_{32}^T \hat{a}_{32} \\ a_{41}^T a_{41} + a_{43}^T a_{43} &= \hat{a}_{41}^T \hat{a}_{41} + \hat{a}_{43}^T \hat{a}_{43}. \end{aligned}$$

What this means is that if one defines $\text{off}(A)$ as the square of the Frobenius norm of the off-diagonal elements of A ,

$$\text{off}(A) = \|A\|_F^2 - \|\text{diag}(A)\|_F^2,$$

then $\text{off}(\hat{A}) = \text{off}(A) - 2\alpha_{31}^2$.

- The good news: every time a Jacobi rotation is used to zero an off-diagonal element, $\text{off}(A)$ decreases by twice the square of that element.
- The bad news: a previously introduced zero may become nonzero in the process.

The original algorithm developed by Jacobi searched for the largest (in absolute value) off-diagonal element and zeroed it, repeating this process until all off-diagonal elements were small. The algorithm was applied by hand by one of his students, Seidel (of Gauss-Seidel fame). The problem with this is that searching for the largest off-diagonal element requires $O(n^2)$ comparisons. Computing and applying one Jacobi rotation as a similarity transformation requires $O(n)$ flops. Thus, for large n this is not practical. Instead, it can be shown that zeroing the off-diagonal elements by columns (or rows) also converges to a diagonal matrix. This is known as the column-cyclic Jacobi algorithm. We illustrate this in Figure 16.10.

16.9.2 Cuppen's Algorithm

To be added at a future time.

16.9.3 The Method of Multiple Relatively Robust Representations (MRRR)

Even once the problem has been reduced to tridiagonal form, the computation of the eigenvalues and eigenvectors via the QR algorithm requires $O(n^3)$ computations. A method that reduces this to $O(n^2)$ time (which can be argued to achieve the lower bound for computation, within a constant, because the n vectors must be at least written) is achieved by the Method of Multiple Relatively Robust Representations (MRRR) by Dhillon and Partlett [14, 13, 15, 16]. The details of that method go beyond the scope of this note.

16.10 The Nonsymmetric QR Algorithm

The QR algorithm that we have described can be modified to compute the Schur decomposition of a nonsymmetric matrix. We briefly describe the high-level ideas.

16.10.1 A variant of the Schur decomposition

Let $A \in \mathbb{R}^{n \times n}$ be nonsymmetric. Recall:

- There exists a unitary matrix $Q \in \mathbb{C}^{n \times n}$ and upper triangular matrix $R \in \mathbb{C}^{n \times n}$ such that $A = QRQ^H$. Importantly: even if A is real valued, the eigenvalues and eigenvectors may be complex valued.
- The eigenvalues will come in conjugate pairs.

A variation of the Schur Decomposition theorem is

Theorem 16.12 *Let $A \in \mathbb{R}^{n \times n}$. Then there exist unitary $Q \in \mathbb{R}^{n \times n}$ and quasi upper triangular matrix $R \in \mathbb{R}^{n \times n}$ such that $A = QRQ^T$.*

Here quasi upper triangular matrix means that the matrix is block upper triangular with blocks of the diagonal that are 1×1 or 2×2 .

Remark 16.13 *The important thing is that this alternative to the Schur decomposition can be computed using only real arithmetic.*

Algorithm: $[A, t] := \text{HESSRED_UNB}(A)$

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

where A_{TL} is 0×0 and t_b has 0 elements

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

Repartition

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

where α_{11} is a scalar

$$[u_{21}, \tau_1, a_{21}] := \text{HouseV}(a_{21})$$

$$y_{01} := A_{02}u_{21}$$

$$A_{02} := A_{02} - \frac{1}{\tau}y_{01}u_{21}^T$$

$$\psi_{11} := a_{12}^T u_{21}$$

$$a_{12}^T := a_{12}^T + \frac{\psi_{11}}{\tau}u_{21}^T$$

$$y_{21} := A_{22}u_{21}$$

$$\beta := u_{21}^T y_{21} / 2$$

$$z_{21} := (y_{21} - \beta u_{21} / \tau_1) / \tau_1$$

$$w_{21} := (A_{22}^T u_{21} - \beta u_{21} / \tau) / \tau$$

$$A_{22} := A_{22} - (u_{21} w_{21}^T + z_{21} u_{21}^T) \quad (\text{rank-2 update})$$

Continue with

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

endwhile

Figure 16.11: Basic algorithm for reduction of a nonsymmetric matrix to upperHessenberg form.

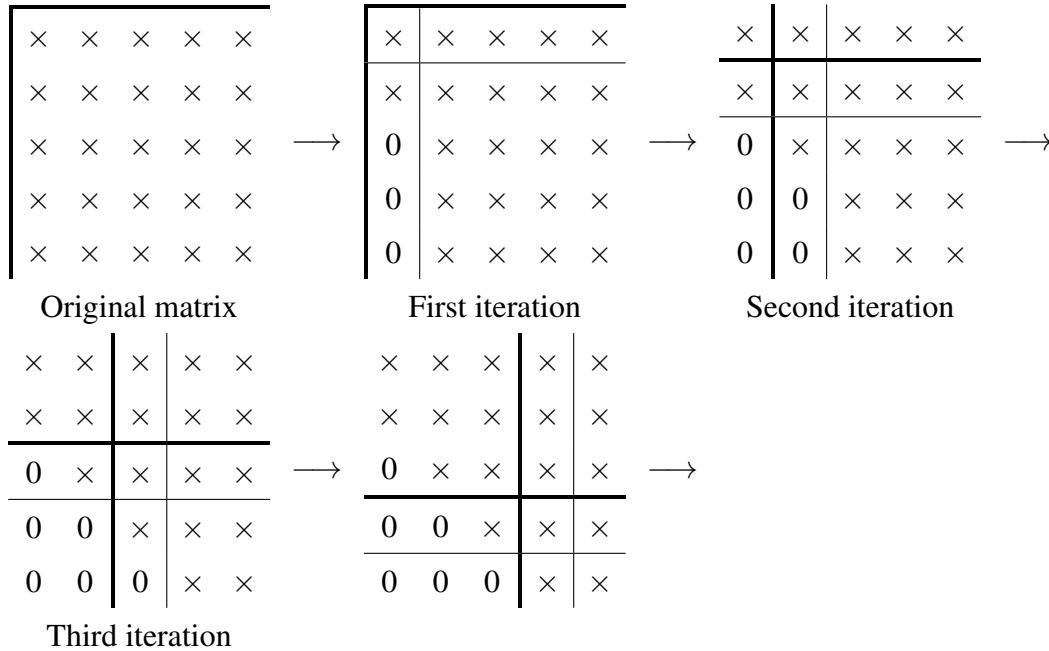


Figure 16.12: Illustration of reduction of a nonsymmetric matrix to upperHessenberg form. The \times s denote nonzero elements in the matrix.

16.10.2 Reduction to upperHessenberg form

The basic algorithm for reducing a real-valued nonsymmetric matrix to upperHessenberg form, overwriting the original matrix with the result, can be explained similar to the explanation of the reduction of a symmetric matrix to tridiagonal form. We assume that the upperHessenberg matrix overwrites the upper-Hessenberg part of A and the Householder vectors used to zero out parts of A overwrite the entries that they annihilate (set to zero).

- Assume that the process has proceeded to where $A = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right)$ with A_{00} being $k \times k$ and upperHessenberg, a_{10}^T being a row vector with only a last nonzero entry, the rest of the submatrices updated according to the application of previous k Householder transformations.

- Let $[u_{21}, \tau, a_{21}] := \text{HouseV}(a_{21})$.³

- Update

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & H \end{array} \right) \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline 0 & a_{21} & A_{22} \end{array} \right) \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & H \end{array} \right)$$

³ Note that the semantics here indicate that a_{21} is overwritten by Ha_{21} .

$$= \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02}H \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T H \\ \hline 0 & Ha_{21} & HA_{22}H \end{array} \right)$$

where $H = H(u_{21})$.

- Note that $a_{21} := Ha_{21}$ need not be executed since this update was performed by the instance of HouseV above.⁴

- The update $A_{02}H$ requires

$$\begin{aligned} A_{02} &:= A_{02}(I - \frac{1}{\tau} u_{21} u_{21}^T) \\ &= A_{02} - \frac{1}{\tau} \underbrace{A_{02} u_{21}}_{y_{01}} u_{21}^T \\ &= A_{02} - \frac{1}{\tau} y_{01} u_{21}^T \quad (\text{ger}) \end{aligned}$$

- The update $a_{12}^T H$ requires

$$\begin{aligned} a_{12}^T &:= a_{12}^T(I - \frac{1}{\tau} u_{21} u_{21}^T) \\ &= a_{12}^T - \frac{1}{\tau} \underbrace{a_{12}^T u_{21}}_{\Psi_{11}} u_{21}^T \\ &= a_{12}^T - \left(\frac{\Psi_{11}}{\tau} \right) u_{21}^T \quad (\text{axpy}) \end{aligned}$$

- The update of A_{22} requires

$$\begin{aligned} A_{22} &:= (I - \frac{1}{\tau} u_{21} u_{21}^T) A_{22} (I - \frac{1}{\tau} u_{21} u_{21}^T) \\ &= (A_{22} - \frac{1}{\tau} u_{21} u_{21}^T A_{22})(I - \frac{1}{\tau} u_{21} u_{21}^T) \\ &= A_{22} - \frac{1}{\tau} u_{21} u_{21}^T A_{22} - \frac{1}{\tau} A_{22} u_{21} u_{21}^T + \frac{1}{\tau^2} u_{21} \underbrace{u_{21}^T A u_{21}}_{2\beta} u_{21}^T \\ &= A_{22} - \left(\frac{1}{\tau} u_{21} u_{21}^T A_{22} - \frac{\beta}{\tau^2} u_{21} u_{21}^T \right) - \left(\frac{1}{\tau} A u_{21} u_{21}^T - \frac{\beta}{\tau^2} u_{21} u_{21}^T \right) \\ &= A_{22} - \underbrace{\frac{1}{\tau} u_{21} \left(u_{21}^T A_{22} - \frac{\beta}{\tau} u_{21}^T \right)}_{w_{21}^T} - \underbrace{\left(A u_{21} - \frac{\beta}{\tau} u_{21} \right) \frac{1}{\tau} u_{21}^T}_{z_{21}} \end{aligned}$$

⁴ In practice, the zeros below the first element of Ha_{21} are not actually written. Instead, the implementation overwrites these elements with the corresponding elements of the vector u_{21} .

$$= \underbrace{A_{22} - \frac{1}{\tau} u_{21} w_{21}^T - \frac{1}{\tau} z_{21} u_{21}^T}_{\text{rank-2 update}}$$

- Continue this process with the updated A .

It doesn't suffice to only This is captured in the algorithm in Figure 16.11. It is also illustrated in Figure 16.12.

Homework 16.14 *Give the approximate total cost for reducing a nonsymmetric $A \in \mathbb{R}^{n \times n}$ to upper-Hessenberg form.*

☞ SEE ANSWER

For a detailed discussion on the blocked algorithm that uses FLAME notation, we recommend [34]

Gregorio Quintana-Ortí and Robert A. van de Geijn.

Improving the performance of reduction to Hessenberg form.

ACM Transactions on Mathematical Software (TOMS) , Vol. 32, No. 2, 2006

and [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. 39, No. 1, 2012

In those papers, citations to earlier work can be found.

16.10.3 The implicitly double-shifted QR algorithm

To be added at a future date!

Chapter **17**

Notes on the Method of Relatively Robust Representations (MRRR)

The purpose of this note is to give some high level idea of how the Method of Relatively Robust Representations (MRRR) works.

17.0.1 Outline

17.0.1	Outline	332
17.1	MRRR, from 35,000 Feet	332
17.2	Cholesky Factorization, Again	335
17.3	The LDL^T Factorization	335
17.4	The UDU^T Factorization	337
17.5	The UDU^T Factorization	340
17.6	The Twisted Factorization	340
17.7	Computing an Eigenvector from the Twisted Factorization	343

17.1 MRRR, from 35,000 Feet

The Method of Relatively Robust Representations (MRRR) is an algorithm that, given a tridiagonal matrix, computes eigenvectors associated with that matrix in $O(n)$ time per eigenvector. This means it computes all eigenvectors in $O(n^2)$ time, which is much faster than the tridiagonal QR algorithm (which requires $O(n^3)$ computation). Notice that highly accurate eigenvalues of a tridiagonal matrix can themselves be computed in $O(n^2)$ time. So, it is legitimate to start by assuming that we have these highly accurate eigenvalues. For our discussion, we only need one.

The MRRR algorithm has at least two benefits of the symmetric QR algorithm for tridiagonal matrices:

- It can compute all eigenvalues and eigenvectors of tridiagonal matrix in $O(n^2)$ time (versus $O(n^3)$ time for the symmetric QR algorithm).
- It can efficiently compute eigenvectors corresponding to a subset of eigenvalues.

The benefit when computing all eigenvalues and eigenvectors of a dense matrix is considerably less because transforming the eigenvectors of the tridiagonal matrix back into the eigenvectors of the dense matrix requires an extra $O(n^3)$ computation, as discussed in [44]. In addition, making the method totally robust has been tricky, since the method does not rely exclusively on unitary similarity transformations.

The fundamental idea is that of a twisted factorization of the tridiagonal matrix. This note builds up to what that factorization is, how to compute it, and how to then compute an eigenvector with it. We start by reminding the reader of what the Cholesky factorization of a symmetric positive definite (SPD) matrix is. Then we discuss the Cholesky factorization of a tridiagonal SPD matrix. This then leads to the LDL^T factorization of an indefinite and indefinite tridiagonal matrix. Next follows a discussion of the UDU^T factorization of an indefinite matrix, which then finally yields the twisted factorization. When the matrix is nearly singular, an approximation of the twisted factorization can then be used to compute an approximate eigenvalue.

Notice that the devil is in the details of the MRRR algorithm. We will not tackle those details.

Algorithm: $A := \text{CHOL}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{array} \right)$
where A_{TL} is 0×0

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

Repartition

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

$\alpha_{11} := \sqrt{\alpha_{11}}$
 $a_{21} := a_{21}/\alpha_{11}$
 $A_{22} := A_{22} - a_{21}a_{21}^T$

Continue with

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

endwhile

Figure 17.1: Unblocked algorithm for computing the Cholesky factorization. Updates to A_{22} affect only the lower triangular part.

Algorithm: $A := \text{CHOL_TRI}(A)$		
Partition $A \rightarrow \left(\begin{array}{c c c} A_{FF} & * & * \\ \hline \alpha_{MFE} e_L^T & \alpha_{MM} & * \\ \hline 0 & \alpha_{LME} e_F & A_{LL} \end{array} \right)$		
where A_{FF} is 0×0		
while $m(A_{FF}) < m(A)$ do		
Repartition		
$\left(\begin{array}{c c c} A_{FF} & * & * \\ \hline \alpha_{MFE} e_L^T & \alpha_{MM} & * \\ \hline 0 & \alpha_{LME} e_F & A_{LL} \end{array} \right) \rightarrow \left(\begin{array}{c c c c} A_{00} & * & * & \\ \hline \alpha_{10} e_L^T & \alpha_{11} & * & * \\ \hline 0 & \alpha_{21} & \alpha_{22} & * \\ \hline 0 & 0 & \alpha_{32} e_F & A_{33} \end{array} \right)$		
<hr/>		
$\alpha_{11} := \sqrt{\alpha_{11}}$		
$\alpha_{21} := \alpha_{21}/\alpha_{11}$		
$\alpha_{22} := \alpha_{22} - \alpha_{21}^2$		
<hr/>		
Continue with		
$\left(\begin{array}{c c c} A_{FF} & * & * \\ \hline \alpha_{MFE} e_L^T & \alpha_{MM} & * \\ \hline 0 & \alpha_{LME} e_F & A_{LL} \end{array} \right) \leftarrow \left(\begin{array}{c c c c} A_{00} & * & * & \\ \hline \alpha_{10} e_L^T & \alpha_{11} & * & * \\ \hline 0 & \alpha_{21} & \alpha_{22} & * \\ \hline 0 & 0 & \alpha_{32} e_F & A_{33} \end{array} \right)$		
endwhile		

Figure 17.2: Algorithm for computing the Cholesky factorization of a tridiagonal matrix.

17.2 Cholesky Factorization, Again

We have discussed in class the Cholesky factorization, $A = LL^T$, which requires a matrix to be symmetric positive definite. (We will restrict our discussion to real matrices.) The following computes the Cholesky factorization:

- Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{21}^T \\ \hline a_{21} & A_{22} \end{array} \right)$.
- Update $\alpha_{11} := \sqrt{\alpha_{11}}$.
- Update $a_{21} := a_{21}/\alpha_{11}$.
- Update $A_{22} := A_{22} - a_{21}a_{21}^T$ (updating only the lower triangular part).
- Continue to compute the Cholesky factorization of the updated A_{22} .

The resulting algorithm is given in Figure 17.1.

In the special case where A is tridiagonal and SPD, the algorithm needs to be modified so that it can take advantage of zero elements:

- Partition $A \rightarrow \left(\begin{array}{c|c|c} \alpha_{11} & * & * \\ \hline \alpha_{21} & \alpha_{22} & * \\ \hline 0 & \alpha_{32}e_F & A_{33} \end{array} \right)$, where $*$ indicates the symmetric part that is not stored. Here e_F indicates the unit basis vector with a “1” as first element.
- Update $\alpha_{11} := \sqrt{\alpha_{11}}$.
- Update $\alpha_{21} := \alpha_{21}/\alpha_{11}$.
- Update $\alpha_{22} := \alpha_{22} - \alpha_{21}^2$.
- Continue to compute the Cholesky factorization of $\left(\begin{array}{c|c} \alpha_{22} & * \\ \hline \alpha_{32}e_F & A_{33} \end{array} \right)$

The resulting algorithm is given in Figure 17.2. In that figure, it helps to interpret F , M , and L as First, Middle, and Last. In that figure, e_F and e_L are the unit basis vectors with a “1” as first and last element, respectively. Notice that the Cholesky factor of a tridiagonal matrix is a lower bidiagonal matrix that overwrites the lower triangular part of the tridiagonal matrix A .

Naturally, the whole matrix needs not be stored. But that detail is not important for our discussion.

17.3 The LDL^T Factorization

Now, one can alternatively compute $A = LDL^T$, where L is unit lower triangular and D is diagonal. We will look at how this is done first and will then note that this factorization can be computed for any indefinite

Algorithm: $A := \text{LDLT}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{array} \right)$
where A_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1

Continue with

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

endwhile

Figure 17.3: Unblocked algorithm for computing $A \rightarrow LDL^T$, overwriting A .

(nonsingular) symmetric matrix. (Notice: the so computed L is *not* the same as the L computed by the Cholesky factorization.) Partition

$$A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{21}^T \\ \hline a_{21} & A_{22} \end{array} \right), L \rightarrow \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right), \text{ and } D \rightarrow \left(\begin{array}{c|c} \delta_1 & 0 \\ \hline 0 & D_{22} \end{array} \right).$$

Then

$$\begin{aligned} \left(\begin{array}{c|c} \alpha_{11} & a_{21}^T \\ \hline a_{21} & A_{22} \end{array} \right) &= \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right) \left(\begin{array}{c|c} \delta_1 & 0 \\ \hline 0 & D_{22} \end{array} \right) \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right)^T \\ &= \left(\begin{array}{c|c} \delta_{11} & 0 \\ \hline \delta_{11}l_{21} & L_{22}D_{22} \end{array} \right) \left(\begin{array}{c|c} 1 & l_{21}^T \\ \hline 0 & L_{22}^T \end{array} \right) \end{aligned}$$

$$= \left(\begin{array}{c|c} \delta_{11} & * \\ \hline \delta_{11}l_{21} & \delta_{11}l_{21}l_{21}^T + L_{22}D_{22}L_{22}^T \end{array} \right)$$

This suggests the following algorithm for overwriting the strictly lower triangular part of A with the strictly lower triangular part of L and the diagonal of A with D :

- Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{21}^T \\ \hline a_{21} & A_{22} \end{array} \right)$.
- $\alpha_{11} := \delta_{11} = \alpha_{11}$ (no-op).
- Compute $l_{21} := a_{21}/\alpha_{11}$.
- Update $A_{22} := A_{22} - l_{21}a_{21}^T$ (updating only the lower triangular part).
- $a_{21} := l_{21}$.
- Continue with computing $A_{22} \rightarrow L_{22}D_{22}L_{22}^T$.

This algorithm will complete as long as $\delta_{11} \neq 0$, which happens when A is nonsingular. This is equivalent to A does not have zero as an eigenvalue. Such a matrix is also called *indefinite*.

Homework 17.1 Modify the algorithm in Figure 17.1 so that it computes the LDL^T factorization. (Fill in Figure 17.3.)

☞ SEE ANSWER

Homework 17.2 Modify the algorithm in Figure 17.2 so that it computes the LDL^T factorization of a tridiagonal matrix. (Fill in Figure 17.4.) What is the approximate cost, in floating point operations, of computing the LDL^T factorization of a tridiagonal matrix? Count a divide, multiply, and add/subtract as a floating point operation each. Show how you came up with the algorithm, similar to how we derived the algorithm for the tridiagonal Cholesky factorization.

☞ SEE ANSWER

Notice that computing the LDL^T factorization of an indefinite matrix is not a good idea when A is nearly singular. This would lead to some δ_{11} being nearly zero, meaning that dividing by it leads to very large entries in l_{21} and corresponding element growth in A_{22} . (In other words, strange things will happen “down stream” from the small δ_{11} .) Not a good thing. Unfortunately, we are going to need something like this factorization specifically for the case where A is nearly singular.

17.4 The UDU^T Factorization

The fact that the LU, Cholesky, and LDL^T factorizations start in the top-left corner of the matrix and work towards the bottom-right is an accident of history. Gaussian elimination works in that direction, hence so does LU factorization, and the rest kind of follow.

One can imagine, given a SPD matrix A , instead computing $A = UU^T$, where U is upper triangular. Such a computation starts in the lower-right corner of the matrix and works towards the top-left. Similarly, an algorithm can be created for computing $A = UDU^T$ where U is unit upper triangular and D is diagonal.

Algorithm: $A := \text{LDLT_TRI}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c|c} A_{FF} & * & * \\ \hline \alpha_{MFE} e_L^T & \alpha_{MM} & * \\ \hline 0 & \alpha_{LME} e_F & A_{LL} \end{array} \right)$

where A_{LL} is 0×0

while $m(A_{FF}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c|c} A_{FF} & * & * \\ \hline \alpha_{MFE} e_L^T & \alpha_{MM} & * \\ \hline 0 & \alpha_{LME} e_F & A_{LL} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c|c} A_{00} & * & * & \\ \hline \alpha_{10} e_L^T & \alpha_{11} & * & * \\ \hline 0 & \alpha_{21} & \alpha_{22} & * \\ \hline 0 & 0 & \alpha_{32} e_F & A_{33} \end{array} \right)$$

where α_{22} is a scalar

Continue with

$$\left(\begin{array}{c|c|c} A_{FF} & * & * \\ \hline \alpha_{MFE} e_L^T & \alpha_{MM} & * \\ \hline 0 & \alpha_{LME} e_F & A_{LL} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c|c} A_{00} & * & * & \\ \hline \alpha_{10} e_L^T & \alpha_{11} & * & * \\ \hline 0 & \alpha_{21} & \alpha_{22} & * \\ \hline 0 & 0 & \alpha_{32} e_F & A_{33} \end{array} \right)$$

endwhile

Figure 17.4: Algorithm for computing the the LDL^T factorization of a tridiagonal matrix.

Algorithm: $A := \text{UDUT}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline * & A_{BR} \end{array} \right)$

where A_{BR} is 0×0

while $m(A_{BR}) < m(A)$ **do**

Repartition

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

where α_{11} is 1×1

Continue with

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

endwhile

Figure 17.5: Unblocked algorithm for computing $A = UDU^T$. Updates to A_{00} affect only the upper triangular part.

Homework 17.3 Derive an algorithm that, given an indefinite matrix A , computes $A = UDU^T$. Overwrite only the upper triangular part of A . (Fill in Figure 17.5.) Show how you came up with the algorithm, similar to how we derived the algorithm for LDL^T .

 SEE ANSWER

17.5 The UDU^T Factorization

Homework 17.4 Derive an algorithm that, given an indefinite tridiagonal matrix A , computes $A = UDU^T$. Overwrite only the upper triangular part of A . (Fill in Figure 17.6.) Show how you came up with the algorithm, similar to how we derived the algorithm for LDL^T .

 SEE ANSWER

Notice that the UDU^T factorization has the exact same shortcomings as does LDL^T when applied to a singular matrix. If D has a small element on the diagonal, it is again “down stream” that this can cause large elements in the factored matrix. But now “down stream” means towards the top-left since the algorithm moves in the opposite direction.

17.6 The Twisted Factorization

Let us assume that A is a tridiagonal matrix and that we are *given* one of its eigenvalues (or rather, a very good approximation), λ , and let us assume that this eigenvalue has multiplicity one. Indeed, we are going to assume that it is well-separated meaning there is no other eigenvalue close to it.

Here is a way to compute an associated eigenvector: Find a nonzero vector in the null space of $B = A - \lambda I$. Let us think back how one was taught how to do this:

- Reduce B to row-echelon form. This may will require pivoting.
- Find a column that has no pivot in it. This identifies an independent (free) variable.
- Set the element in vector x corresponding to the independent variable to *one*.
- Solve for the dependent variables.

There are a number of problems with this approach:

- It does not take advantage of symmetry.
- It is inherently unstable since you are working with a matrix, $B = A - \lambda I$ that inherently has a bad condition number. (Indeed, it is infinity if λ is an exact eigenvalue.)
- λ is not an exact eigenvalue when it is computed by a computer and hence B is not exactly singular. So, no independent variables will even be found.

These are only some of the problems. To overcome this, one computes something called a twisted factorization.

Again, let B be a tridiagonal matrix. Assume that λ is an approximate eigenvalue of B and let $A = B - \lambda I$. We are going to compute an approximate eigenvector of B associated with λ by computing a vector that is in the null space of a singular matrix that is close to A . We will assume that λ is an eigenvalue of

Algorithm: $A := \text{UDU}^T \text{.TRI}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c|c} A_{FF} & \alpha_{FME}e_L^T & 0 \\ \hline * & \alpha_{MM} & \alpha_{MLE}e_F^T \\ \hline * & * & A_{LL} \end{array} \right)$

where A_{FF} is 0×0

while $m(A_{LL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c|c} A_{FF} & \alpha_{FME}e_L & 0 \\ \hline * & \alpha_{MM} & \alpha_{MLE}e_F^T \\ \hline * & * & A_{LL} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c|c} A_{00} & \alpha_{01}e_L & 0 & 0 \\ \hline * & \alpha_{11} & \alpha_{12} & 0 \\ \hline * & * & \alpha_{22} & \alpha_{23}e_F^T \\ \hline * & * & * & A_{33} \end{array} \right)$$

where

Continue with

$$\left(\begin{array}{c|c|c} A_{FF} & \alpha_{FME}e_L & 0 \\ \hline * & \alpha_{MM} & \alpha_{MLE}e_F^T \\ \hline * & * & A_{LL} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c|c} A_{00} & \alpha_{01}e_L & 0 & 0 \\ \hline * & \alpha_{11} & \alpha_{12} & 0 \\ \hline * & * & \alpha_{22} & \alpha_{23}e_F^T \\ \hline * & * & * & A_{33} \end{array} \right)$$

endwhile

Figure 17.6: Algorithm for computing the the UDU^T factorization of a tridiagonal matrix.

B that has multiplicity one, and is well-separated from other eigenvalues. Thus, the singular matrix close to A has a null space with dimension one. Thus, we would expect $A = LDL^T$ to have one element on the diagonal of D that is essentially zero. Ditto for $A = UEU^T$, where E is diagonal and we use this letter to be able to distinguish the two diagonal matrices.

Thus, we have

- λ is an approximate (but not exact) eigenvalue of B .
- $A = B - \lambda I$ is indefinite.
- $A = LDL^T$. L is bidiagonal, unit lower triangular, and D is diagonal with one small element on the diagonal.
- $A = UEU^T$. U is bidiagonal, unit upper triangular, and E is diagonal with one small element on the diagonal.

Let

$$A = \begin{pmatrix} A_{00} & \alpha_{10}e_L & 0 \\ \alpha_{10}e_L^T & \alpha_{11} & \alpha_{21}e_F^T \\ 0 & \alpha_{21}e_F & A_{22} \end{pmatrix}, L = \begin{pmatrix} L_{00} & 0 & 0 \\ \lambda_{10}e_L^T & 1 & 0 \\ 0 & \lambda_{21}e_F & L_{22} \end{pmatrix}, U = \begin{pmatrix} U_{00} & v_{01}e_L & 0 \\ 0 & 1 & v_{21}e_F^T \\ 0 & 0 & U_{22} \end{pmatrix},$$

$$D = \begin{pmatrix} D_{00} & 0 & 0 \\ 0 & \delta_1 & 0 \\ 0 & 0 & D_{22} \end{pmatrix}, \text{ and } E = \begin{pmatrix} E_{00} & 0 & 0 \\ 0 & \varepsilon_1 & 0 \\ 0 & 0 & E_{22} \end{pmatrix},$$

where all the partitioning is “conformal”.

Homework 17.5 Show that, provided ϕ_1 is chosen appropriately,

$$\begin{pmatrix} L_{00} & 0 & 0 \\ \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ 0 & 0 & U_{22} \end{pmatrix} \begin{pmatrix} D_{00} & 0 & 0 \\ 0 & \phi_1 & 0 \\ 0 & 0 & E_{22} \end{pmatrix} \begin{pmatrix} L_{00} & 0 & 0 \\ \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ 0 & 0 & U_{22} \end{pmatrix}^T$$

$$= \begin{pmatrix} A_{00} & \alpha_{01}e_L & 0 \\ \alpha_{01}e_L^T & \alpha_{11} & \alpha_{21}e_F^T \\ 0 & \alpha_{21}e_F & A_{22} \end{pmatrix}.$$

(Hint: multiply out $A = LDL^T$ and $A = UEU^T$ with the partitioned matrices first. Then multiply out the above. Compare and match...) How should ϕ_1 be chosen? What is the cost of computing the twisted factorization given that you have already computed the LDL^T and UDU^T factorizations? A “Big O” estimate is sufficient. Be sure to take into account what $e_L^T D_{00} e_L$ and $e_F^T E_{22} e_F$ equal in your cost estimate.

SEE ANSWER

17.7 Computing an Eigenvector from the Twisted Factorization

The way the method now works for computing the desired approximate eigenvector is to find the ϕ_1 that is smallest in value of all possible such values. In other words, you can partition A , L , U , D , and E in many ways, singling out any of the diagonal values of these matrices. The partitioning chosen is the one that makes ϕ_1 the smallest of all possibilities. It is then set to zero so that

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right)^T \approx \left(\begin{array}{c|c|c} A_{00} & \alpha_{01}e_L & 0 \\ \hline \alpha_{01}e_L^T & \alpha_{11} & \alpha_{21}e_F^T \\ \hline 0 & \alpha_{21}e_F & A_{22} \end{array} \right).$$

Because λ was assumed to have multiplicity one and well-separated, the resulting twisted factorization has “nice” properties. We won’t get into what that exactly means.

Homework 17.6 Compute x_0 , χ_1 , and x_2 so that

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \underbrace{\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right)^T \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{array} \right)}_{\text{Hint: } \left(\begin{array}{c} 0 \\ \hline 1 \\ \hline 0 \end{array} \right)} = \left(\begin{array}{c} 0 \\ \hline 0 \\ \hline 0 \end{array} \right),$$

where $x = \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{array} \right)$ is not a zero vector. What is the cost of this computation, given that L_{00} and U_{22} have special structure?

[SEE ANSWER](#)

The vector x that is so computed is the desired approximate eigenvector of B .

Now, if the eigenvalues of B are well separated, and one follows essentially this procedure to find eigenvectors associated with each eigenvalue, the resulting eigenvectors are quite orthogonal to each other. We know that the eigenvectors of a symmetric matrix with distinct eigenvalues should be orthogonal, so this is a desirable property.

The approach becomes very tricky when eigenvalues are “clustered”, meaning that some of them are very close to each other. A careful scheme that shifts the matrix is then used to make eigenvalues in a cluster relatively well separated. But that goes beyond this note.

Chapter 18

Notes on Computing the SVD of a Matrix

For simplicity we will focus on the case where $A \in \mathbb{R}^{n \times n}$. We will assume that the reader has read Chapter 16.

18.0.1 Outline

18.0.1	Outline	346
18.1	Background	346
18.2	Reduction to Bidiagonal Form	346
18.3	The QR Algorithm with a Bidiagonal Matrix	350
18.4	Putting it all together	352

18.1 Background

Recall:

Theorem 18.1 *If $A \in \mathbb{R}^{m \times m}$ then there exists unitary matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$, and diagonal matrix $\Sigma \in \mathbb{R}^{m \times n}$ such that $A = U\Sigma V^T$. This is known as the Singular Value Decomposition.*

There is a relation between the SVD of a matrix A and the Spectral Decomposition of $A^T A$:

Homework 18.2 *If $A = U\Sigma V^T$ is the SVD of A then $A^T A = V\Sigma^2 V^T$ is the Spectral Decomposition of $A^T A$.*

☞ SEE ANSWER

The above exercise suggests steps for computing the Reduced SVD:

- Form $C = A^T A$.
- Compute unitary V and diagonal Λ such that $C = Q\Lambda Q^T$, ordering the eigenvalues from largest to smallest on the diagonal of Λ .
- Form $W = AV$. Then $W = U\Sigma$ so that U and Σ can be computed from W by choosing the diagonal elements of Σ to equal the lengths of the corresponding columns of W and normalizing those columns by those lengths.

The problem with this approach is that forming $A^T A$ squares the condition number of the problem. We will show how to avoid this.

18.2 Reduction to Bidiagonal Form

In the last chapter, we saw that it is beneficial to start by reducing a matrix to tridiagonal or upperHessenberg form when computing the Spectral or Schur decompositions. The corresponding step when computing the SVD of a given matrix is to reduce the matrix to bidiagonal form.

We assume that the bidiagonal matrix overwrites matrix A . Householder vectors will again play a role, and will again overwrite elements that are annihilated (set to zero).

Algorithm: $[A, t, r] := \text{BiDRED_UNB}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ t_B \end{array} \right), r \rightarrow \left(\begin{array}{c} r_T \\ r_B \end{array} \right)$

where A_{TL} is 0×0 and t_B has 0 elements

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

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} r_T \\ r_B \end{array} \right) \rightarrow \left(\begin{array}{c} r_0 \\ \rho_1 \\ r_2 \end{array} \right)$$

where α_{11} , τ_1 , and ρ_1 are scalars

$$[\left(\begin{array}{c} 1 \\ u_{21} \end{array} \right), \tau_1, \left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right)] := \text{HouseV}(\left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right))$$

$$a_{12}^T := a_{12}^T - w_{12}^T$$

$$z_{21} := (y_{21} - \beta u_{21}/\tau_1)/\tau_1$$

$$A_{22} := A_{22} - u_{21}w_{21}^T \quad (\text{rank-1 update})$$

$$[v_{12}, \rho_1, a_{12}] := \text{HouseV}(a_{12})$$

$$w_{21} := A_{22}v_{12}/\rho_1$$

$$A_{22} := A_{22} - w_{21}v_{12}^T \quad (\text{rank-1 update})$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} t_T \\ t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right), \left(\begin{array}{c} r_T \\ r_B \end{array} \right) \leftarrow \left(\begin{array}{c} r_0 \\ \rho_1 \\ r_2 \end{array} \right)$$

endwhile

Figure 18.1: Basic algorithm for reduction of a nonsymmetric matrix to bidiagonal form.

$\begin{array}{ cccc} \hline & \times & \times & \times & \times \\ \hline & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ \hline \end{array}$ <p>Original matrix</p>	$\begin{array}{ ccccc} \hline & \times & \times & 0 & 0 \\ \hline & 0 & \times & \times & \times \\ & 0 & \times & \times & \times \\ & 0 & \times & \times & \times \\ & 0 & \times & \times & \times \\ \hline \end{array}$ <p>First iteration</p>	$\begin{array}{ ccccc} \hline & \times & \times & 0 & 0 \\ \hline & 0 & \times & \times & 0 \\ & 0 & 0 & \times & \times \\ & 0 & 0 & \times & \times \\ & 0 & 0 & \times & \times \\ \hline \end{array}$ <p>Second iteration</p>
$\begin{array}{ ccccc} \hline & \times & \times & 0 & 0 \\ \hline & 0 & \times & \times & 0 \\ \hline & 0 & 0 & \times & \times \\ \hline & 0 & 0 & 0 & \times \\ & 0 & 0 & 0 & \times \\ \hline \end{array}$ <p>Third iteration</p>	$\begin{array}{ ccccc} \hline & \times & \times & 0 & 0 \\ \hline & 0 & \times & \times & 0 \\ \hline & 0 & 0 & \times & \times \\ \hline & 0 & 0 & 0 & \times \\ \hline & 0 & 0 & 0 & 0 \\ \hline \end{array}$ <p>Fourth iteration</p>	

Figure 18.2: Illustration of reduction to bidiagonal form.

- Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right)$. In the first iteration, this can be visualized as

$\begin{array}{ cccc} \hline & \times & \times & \times \\ \hline & \times & \times & \times \\ \hline \end{array}$

- We introduce zeroes below the “diagonal” in the first column by computing a Householder transformation and applying this from the left:

- Let $\left[\begin{array}{c} 1 \\ u_{21} \end{array} \right], \tau_1, \left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right) := \text{HouseV}(\left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array} \right))$. This not only computes the Householder vector, but also zeroes the entries in a_{21} and updates α_{11} .
- Update

$$\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) := (I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right) \left(\begin{array}{c} 1 \\ u_{21} \end{array} \right)^T) \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c} \tilde{\alpha}_{11} & \tilde{a}_{12}^T \\ \hline 0 & \tilde{A}_{22} \end{array} \right)$$

where

- * $\tilde{\alpha}_{11}$ is updated as part of the computation of the Householder vector;
- * $w_{12}^T := (a_{12}^T + u_{21}^T A_{22})/\tau_1$;
- * $\tilde{a}_{12}^T := a_{12}^T - w_{12}^T$; and
- * $\tilde{A}_{22} := A_{22} - u_{21} w_{12}^T$.

This introduces zeroes below the "diagonal" in the first column:

$$\begin{array}{c|cccc} \times & \times & \times & \times \\ \hline \times & \times & \times & \times \\ \end{array} \xrightarrow{\quad} \begin{array}{c|cccc} \times & \times & \times & \times \\ \hline 0 & \times & \times & \times \\ \end{array}$$

The zeroes below α_{11} are not actually written. Instead, the implementation overwrites these elements with the corresponding elements of the vector u_{21}^T .

- Next, we introduce zeroes in the first row to the right of the element on the superdiagonal by computing a Householder transformation from \tilde{a}_{12}^T :
 - Let $[v_{12}, \rho_1, a_{12}] := \text{HouseV}(a_{12})$. This not only computes the Householder vector, but also zeroes all but the first entry in \tilde{a}_{12}^T , updating that first entry.
 - Update

$$\begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} := \begin{pmatrix} a_{12}^T \\ A_{22} \end{pmatrix} \left(I - \frac{1}{\rho_1} v_{12} v_{12}^T \right) = \begin{pmatrix} \tilde{\alpha}_{12} e_0^T \\ \tilde{A}_{22} \end{pmatrix}$$

where

- * $\tilde{\alpha}_{12}$ equals the first element of the updated a_{12}^T ;
- * $w_{21} := A_{22} v_{12} / \rho_1$;
- * $\tilde{A}_{22} := A_{22} - w_{21} v_{12}^T$.

This introduces zeroes to the right of the "superdiagonal" in the first row:

$$\begin{array}{c|ccc} \times & \times & \times & \times \\ \hline 0 & \times & \times & \times \\ \end{array} \xrightarrow{\quad} \begin{array}{c|ccc} \times & \times & 0 & 0 \\ \hline 0 & \times & \times & \times \\ \end{array}$$

- The zeros to the right of the first element of a_{12}^T are not actually written. Instead, the implementation overwrites these elements with the corresponding elements of the vector v_{12}^T .

- Continue this process with the updated A_{22} .

The above observations are captured in the algorithm in Figure 18.1. It is also illustrated in Figure 18.2.

Homework 18.3 *Homework 18.4* Give the approximate total cost for reducing $A \in \mathbb{R}^{n \times n}$ to bidiagonal form.

SEE ANSWER

For a detailed discussion on the blocked algorithm that uses FLAME notation, we recommend [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. 39, No. 1, 2012

18.3 The QR Algorithm with a Bidiagonal Matrix

Let $B = U_B^T A V_B$ be bidiagonal where U_B and V_B have orthonormal columns.

Lemma 18.5 Let $B \in \mathbb{R}^{m \times n}$ be upper bidiagonal and $T \in \mathbb{R}^{n \times n}$ with $T = B^T B$. Then T is tridiagonal.

Proof: Partition

$$B = \left(\begin{array}{c|c} B_{00} & \beta_{10} e_l \\ \hline 0 & \beta_{11} \end{array} \right),$$

where e_l denotes the unit basis vector with a “1” in the last entry. Then

$$\begin{aligned} B^T B &= \left(\begin{array}{c|c} B_{00} & \beta_{10} e_l \\ \hline 0 & \beta_{11} \end{array} \right)^T \left(\begin{array}{c|c} B_{00} & \beta_{10} e_l \\ \hline 0 & \beta_{11} \end{array} \right) = \left(\begin{array}{c|c} B_{00}^T & 0 \\ \hline \beta_{10} e_l^T & \beta_{11} \end{array} \right) \left(\begin{array}{c|c} B_{00} & \beta_{10} e_l \\ \hline 0 & \beta_{11} \end{array} \right) \\ &= \left(\begin{array}{c|c} B_{00}^T B_{00} & \beta_{10} B_{00}^T e_l \\ \hline \beta_{10} e_l^T B & \beta_{10}^2 + \beta_{11}^2 \end{array} \right), \end{aligned}$$

where $\beta_{10} B_{00}^T e_l$ is (clearly) a vector with only a nonzero in the last entry.

The above proof also shows that if B has no zeroes on its superdiagonal, then neither does $B^T B$.

This means that one can apply the QR algorithm to matrix $B^T B$. The problem, again, is that this squares the condition number of the problem.

The following extension of the Implicit Q Theorem comes to the rescue:

Theorem 18.6 Let $C, B \in \mathbb{R}^{m \times n}$. If there exist unitary matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ so that $B = U^T C V$ is upper bidiagonal and has positive values on its superdiagonal then B and V are uniquely determined by C and the first column of V .

The above theorem supports an algorithm that, starting with an upper bidiagonal matrix B , implicitly performs a shifted QR algorithm with $T = B^T B$.

Consider the 5×5 bidiagonal matrix

$$B = \left(\begin{array}{ccccc} \beta_{0,0} & \beta_{0,1} & 0 & 0 & 0 \\ 0 & \beta_{1,1} & \beta_{1,2} & 0 & 0 \\ 0 & 0 & \beta_{2,2} & \beta_{2,3} & 0 \\ 0 & 0 & 0 & \beta_{3,3} & \beta_{3,4} \\ 0 & 0 & 0 & 0 & \beta_{4,4} \end{array} \right).$$

Then $T = B^T B$ equals

$$T = \begin{pmatrix} \beta_{0,0}^2 & * & 0 & 0 & 0 \\ \beta_{0,1}\beta_{0,0} & * & * & 0 & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & \beta_{3,4}^2 + \beta_{4,4}^2 \end{pmatrix},$$

where the $*$ s indicate “don’t care” entries. Now, if an iteration of the implicitly shifted QR algorithm were executed with this matrix, then the shift would be $\mu_k = \beta_{3,4}^2 + \beta_{4,4}^2$ (actually, it is usually computed from the bottom-right 2×2 matrix, a minor detail) and the first Givens’ rotation would be computed so that

$$\begin{pmatrix} \gamma_{0,1} & \sigma_{0,1} \\ -\sigma_{0,1} & \gamma_{0,1} \end{pmatrix} \begin{pmatrix} \beta_{0,0}^2 - \mu I \\ \beta_{0,1}\beta_{1,1} \end{pmatrix}$$

has a zero second entry. Let us call the $n \times n$ matrix with this as its top-left submatrix G_0 . Then applying this from the left and right of T means forming $G_0 T G_0 = G_0 B^T B G_0^T = (B G_0^T)^T (B G_0^T)$. What if we only apply it to B ? Then

$$\left(\begin{array}{|c|c|c|c|c|} \hline \tilde{\beta}_{0,0} & \tilde{\beta}_{0,1} & 0 & 0 & 0 \\ \hline \tilde{\beta}_{1,0} & \tilde{\beta}_{1,1} & \beta_{1,2} & 0 & 0 \\ \hline 0 & 0 & \beta_{2,2} & \beta_{2,3} & 0 \\ \hline 0 & 0 & 0 & \beta_{3,3} & \beta_{3,4} \\ \hline 0 & 0 & 0 & 0 & \beta_{4,4} \\ \hline \end{array} \right) = \left(\begin{array}{|c|c|c|c|c|} \hline \beta_{0,0} & \beta_{0,1} & 0 & 0 & 0 \\ \hline 0 & \beta_{1,1} & \beta_{1,2} & 0 & 0 \\ \hline 0 & 0 & \beta_{2,2} & \beta_{2,3} & 0 \\ \hline 0 & 0 & 0 & \beta_{3,3} & \beta_{3,4} \\ \hline 0 & 0 & 0 & 0 & \beta_{4,4} \\ \hline \end{array} \right) \left(\begin{array}{|c|c|c|c|c|} \hline \gamma_{0,1} & -\sigma_{0,1} & 0 & 0 & 0 \\ \hline \sigma_{0,1} & \gamma_{0,1} & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \\ \hline \end{array} \right).$$

The idea now is to apply Givens’ rotation from the left and right to “chase the bulge” except that now the rotations applied from both sides need not be the same. Thus, next, from $\begin{pmatrix} \tilde{\beta}_{0,0} \\ \tilde{\beta}_{1,0} \end{pmatrix}$ one can compute $\gamma_{1,0}$

and $\sigma_{1,0}$ so that $\begin{pmatrix} \gamma_{1,0} & \sigma_{1,0} \\ -\sigma_{1,0} & \gamma_{1,0} \end{pmatrix} \begin{pmatrix} \tilde{\beta}_{0,0} \\ \tilde{\beta}_{1,0} \end{pmatrix} = \begin{pmatrix} \tilde{\beta}_{0,0} \\ 0 \end{pmatrix}$. Then

$$\left(\begin{array}{|c|c|c|c|c|} \hline \tilde{\beta}_{0,0} & \tilde{\beta}_{0,1} & \tilde{\beta}_{0,2} & 0 & 0 \\ \hline 0 & \tilde{\beta}_{1,1} & \tilde{\beta}_{1,2} & 0 & 0 \\ \hline 0 & 0 & \beta_{2,2} & \beta_{2,3} & 0 \\ \hline 0 & 0 & 0 & \beta_{3,3} & \beta_{3,4} \\ \hline 0 & 0 & 0 & 0 & \beta_{4,4} \\ \hline \end{array} \right) = \left(\begin{array}{|c|c|c|c|c|} \hline \gamma_{1,0} & \sigma_{1,0} & 0 & 0 & 0 \\ \hline -\sigma_{1,0} & \gamma_{1,0} & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \\ \hline \end{array} \right) \left(\begin{array}{|c|c|c|c|c|} \hline \tilde{\beta}_{0,0} & \tilde{\beta}_{0,1} & 0 & 0 & 0 \\ \hline \tilde{\beta}_{1,0} & \tilde{\beta}_{1,1} & \beta_{1,2} & 0 & 0 \\ \hline 0 & 0 & \beta_{2,2} & \beta_{2,3} & 0 \\ \hline 0 & 0 & 0 & \beta_{3,3} & \beta_{3,4} \\ \hline 0 & 0 & 0 & 0 & \beta_{4,4} \\ \hline \end{array} \right).$$

Continuing, from $\begin{pmatrix} \tilde{\beta}_{0,1} \\ \tilde{\beta}_{0,2} \end{pmatrix}$ one can compute $\gamma_{0,2}$ and $\sigma_{0,2}$ so that $\begin{pmatrix} \gamma_{0,2} & \sigma_{0,2} \\ -\sigma_{0,2} & \gamma_{0,2} \end{pmatrix} \begin{pmatrix} \tilde{\beta}_{0,1} \\ \tilde{\beta}_{0,2} \end{pmatrix} =$

$\begin{pmatrix} \tilde{\beta}_{0,2} \\ 0 \end{pmatrix}$. Then

$$\left(\begin{array}{c|cc|cc} \tilde{\beta}_{0,0} & \tilde{\beta}_{0,1} & 0 & 0 & 0 \\ \hline 0 & \tilde{\beta}_{1,1} & \tilde{\beta}_{1,2} & 0 & 0 \\ \hline 0 & \tilde{\beta}_{2,1} & \tilde{\beta}_{2,2} & \beta_{2,3} & 0 \\ \hline 0 & 0 & 0 & \beta_{3,3} & \beta_{3,4} \\ \hline 0 & 0 & 0 & 0 & \beta_{4,4} \end{array} \right) = \left(\begin{array}{c|cc|cc} \tilde{\beta}_{0,0} & \tilde{\beta}_{0,1} & \tilde{\beta}_{02} & 0 & 0 \\ \hline 0 & \tilde{\beta}_{1,1} & \tilde{\beta}_{1,2} & 0 & 0 \\ \hline 0 & 0 & \beta_{2,2} & \beta_{2,3} & 0 \\ \hline 0 & 0 & 0 & \beta_{3,3} & \beta_{3,4} \\ \hline 0 & 0 & 0 & 0 & \beta_{4,4} \end{array} \right) \left(\begin{array}{c|cc|cc} 1 & 0 & 0 & 0 & 0 \\ \hline 0 & \gamma_{1,0} & -\sigma_{1,0} & 0 & 0 \\ \hline 0 & \sigma_{1,0} & \gamma_{1,0} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \end{array} \right)$$

And so forth, as illustrated in Figure 18.3.

18.4 Putting it all together

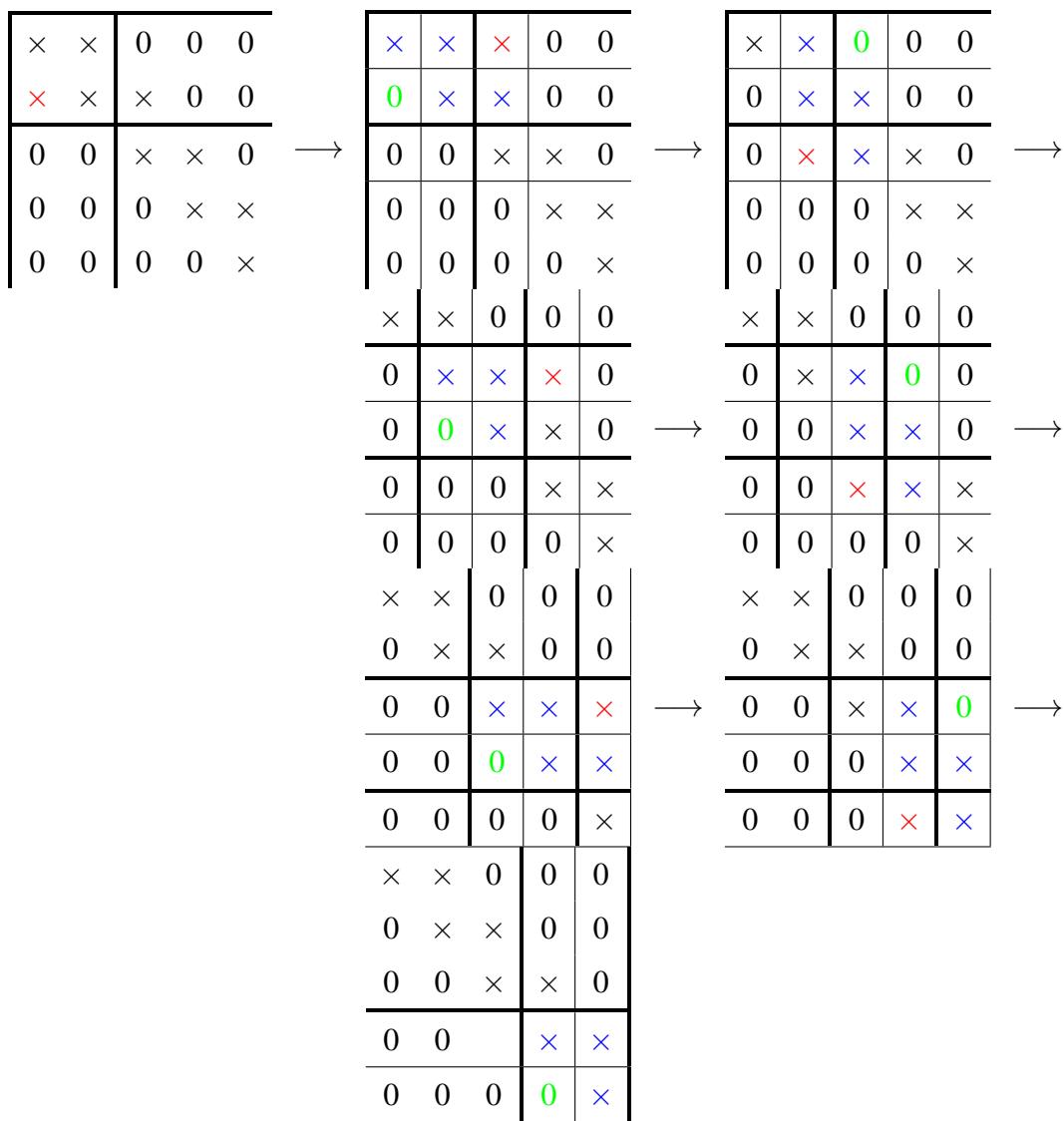


Figure 18.3: Illustration of one sweep of an implicit QR algorithm with a bidiagonal matrix.

Algorithm: $B := \text{CHASEBULGEBiD}(B)$

$$\text{Partition } B \rightarrow \left(\begin{array}{c|c|c} B_{TL} & B_{TM} & \star \\ \hline 0 & B_{MM} & B_{MR} \\ \hline 0 & 0 & B_{BR} \end{array} \right)$$

where B_{TL} is 0×0 and B_{MM} is 2×2

while $m(B_{BR}) \geq 0$ **do**

Repartition

$$\left(\begin{array}{c|c|c} B_{TL} & B_{TM} & \star \\ \hline 0 & B_{MM} & B_{MR} \\ \hline 0 & 0 & B_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c|c|c} B_{00} & \beta_{01}e_l & 0 & 0 & 0 \\ \hline 0 & \beta_{11} & \beta_{12} & 0 & 0 \\ \hline 0 & \beta_{21} & \beta_{22} & \beta_{23} & 0 \\ \hline 0 & 0 & 0 & \beta_{33} & \beta_{34}e_0^T \\ \hline 0 & 0 & 0 & 0 & B_{44} \end{array} \right)$$

where τ_{11} and τ_{33} are scalars
(during final step, τ_{33} is 0×0)

$$\text{Compute } (\gamma_1^L, \sigma_1^L) \text{ s.t. } \left(\begin{array}{c|c} \gamma_1^L & \sigma_1^L \\ \hline -\sigma_1^L & \gamma_1^L \end{array} \right) \left(\begin{array}{c} \beta_{1,1} \\ \hline \beta_{2,1} \end{array} \right) = \left(\begin{array}{c} \tilde{\beta}_{1,1} \\ \hline 0 \end{array} \right)$$

overwriting $\beta_{1,1}$ with $\tilde{\beta}_{1,1}$

$$\left(\begin{array}{c|c} \beta_{1,2} & \beta_{1,3} \\ \hline \beta_{2,2} & \beta_{2,3} \end{array} \right) := \left(\begin{array}{c|c} \gamma_1^L & \sigma_1^L \\ \hline -\sigma_1^L & \gamma_1^L \end{array} \right) \left(\begin{array}{c|c} \beta_{1,2} & 0 \\ \hline \beta_{2,2} & \beta_{2,3} \end{array} \right)$$

if $m(B_{BR}) \neq 0$

$$\text{Compute } (\gamma_1^R, \sigma_1^R) \text{ s.t. } \left(\begin{array}{c|c} \gamma_1^R & \sigma_1^R \\ \hline -\sigma_1^R & \gamma_1^R \end{array} \right) \left(\begin{array}{c} \beta_{1,2} \\ \hline \beta_{1,3} \end{array} \right) = \left(\begin{array}{c} \tilde{\beta}_{1,2} \\ \hline 0 \end{array} \right)$$

overwriting $\beta_{1,2}$ with $\tilde{\beta}_{1,2}$

$$\left(\begin{array}{c|c} \beta_{1,2} & 0 \\ \hline \beta_{2,2} & \beta_{2,3} \\ \hline \beta_{3,2} & \beta_{3,3} \end{array} \right) := \left(\begin{array}{c|c} \beta_{1,2} & \beta_{1,3} \\ \hline \beta_{2,2} & \beta_{2,3} \\ \hline 0 & \beta_{3,3} \end{array} \right) \left(\begin{array}{c|c} \gamma_1^R & -\sigma_1^R \\ \hline \sigma_1^R & \gamma_1^R \end{array} \right)$$

Continue with

$$\left(\begin{array}{c|c|c} B_{TL} & B_{TM} & \star \\ \hline 0 & B_{MM} & B_{MR} \\ \hline 0 & 0 & B_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c|c|c} B_{00} & \beta_{01}e_l & 0 & 0 & 0 \\ \hline 0 & \beta_{11} & \beta_{12} & 0 & 0 \\ \hline 0 & \beta_{21} & \beta_{22} & \beta_{23} & 0 \\ \hline 0 & 0 & 0 & \beta_{33} & \beta_{34}e_0^T \\ \hline 0 & 0 & 0 & 0 & B_{44} \end{array} \right)$$

endwhile

Figure 18.4: Chasing the bulge.

More Chapters to be Added in the Future!

Answers

Chapter 1. Notes on Simple Vector and Matrix Operations (Answers)

Homework 1.1 Partition A

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

Convince yourself that the following hold:

$$\bullet \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)^T = \begin{pmatrix} \overline{a_0^T} \\ \overline{a_1^T} \\ \vdots \\ \overline{a_{m-1}^T} \end{pmatrix}.$$

$$\bullet \begin{pmatrix} \overline{\hat{a}_0^T} \\ \overline{\hat{a}_1^T} \\ \vdots \\ \overline{\hat{a}_{m-1}^T} \end{pmatrix}^T = \left(\begin{array}{c|c|c|c} \hat{a}_0 & \hat{a}_1 & \cdots & \hat{a}_{n-1} \end{array} \right).$$

$$\bullet \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)^H = \begin{pmatrix} \overline{a_0^H} \\ \overline{a_1^H} \\ \vdots \\ \overline{a_{m-1}^H} \end{pmatrix}.$$

- $\left(\begin{array}{c} \widehat{a}_0^T \\ \hline \widehat{a}_1^T \\ \vdots \\ \hline \widehat{a}_{m-1}^T \end{array} \right)^H = \left(\begin{array}{c|c|c|c} \overline{\widehat{a}_0} & \overline{\widehat{a}_1} & \cdots & \overline{\widehat{a}_{n-1}} \end{array} \right).$

[◀ BACK TO TEXT](#)

Homework 1.2 Partition x into subvectors:

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

Convince yourself that the following hold:

- $\bar{x} = \left(\begin{array}{c} \overline{x_0} \\ \hline \overline{x_1} \\ \vdots \\ \hline \overline{x_{N-1}} \end{array} \right).$
- $x^T = \left(\begin{array}{c|c|c|c} x_0^T & x_1^T & \cdots & x_{N-1}^T \end{array} \right).$
- $x^H = \left(\begin{array}{c|c|c|c} x_0^H & x_1^H & \cdots & x_{N-1}^H \end{array} \right).$

[◀ BACK TO TEXT](#)

Homework 1.3 Partition A

$$A = \begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{pmatrix},$$

where $A_{i,j} \in \mathbb{C}^{m_i \times n_j}$. Here $\sum_{i=0}^{M-1} m_i = m$ and $\sum_{j=0}^{N-1} n_i = n$.

Convince yourself that the following hold:

- $$\left(\begin{array}{cccc} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{array} \right)^T = \left(\begin{array}{cccc} A_{0,0}^T & A_{1,0}^T & \cdots & A_{M-1,0}^T \\ A_{0,1}^T & A_{1,1}^T & \cdots & A_{M-1,1}^T \\ \vdots & \vdots & \cdots & \vdots \\ A_{0,N-1}^T & A_{1,N-1}^T & \cdots & A_{M-1,N-1}^T \end{array} \right).$$
- $$\overline{\left(\begin{array}{cccc} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{array} \right)} = \left(\begin{array}{cccc} \overline{A_{0,0}} & \overline{A_{0,1}} & \cdots & \overline{A_{0,N-1}} \\ \overline{A_{1,0}} & \overline{A_{1,1}} & \cdots & \overline{A_{1,N-1}} \\ \vdots & \vdots & \cdots & \vdots \\ \overline{A_{M-1,0}} & \overline{A_{M-1,1}} & \cdots & \overline{A_{M-1,N-1}} \end{array} \right).$$
- $$\left(\begin{array}{cccc} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{array} \right)^H = \left(\begin{array}{cccc} A_{0,0}^H & A_{1,0}^H & \cdots & A_{M-1,0}^H \\ A_{0,1}^H & A_{1,1}^H & \cdots & A_{M-1,1}^H \\ \vdots & \vdots & \cdots & \vdots \\ A_{0,N-1}^H & A_{1,N-1}^H & \cdots & A_{M-1,N-1}^H \end{array} \right).$$

[◀ BACK TO TEXT](#)

Homework 1.4 Convince yourself of the following:

- $\alpha x^T = (\alpha x_0 \mid \alpha x_1 \mid \cdots \mid \alpha x_{n-1})$.
- $(\alpha x)^T = \alpha x^T$.
- $(\alpha x)^H = \bar{\alpha} x^H$.

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

[◀ BACK TO TEXT](#)

Homework 1.5 Convince yourself of the following:

- $\alpha \begin{pmatrix} \frac{x_0}{x_1} \\ \vdots \\ \frac{x_{N-1}}{} \end{pmatrix} + \begin{pmatrix} \frac{y_0}{y_1} \\ \vdots \\ \frac{y_{N-1}}{} \end{pmatrix} = \begin{pmatrix} \frac{\alpha x_0 + y_0}{\alpha x_1 + y_1} \\ \vdots \\ \frac{\alpha x_{N-1} + y_{N-1}}{} \end{pmatrix}. \text{ (Provided } x_i, y_i \in \mathbb{C}^{n_i} \text{ and } \sum_{i=0}^{N-1} n_i = n\text{.)}$

[BACK TO TEXT](#)

Homework 1.6 Convince yourself of the following:

$$\bullet \quad \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix}^H \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{pmatrix} = \sum_{i=0}^{N-1} x_i^H y_i. \text{ (Provided } x_i, y_i \in \mathbb{C}^{n_i} \text{ and } \sum_{i=0}^{N-1} n_i = n.)$$

[BACK TO TEXT](#)

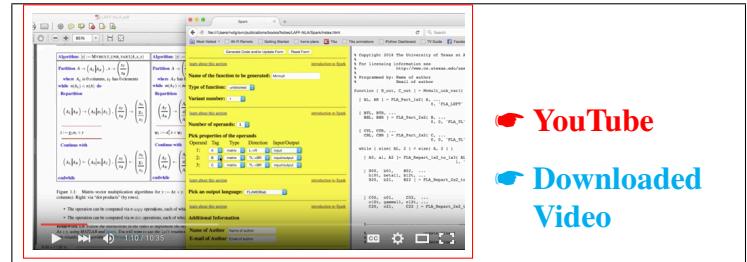
Homework 1.7 Prove that $x^H y = \overline{y^H x}$.

Answer:

$$x^H y = \sum_{i=0}^{n-1} \bar{\chi}_i \psi_i = \sum_{i=0}^{n-1} \bar{\chi}_i \bar{\Psi}_i = \sum_{i=0}^{n-1} \bar{\Psi}_i \bar{\chi}_i = \overline{\sum_{i=0}^{n-1} \bar{\Psi}_i \bar{\chi}_i} = \overline{y^H x}$$

[BACK TO TEXT](#)

Homework 1.8 Follow the instructions in the below video to implement the two algorithms for computing $y := Ax + b$, using MATLAB and **Spark**. You will want to use the `laff` routines summarized in Appendix B. You can visualize the algorithm with **PictureFLAME**.



[YouTube](#)

[Downloaded Video](#)

Answer: Implementations are given in Figure 1.5. They can also be found in

[Programming/chapter01_answers/Mvmult_unb_var1.m](#) (see file only) (view in MATLAB)

and

[Programming/chapter01_answers/Mvmult_unb_var2.m](#) (see file only) (view in MATLAB)

[BACK TO TEXT](#)

Homework 1.9 Implement the two algorithms for computing $A := yx^T + A$, using MATLAB and **Spark**. You will want to use the `laff` routines summarized in Appendix B. You can visualize the algorithm with **PictureFLAME**.

Answer: Implementations can be found in

```

function [ y_out ] = Mvmult_unb_var1( A, x, y )
    [ AL, AR ] = FLA_Part_1x2( A, ...
        0, 'FLA.LEFT' );
    [ xT, ...
    xB ] = FLA_Part_2x1( x, ...
        0, 'FLA.TOP' );
    while ( size( AL, 2 ) < size( A, 2 ) )
        [ A0, a1, A2 ]= FLA_Report_1x2_to_1x3( AL, AR, ...
            1, 'FLA.RIGHT' );
        [ x0, ...
        ch1, ...
        x2 ] = FLA_Report_2x1_to_3x1( xT, ...
            xB, ...
            1, 'FLA.BOTTOM' );
        %—————%
        % y = ch1 * a1 + y;
        y = laff_axpy( ch1, a1, y );
        %—————%
        [ AL, AR ] = FLA_Cont_with_1x3_to_1x2( A0, a1, A2, ...
            'FLA.LEFT' );
        [ xT, ...
        xB ] = FLA_Cont_with_3x1_to_2x1( x0, ...
            ch1, ...
            x2, ...
            'FLA.TOP' );
    end
    y_out = y;
return

function [ y_out ] = Mvmult_unb_var2( A, x, y )
    [ AT, ...
    AB ] = FLA_Part_2x1( A, ...
        0, 'FLA.TOP' );
    [ yT, ...
    yB ] = FLA_Part_2x1( y, ...
        0, 'FLA.TOP' );
    while ( size( AT, 1 ) < size( A, 1 ) )
        [ A0, ...
        alt, ...
        A2 ] = FLA_Report_2x1_to_3x1( AT, ...
            AB, ...
            1, 'FLA.BOTTOM' );
        [ y0, ...
        psi1, ...
        y2 ] = FLA_Report_2x1_to_3x1( yT, ...
            yB, ...
            1, 'FLA.BOTTOM' );
        %—————%
        % psi1 = alt * x + psi1;
        psi1 = laff_dots( alt, x, psi1 );
        %—————%
        [ AT, ...
        AB ] = FLA_Cont_with_3x1_to_2x1( A0, ...
            alt, ...
            A2, ...
            'FLA.TOP' );
        [ yT, ...
        yB ] = FLA_Cont_with_3x1_to_2x1( y0, ...
            psi1, ...
            y2, ...
            'FLA.TOP' );
    end
    y_out = [ yT
              yB ];
return

```

Figure 1.5: Implementations of the two unblocked variants for computing $y := Ax + y$.

[Programming/chapter01_answers/Rank1_unb_var1.m](#) (see file only) (view in MATLAB)

and

[Programming/chapter01_answers/Rank1_unb_var2.m](#) (see file only) (view in MATLAB)

◀ BACK TO TEXT

Homework 1.10 Prove that the matrix xy^T where x and y are vectors has rank at most one, thus explaining the name “rank-1 update”.

Answer: There are a number of ways of proving this:

The rank of a matrix equals the number of linearly independent columns it has. But each column of xy^T is a multiple of vector x and hence there is at most one linearly independent column.

The rank of a matrix equals the number of linearly independent rows it has. But each row of xy^T is a multiple of row vector y^T and hence there is at most one linearly independent row.

The rank of a matrix equals the dimension of the column space of the matrix. If z is in the column space of xy^T then there must be a vector w such that $z = (xy^T)w$. But then $z = y^Twx$ and hence is a multiple of x . This means the column space is at most one dimensional and hence the rank is at most one.

There are probably a half dozen other arguments... For now, do not use an argument that utilizes the SVD, since we have not yet learned about it.

◀ BACK TO TEXT

Homework 1.11 Implement $C := AB + C$ via matrix-vector multiplications, using MATLAB and [Spark](#). You will want to use the `laff` routines summarized in Appendix B. You can visualize the algorithm with [PictureFLAME](#).

Answer: An implementation can be found in

[Programming/chapter01_answers/Gemm_unb_var1.m](#) (see file only) (view in MATLAB)

◀ BACK TO TEXT

Homework 1.12 Argue that the given matrix-matrix multiplication algorithm with $m \times n$ matrix C , $m \times k$ matrix A , and $k \times n$ matrix B costs, approximately, $2mnk$ flops. **Answer:** This can be easily argued by noting that each update of a row of matrix C costs, approximately, $2nk$ flops. There are m such rows to be computed.

◀ BACK TO TEXT

Homework 1.13 Implement $C := AB + C$ via row vector-matrix multiplications, using MATLAB and **Spark**. You will want to use the `laff` routines summarized in Appendix B. You can visualize the algorithm with **PictureFLAME**. Hint: $y^T := x^T A + y^T$ is not supported by a `laff` routine. You can use `laff_gemv` instead.

Answer: An implementation can be found in

[Programming/chapter01_answers/Gemm_unb_var2.m](#) (see file only) (view in MATLAB)

 [BACK TO TEXT](#)

Homework 1.9 Argue that the given matrix-matrix multiplication algorithm with $m \times n$ matrix C , $m \times k$ matrix A , and $k \times n$ matrix B costs, approximately, $2mnk$ flops. **Answer:** This can be easily argued by noting that each rank-1 update of matrix C costs, approximately, $2mn$ flops. There are k such rank-1 updates to be computed.

 [BACK TO TEXT](#)

Homework 1.10 Implement $C := AB + C$ via rank-1 updates, using MATLAB and **Spark**. You will want to use the `laff` routines summarized in Appendix B. You can visualize the algorithm with **PictureFLAME**.

Answer: An implementation can be found in

[Programming/chapter01_answers/Gemm_unb_var3.m](#) (see file only) (view in MATLAB)

 [BACK TO TEXT](#)

Chapter 2. Notes on Vector and Matrix Norms (Answers)

Homework 2.2 Prove that if $v : \mathbb{C}^n \rightarrow \mathbb{R}$ is a norm, then $v(0) = 0$ (where the first 0 denotes the zero vector in \mathbb{C}^n).

Answer: Let $x \in \mathbb{C}^n$ and $\vec{0}$ the zero vector of size n and 0 the scalar zero. Then

$$\begin{aligned} v(\vec{0}) &= v(0 \cdot x) & 0 \cdot x = \vec{0} \\ &= |0|v(x) & v(\cdot) \text{ is homogeneous} \\ &= 0 & \text{algebra} \end{aligned}$$

 [BACK TO TEXT](#)

Homework 2.7 The vector 1-norm is a norm.

Answer: We show that the three conditions are met:

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

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

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

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

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

$$\begin{aligned} \|\alpha x\|_1 &= |\alpha \chi_0| + \dots + |\alpha \chi_{n-1}| = |\alpha| |\chi_0| + \dots + |\alpha| |\chi_{n-1}| = |\alpha| (|\chi_0| + \dots + |\chi_{n-1}|) = \\ &= |\alpha| (|\chi_0| + \dots + |\chi_{n-1}|) = |\alpha| \|x\|_1. \end{aligned}$$

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

$$\begin{aligned} \|x+y\|_1 &= |\chi_0 + \psi_0| + |\chi_1 + \psi_1| + \dots + |\chi_{n-1} + \psi_{n-1}| \\ &\leq |\chi_0| + |\psi_0| + |\chi_1| + |\psi_1| + \dots + |\chi_{n-1}| + |\psi_{n-1}| \\ &= |\chi_0| + |\chi_1| + \dots + |\chi_{n-1}| + |\psi_0| + |\psi_1| + \dots + |\psi_{n-1}| \\ &= \|x\|_1 + \|y\|_1. \end{aligned}$$

 [BACK TO TEXT](#)

Homework 2.9 The vector ∞ -norm is a norm.

Answer: We show that the three conditions are met:

Let $x, y \in \mathbb{C}^n$ 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

$$\|x\|_\infty = \max_i |\chi_i| \geq |\chi_j| > 0.$$

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

$$\|\alpha x\|_\infty = \max_i |\alpha \chi_i| = \max_i |\alpha| |\chi_i| = |\alpha| \max_i |\chi_i| = |\alpha| \|x\|_\infty.$$

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

$$\begin{aligned} \|x+y\|_\infty &= \max_i |\chi_i + \psi_i| \\ &\leq \max_i (|\chi_i| + |\psi_i|) \\ &\leq \max_i (|\chi_i| + \max_j |\psi_j|) \\ &= \max_i |\chi_i| + \max_j |\psi_j| = \|x\|_\infty + \|y\|_\infty. \end{aligned}$$

[BACK TO TEXT](#)

Homework 2.13 Show that the Frobenius norm is a norm.

Answer: The answer is to realize that if $A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$ then

$$\|A\|_F = \sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |\alpha_{i,j}|^2} = \sqrt{\sum_{j=0}^{n-1} \sum_{i=0}^{m-1} |\alpha_{i,j}|^2} = \sqrt{\sum_{j=0}^{n-1} \|a_j\|_2^2} = \sqrt{\left\| \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \right\|_2^2}.$$

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. The fact that the Frobenius norm is a norm then comes from realizing this connection and exploiting it.

Alternatively, just grind through the three conditions!

[BACK TO TEXT](#)

$$\begin{pmatrix} \widehat{a}_0^T \\ \widehat{a}_1^T \\ \vdots \\ \widehat{a}_{m-1}^T \end{pmatrix}$$

Homework 2.20 Let $A \in \mathbb{C}^{m \times n}$ and partition $A = \begin{pmatrix} \widehat{a}_0^T \\ \widehat{a}_1^T \\ \vdots \\ \widehat{a}_{m-1}^T \end{pmatrix}$. Show that

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

Answer: Partition $A = \begin{pmatrix} \widehat{a}_0^T \\ \widehat{a}_1^T \\ \vdots \\ \widehat{a}_{m-1}^T \end{pmatrix}$. Then

$$\begin{aligned}
\|A\|_\infty &= \max_{\|x\|_\infty=1} \|Ax\|_\infty = \max_{\|x\|_\infty=1} \left\| \begin{pmatrix} \widehat{a}_0^T \\ \widehat{a}_1^T \\ \vdots \\ \widehat{a}_{m-1}^T \end{pmatrix} x \right\|_\infty = \max_{\|x\|_\infty=1} \left\| \begin{pmatrix} \widehat{a}_0^T x \\ \widehat{a}_1^T x \\ \vdots \\ \widehat{a}_{m-1}^T x \end{pmatrix} \right\|_\infty \\
&= \max_{\|x\|_\infty=1} \left(\max_i |\alpha_i^T x| \right) = \max_{\|x\|_\infty=1} \max_i \left| \sum_{p=0}^{n-1} \alpha_{i,p} \chi_p \right| \leq \max_{\|x\|_\infty=1} \max_i \sum_{p=0}^{n-1} |\alpha_{i,p} \chi_p| \\
&= \max_{\|x\|_\infty=1} \max_i \sum_{p=0}^{n-1} (|\alpha_{i,p}| |\chi_p|) \leq \max_{\|x\|_\infty=1} \max_i \sum_{p=0}^{n-1} (|\alpha_{i,p}| (\max_k |\chi_k|)) \leq \max_{\|x\|_\infty=1} \max_i \sum_{p=0}^{n-1} (|\alpha_{i,p}| \|x\|_\infty) \\
&= \max_i \sum_{p=0}^{n-1} (|\alpha_{i,p}|) = \|\widehat{a}_i\|_1
\end{aligned}$$

so that $\|A\|_\infty \leq \max_i \|\widehat{a}_i\|_1$.

We also want to show that $\|A\|_\infty \geq \max_i \|\widehat{a}_i\|_1$. Let k be such that $\max_i \|\widehat{a}_i\|_1 = \|\widehat{a}_k\|_1$ and pick $y =$

$$\begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} \text{ so that } \widehat{a}_k^T y = |\alpha_{k,0}| + |\alpha_{k,1}| + \cdots + |\alpha_{k,n-1}| = \|\widehat{a}_k\|_1. \text{ (This is a matter of picking } \psi_i \text{ so that } |\psi_i| = 1 \text{ and } \psi_i \alpha_{k,i} = |\alpha_{k,i}|.) \text{ Then}$$

$$\begin{aligned}
\|A\|_\infty &= \max_{\|x\|_1=1} \|Ax\|_\infty = \max_{\|x\|_1=1} \left\| \begin{pmatrix} \widehat{a}_0^T \\ \widehat{a}_1^T \\ \vdots \\ \widehat{a}_{m-1}^T \end{pmatrix} x \right\|_\infty \geq \left\| \begin{pmatrix} \widehat{a}_0^T \\ \widehat{a}_1^T \\ \vdots \\ \widehat{a}_{m-1}^T \end{pmatrix} y \right\|_\infty \\
&= \left\| \begin{pmatrix} \widehat{a}_0^T y \\ \widehat{a}_1^T y \\ \vdots \\ \widehat{a}_{m-1}^T y \end{pmatrix} \right\|_\infty \geq |\widehat{a}_k^T y| = \|\widehat{a}_k\|_1 = \max_i \|\widehat{a}_i\|_1
\end{aligned}$$

◀ BACK TO TEXT

Homework 2.21 Let $y \in \mathbb{C}^m$ and $x \in \mathbb{C}^n$. Show that $\|yx^H\|_2 = \|y\|_2\|x\|_2$.

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

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

$$\|yx^H\|_2 = \max_{\|z\|_2=1} \|yx^H z\|_2 = \max_{\|z\|_2=1} |x^H z| \|y\|_2 \leq \max_{\|z\|_2=1} \|x\|_2\|z\|_2\|y\|_2 = \|x\|_2\|y\|_2.$$

But also

$$\|yx^H\|_2 = \max_{z \neq 0} \|yx^H z\|_2/\|z\|_2 \geq \|yx^H x\|_2/\|x\|_2 = \|y\|_2\|x\|_2.$$

Hence

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

◀ BACK TO TEXT

Homework 2.25 Show that $\|Ax\|_\mu \leq \|A\|_{\mu,v}\|x\|_v$.

Answer: W.l.o.g. let $x \neq 0$.

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

Rearranging this establishes the result.

◀ BACK TO TEXT

Homework 2.26 Show that $\|AB\|_\mu \leq \|A\|_{\mu,v}\|B\|_v$.

Answer:

$$\|AB\|_{\mu,v} = \max_{\|x\|_v=1} \|ABx\|_\mu \leq \max_{\|x\|_v=1} \|A\|_{\mu,v}\|Bx\|_v = \|A\|_{\mu,v} \max_{\|x\|_v=1} \|Bx\|_v = \|A\|_{\mu,v}\|B\|_v$$

◀ BACK TO TEXT

Homework 2.27 Show that the Frobenius norm, $\|\cdot\|_F$, is submultiplicative.

Answer:

$$\begin{aligned} \|AB\|_F^2 &= \left\| \begin{pmatrix} \hat{a}_0^H \\ \hat{a}_1^H \\ \vdots \\ \hat{a}_{m-1}^H \end{pmatrix} \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right) \right\|_F^2 = \left\| \begin{pmatrix} \hat{a}_0^H b_0 & \hat{a}_0^H b_1 & \cdots & \hat{a}_0^H b_{n-1} \\ \hline \hat{a}_0^H b_0 & \hat{a}_0^H b_1 & \cdots & \hat{a}_0^H b_{n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline \hat{a}_{m-1}^H b_0 & \hat{a}_{m-1}^H b_1 & \cdots & \hat{a}_{m-1}^H b_{n-1} \end{pmatrix} \right\|_F^2 \\ &= \sum_i \sum_j |\hat{a}_i^H b_j|^2 \\ &\leq \sum_i \sum_j \|\hat{a}_i\|_2^2 \|b_j\|_2^2 \quad (\text{Cauchy-Schwartz}) \\ &= \left(\sum_i \|\hat{a}_i\|_2^2 \right) \left(\sum_j \|b_j\|^2 \right) = \|A\|_F^2 \|B\|_F^2. \end{aligned}$$

so that $\|AB\|_F^2 \leq \|A\|_2^2 \|B\|_2^2$. Taking the square-root of both sides established the desired result.

 [BACK TO TEXT](#)

Homework 2.28 Let $\|\cdot\|$ be a matrix norm induced by the $\|\cdot\|$ vector norm. Show that $\kappa(A) = \|A\| \|A^{-1}\| \geq 1$.

Answer:

$$\|I\| = \|AA^{-1}\| \leq \|A\| \|A^{-1}\|.$$

But

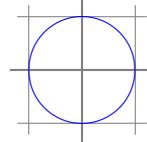
$$\|I\| = \max_{\|x\|=1} \|Ix\| = \max_{\|x\|} \|x\| = 1.$$

Hence $1 \leq \|A\| \|A^{-1}\|$.

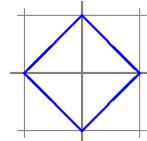
 [BACK TO TEXT](#)

Homework 2.29 A vector $x \in \mathbb{R}^2$ can be represented by the point to which it points when rooted at the origin. For example, the vector $x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ can be represented by the point $(2, 1)$. With this in mind, plot

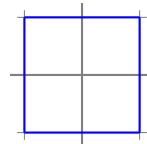
1. The points corresponding to the set $\{x \mid \|x\|_2 = 1\}$.



2. The points corresponding to the set $\{x \mid \|x\|_1 = 1\}$.



3. The points corresponding to the set $\{x \mid \|x\|_\infty = 1\}$.



 [BACK TO TEXT](#)

Homework 2.30 Consider

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

1. Compute the 1-norm of each column and pick the largest of these:
 $\|A\|_1 = \max(|1| + |-1|, |2| + |1|, |-1| + |0|) = \max(2, 3, 1) = 3$
2. Compute the 1-norm of each row and pick the largest of these:
 $\|A\|_\infty = \max(|1| + |2| + |-1|, |-1| + |1| + |0|) = \max(2, 4) = 4$
3. $\|A\|_F = \sqrt{1^2 + 2^2 + (-1)^2 + (-1)^2 + 1^2 + 0^2} = \sqrt{8} = 2\sqrt{2}$

◀ BACK TO TEXT

Homework 2.31 Show that for all $x \in \mathbb{C}^n$

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

Answer:

$\|x\|_2 \leq \|x\|_1$: We will instead prove that $\|x\|_2^2 \leq \|x\|_1^2$:

$$\|x\|_2^2 = \sum_i |\chi_i|^2 \leq \left(\sum_i |\chi_i| \right)^2 = \|x\|_1^2.$$

Hence $\|x\|_2 \leq \|x\|_1$.

$\|x\|_1 \leq \sqrt{n}\|x\|_2$: This requires the following trick: Let e be the vector of size n with only ones as its entry and $|x|$ be the vector x except with its elements replaced by their absolute values. Notice that $\|x\|_1 = |x^T e| = ||x|e||$. Also, $\| |x| \|_2 = \|x\|_2$.

Now

$$\|x\|_1 = |x^T e| = ||x|e|| \leq \| |x| \|_2 \|e\|_2 = \sqrt{n}\|x\|_2.$$

Here, the inequality comes from applying the Cauchy-Schwartz inequality.

2. $\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty.$

Answer:

$\|x\|_\infty \leq \|x\|_1$:

Let k be the index such that $|x_k| = \|x\|_\infty$. Then

$$\|x\|_\infty = |x_k| \leq \sum_i |x_i| = \|x\|_1.$$

$\|x\|_1 \leq n\|x\|_\infty$:

Let k be the index such that $|x_k| = \|x\|_\infty$. $\|x\|_1 = \sum_i |x_i| \leq \sum_i |x_k| = n|x_k| = n\|x\|_\infty$.

3. $(1/\sqrt{n})\|x\|_2 \leq \|x\|_\infty \leq \sqrt{n}\|x\|_2$.

Answer:

$(1/\sqrt{n})\|x\|_2 \leq \|x\|_\infty$: Let k be the index such that $|x_k| = \|x\|_\infty$.

$$\|x\|_\infty = |x_k| = \sqrt{|x_k|^2} = \sqrt{\frac{\sum_i |x_i|^2}{n}} \geq \frac{\sqrt{\sum_i |x_i|^2}}{\sqrt{n}} = \frac{\|x\|_2}{\sqrt{n}}$$

$\|x\|_\infty \leq \sqrt{n}\|x\|_2$:

This follows from the fact that $\|x\|_\infty \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$.

BACK TO TEXT

Homework 2.32 (I need to double check that this is true!)

Prove that if for all x $\|x\|_v \leq \beta\|x\|_\mu$ then $\|A\|_v \leq \beta\|A\|_{\mu,v}$.

Answer:

$$\|A\|_v = \max_{\|x\|_v=1} \|Ax\|_v \leq \max_{\|x\|_v=1} \beta\|Ax\|_v = \beta \max_{\|x\|_v=1} \|Ax\|_v = \beta\|A\|_{\mu,v}.$$

BACK TO TEXT

Homework 2.33 Partition

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

Prove that

$$1. \|A\|_F = \|A^T\|_F.$$

Answer:

$$\|A\|_F^2 = \sum_{i=0}^{m-1} \left(\sum_{j=0}^{n-1} |\alpha_{i,j}|^2 \right) = \sum_{j=0}^{n-1} \left(\sum_{i=0}^{m-1} |\alpha_{i,j}|^2 \right) = \|A^T\|_F^2$$

$$2. \|A\|_F = \sqrt{\|a_0\|_2^2 + \|a_1\|_2^2 + \cdots + \|a_{n-1}\|_2^2}.$$

Answer:

$$\|A\|_F^2 = \sum_{i=0}^{m-1} \left(\sum_{j=0}^{n-1} |\alpha_{i,j}|^2 \right) = \sum_{j=0}^{n-1} \left(\sum_{i=0}^{m-1} |\alpha_{i,j}|^2 \right) = \sum_{j=0}^{n-1} \|a_j\|_2^2$$

$$3. \|A\|_F = \sqrt{\|\tilde{a}_0\|_2^2 + \|\tilde{a}_1\|_2^2 + \cdots + \|\tilde{a}_{m-1}\|_2^2}.$$

Answer:

$$\|A\|_F^2 = \|A^T\|_F^2 = \left\| \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix}^T \right\|_F^2 = \left\| \begin{pmatrix} \tilde{a}_0 & | & \tilde{a}_1 & | & \cdots & | & \tilde{a}_{m-1} \end{pmatrix} \right\|_F^2 = \|\tilde{a}_0\|_2^2 + \|\tilde{a}_1\|_2^2 + \cdots + \|\tilde{a}_{m-1}\|_2^2.$$

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

[BACK TO TEXT](#)

Homework 2.34 Let for $e_j \in \mathbb{R}^n$ (a standard basis vector), compute

- $\|e_j\|_2 = 1$
- $\|e_j\|_1 = 1$
- $\|e_j\|_\infty = 1$
- $\|e_j\|_p = 1$

[BACK TO TEXT](#)

Homework 2.35 Let for $I \in \mathbb{R}^{n \times n}$ (the identity matrix), compute

- $\|I\|_F = \sqrt{n}$
- $\|I\|_1 = 1$
- $\|I\|_\infty = 1$

[BACK TO TEXT](#)

Homework 2.36 Let $\|\cdot\|$ be a vector norm defined for a vector of any size (arbitrary n). Let $\|\cdot\|$ be the induced matrix norm. Prove that $\|I\| = 1$ (where I equals the identity matrix).

Answer:

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

Conclude that $\|I\|_p = 1$ for any p-norm.

[BACK TO TEXT](#)

Homework 2.37 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

- $\|D\|_1 = \max_i |\delta_i|$
- $\|D\|_\infty = \max_i |\delta_i|$

[◀ BACK TO TEXT](#)

Homework 2.38 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). Then

$$\|D\|_p = \max_i |\delta_i|.$$

Prove your answer.

Answer:

$$\begin{aligned}
 \|D\|_p &= \max_{\|x\|_p=1} \|Dx\|_p \\
 &= \max_{\|x\|_p=1} \left\| \begin{pmatrix} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & \delta_{n-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \right\|_p \\
 &= \max_{\|x\|_p=1} \left\| \begin{pmatrix} \delta_0 \chi_0 \\ \delta_1 \chi_1 \\ \vdots \\ \delta_{n-1} \chi_{n-1} \end{pmatrix} \right\|_p = \max_{\|x\|_p=1} \sqrt[p]{|\delta_0 \chi_0|^p + \cdots + |\delta_{n-1} \chi_{n-1}|^p} \\
 &= \max_{\|x\|_p=1} \sqrt[p]{|\delta_0|^p |\chi_0|^p + \cdots + |\delta_{n-1}|^p |\chi_{n-1}|^p} \leq \max_{\|x\|_p=1} \sqrt[p]{\max_k |\delta_k|^p |\chi_0|^p + \cdots + \max_k |\delta_k|^p |\chi_{n-1}|^p} \\
 &= \max_k |\delta_k| \max_{\|x\|_p=1} \sqrt[p]{|\chi_0|^p + \cdots + |\chi_{n-1}|^p} = \max_k |\delta_k| \max_{\|x\|_p=1} \|x\|_p = \max_k |\delta_k|.
 \end{aligned}$$

Also,

$$\|D\|_p = \max_{\|x\|_p=1} \|Dx\|_p \geq \|De_J\|_p = \|\delta_J e_J\|_p = |\delta_J| \|e_J\|_p = |\delta_J| = \max_k |\delta_k|$$

where J is chosen to be the index so that $= |\delta_J| = \max_k |\delta_k|$. Thus

$$\max_k |\delta_k| \leq \|D\|_p \leq \max_k |\delta_k|$$

from which we conclude that $\|D\|_p = \max_k |\delta_k|$.

 BACK TO TEXT

Homework 2.39 Let $y \in \mathbb{C}^n$. Show that $\|y^H\|_2 = \|y\|_2$.

Answer:

$$\begin{aligned}\|y^H\|_2 &= \max_{\|x\|_2=1} \|y^H x\|_2 = \max_{\|x\|_2=1} |y^H x| \\ &\leq \max_{\|x\|_2=1} \|y\|_2 \|x\|_2 = \|y\|_2 \max_{\|x\|_2=1} \|x\|_2 = \|y\|_2.\end{aligned}$$

Also,

$$\|y^H\|_2 = \max_{x \neq 0} \frac{\|y^H x\|_2}{\|x\|_2} \geq \frac{\|y^H y\|_2}{\|y\|_2} = \frac{\|y\|_2^2}{\|y\|_2} = \|y\|_2.$$

Hence $\|y^H\|_2 = \|y\|_2$.

 BACK TO TEXT

Chapter 3. Notes on Orthogonality and the SVD (Answers)

Homework 3.4 Let $Q \in \mathbb{C}^{m \times n}$ (with $n \leq m$). Partition $Q = \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array} \right)$. Show that Q is an orthonormal matrix if and only if q_0, q_1, \dots, q_{n-1} are mutually orthonormal.

Answer: 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

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

Now consider $Q^H Q = I$:

$$\begin{pmatrix} q_0^H q_0 & q_0^H q_1 & \cdots & q_0^H q_{n-1} \\ \hline q_1^H q_0 & q_1^H q_1 & \cdots & q_1^H q_{n-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline q_{n-1}^H q_0 & q_{n-1}^H q_1 & \cdots & q_{n-1}^H q_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}.$$

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

[◀ BACK TO TEXT](#)

Homework 3.6 Let $Q \in \mathbb{C}^{m \times m}$. Show that if Q is unitary then $Q^{-1} = Q^H$ and $QQ^H = I$.

Answer: If Q is unitary, then $Q^H Q = I$. If $A, B \in \mathbb{C}^{m \times m}$, the matrix B such that $BA = I$ is the inverse of A . Hence $Q^{-1} = Q^H$. Also, if $BA = I$ then $AB = I$ and hence $QQ^H = I$.

[◀ BACK TO TEXT](#)

Homework 3.7 Let $Q_0, Q_1 \in \mathbb{C}^{m \times m}$ both be unitary. Show that their product, $Q_0 Q_1$, is unitary.

Answer: Obviously, $Q_0 Q_1$ is a square matrix.

$$(Q_0 Q_1)^H (Q_0 Q_1) = Q_1^H \underbrace{Q_0^H Q_0}_I Q_1 = \underbrace{Q_1^H Q_1}_I = I.$$

Hence $Q_0 Q_1$ is unitary.

[◀ BACK TO TEXT](#)

Homework 3.8 Let $Q_0, Q_1, \dots, Q_{k-1} \in \mathbb{C}^{m \times m}$ all be unitary. Show that their product, $Q_0 Q_1 \cdots Q_{k-1}$, is unitary.

Answer: Strictly speaking, we should do a proof by induction. But instead we will make the more informal argument that

$$\begin{aligned}(Q_0 Q_1 \cdots Q_{k-1})^H Q_0 Q_1 \cdots Q_{k-1} &= Q_{k-1}^H \cdots Q_1^H Q_0^H Q_0 Q_1 \cdots Q_{k-1} \\ &= Q_{k-1}^H \cdots Q_1^H \underbrace{Q_0^H Q_0}_I Q_1 \cdots Q_{k-1} = I.\end{aligned}$$

I
 I
 I
 I

[◀ BACK TO TEXT](#)

Homework 3.9 Let $U \in \mathbb{C}^{m \times m}$ be unitary and $x \in \mathbb{C}^m$, then $\|Ux\|_2 = \|x\|_2$.

Answer: $\|Ux\|_2^2 = (Ux)^H (Ux) = x^H \underbrace{U^H U}_I x = x^H x = \|x\|_2^2$. Hence $\|Ux\|_2 = \|x\|_2$.

[◀ BACK TO TEXT](#)

Homework 3.10 Let $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ be unitary matrices and $A \in \mathbb{C}^{m \times n}$. Then

$$\|UA\|_2 = \|AV\|_2 = \|A\|_2.$$

Answer:

- $\|UA\|_2 = \max_{\|x\|_2=1} \|UAx\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \|A\|_2$.
- $\|AV\|_2 = \max_{\|x\|_2=1} \|AVx\|_2 = \max_{\|Vx\|_2=1} \|A(Vx)\|_2 = \max_{\|y\|_2=1} \|Ay\|_2 = \|A\|_2$.

[◀ BACK TO TEXT](#)

Homework 3.11 Let $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ be unitary matrices and $A \in \mathbb{C}^{m \times n}$. Then

$$\|UA\|_F = \|AV\|_F = \|A\|_F.$$

Answer:

- Partition $A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$. Then it is easy to show that $\|A\|_F^2 = \sum_{j=0}^{n-1} \|a_j\|_2^2$. Thus

$$\begin{aligned}\|UA\|_F^2 &= \|U \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)\|_F^2 = \left\| \left(\begin{array}{c|c|c|c} Ua_0 & Ua_1 & \cdots & Ua_{n-1} \end{array} \right) \right\|_F^2 \\ &= \sum_{j=0}^{n-1} \|Ua_j\|_2^2 = \sum_{j=0}^{n-1} \|a_j\|_2^2 = \|A\|_F^2.\end{aligned}$$

Hence $\|UA\|_F = \|A\|_F$.

- Partition $A = \begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix}$. Then it is easy to show that $\|A\|_F^2 = \sum_{i=0}^{m-1} \|(\hat{a}_i^T)^H\|_2^2$. Thus

$$\begin{aligned}\|AV\|_F^2 &= \left\| \begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix} V \right\|_F^2 = \left\| \begin{pmatrix} \hat{a}_0^T V \\ \hat{a}_1^T V \\ \vdots \\ \hat{a}_{m-1}^T V \end{pmatrix} \right\|_F^2 = \sum_{i=0}^{m-1} \|(\hat{a}_i^T V)^H\|_2^2 \\ &= \sum_{i=0}^{m-1} \|V^H (\hat{a}_i^T)^H\|_2^2 = \sum_{i=0}^{m-1} \|(\hat{a}_i^T)^H\|_2^2 = \|A\|_F^2.\end{aligned}$$

Hence $\|AV\|_F = \|A\|_F$.

 [BACK TO TEXT](#)

Homework 3.13 Let $D = \text{diag}(\delta_0, \dots, \delta_{n-1})$. Show that $\|D\|_2 = \max_{i=0}^{n-1} |\delta_i|$.

Answer:

$$\begin{aligned}\|D\|_2^2 &= \max_{\|x\|_2=1} \|Dx\|_2^2 = \max_{\|x\|_2=1} \left\| \begin{pmatrix} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_{n-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} \right\|_2^2 \\ &= \max_{\|x\|_2=1} \left\| \begin{pmatrix} \delta_0 \chi_0 \\ \delta_1 \chi_1 \\ \vdots \\ \delta_{n-1} \chi_{n-1} \end{pmatrix} \right\|_2^2 = \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} |\delta_j \chi_j|^2 \right] = \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} [|\delta_j|^2 |\chi_j|^2] \right] \\ &\leq \max_{\|x\|_2=1} \left[\sum_{j=0}^{n-1} \left[\max_{i=0}^{n-1} |\delta_i|^2 |\chi_j|^2 \right] \right] = \max_{\|x\|_2=1} \left[\max_{i=0}^{n-1} |\delta_i|^2 \sum_{j=0}^{n-1} |\chi_j|^2 \right] = \left(\max_{i=0}^{n-1} |\delta_i| \right)^2 \max_{\|x\|_2=1} \|x\|_2^2 \\ &= \left(\max_{i=0}^{n-1} |\delta_i| \right)^2.\end{aligned}$$

so that $\|D\|_2 \leq \max_{i=0}^{n-1} |\delta_i|$.

Also, choose j so that $|\delta_j| = \max_{i=0}^{n-1} |\delta_i|$. Then

$$\|D\|_2 = \max_{\|x\|_2=1} \|Dx\|_2 \geq \|De_j\|_2 = \|\delta_j e_j\|_2 = |\delta_j| \|e_j\|_2 = |\delta_j| = \max_{i=0}^{n-1} |\delta_i|.$$

so that $\max_{i=0}^{n-1} |\delta_i| \leq \|D\|_2 \leq \max_{i=0}^{n-1} |\delta_i|$, which implies that $\|D\|_2 = \max_{i=0}^{n-1} |\delta_i|$.

BACK TO TEXT

Homework 3.14 Let $A = \begin{pmatrix} A_T \\ 0 \end{pmatrix}$. Use the SVD of A_T to show that $\|A\|_2 = \|A_T\|_2$.

Answer: Let $A_T = U_T \Sigma_T V_T^H$ be the SVD of A_T . Then

$$A = \begin{pmatrix} A_T \\ 0 \end{pmatrix} = \begin{pmatrix} U_T \Sigma_T V_T^H \\ 0 \end{pmatrix} = \begin{pmatrix} U_T \\ 0 \end{pmatrix} \Sigma_T V_T^H,$$

which is the SVD of A . As a result, clearly the largest singular value of A_T equals the largest singular value of A and hence $\|A\|_2 = \|A_T\|_2$.

BACK TO TEXT

Homework 3.15 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 .

Answer: 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 .

BACK TO TEXT

Homework 3.16 Let $A \in \mathbb{C}^{m \times n}$ with $A = \begin{pmatrix} \sigma_0 & | & 0 \\ 0 & | & B \end{pmatrix}$ and assume that $\|A\|_2 = \sigma_0$. Show that $\|B\|_2 \leq \|A\|_2$. (Hint: Use the SVD of B .)

Answer: Let $B = U_B \Sigma_B V_B^H$ be the SVD of B . Then

$$A = \begin{pmatrix} \sigma_0 & | & 0 \\ 0 & | & B \end{pmatrix} = \begin{pmatrix} \sigma_0 & | & 0 \\ 0 & | & U_B \Sigma_B V_B^H \end{pmatrix} = \begin{pmatrix} 1 & | & 0 \\ 0 & | & U_B \end{pmatrix} \begin{pmatrix} \sigma_0 & | & 0 \\ 0 & | & \Sigma_B \end{pmatrix} \begin{pmatrix} 1 & | & 0 \\ 0 & | & V_B \end{pmatrix}^H$$

which shows the relationship between the SVD of A and B . Since $\|A\|_2 = \sigma_0$, it must be the case that the diagonal entries of Σ_B are less than or equal to σ_0 in magnitude, which means that $\|B\|_2 \leq \|A\|_2$.

BACK TO TEXT

Homework 3.17 Prove Lemma 3.12 for $m \leq n$.

Answer: You can use the following as an outline for your proof:

Proof: First, let us observe that if $A = 0$ (the zero matrix) then the theorem trivially holds: $A = UDV^H$

where $U = I_{m \times m}$, $V = I_{n \times n}$, and $D = \begin{pmatrix} \cdot & & \\ & \ddots & \\ & & 0 \end{pmatrix}$, so that D_{TL} is 0×0 . Thus, w.l.o.g. assume that $A \neq 0$.

We will employ a proof by induction on m .

- **Base case:** $m = 1$. In this case $A = \begin{pmatrix} \hat{a}_0^T \end{pmatrix}$ where $\hat{a}_0^T \in \mathbb{R}^{1 \times n}$ is its only row. By assumption, $\hat{a}_0^T \neq 0$. Then

$$A = \begin{pmatrix} \hat{a}_0^T \end{pmatrix} = \begin{pmatrix} 1 \end{pmatrix} (\|\hat{a}_0^T\|_2) \begin{pmatrix} v_0 \end{pmatrix}^H$$

where $v_0 = (\hat{a}_0^T)^H / \|\hat{a}_0^T\|_2$. Choose $V_1 \in \mathbb{C}^{n \times (n-1)}$ so that $V = \begin{pmatrix} v_0 & V_1 \end{pmatrix}$ is unitary. Then

$$A = \begin{pmatrix} \hat{a}_0^T \end{pmatrix} = \begin{pmatrix} 1 \end{pmatrix} \begin{pmatrix} \|\hat{a}_0^T\|_2 & 0 \end{pmatrix} \begin{pmatrix} v_0 & V_1 \end{pmatrix}^H = UDV^H$$

where $D_{TL} = \begin{pmatrix} \delta_0 \end{pmatrix} = \begin{pmatrix} \|\hat{a}_0^T\|_2 \end{pmatrix}$ and $U = \begin{pmatrix} 1 \end{pmatrix}$.

- **Inductive step:** Similarly modify the inductive step of the proof of the theorem.
- **By the Principle of Mathematical Induction** the result holds for all matrices $A \in \mathbb{C}^{m \times n}$ with $m \geq n$.

◀ BACK TO TEXT

Homework 3.35 Show that if $A \in \mathbb{C}^{m \times m}$ is nonsingular, then

- $\|A\|_2 = \sigma_0$, the largest singular value;
- $\|A^{-1}\|_2 = 1/\sigma_{m-1}$, the inverse of the smallest singular value; and
- $\kappa_2(A) = \sigma_0/\sigma_{m-1}$.

Answer: Answer needs to be filled in

◀ BACK TO TEXT

Homework 3.36 Let $A \in \mathbb{C}^{n \times n}$ have singular values $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{n-1}$ (where σ_{n-1} may equal zero). Prove that $\sigma_{n-1} \leq \|Ax\|_2/\|x\|_2 \leq \sigma_0$ (assuming $x \neq 0$).

Answer:

$$\frac{\|Ax\|_2}{\|x\|_2} \leq \max_{y \neq 0} \frac{\|Ay\|_2}{\|y\|_2} = \sigma_0.$$

(for example, as a consequence of how σ_0 shows up in the derivation of the SVD).

$$\begin{aligned}\frac{\|Ax\|_2}{\|x\|_2} \geq \min_{y \neq 0} \frac{\|Ay\|_2}{\|y\|_2} &= \min_{y \neq 0} \frac{\|U\Sigma V^H y\|_2}{\|y\|_2} = \min_{y \neq 0} \frac{\|\Sigma V^H y\|_2}{\|y\|_2} \\ &= \min_{y \neq 0} \frac{\|\Sigma V^H y\|_2}{\|V^H y\|_2} = \min_{z \neq 0} \frac{\|\Sigma z\|_2}{\|z\|_2} = \sigma_{n-1}\end{aligned}$$

(through a simple argument about the smallest element (in magnitude) on the diagonal of a diagonal matrix).

BACK TO TEXT

Homework 3.37 Let $D \in \mathbb{R}^{n \times n}$ be a diagonal matrix with entries $\delta_0, \delta_1, \dots, \delta_{n-1}$ on its diagonal. Show that

$$1. \max_{x \neq 0} \|Dx\|_2/\|x\|_2 = \max_{0 \leq i < n} |\delta_i|. \text{ Answer: It is usually simpler to square both sides: } \max_{x \neq 0} \|Dx\|_2^2/\|x\|_2^2 = \max_{0 \leq i < n} \delta_i^2$$

$$\begin{aligned}\max_{x \neq 0} \|Dx\|_2^2/\|x\|_2^2 &= \max_{\|x\|_2=1} \|Dx\|_2^2 = \max_{\|x\|_2=1} (\delta_0^2 \chi_0^2 + \dots + \delta_{n-1}^2 \chi_{n-1}^2) \\ &\leq \max_{\|x\|_2=1} \left(\max_{0 \leq i < n} \delta_i^2 \chi_0^2 + \dots + \max_{0 \leq i < n} \delta_i^2 \chi_{n-1}^2 \right) = \max_{0 \leq i < n} \delta_i^2 \max_{\|x\|_2=1} \|x\|_2^2 \\ &= \max_{0 \leq i < n} \delta_i^2\end{aligned}$$

Also, letting k be the index such that $\delta_k = \max_{0 \leq i < n} \delta_i^2$,

$$\begin{aligned}\max_{x \neq 0} \|Dx\|_2^2/\|x\|_2^2 &\geq \|De_k\|_2^2/\|e_k\|_2^2 = \|De_k\|_2^2 \\ &= e_k^T D^2 e_k = \delta_k^2 = \max_{0 \leq i < n} \delta_i^2.\end{aligned}$$

$$2. \min_{x \neq 0} \|Dx\|_2/\|x\|_2 = \min_{0 \leq i < n} |\delta_i|. \text{ Answer: Again, it is usually simpler to square both sides: } \min_{x \neq 0} \|Dx\|_2^2/\|x\|_2^2 = \min_{0 \leq i < n} \delta_i^2$$

$$\begin{aligned}\min_{x \neq 0} \|Dx\|_2^2/\|x\|_2^2 &= \min_{\|x\|_2=1} \|Dx\|_2^2 = \min_{\|x\|_2=1} (\delta_0^2 \chi_0^2 + \dots + \delta_{n-1}^2 \chi_{n-1}^2) \\ &\geq \min_{\|x\|_2=1} \left(\min_{0 \leq i < n} \delta_i^2 \chi_0^2 + \dots + \min_{0 \leq i < n} \delta_i^2 \chi_{n-1}^2 \right) = \min_{0 \leq i < n} \delta_i^2 \min_{\|x\|_2=1} \|x\|_2^2 \\ &= \min_{0 \leq i < n} \delta_i^2\end{aligned}$$

Also, letting k be the index such that $\delta_k = \min_{0 \leq i < n} \delta_i^2$,

$$\begin{aligned}\min_{x \neq 0} \|Dx\|_2^2/\|x\|_2^2 &\leq \|De_k\|_2^2/\|e_k\|_2^2 = \|De_k\|_2^2 \\ &= e_k^T D^2 e_k = \delta_k^2 = \min_{0 \leq i < n} \delta_i^2.\end{aligned}$$

*

BACK TO TEXT

Homework 3.38 Let $A \in \mathbb{C}^{n \times n}$ have the property that for all vectors $x \in \mathbb{C}^n$ it holds that $\|Ax\|_2 = \|x\|_2$. Use the SVD to prove that A is unitary.

Answer: Let $A = U\Sigma V^H$ be the singular value decomposition. It suffices to show that $\Sigma = I$ or, equivalently, that $\sigma_0 = \dots = \sigma_{n-1}$. This then means A is the result of multiplying three unitary matrices, and hence unitary itself.

Answer 1: Partition

$$U = \begin{pmatrix} u_0 & \cdots & u_{n-1} \end{pmatrix} \quad \text{and} \quad V = \begin{pmatrix} v_0 & \cdots & v_{n-1} \end{pmatrix}$$

We will consider $\|Ax\|_2^2 = \|x\|_2^2$. Then

$$\|x\|_2^2 = \|U\Sigma V^H x\|_2^2 = x^H V \Sigma U^H U \Sigma V^H x = x^H V \Sigma^2 V^H x.$$

Recall that, by convention, Σ has nonnegative real values on its diagonal.

Now, pick $x = v_i$. Then

$$1 = \|v_i\|_2^2 = v_i^H V \Sigma^2 V^H v_i = e_i^H \Sigma^2 e_i = \sigma_i^2.$$

Hence $\sigma_i = 1$.

Answer 2: $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{n-1}$.

$$\sigma_0 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = 1.$$

$$\sigma_{n-1} = \min_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = 1.$$

Hence $\sigma_i = 1$ for $0 \leq i < n$.

Answer 3: (Does not use the SVD) $\|Ax\|_2 = \|x\|_2$ is equivalent to $\|Ax\|_2^2 = \|x\|_2^2$ and to $x^H A^H A x = x^H x$.

Let a_i denote the i th column of A . Pick $x = e_i$. Then we see that $e_i^H A^H A e_i = (Ae_i)^H Ae_i = a_i^H a_i = 1$ and hence the columns of A are all of length one. Now, consider $x = e_i + e_j$ where $i \neq j$:

$$2 = \|e_i + e_j\|_2^2 = (e_i + e_j)^H A^H A (e_i + e_j) = e_i^H A^H A e_i + 2e_i^H A^H A e_j + e_j^H A^H A e_j = 2 + 2a_i^H a_j.$$

This means that $a_i^H a_j = 0$. Hence the columns of A are mutually orthonormal and A is unitary.

BACK TO TEXT

Homework 3.39 Use the SVD to prove that $\|A\|_2 = \|A^T\|_2$

Answer:

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

BACK TO TEXT

Homework 3.40 Compute the SVD of $\begin{pmatrix} 1 \\ -2 \end{pmatrix} \begin{pmatrix} 2 & 1 \end{pmatrix}$.

Answer: The trick is to recognize that you need to normalize each of the vectors:

$$\begin{aligned} \begin{pmatrix} 1 \\ -2 \end{pmatrix} \begin{pmatrix} -1 & 1 \end{pmatrix} &= \left(\frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ -2 \end{pmatrix} \sqrt{5} \right) \left(\sqrt{2} \frac{1}{\sqrt{2}} \begin{pmatrix} -1 & 1 \end{pmatrix} \right) \\ &= \underbrace{\left(\frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ -2 \end{pmatrix} \right)}_U \underbrace{\Sigma}_{(\sqrt{5}\sqrt{2})} \underbrace{\left(\frac{1}{\sqrt{2}} \begin{pmatrix} -1 & 1 \end{pmatrix} \right)}_{V^H} \end{aligned}$$

◀ BACK TO TEXT

Chapter 4. Notes on Gram-Schmidt QR Factorization (Answers)

Homework 4.1 What happens in the Gram-Schmidt algorithm if the columns of A are NOT linearly independent? **Answer:** If a_j is the first column such that $\{a_0, \dots, a_j\}$ are linearly dependent, then a_j^\perp will equal the zero vector and the process breaks down. How might one fix this? **Answer:** When a vector with a_j^\perp is encountered, the columns can be rearranged so that that column (or those columns) come last.

How can the Gram-Schmidt algorithm be used to identify which columns of A are linearly independent? **Answer:** Again, if $a_j^\perp = 0$ for some j , then the columns are linearly dependent.

BACK TO TEXT

Homework 4.2 Convince yourself that the relation between the vectors $\{a_j\}$ and $\{q_j\}$ in the algorithms in Figure 4.2 is given by

$$\left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} q_0 & q_1 & \cdots & q_{n-1} \end{array} \right) \left(\begin{array}{c|c|c|c} \rho_{0,0} & \rho_{0,1} & \cdots & \rho_{0,n-1} \\ \hline 0 & \rho_{1,1} & \cdots & \rho_{1,n-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & \rho_{n-1,n-1} \end{array} \right),$$

where

$$q_i^H q_j = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \rho_{i,j} = \begin{cases} q_i^H a_j & \text{for } i < j \\ \|a_j - \sum_{i=0}^{j-1} \rho_{i,j} q_i\|_2 & \text{for } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Answer: Just watch the video for this lecture!

BACK TO TEXT

Homework 4.5 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 $k \times k$. Show that

1. $A_L = Q_L R_{TL}$: $Q_L R_{TL}$ equals the QR factorization of A_L ,
2. $C(A_L) = C(Q_L)$: 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. $(A_R - Q_L R_{TR})^H Q_L = 0$,

5. $A_R - Q_L R_{TR} = Q_R R_{BR}$, and
6. $\mathcal{C}(A_R - Q_L R_{TR}) = \mathcal{C}(Q_R)$.

Answer: Consider the fact that $A = QR$. Then

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

Hence

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

- The left equation answers 1.
- Rearranging the right equation yields $A_R - Q_L R_{TR} = Q_R R_{BR}$, which answers 5.
- $\mathcal{C}(A_L) = \mathcal{C}(Q_L)$ can be shown by showing 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 $y \in \mathcal{C}(A_L)$. (Notice that R_{TL} is nonsingular because it is a triangular matrix that has only nonzeros on its diagonal.)

This answer 2.

- 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.

- 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

$$(A_R - Q_L R_{TR})^H Q_L = R_{BR}^H Q_R^H Q_L = 0.$$

- The proof of 6. follows similar to the proof of 2.

◀ BACK TO TEXT

Homework 4.6 Implement `GS_unb_var1` using MATLAB and **Spark**. You will want to use the `laff` routines summarized in Appendix B. (I'm not sure if you can visualize the algorithm with **PictureFLAME**. Try it!)

Answer: Implementations can be found in `Programming/chapter04_answers`.

◀ BACK TO TEXT

Homework 4.7 Implement MGS_unb_var1 using MATLAB and Spark. You will want to use the laff routines summarized in Appendix B. (I'm not sure if you can visualize the algorithm with PictureFLAME. Try it!)

Answer: Implementations can be found in Programming/chapter04_answers.

 [BACK TO TEXT](#)

Chapter 6. Notes on Householder QR Factorization (Answers)

Homework 6.2 Show that if H is a reflector, then

- $HH = I$ (reflecting the reflection of a vector results in the original vector).

Answer:

$$(I - 2uu^H)(I - 2uu^H) = I - 2uu^H - 2uu^H + 4u \underbrace{u^H u}_1 u^H = I - 4uu^H + 4uu^H = I$$

- $H = H^H$.

Answer:

$$(I - 2uu^H)^H = I - 2(u^H)^H u^H = I - 2uu^H$$

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

Answer:

$$H^H H = HH = I$$

[BACK TO TEXT](#)

Homework 6.4 Show that if $x \in \mathbb{R}^n$, $v = x \mp \|x\|_2 e_0$, and $\tau = v^T v / 2$ then $(I - \frac{1}{\tau} vv^T)x = \pm \|x\|_2 e_0$.

Answer: This is surprisingly messy...

[BACK TO TEXT](#)

Homework 6.5 Verify that

$$\left(I - \frac{1}{\tau} \left(\frac{1}{u_2} \right) \left(\frac{1}{u_2} \right)^H \right) \begin{pmatrix} \chi_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \rho \\ 0 \end{pmatrix}$$

where $\tau = u^H u / 2 = (1 + u_2^H u_2) / 2$ and $\rho = \oplus \|x\|_2$.

Hint: $\rho \bar{\rho} = |\rho|^2 = \|x\|_2^2$ since H preserves the norm. Also, $\|x\|_2^2 = |\chi_1|^2 + \|x_2\|_2^2$ and $\sqrt{\frac{z}{\bar{z}}} = \frac{z}{|\bar{z}|}$.

Answer: Again, surprisingly messy...

[BACK TO TEXT](#)

Homework 6.6 Function `Housev.m` implements the steps in Figure 6.2 (left). Update this implementation with the equivalent steps in Figure 6.2 (right), which is closer to how it is implemented in practice.

Answer: See `Programming/chapter06_answers/Housev_alt.m`

[BACK TO TEXT](#)

Homework 6.7 Show that

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

Answer:

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

[BACK TO TEXT](#)

Homework 6.8 Given $A \in \mathbb{R}^{m \times n}$ show that the cost of the algorithm in Figure 6.4 is given by

$$C_{\text{HQR}}(m, n) \approx 2mn^2 - \frac{2}{3}n^3 \text{ flops.}$$

Answer: The bulk of the computation is in $w_{12}^T = (a_{12}^T + u_{21}^H A_{22})/\tau_1$ and $A_{22} - u_{21} w_{12}^T$. During the k th iteration (when R_{TL} is $k \times k$), this means a matrix-vector multiplication ($u_{21}^H A_{22}$) and rank-1 update with

matrix A_{22} which is of size approximately $(m-k) \times (n-k)$ for a cost of $4(m-k)(n-k)$ flops. Thus the total cost is approximately

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

[BACK TO TEXT](#)

Homework 6.9 Implement the algorithm in Figure 6.4 as

```
function [ Aout, tout ] = HQR_unb_var1( A, t )
```

Input is an $m \times n$ matrix A and vector t of size n . Output is the overwritten matrix A and the vector of scalars that define the Householder transformations. You may want to use `Programming/chapter06/test_HQR_unb_var1.m` to check your implementation.

Answer: See `Programming/chapter06_answers/HQR_unb_var1.m`.

[BACK TO TEXT](#)

Homework 6.12 Implement the algorithm in Figure 6.6 as

```
function Aout = FormQ_unb_var1( A, t )
```

You may want to use `Programming/chapter06/test_HQR_unb_var1.m` to check your implementation.

Answer: See `Programming/chapter06_answers/FormQ_unb_var1.m`.

[BACK TO TEXT](#)

Homework 6.13 Given $A \in \mathbb{C}^{m \times n}$ the cost of the algorithm in Figure 6.6 is given by

$$C_{\text{FormQ}}(m, n) \approx 2mn^2 - \frac{2}{3}n^3 \text{ flops.}$$

Answer: The answer for Homework 6.8 can be easily modified to establish this result.

[BACK TO TEXT](#)

Homework 6.14 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 6.6.

Answer: The benefit of accumulating Q via the sequence $\cdots (H_1(H_0I))$ is that many zeroes are preserved, thus reducing the necessary computations. In contrast, IH_0 creates a dense matrix, after which application of subsequent Householder transformations benefits less from zeroes.

BACK TO TEXT

Homework 6.15 Consider $u_1 \in \mathbb{C}^m$ with $u_1 \neq 0$ (the zero vector), $U_0 \in \mathbb{C}^{m \times k}$, and nonsingular $T_{00} \in \mathbb{C}^{k \times k}$. Define $\tau_1 = (u_1^H u_1)/2$, so that

$$H_1 = I - \frac{1}{\tau_1} u_1 u_1^H$$

equals a Householder transformation, and let

$$Q_0 = I - U_0 T_{00}^{-1} U_0^H.$$

Show that

$$Q_0 H_1 = (I - U_0 T_{00}^{-1} U_0^H)(I - \frac{1}{\tau_1} u_1 u_1^H) = I - \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} T_{00} & t_{01} \\ \hline 0 & \tau_1 \end{array} \right)^{-1} \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right)^H,$$

where $t_{01} = Q_0^H u_1$.

Answer:

$$\begin{aligned} Q_0 H_1 &= (I - U_0 T_{00}^{-1} U_0^H)(I - \frac{1}{\tau_1} u_1 u_1^H) \\ &= I - U_0 T_{00}^{-1} U_0^H - \frac{1}{\tau_1} u_1 u_1^H + U_0 T_{00}^{-1} U_0^H \frac{1}{\tau_1} u_1 u_1^H \\ &= I - U_0 T_{00}^{-1} U_0^H - \frac{1}{\tau_1} u_1 u_1^H + \frac{1}{\tau_1} U_0 T_{00}^{-1} t_{01} u_1^H \end{aligned}$$

Also

$$\begin{aligned} &I - \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} T_{00} & t_{01} \\ \hline 0 & \tau_1 \end{array} \right)^{-1} \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right)^H \\ &= I - \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right) \left(\begin{array}{c|c} T_{00}^{-1} & -T_{00}^{-1} t_{01}/\tau_1 \\ \hline 0 & 1/\tau_1 \end{array} \right) \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right)^H \\ &= I - \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right) \left(\begin{array}{c} T_{00}^{-1} U_0^H - T_{00}^{-1} t_{01} u_1^H / \tau_1 \\ \hline 1/\tau_1 u_1^H \end{array} \right) \\ &= I - U_0 (T_{00}^{-1} U_0^H - T_{00}^{-1} t_{01} u_1^H / \tau_1) - 1/\tau_1 u_1 u_1^H \\ &= I - U_0 T_{00}^{-1} U_0^H - \frac{1}{\tau_1} u_1 u_1^H + \frac{1}{\tau_1} U_0 T_{00}^{-1} t_{01} u_1^H \end{aligned}$$

[◀ BACK TO TEXT](#)

Homework 6.16 Consider $u_i \in \mathbb{C}^m$ with $u_i \neq 0$ (the zero vector). Define $\tau_i = (u_i^H u_i)/2$, so that

$$H_i = I - \frac{1}{\tau_i} u_i u_i^H$$

equals a Householder transformation, and let

$$U = \left(\begin{array}{c|c|c|c} u_0 & u_1 & \cdots & u_{k-1} \end{array} \right).$$

Show that

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

where T is an upper triangular matrix.

Answer: The results follows from Homework 6.15 via a proof by induction.

[◀ BACK TO TEXT](#)

Homework 6.17 Implement the algorithm in Figure 6.8 as

```
function T = FormT_unb_var1( U, t, T )
```

[◀ BACK TO TEXT](#)

Homework 6.18 Assuming all inverses exist, show that

$$\left(\begin{array}{c|c} T_{00} & t_{01} \\ \hline 0 & \tau_1 \end{array} \right)^{-1} = \left(\begin{array}{c|c} T_{00}^{-1} & -T_{00}^{-1}t_{01}/\tau_1 \\ \hline 0 & 1/\tau_1 \end{array} \right).$$

Answer:

$$\left(\begin{array}{c|c} T_{00} & t_{01} \\ \hline 0 & \tau_1 \end{array} \right) \left(\begin{array}{c|c} T_{00}^{-1} & -T_{00}^{-1}t_{01}/\tau_1 \\ \hline 0 & 1/\tau_1 \end{array} \right) = \left(\begin{array}{c|c} T_{00}T_{00}^{-1} & -T_{00}T_{00}^{-1}t_{01}/\tau_1 + \tau_1/\tau_1 \\ \hline 0 & \tau_1/\tau_1 \end{array} \right) = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & 1 \end{array} \right).$$

[◀ BACK TO TEXT](#)

Homework 6.19 Let $A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$, $v = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{pmatrix} = \begin{pmatrix} \|a_0\|_2^2 \\ \|a_1\|_2^2 \\ \vdots \\ \|a_{n-1}\|_2^2 \end{pmatrix}$, $q^T q = 1$ (of 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

$$\begin{pmatrix} \|b_0\|_2^2 \\ \|b_1\|_2^2 \\ \vdots \\ \|b_{n-1}\|_2^2 \end{pmatrix} = \begin{pmatrix} v_0 - \rho_0^2 \\ v_1 - \rho_1^2 \\ \vdots \\ v_{n-1} - \rho_{n-1}^2 \end{pmatrix}.$$

Answer:

$$a_i = (a_i - a_i^T qq) + a_i^T qq$$

and

$$(a_i - a_i^T qq)^T q = a_i^T q - a_i^T qq^T q = a_i^T q - a_i^T q = 0.$$

This means that

$$\|a_i\|_2^2 = \|(a_i - a_i^T qq) + a_i^T qq\|_2^2 = \|a_i - a_i^T qq\|_2^2 + \|a_i^T qq\|_2^2 = \|a_i - \rho_i q\|_2^2 + \|\rho_i q\|_2^2 = \|b_i\|_2^2 + \rho_i^2$$

so that

$$\|b_i\|_2^2 = \|a_i\|_2^2 - \rho_i^2 = v_i - \rho_i^2.$$

 [BACK TO TEXT](#)

Homework 6.20 In Section 4.4 we discuss how MGS yields a higher quality solution than does CGS in terms of the orthogonality of the computed columns. The classic example, already mentioned in Chapter 4, that illustrates this is

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

where $\epsilon = \sqrt{\epsilon_{\text{mach}}}$. In this exercise, you will compare and contrast the quality of the computed matrix Q for CGS, MGS, and Householder QR factorization.

- Start with the matrix

```

format long      % print out 16 digits
eps = 1.0e-8    % roughly the square root of the machine epsilon
A = [
1   1   1
eps  0   0
0   eps  0
0   0   eps
]

```

- With the various routines you implemented for Chapter 4 and the current chapter compute

```

[ Q_CGS, R_CGS ] = CGS_unb_var1( A )
[ Q_MGS, R_MGS ] = MGS_unb_var1( A )
[ A_HQR, t_HQR ] = HQR_unb_var1( A )
Q_HQR = FormQ_unb_var1( A_HQR, t_HQR )

```

- Finally, check whether the columns of the various computed matrices Q are mutually orthonormal:

```

Q_CGS' * Q_CGS
Q_MGS' * Q_MGS
Q_HQR' * Q_HQR

```

What do you notice? When we discuss numerical stability of algorithms we will gain insight into why HQR produces high quality mutually orthogonal columns.

- Check how well QR approximates A :

```

A = Q_CGS * triu( R_CGS )
A = Q_MGS * triu( R_MGS )
A = Q_HQR * triu( A_HQR( 1:3, 1:3 ) )

```

What you notice is that all approximate A well.

Later, we will see how the QR factorization can be used to solve $Ax = b$ and linear least-squares problems. At that time we will examine how accurate the solution is depending on which method for QR factorization is used to compute Q and R .

Answer:

```

>> Q_CGS' * Q_CGS

ans =
1.000000000000000 -0.000000007071068 -0.000000007071068
-0.000000007071068 1.000000000000000 0.500000000000000
-0.000000007071068 0.500000000000000 1.000000000000000

```

```

>> Q_MGS' * Q_MGS
ans =
1.000000000000000 -0.000000007071068 -0.000000004082483
-0.000000007071068 1.000000000000000 0.000000000000000
-0.000000004082483 0.000000000000000 1.000000000000000

>> Q_HQR' * Q_HQR
ans =
1.000000000000000 0 0
0 1.000000000000000 0
0 0 1.000000000000000

```

Clearly, the orthogonality of the matrix Q computed via Householder QR factorization is much more accurate.

[BACK TO TEXT](#)

Homework 6.21 Consider the matrix $\begin{pmatrix} \overline{A} \\ \overline{B} \end{pmatrix}$ where A has linearly independent columns. Let

- $A = Q_A R_A$ be the QR factorization of A .
- $\begin{pmatrix} R_A \\ \overline{B} \end{pmatrix} = Q_B R_B$ be the QR factorization of $\begin{pmatrix} R_A \\ \overline{B} \end{pmatrix}$.
- $\begin{pmatrix} A \\ \overline{B} \end{pmatrix} = QR$ be the QR factorization of $\begin{pmatrix} A \\ \overline{B} \end{pmatrix}$.

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

Answer:

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

Also, $\begin{pmatrix} A \\ \overline{B} \end{pmatrix} = QR$. By the uniqueness of the QR factorization, $Q = \left(\begin{array}{c|c} Q_A & 0 \\ \hline 0 & I \end{array} \right) Q_B$ and $R = R_B$.

[BACK TO TEXT](#)

Homework 6.22 Consider the matrix $\begin{pmatrix} R \\ \hline \hline B \end{pmatrix}$ where R is an upper triangular matrix. Propose a modification of HQR_UNB_VAR1 that overwrites R and B with Householder vectors and updated matrix R . Importantly, the algorithm should take advantage of the zeros in R (in other words, it should avoid computing with the zeroes below its diagonal). An outline for the algorithm is given in Figure 6.15.

Answer: See Figure 6.6.

 [BACK TO TEXT](#)

Homework 6.23 Implement the algorithm from Homework 6.22.

 [BACK TO TEXT](#)

Algorithm: $[R, B, t] := \text{HQR_UPDATE_UNB_VAR1}(R, B, t)$

Partition $R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array} \right), B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array} \right), t \rightarrow \left(\begin{array}{c} t_T \\ t_B \end{array} \right)$

where R_{TL} is 0×0 , B_L has 0 columns, t_T has 0 rows

while $m(R_{TL}) < m(R)$ **do**

Repartition

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

$$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right), \left(\begin{array}{c} t_T \\ t_B \end{array} \right) \rightarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right)$$

where ρ_{11} is 1×1 , b_1 has 1 column, τ_1 has 1 row

$$\left[\left(\begin{array}{c} \rho_{11} \\ b_1 \end{array} \right), \tau_1 \right] := \text{Housev} \left(\begin{array}{c} \rho_{11} \\ b_1 \end{array} \right)$$

Update $\left(\begin{array}{c} r_{12}^T \\ B_2 \end{array} \right) := \left(I - \frac{1}{\tau_1} \left(\begin{array}{c} 1 \\ b_1 \end{array} \right) \left(\begin{array}{c|c} 1 & b_1^H \end{array} \right) \right) \left(\begin{array}{c} r_{12}^T \\ B_2 \end{array} \right)$

via the steps

- $w_{12}^T := (r_{12}^T + b_1^H B_2) / \tau_1$
- $\left(\begin{array}{c} r_{12}^T \\ B_2 \end{array} \right) := \left(\begin{array}{c} r_{12}^T - w_{12}^T \\ B_2 - b_2 w_{12}^T \end{array} \right)$

Continue with

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

$$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right), \left(\begin{array}{c} t_T \\ t_B \end{array} \right) \leftarrow \left(\begin{array}{c} t_0 \\ \tau_1 \\ t_2 \end{array} \right)$$

endwhile

Figure 6.6: Algorithm that compute the QR factorization of a triangular matrix, R , appended with a matrix B . Upon completion, the Householder vectors are stored in R and the scalars associated with the

Chapter 8. Notes on Solving Linear Least-squares Problems (Answers)

Homework 8.2 Let $A \in \mathbb{C}^{m \times n}$ with $m < n$ have linearly independent rows. Show that there exist a lower triangular matrix $L_L \in \mathbb{C}^{m \times m}$ and a matrix $Q_T \in \mathbb{C}^{m \times n}$ with orthonormal rows such that $A = L_L Q_T$, noting that L_L does not have any zeroes on the diagonal. Letting $L = \left(\begin{array}{c|c} L_L & 0 \end{array} \right)$ be $\mathbb{C}^{m \times n}$ and unitary $Q = \left(\begin{array}{c|c} Q_T \\ \hline Q_B \end{array} \right)$, reason that $A = LQ$.

Don't overthink the problem: use results you have seen before.

Answer: We know that $A^H \in \mathbb{C}^{n \times m}$ with linearly independent columns (and $n \leq m$). Hence there exists a unitary matrix $\check{Q} = \text{FlaOneByTwo}Q_L Q_R$ and $R = \left(\begin{array}{c|c} R_T \\ \hline 0 \end{array} \right)$ with upper triangular matrix R_T that has no zeroes on its diagonal such that $A^H = \check{Q}_L R_T$. It is easy to see that the desired Q_T equals $Q_T = \check{Q}_L^H$ and the desired L_L equals R_T^H .

[BACK TO TEXT](#)

Homework 8.3 Let $A \in \mathbb{C}^{m \times n}$ with $m < n$ have linearly independent rows. Consider

$$\|Ax - y\|_2 = \min_z \|Az - y\|_2.$$

Use the fact that $A = L_L Q_T$, where $L_L \in \mathbb{C}^{m \times m}$ is lower triangular and Q_T has orthonormal rows, to argue that any vector of the form $Q_T^H L_L^{-1} y + Q_B^H w_B$ (where w_B is any vector in \mathbb{C}^{n-m}) is a solution to the LLS problem. Here $Q = \left(\begin{array}{c|c} Q_T \\ \hline Q_B \end{array} \right)$.

Answer:

$$\begin{aligned} \min_z \|Az - y\|_2 &= \min_z \|L_Q z - y\|_2 \\ &= \min_{\substack{w \\ z = Q^H w}} \|Lw - y\|_2 \\ &= \min_{\substack{w \\ z = Q^H w}} \left\| \left(\begin{array}{c|c} L_L & 0 \end{array} \right) \left(\begin{array}{c} w_T \\ \hline w_B \end{array} \right) - y \right\|_2 \\ &= \min_{\substack{w \\ z = Q^H w}} \|L_L w_T - y\|_2. \end{aligned}$$

Hence $w_T = L_L^{-1} y$ minimizes. But then

$$z = Q^H w = \left(\begin{array}{c|c} Q_T^H & Q_B^H \end{array} \right) \left(\begin{array}{c} w_T \\ \hline w_B \end{array} \right) = Q_T^H w_T + Q_B w_B.$$

describes all solutions to the LLS problem.

 [BACK TO TEXT](#)

Homework 8.4 Continuing Exercise 8.2, use Figure 8.1 to give a Classical Gram-Schmidt inspired algorithm for computing L_L and Q_T . (The best way to check you got the algorithm right is to implement it!)

Answer:

 [BACK TO TEXT](#)

Homework 8.5 Continuing Exercise 8.2, use Figure 8.2 to give a Householder QR factorization inspired algorithm for computing L and Q , leaving L in the lower triangular part of A and Q stored as Householder vectors above the diagonal of A . (The best way to check you got the algorithm right is to implement it!)

Answer:

 [BACK TO TEXT](#)

Chapter 8. Notes on the Condition of a Problem (Answers)

Homework 9.1 Show that, if A is a nonsingular matrix, for a consistent matrix norm, $\kappa(A) \geq 1$.

Answer: Hmm, I need to go and check what the exact definition of a consistent matrix norm is here... What I mean is a matrix norm induced by a vector norm. The reason is that then $\|I\| = 1$.

Let $\|\cdot\|$ be the norm that is used to define $\kappa(A) = \|AA^{-1}\|$. Then

$$1 = \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = \kappa(A).$$

 [BACK TO TEXT](#)

Homework 9.2 If A has linearly independent columns, show that $\|(A^H A)^{-1} A^H\|_2 = 1/\sigma_{n-1}$, where σ_{n-1} equals the smallest singular value of A . Hint: Use the SVD of A .

Answer: Let $A = U\Sigma V^H$ be the reduced SVD of A . Then

$$\begin{aligned} \|(A^H A)^{-1} A^H\|_2 &= \|((U\Sigma V^H)^H U\Sigma V^H)^{-1} (U\Sigma V^H)^H\|_2 \\ &= \|(V\Sigma U^H U\Sigma V^H)^{-1} V\Sigma U^H\|_2 \\ &= \|(V\Sigma^{-1} \Sigma^{-1} V^H) V\Sigma U^H\|_2 \\ &= \|V\Sigma^{-1} U^H\|_2 \\ &= \|\Sigma^{-1}\|_2 \\ &= 1/\sigma_{n-1} \end{aligned}$$

(since the two norm of a diagonal matrix equals its largest diagonal element in absolute value).

 [BACK TO TEXT](#)

Homework 9.3 Let A have linearly independent columns. Show that $\kappa_2(A^H A) = \kappa_2(A)^2$.

Answer: Let $A = U\Sigma V^H$ be the reduced SVD of A . Then

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

 [BACK TO TEXT](#)

Homework 9.4 Let $A \in \mathbb{C}^{n \times n}$ have linearly independent columns.

- Show that $Ax = y$ if and only if $A^H A x = A^H y$.

Answer: Since A has linearly independent columns and is square, we know that A^{-1} and A^{-H} exist. If $Ax = y$, then multiplying both sides by A^H yields $A^H A x = A^H y$. If $A^H A x = A^H y$ then multiplying both sides by A^{-H} yields $Ax = y$.

- Reason that using the method of normal equations to solve $Ax = y$ has a condition number of $\kappa_2(A)^2$.

◀ BACK TO TEXT

Homework 9.5 Let $U \in \mathbb{C}^{n \times n}$ be unitary. Show that $\kappa_2(U) = 1$.

Answer: The SVD of U is given by $U = U\Sigma V^H$ where $\Sigma = I$ and $V = I$. Hence $\sigma_0 = \sigma_{n-1} = 1$ and $\kappa_2(U) = \sigma_0/\sigma_{n-1} = 1$.

◀ BACK TO TEXT

Homework 9.6 Characterize the set of all square matrices A with $\kappa_2(A) = 1$.

Answer: If $\kappa_2(A) = 1$ then $\sigma_0/\sigma_{n-1} = 1$ which means that $\sigma_0 = \sigma_{n-1}$ and hence $\sigma_0 = \sigma_1 = \dots = \sigma_{n-1}$. Hence the singular value decomposition of A must equal $A = U\Sigma V^H = \sigma_0 UV^H$. But UV^H is unitary since U and V^H are, and hence we conclude that A must be a nonzero multiple of a unitary matrix.

◀ BACK TO TEXT

Homework 9.7 Let $\|\cdot\|$ be a vector norm with corresponding induced matrix norm. If $A \in \mathbb{C}^{n \times n}$ is nonsingular and

$$\frac{\|\Delta A\|}{\|A\|} < \frac{1}{\kappa(A)}. \quad (9.1)$$

then $A + \Delta A$ is nonsingular.

Answer: Equation (9.1) can be rewritten as $\|\Delta A\| \|A^{-1}\| < 1$. We will prove that if $A + \Delta A$ is singular, then $\|\Delta A\| \|A^{-1}\| \geq 1$.

$$\begin{aligned}
 & A + \Delta A \text{ is singular} \\
 \Rightarrow & \quad < \text{equivalent condition} > \\
 & (A + \Delta A)y = 0 \text{ for some } y \neq 0 \\
 \Rightarrow & \quad < \text{linear algebra} > \\
 & y = -A^{-1}\Delta A y \\
 \Rightarrow & \quad < \text{take norms} > \\
 & \|y\| = \|A^{-1}\Delta A y\| \\
 \Rightarrow & \quad < \text{property of induced matrix norms} > \\
 & \|y\| \leq \|A^{-1}\| \|\Delta A\| \|y\| \\
 \Rightarrow & \quad < \|y\| > 0 > \\
 & 1 \leq \|A^{-1}\| \|\Delta A\|.
 \end{aligned}$$

[BACK TO TEXT](#)

Homework 9.8 Let $A \in \mathbb{C}^{n \times n}$ be nonsingular, $Ax = y$, and $(A + \Delta A)(x + \delta x) = y$. Show that (for induced norm)

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leq \kappa(A) \frac{\|\Delta A\|}{\|A\|}.$$

Answer: $(A + \Delta A)(x + \delta x) = y$ can be rewritten as $Ax + A\delta x + \Delta A(x + \delta x) = y$. Since $Ax = y$ we find that $\delta x = -A^{-1}\Delta A(x + \delta x)$ and hence

$$\|\delta x\| \leq \|A^{-1}\| \|\Delta A\| \|x + \delta x\|.$$

Thus

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leq \|A^{-1}\| \|\Delta A\| = \|A^{-1}\| \frac{\kappa(A)}{\|A\| \|A^{-1}\|} \|\Delta A\| = \kappa(A) \frac{\|\Delta A\|}{\|A\|}.$$

[BACK TO TEXT](#)

Homework 9.9 Let $A \in \mathbb{C}^{n \times n}$ be nonsingular, $\|\Delta A\|/\|A\| < 1/\kappa(A)$, $y \neq 0$, $Ax = y$, and $(A + \Delta A)(x + \delta x) = y$. Show that (for induced norm)

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A) \frac{\|\Delta A\|}{\|A\|}}{1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}}.$$

Answer:

[BACK TO TEXT](#)

Homework 9.10 Let $A \in \mathbb{C}^{n \times n}$ be nonsingular, $\|\Delta A\|/\|A\| < 1$, $y \neq 0$, $Ax = y$, and $(A + \Delta A)(x + \delta x) = y + \delta y$. Show that (for induced norm)

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A) \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\delta y\|}{\|y\|} \right)}{1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}}.$$

Answer:

$$(A + \Delta A)(x + \delta x) = y + \delta y$$

is equivalent to

$$Ax + \Delta Ax + A\delta x + \Delta A\delta x = y + \delta y.$$

Since $Ax = y$ we can simplify and rearrange this to yield

$$\delta x = A^{-1}(\delta y - \Delta Ax - \Delta A\delta x).$$

Taking the norm on both side yields

$$\begin{aligned}\|\delta x\| &= \|A^{-1}\| \|\delta y - \Delta A x - \Delta A \delta x\| \leq \|A^{-1}\| (\|\delta y\| + \|\Delta A\| \|x\| + \|\Delta A\| \|\delta x\|) \\ &= \kappa(A) \left(\frac{\|\delta y\|}{\|A\|} + \frac{\|\Delta A\| \|x\|}{\|A\|} \right) + \kappa(A) \frac{\|\Delta A\|}{\|A\|} \|\delta x\|\end{aligned}$$

Bringing terms that involve δx to the left and dividing by $\|x\|$ yields

$$\begin{aligned}(1 - \kappa(A) \frac{\|\Delta A\|}{\|A\|}) \frac{\|\delta x\|}{\|x\|} &\leq \kappa(A) \left(\frac{\|\delta y\|}{\|A\| \|x\|} + \frac{\|\Delta A\|}{\|A\|} \right) \\ &\leq \kappa(A) \left(\frac{\|\delta y\|}{\|y\|} + \frac{\|\Delta A\|}{\|A\|} \right)\end{aligned}$$

since $\|y\| = \|Ax\| \leq \|A\| \|x\|$. Rearranging this yields the desired result.

 BACK TO TEXT

Chapter 9. Notes on the Stability of an Algorithm (Answers)

Homework 10.3 Assume a floating point number system with $\beta = 2$ and a mantissa with t digits so that a typical positive number is written as $.d_0d_1\dots d_{t-1} \times 2^e$, with $d_i \in \{0, 1\}$.

- Write the number 1 as a floating point number.

Answer: $. \underbrace{10\cdots 0}_t \times 2^1$.
digits

- What is the largest positive real number \mathbf{u} (represented as a binary fraction) such that the floating point representation of $1 + \mathbf{u}$ equals the floating point representation of 1? (Assume rounded arithmetic.)

Answer: $. \underbrace{10\cdots 0}_t \times 2^1 + . \underbrace{00\cdots 0}_t 1 \times 2^1 = . \underbrace{10\cdots 0}_t 1 \times 2^1$ which rounds to $. \underbrace{10\cdots 0}_t \times 2^1 = 1$.
digits digits digits digits

It is not hard to see that any larger number rounds to $. \underbrace{10\cdots 01}_t \times 2^1 > 1$.
digits

- Show that $\mathbf{u} = \frac{1}{2}2^{1-t}$.

Answer: $. \underbrace{0\cdots 00}_t 1 \times 2^1 = .1 \times 2^{1-t} = \frac{1}{2} \times 2^{1-t}$.
digits

 [BACK TO TEXT](#)

Homework 10.10 Prove Lemma 10.9.

Answer: Let $C = AB$. Then the (i, j) entry in $|C|$ is given by

$$|\gamma_{i,j}| = \left| \sum_{p=0}^k \alpha_{i,p} \beta_{p,j} \right| \leq \sum_{p=0}^k |\alpha_{i,p} \beta_{p,j}| = \sum_{p=0}^k |\alpha_{i,p}| |\beta_{p,j}|$$

which equals the (i, j) entry of $|A||B|$. Thus $|AB| \leq |A||B|$.

 [BACK TO TEXT](#)

Homework 10.12 Prove Theorem 10.11.

Answer:

Show that if $|A| \leq |B|$ then $\|A\|_1 \leq \|B\|_1$:

Let

$$A = \left(\begin{array}{c|c|c} a_0 & \cdots & a_{n-1} \end{array} \right) \quad \text{and} \quad B = \left(\begin{array}{c|c|c} b_0 & \cdots & b_{n-1} \end{array} \right).$$

Then

$$\begin{aligned} \|A\|_1 &= \max_{0 \leq j < n} \|a_j\|_1 = \max_{0 \leq j < n} \left(\sum_{i=0}^{m-1} |\alpha_{i,j}| \right) \\ &= \left(\sum_{i=0}^{m-1} |\alpha_{i,k}| \right) \text{ where } k \text{ is the index that maximizes} \\ &\leq \left(\sum_{i=0}^{m-1} |\beta_{i,k}| \right) \text{ since } |A| \leq |B| \\ &\leq \max_{0 \leq j < n} \left(\sum_{i=0}^{m-1} |\beta_{i,j}| \right) = \max_{0 \leq j < n} \|b_j\|_1 = \|B\|_1. \end{aligned}$$

Show that if $|A| \leq |B|$ then $\|A\|_\infty \leq \|B\|_\infty$:

Note: $\|A\|_\infty = \|A^T\|_1$ and $\|B\|_\infty = \|B^T\|_1$. Also, if $|A| \leq |B|$ then, clearly, $|A^T| \leq |B^T|$. Hence $\|A\|_\infty = \|A^T\|_1 \leq \|B^T\|_1 = \|B\|_\infty$.

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 $\|A\|_F \leq \|B\|_F$.

 [BACK TO TEXT](#)

Homework 10.13 Repeat the above steps for the computation

$$\kappa := ((\chi_0 \psi_0 + \chi_1 \psi_1) + \chi_2 \psi_2),$$

computing in the indicated order.

Answer:

 [BACK TO TEXT](#)

Homework 10.15 Complete the proof of Lemma 10.14.

Answer: We merely need to fill in the details for Case 1 in the proof:

Case 1: $\prod_{i=0}^n (1 + \varepsilon_i)^{\pm 1} = (\prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1})(1 + \varepsilon_n)$. By the I.H. there exists a θ_n such that $(1 + \theta_n) = \prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1}$ and $|\theta_n| \leq n\mathbf{u}/(1 - n\mathbf{u})$. Then

$$\left(\prod_{i=0}^{n-1} (1 + \varepsilon_i)^{\pm 1} \right) (1 + \varepsilon_n) = (1 + \theta_n)(1 + \varepsilon_n) = 1 + \underbrace{\theta_n + \varepsilon_n + \theta_n \varepsilon_n}_{\theta_{n+1}},$$

which tells us how to pick θ_{n+1} . Now

$$\begin{aligned} |\theta_{n+1}| &= |\theta_n + \varepsilon_n + \theta_n \varepsilon_n| \leq |\theta_n| + |\varepsilon_n| + |\theta_n||\varepsilon_n| \leq \frac{n\mathbf{u}}{1 - n\mathbf{u}} + \mathbf{u} + \frac{n\mathbf{u}}{1 - n\mathbf{u}} \mathbf{u} \\ &= \frac{n\mathbf{u} + \mathbf{u}(1 - n\mathbf{u}) + n\mathbf{u}^2}{1 - n\mathbf{u}} = \frac{(n+1) + \mathbf{u}}{1 - n\mathbf{u}} \leq \frac{(n+1) + \mathbf{u}}{1 - (n+1)\mathbf{u}}. \end{aligned}$$

◀ BACK TO TEXT

Homework 10.18 Prove Lemma 10.17.

Answer:

$$\gamma_n = \frac{n\mathbf{u}}{1 - n\mathbf{u}} \leq \frac{(n+b)\mathbf{u}}{1 - n\mathbf{u}} \leq \frac{(n+b)\mathbf{u}}{1 - (n+b)\mathbf{u}} = \gamma_{n+b}.$$

$$\begin{aligned} \gamma_n + \gamma_b + \gamma_n \gamma_b &= \frac{n\mathbf{u}}{1 - n\mathbf{u}} + \frac{b\mathbf{u}}{1 - b\mathbf{u}} + \frac{n\mathbf{u}}{(1 - n\mathbf{u})(1 - b\mathbf{u})} \frac{b\mathbf{u}}{(1 - b\mathbf{u})} \\ &= \frac{n\mathbf{u}(1 - b\mathbf{u}) + (1 - n\mathbf{u})b\mathbf{u} + bn\mathbf{u}^2}{(1 - n\mathbf{u})(1 - b\mathbf{u})} \\ &= \frac{n\mathbf{u} - bn\mathbf{u}^2 + b\mathbf{u} - bn\mathbf{u}^2 + bn\mathbf{u}^2}{1 - (n+b)\mathbf{u} + bn\mathbf{u}^2} = \frac{(n+b)\mathbf{u} - bn\mathbf{u}^2}{1 - (n+b)\mathbf{u} + bn\mathbf{u}^2} \\ &\leq \frac{(n+b)\mathbf{u}}{1 - (n+b)\mathbf{u} + bn\mathbf{u}^2} \leq \frac{(n+b)\mathbf{u}}{1 - (n+b)\mathbf{u}} = \gamma_{n+b}. \end{aligned}$$

◀ BACK TO TEXT

Homework 10.19 Let $k \geq 0$ and assume that $|\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}$, with $\varepsilon_1 = 0$ if $k = 0$. Show that

$$\left(\begin{array}{c|c} I + \Sigma^{(k)} & 0 \\ \hline 0 & (1 + \varepsilon_1) \end{array} \right) (1 + \varepsilon_2) = (I + \Sigma^{(k+1)}).$$

Hint: reason the case where $k = 0$ separately from the case where $k > 0$.

Answer:

Case: $k = 0$. Then

$$\left(\begin{array}{c|c} I + \Sigma^{(k)} & 0 \\ \hline 0 & (1 + \varepsilon_1) \end{array} \right) (1 + \varepsilon_2) = (1 + 0)(1 + \varepsilon_2) = (1 + \varepsilon_2) = (1 + \theta_1) = (I + \Sigma^{(1)}).$$

Case: $k = 1$. Then

$$\begin{aligned} \left(\begin{array}{c|c} I + \Sigma^{(k)} & 0 \\ \hline 0 & (1 + \varepsilon_1) \end{array} \right) (1 + \varepsilon_2) &= \left(\begin{array}{c|c} 1 + \theta_1 & 0 \\ \hline 0 & (1 + \varepsilon_1) \end{array} \right) (1 + \varepsilon_2) \\ &= \left(\begin{array}{c|c} (1 + \theta_1)(1 + \varepsilon_2) & 0 \\ \hline 0 & (1 + \varepsilon_1)(1 + \varepsilon_2) \end{array} \right) \\ &= \left(\begin{array}{c|c} (1 + \theta_2) & 0 \\ \hline 0 & (1 + \theta_2) \end{array} \right) = (I + \Sigma^{(2)}). \end{aligned}$$

Case: $k > 1$. Notice that

$$\begin{aligned} (I + \Sigma^{(k)})(1 + \varepsilon_2) &= \text{diag}((0)(1 + \theta_k), (1 + \theta_k), (1 + \theta_{k-1}), \dots, (1 + \theta_2))(1 + \varepsilon_2) \\ &= \text{diag}((0)(1 + \theta_{k+1}), (1 + \theta_{k+1}), (1 + \theta_k), \dots, (1 + \theta_3)) \end{aligned}$$

Then

$$\begin{aligned} \left(\begin{array}{c|c} I + \Sigma^{(k)} & 0 \\ \hline 0 & (1 + \varepsilon_1) \end{array} \right) (1 + \varepsilon_2) &= \left(\begin{array}{c|c} (I + \Sigma^{(k)})(1 + \varepsilon_2) & 0 \\ \hline 0 & (1 + \varepsilon_1)(1 + \varepsilon_2) \end{array} \right) \\ &= \left(\begin{array}{c|c} (I + \Sigma^{(k)})(1 + \varepsilon_2) & 0 \\ \hline 0 & (1 + \theta_2) \end{array} \right) = (I + \Sigma^{(k+1)}). \end{aligned}$$

◀ BACK TO TEXT

Homework 10.23 Prove R1-B.

Answer: From Theorem 10.20 we know that

$$\check{\kappa} = x^T (I + \Sigma^{(n)}) y = (x + \frac{\Sigma^{(n)} x}{\delta x})^T y.$$

Then

$$\begin{aligned} |\delta x| &= |\Sigma^{(n)} x| = \left| \begin{pmatrix} \theta_n \chi_0 \\ \theta_n \chi_1 \\ \theta_{n-1} \chi_2 \\ \vdots \\ \theta_2 \chi_{n-1} \end{pmatrix} \right| = \left| \begin{pmatrix} |\theta_n \chi_0| \\ |\theta_n \chi_1| \\ |\theta_{n-1} \chi_2| \\ \vdots \\ |\theta_2 \chi_{n-1}| \end{pmatrix} \right| = \left| \begin{pmatrix} |\theta_n| |\chi_0| \\ |\theta_n| |\chi_1| \\ |\theta_{n-1}| |\chi_2| \\ \vdots \\ |\theta_2| |\chi_{n-1}| \end{pmatrix} \right| \\ &\leq \left| \begin{pmatrix} |\theta_n| |\chi_0| \\ |\theta_n| |\chi_1| \\ |\theta_n| |\chi_2| \\ \vdots \\ |\theta_n| |\chi_{n-1}| \end{pmatrix} \right| = |\theta_n| \left| \begin{pmatrix} |\chi_0| \\ |\chi_1| \\ |\chi_2| \\ \vdots \\ |\chi_{n-1}| \end{pmatrix} \right| \leq \gamma_n |x|. \end{aligned}$$

(Note: strictly speaking, one should probably treat the case $n = 1$ separately.)!.

◀ BACK TO TEXT

Homework 10.25 In the above theorem, could one instead prove the result

$$\check{y} = A(x + \delta x),$$

where δx is “small”?

Answer: The answer is “no”. The reason is that for each individual element of y

$$\check{\psi}_i = a_i^T(x + \delta x)$$

which would appear to support that

$$\begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} a_0^T(x + \delta x) \\ a_1^T(x + \delta x) \\ \vdots \\ a_{m-1}^T(x + \delta x) \end{pmatrix}.$$

However, the δx for each entry $\check{\psi}_i$ is different, meaning that we cannot factor out $x + \delta x$ to find that $\check{y} = A(x + \delta x)$.

◀ BACK TO TEXT

Homework 10.27 In the above theorem, could one instead prove the result

$$\check{C} = (A + \Delta A)(B + \Delta B),$$

where ΔA and ΔB are “small”?

Answer: The answer is “no” for reasons similar to why the answer is “no” for Exercise 10.25.

◀ BACK TO TEXT

Chapter 10. Notes on Performance (Answers)

No exercises to answer yet.

Chapter 11. Notes on Gaussian Elimination and LU Factorization (Answers)

Homework 12.6 Show that

$$\left(\begin{array}{c|cc|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array} \right)^{-1} = \left(\begin{array}{c|cc|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & l_{21} & I \end{array} \right)$$

Answer:

$$\left(\begin{array}{c|cc|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array} \right) \left(\begin{array}{c|cc|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & l_{21} & I \end{array} \right) = \left(\begin{array}{c|ccc} I_k & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & -l_{21} + l_{21} & I & 0 \end{array} \right) = \left(\begin{array}{c|cc|c} I_k & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & I \end{array} \right)$$

[◀ BACK TO TEXT](#)

Homework 12.7 Let $\tilde{L}_k = L_0 L_1 \dots L_k$. Assume that \tilde{L}_k has the form $\tilde{L}_{k-1} = \left(\begin{array}{c|cc|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & 0 & I \end{array} \right)$, where \tilde{L}_{00} is $k \times k$. Show that \tilde{L}_k is given by $\tilde{L}_k = \left(\begin{array}{c|cc|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & l_{21} & I \end{array} \right) \dots$ (Recall: $\hat{L}_k = \left(\begin{array}{c|cc|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & -l_{21} & I \end{array} \right)$.)

Answer:

$$\begin{aligned} \tilde{L}_k &= \tilde{L}_{k-1} L_k = \tilde{L}_{k-1} \hat{L}_k^{-1} \\ &= \left(\begin{array}{c|cc|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & 0 & I \end{array} \right) \cdot \left(\begin{array}{c|ccc} I & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & -l_{21} & I & 0 \end{array} \right)^{-1} = \left(\begin{array}{c|cc|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & 0 & I \end{array} \right) \cdot \left(\begin{array}{c|ccc} I & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & l_{21} & I & 0 \end{array} \right) \\ &= \left(\begin{array}{c|cc|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & l_{21} & I \end{array} \right). \end{aligned}$$

[◀ BACK TO TEXT](#)

```

function [ A_out ] = LU_unb_var5( A )

[ ATL, ATR, ...
ABL, ABR ] = FLA_Part_2x2( A, ...
0, 0, 'FLA_TL' );

while ( size( ATL, 1 ) < size( A, 1 ) )

[ A00, a01, A02, ...
a10t, alphai1, a12t, ...
A20, a21, A22 ] = FLA_Repart_2x2_to_3x3( ATL, ATR, ...
ABL, ABR, ...
1, 1, 'FLA_BR' );

%-----%
a21 = a21/ alphai1;
A22 = A22 - a21 * a12t;

%-----%
[ ATL, ATR, ...
ABL, ABR ] = FLA_Cont_with_3x3_to_2x2( A00, a01, A02, ...
a10t, alphai1, a12t, ...
A20, a21, A22, ...
'FLA_TL' );

end

A_out = [ ATL, ATR
ABL, ABR ];

return

```

Figure 12.7: Answer to Exercise 12.12.

Homework 12.12 Implement LU factorization with partial pivoting with the FLAME@lab API, in M-script.

Answer: See Figure 12.7.

 [BACK TO TEXT](#)

Homework 12.13 Derive an algorithm for solving $Ux = y$, overwriting y with the solution, that casts most computation in terms of DOT products. Hint: Partition

$$U \rightarrow \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right).$$

Call this Variant 1 and use Figure 12.6 to state the algorithm.

Algorithm: Solve $Uz = y$, overwriting y (Variant 1)

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right)$

where U_{BR} is 0×0 , y_B has 0 rows

while $m(U_{BR}) < m(U)$ **do**

Repartition

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

$$\left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \Psi_1 \\ \hline y_2 \end{array} \right)$$

$$\Psi_1 := \Psi_1 - u_{12}^T x_2$$

$$y_1 := \Psi_1 / v_{11}$$

Continue with

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

$$\left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline \Psi_1 \\ \hline y_2 \end{array} \right)$$

endwhile

Algorithm: Solve $Uz = y$, overwriting y (Variant 2)

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right)$

where U_{BR} is 0×0 , y_B has 0 rows

while $m(U_{BR}) < m(U)$ **do**

Repartition

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

$$\left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \Psi_1 \\ \hline y_2 \end{array} \right)$$

$$\Psi_1 := \chi_1 / v_{11}$$

$$y_0 := y_0 - \chi_1 u_{01}$$

Continue with

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

$$\left(\begin{array}{c} y_T \\ \hline y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \hline \Psi_1 \\ \hline y_2 \end{array} \right)$$

endwhile

Figure 6 (Answer): Algorithms for the solution of upper triangular system $Ux = y$ that overwrite y with x .

Answer:

Partition

$$\left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right) \left(\begin{array}{c} \chi_1 \\ x_2 \end{array} \right) = \left(\begin{array}{c} \psi_1 \\ y_2 \end{array} \right).$$

Multiplying this out yields

$$\left(\begin{array}{c} v_{11}\chi_1 + u_{12}^T x_2 \\ \hline U_{22}x_2 \end{array} \right) = \left(\begin{array}{c} \psi_1 \\ y_2 \end{array} \right).$$

So, if we assume that x_2 has already been computed and has overwritten y_2 , then χ_1 can be computed as

$$\chi_1 = (\psi_1 - u_{12}^T x_2) / v_{11}$$

which can then overwrite ψ_1 . The resulting algorithm is given in Figure 12.6 (Answer) (left).

BACK TO TEXT

Homework 12.14 Derive an algorithm for solving $Ux = y$, overwriting y with the solution, that casts most computation in terms of AXPY operations. Call this Variant 2 and use Figure 12.6 to state the algorithm.

Answer: Partition

$$\left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & v_{11} \end{array} \right) \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right) = \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right).$$

Multiplying this out yields

$$\left(\begin{array}{c} U_{00}x_0 + u_{01}\chi_1 \\ \hline v_{11}\chi_1 \end{array} \right) = \left(\begin{array}{c} y_0 \\ \psi_1 \end{array} \right).$$

So, $\chi_1 = \psi_1 / v_{11}$ after which x_0 can be computed by solving $U_{00}x_0 = y_0 - \chi_1 u_{01}$. The resulting algorithm is given in Figure 12.6 (Answer) (right).

BACK TO TEXT

Homework 12.15 If A is an $n \times n$ matrix, show that the cost of Variant 1 is approximately $\frac{2}{3}n^3$ flops.

Answer: During the k th iteration, L_{00} is $k \times k$, for $k = 0, \dots, n-1$. Then the (approximate) cost of the steps are given by

- Solve $L_{00}u_{01} = a_{01}$ for u_{01} , overwriting a_{01} with the result. Cost: 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: k^2 flops.
- Compute $v_{11} = \alpha_{11} - l_{10}^T u_{01}$, overwriting α_{11} with the result. Cost: $2k$ flops.

Thus, the total cost is given by

$$\sum_{k=0}^{n-1} (k^2 + k^2 + 2k) \approx 2 \sum_{k=0}^{n-1} k^2 \approx 2 \frac{1}{3} n^3 = \frac{2}{3} n^3.$$

BACK TO TEXT

Homework 12.16 Derive the up-looking variant for computing the LU factorization.

Answer: Consider the loop invariant:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c} L \setminus U_{TL} & U_{TR} \\ \hline \widehat{A}_{BL} & \widehat{A}_{BR} \end{array} \right)$$

$$\wedge \frac{L_{TL}U_{TL} = \widehat{A}_{TL}}{L_{BL}U_{TL} = \widehat{A}_{BL}} \quad \frac{L_{TL}U_{TR} = \widehat{A}_{TR}}{L_{BL}U_{TR} + L_{BR}U_{BR} = \widehat{A}_{BR}}$$

At the top of the loop, after repartitioning, A contains

$$\begin{array}{c|c|c} L \setminus U_{00} & u_{01} & U_{02} \\ \hline \widehat{a}_{10}^T & \widehat{\alpha}_{11} & \widehat{a}_{12}^T \\ \hline \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array}$$

while at the bottom it must contain

$$\begin{array}{c|c|c} L \setminus U_{00} & u_{01} & \widehat{A}_{02} \\ \hline l_{10}^T & v_{11} & u_{12}^T \\ \hline \widehat{A}_{20} & \widehat{a}_{21} & \widehat{A}_{22} \end{array}$$

where the entries in blue are to be computed. Now, considering $LU = \widehat{A}$ we notice that

$$\begin{array}{c|c|c} L_{00}U_{00} = \widehat{A}_{00} & L_{00}u_{01} = \widehat{a}_{01} & L_{00}U_{02} = \widehat{A}_{02} \\ \hline l_{10}^T U_{00} = \widehat{a}_{10}^T & l_{10}^T u_{01} + v_{11} = \widehat{\alpha}_{11} & l_{10}^T U_{02} + u_{12}^T = \widehat{a}_{12}^T \\ \hline L_{20}U_{00} = \widehat{A}_{20} & L_{20}u_{01} + v_{11}l_{21} = \widehat{a}_{21} & L_{20}U_{02} + l_{21}u_{12}^T + L_{22}U_{22} = \widehat{A}_{22} \end{array}$$

The equalities in yellow can be used to compute the desired parts of L and U :

- Solve $l_{10}^T U_{00} = \widehat{a}_{10}^T$ for l_{10}^T , overwriting \widehat{a}_{10}^T with the result.
- Compute $v_{11} = \widehat{\alpha}_{11} - l_{10}^T u_{01}$, overwriting $\widehat{\alpha}_{11}$ with the result.
- Compute $u_{12}^T := \widehat{a}_{12}^T - l_{10}^T U_{02}$, overwriting \widehat{a}_{12}^T with the result.

BACK TO TEXT

Homework 12.17 Implement all five LU factorization algorithms with the FLAME@lab API, in M-script.

BACK TO TEXT

Homework 12.18 Which of the five variants can be modified to incorporate partial pivoting?

Answer: Variants 2, 4, and 5.

BACK TO TEXT

Homework 12.23 Apply LU with partial pivoting to

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & & \cdots & 1 & 1 \\ -1 & -1 & & \cdots & -1 & 1 \end{pmatrix}.$$

Pivot only when necessary.

Answer: Notice that no pivoting is necessary: 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 yields:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 2 \\ 0 & 0 & 1 & \cdots & 0 & 4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & & \cdots & 1 & 4 \\ 0 & 0 & & \cdots & -1 & 4 \end{pmatrix}.$$

Eliminating the entries below the diagonal in the $(n - 1)$ st column yields:

$$\left(\begin{array}{cccccc} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 2 \\ 0 & 0 & 1 & \cdots & 0 & 4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 2^{n-2} & \\ 0 & 0 & \cdots & 0 & 2^{n-1} & \end{array} \right).$$

[BACK TO TEXT](#)

Homework 12.24 Let L be a unit lower triangular matrix partitioned as

$$L = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \end{array} \right).$$

Show that

$$L^{-1} = \left(\begin{array}{c|c} L_{00}^{-1} & 0 \\ \hline -l_{10}^T L_{00}^{-1} & 1 \end{array} \right).$$

Answer:

Answer 1:

$$\left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c|c} L_{00}^{-1} & 0 \\ \hline -l_{10}^T L_{00}^{-1} & 1 \end{array} \right) = \left(\begin{array}{c|c} L_{00} L_{00}^{-1} & 0 \\ \hline l_{10}^T L_{00}^{-1} - l_{10}^T L_{00}^{-1} & 1 \end{array} \right) = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & 1 \end{array} \right)$$

Answer 2: We know that the inverse of a unit lower triangular matrix is unit lower triangular. Let's call the inverse matrix X . Then $LX = I$ and hence

$$\left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & 1 \end{array} \right) \left(\begin{array}{c|c} X_{00} & 0 \\ \hline x_{10}^T & 1 \end{array} \right) = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & 1 \end{array} \right).$$

Now, this means that

- $L_{00}X_{00} = I$ and hence $X_{00} = L_{00}^{-1}$.
- $l_{10}^T X_{00} + x_{10}^T = 0$ and hence $x_{10}^T = -l_{10}^T L_{00}^{-1}$.

[BACK TO TEXT](#)

Algorithm: $[L] := \text{LINV_UNB_VAR1}(L)$

Partition $L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right)$

where L_{TL} is 0×0

while $m(L_{TL}) < m(L)$ **do**

Repartition

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right)$$

where λ_{11} is 1×1

$$l_{10}^T := l_{10}^T L_{00}$$

Continue with

$$\left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right)$$

endwhile

Figure 12.8: Algorithm for inverting a unit lower triangular matrix. The algorithm assumes that in previous iterations L_{00} has been overwritten with L_{00}^{-1} .

Homework 12.25 Use Homework 12.24 and Figure 12.12 to propose an algorithm for overwriting a unit lower triangular matrix L with its inverse.

Answer: See Figure 12.8

 [BACK TO TEXT](#)

Homework 12.26 Show that the cost of the algorithm in Homework 12.25 is, approximately, $\frac{1}{3}n^3$, where L is $n \times n$.

Answer: Assume that L_{00} is $k \times k$. Then the cost of the current iteration is, approximately, k^2 flops (the cost multiplying a lower triangular matrix times a vector). Thus, the total cost is, approximately

$$\sum_{k=0}^{n-1} k^2 \approx \int_0^n x^2 dx = \frac{1}{3}n^3.$$

Homework 12.27 Given $n \times n$ upper triangular matrix U and $n \times n$ unit lower triangular matrix L , propose an algorithm for computing Y where $UY = L$. Be sure to take advantage of zeros in U and L . The cost of the resulting algorithm should be, approximately, n^3 flops.

There is a way of, given that L and U have overwritten matrix A , overwriting this matrix with Y . Try to find that algorithm. It is not easy... [Answer:](#)

Variant 1 Partition

$$U \rightarrow \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right), L \rightarrow \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array} \right), \text{ and } Y \rightarrow \left(\begin{array}{c|c} \psi_{11} & y_{12}^T \\ \hline y_{21} & Y_{22} \end{array} \right).$$

Then, since $UY = L$,

$$\left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c} \psi_{11} & y_{12}^T \\ \hline y_{21} & Y_{22} \end{array} \right) = \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array} \right)$$

and hence

$$\left(\begin{array}{c|c} v_{11}\psi_{11} + u_{12}^T y_{21} & v_{11}y_{12}^T + u_{12}^T Y_{22} \\ \hline U_{22}y_{21} & U_{22}Y_{22} \end{array} \right) = \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array} \right)$$

This suggests the following steps. We also give the approximate cost of the step when U_{22} is $k \times k$.

- Assume that Y_{22} has already been computed.
- Solve $U_{22}y_{21} = l_{21}$.
Cost: k^2 flops.
- $y_{12}^T := -u_{12}^T Y_{22}/v_{22}$.
Cost: $2k^2$ flops.
- $\psi_{11} := (1 - u_{12}^T y_{21})/v_{22}$.
Cost: negligible.

The total cost is then, approximately,

$$\sum_{k=0}^{n-1} 3k^2 \approx 3 \int_0^n k^2 = n^3.$$

The algorithm is given in Figure 12.9.

Algorithm: $[Y] := \text{SOLVE_UY_EQ_L_UNB_VAR1}(U, L, Y)$

Partition $U \rightarrow \begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix}, L \rightarrow \begin{pmatrix} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{pmatrix}, Y \rightarrow \begin{pmatrix} Y_{TL} & Y_{TR} \\ \hline Y_{BL} & Y_{BR} \end{pmatrix}$

where U_{BR} , L_{BR} , and Y_{BR} are 0×0

while $m(U_{BR}) < m(U)$ **do**

Repartition

$$\begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{pmatrix}, \begin{pmatrix} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{pmatrix},$$

$$\begin{pmatrix} Y_{TL} & Y_{TR} \\ \hline Y_{BL} & Y_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} Y_{00} & y_{01} & Y_{02} \\ \hline y_{10}^T & \Psi_{11} & y_{12}^T \\ \hline Y_{20} & y_{21} & Y_{22} \end{pmatrix}$$

where v_{11} is 1×1 , λ_{11} is 1×1 , Ψ_{11} is 1×1

Solve $U_{22}y_{21} = l_{21}$

$$y_{12}^T := -u_{12}^T Y_{22} / v_{22}$$

$$\Psi_{11} := (1 - u_{12}^T y_{21}) / v_{22}$$

Continue with

$$\begin{pmatrix} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{pmatrix}, \begin{pmatrix} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{pmatrix},$$

$$\begin{pmatrix} Y_{TL} & Y_{TR} \\ \hline Y_{BL} & Y_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} Y_{00} & y_{01} & Y_{02} \\ \hline y_{10}^T & \Psi_{11} & y_{12}^T \\ \hline Y_{20} & y_{21} & Y_{22} \end{pmatrix}$$

endwhile

Figure 12.9: Algorithm for solving $UY = L$.

Chapter 12. Notes on Cholesky Factorization (Answers)

Homework 13.3 Let $B \in \mathbb{C}^{m \times n}$ have linearly independent columns. Prove that $A = B^H B$ is HPD.

Answer: 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 $(Bx)^H (Bx) > 0$.

[BACK TO TEXT](#)

Homework 13.4 Let $A \in \mathbb{C}^{m \times m}$ be HPD. Show that its diagonal elements are real and positive.

Answer: Let e_j be the j th unit basis vectors. Then $0 < e_j^H A e_j = \alpha_{j,j}$.

[BACK TO TEXT](#)

Homework 13.14 Implement the Cholesky factorization with M-script.

[BACK TO TEXT](#)

Homework 13.15 Consider $B \in \mathbb{C}^{m \times n}$ with linearly independent columns. Recall that B has a QR factorization, $B = QR$ where Q has orthonormal columns and R is an upper triangular matrix with positive diagonal elements. How are the Cholesky factorization of $B^H B$ and the QR factorization of B related?

Answer:

$$B^H B = (QR)^H QR = R^H \underbrace{Q^H Q}_{I} R = \underbrace{R^H}_{L} \underbrace{R}_{L^H}.$$

[BACK TO TEXT](#)

Homework 13.16 Let A be SPD and partition

$$A \rightarrow \left(\begin{array}{c|c} A_{00} & a_{10} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right)$$

(Hint: For this exercise, use techniques similar to those in Section 13.5.)

1. Show that A_{00} is SPD.

Answer: Assume that A is $n \times n$ so that A_{00} is $(n-1) \times (n-1)$. Let $x_0 \in \mathbb{R}^{n-1}$ be a nonzero vector. Then

$$x_0^T A_{00} x_0 = \left(\begin{array}{c} x_0 \\ 0 \end{array} \right)^T \left(\begin{array}{c|c} A_{00} & a_{10} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) \left(\begin{array}{c} x_0 \\ 0 \end{array} \right) > 0$$

since $\left(\begin{array}{c} x_0 \\ 0 \end{array} \right)^T$ is a nonzero vector and A is SPD.

2. Assuming that $A_{00} = L_{00}L_{00}^T$, where L_{00} is lower triangular and nonsingular, argue that the assignment $l_{10}^T := a_{10}^T L_{00}^{-T}$ is well-defined.

Answer: The computation is well-defined because L_{00}^{-1} exists and hence $l_{10}^T := a_{10}^T L_{00}^{-T}$ uniquely computes l_{10}^T .

3. Assuming that A_{00} is SPD, $A_{00} = L_{00}L_{00}^T$ where L_{00} is lower triangular and nonsingular, and $l_{10}^T = a_{10}^T L_{00}^{-T}$, show that $\alpha_{11} - l_{10}^T l_{10} > 0$ so that $\lambda_{11} := \sqrt{\alpha_{11} - l_{10}^T l_{10}}$ is well-defined.

Answer: We want to show that $\alpha_{11} - l_{10}^T l_{10} > 0$. To do so, we are going to construct a nonzero vector x so that $x^T A x = \alpha_{11} - l_{10}^T l_{10}$, at which point we can invoke the fact that A is SPD.

Rather than just giving the answer, we go through the steps that will give insight into the thought process leading up to the answer as well. Consider

$$\begin{aligned} \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right)^T \left(\begin{array}{c|c} A_{00} & a_{10} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right) &= \left(\begin{array}{c} x_0 \\ \chi_1 \end{array} \right)^T \left(\begin{array}{c} A_{00}x_0 + \chi_1 a_{10} \\ \hline a_{10}^T x_0 + \alpha_{11} \chi_1 \end{array} \right) \\ &= x_0^T A_{00} x_0 + \chi_1 a_{10} + \chi_1 a_{10}^T x_0 + \alpha_{11} \chi_1 \\ &= x_0^T A_{00} x_0 + \chi_1 x_0^T a_{10} + \chi_1 a_{10}^T x_0 + \alpha_{11} \chi_1^2. \end{aligned}$$

Since we are trying to get to $\alpha_{11} - l_{10}^T l_{10}$, perhaps we should pick $\chi_1 = 1$. Then

$$\left(\begin{array}{c} x_0 \\ 1 \end{array} \right)^T \left(\begin{array}{c|c} A_{00} & a_{10} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) \left(\begin{array}{c} x_0 \\ 1 \end{array} \right) = x_0^T A_{00} x_0 + x_0^T a_{10} + a_{10}^T x_0 + \alpha_{11}.$$

The question now becomes how to pick x_0 so that

$$x_0^T A_{00} x_0 + x_0^T a_{10} + a_{10}^T x_0 = -l_{10}^T l_{10}.$$

Let's try $x_0 = -L_{00}^{-1} l_{10}$ and recall that $l_{10}^T = a_{10}^T L_{00}^{-T}$ so that $L_{00} l_{10} = a_{10}$. Then

$$\begin{aligned} x_0^T A_{00} x_0 + x_0^T a_{10} + a_{10}^T x_0 &= x_0^T L_{00} L_{00}^T x_0 + x_0^T a_{10} + a_{10}^T x_0 \\ &= (-L_{00}^{-1} l_{10})^T L_{00} L_{00}^T (-L_{00}^{-1} l_{10}) + (-L_{00}^{-1} l_{10})^T (L_{00} l_{10}) + (L_{00} l_{10})^T (-L_{00}^{-1} l_{10}) \\ &= l_{10}^T L_{00}^{-T} L_{00} L_{00}^T L_{00}^{-1} l_{10} - l_{10}^T L_{00}^{-T} L_{00} l_{10} - l_{10}^T L_{00}^T L_{00}^{-1} l_{10} \\ &= -l_{10}^T l_{10}. \end{aligned}$$

We now put all these insights together:

$$\begin{aligned} 0 &< \left(\begin{array}{c} -L_{00}^{-1} l_{10} \\ 1 \end{array} \right)^T \left(\begin{array}{c|c} A_{00} & a_{10} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) \left(\begin{array}{c} -L_{00}^{-1} l_{10} \\ 1 \end{array} \right) \\ &= (-L_{00}^{-1} l_{10})^T L_{00} L_{00}^T (-L_{00}^{-1} l_{10}) + (-L_{00}^{-1} l_{10})^T (L_{00} l_{10}) + (L_{00} l_{10})^T (-L_{00}^{-1} l_{10}) + \alpha_{11} \\ &= \alpha_{11} - l_{10}^T l_{10}. \end{aligned}$$

Algorithm: $A := \text{CHOL_UNB}(A)$ Bordered algorithm

$$\text{Partition } A \rightarrow \left(\begin{array}{c|c} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{array} \right)$$

where A_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1

$(A_{00} = L_{00}L_{00}^T$ has already been computed and
 L_{00} has overwritten the lower triangular part of A_{00})

$$\begin{aligned} a_{10}^T &:= a_{10}^T L_{00}^{-T} && (\text{Solve } L_{00}l_{10} = a_{10}, \text{ overwriting } a_{10}^T \text{ with } l_{10}.) \\ \alpha_{11} &:= \alpha_{11} - a_{10}^T a_{10} \\ \alpha_{11} &:= \sqrt{\alpha_{11}} \end{aligned}$$

Continue with

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

endwhile

Figure 13.10: Answer for Homework 13.17.

4. Show that

$$\left(\begin{array}{c|c} A_{00} & a_{10} \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array} \right) \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array} \right)^T$$

Answer: This is just a matter of multiplying it all out.

 **BACK TO TEXT**

Homework 13.17 Use the results in the last exercise to give an alternative proof by induction of the Cholesky Factorization Theorem and to give an algorithm for computing it by filling in Figure 13.3. This algorithm is often referred to as the *bordered* Cholesky factorization algorithm.

Answer:

Proof by induction.

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 SPD 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.

Inductive step: Inductive Hypothesis (I.H.): Assume the result is true for SPD matrix $A \in \mathbb{C}^{(n-1) \times (n-1)}$.

We will show that it holds for $A \in \mathbb{C}^{n \times n}$. Let $A \in \mathbb{C}^{n \times n}$ be SPD. Partition A and L like

$$A = \left(\begin{array}{c|c} A_{00} & * \\ \hline a_{10}^T & \alpha_{11} \end{array} \right) \quad \text{and} \quad L = \left(\begin{array}{c|c} L_{00} & 0 \\ \hline l_{10}^T & \lambda_{11} \end{array} \right).$$

Assume that $A_{00} = L_{00}L_{00}^T$ is the Cholesky factorization of A_{00} . By the I.H., we know this exists since A_{00} is $(n-1) \times (n-1)$ and that L_{00} is nonsingular. By the previous homework, we then know that

- $l_{10}^T = a_{01}^T L_{00}^{-T}$ is well-defined,
- $\lambda_{11} = \sqrt{\alpha_{11} - l_{10}^T l_{10}}$ is well-defined, and
- $A = LL^T$.

Hence L is the desired Cholesky factor of A .

By the principle of mathematical induction, the theorem holds.

The algorithm is given in 13.10.

BACK TO TEXT

Homework 13.18 Show that the cost of the bordered algorithm is, approximately, $\frac{1}{3}n^3$ flops.

Answer: During the k th iteration, $k = 0, \dots, n-1$ assume that A_{00} is $k \times k$. Then

- $a_{10}^T = a_{10}^T L_{00}^{-T}$ (implemented as the lower triangular solve $L_{00}l_{10} = a_{10}$) requires, approximately, k^2 flops.
- The cost of update α_{11} can be ignored.

Thus, the total cost equals (approximately)

$$\sum_{k=0}^{n-1} k^2 \approx \int_0^n x^2 dx = \frac{1}{3}n^3.$$

BACK TO TEXT

Chapter 13. Notes on Eigenvalues and Eigenvectors (Answers)

Homework 14.3 Eigenvectors are not unique.

Answer: If $Ax = \lambda x$ for $x \neq 0$, then $A(\alpha x) = \alpha Ax = \alpha\lambda x = \lambda(\alpha x)$. Hence any (nonzero) scalar multiple of x is also an eigenvector. This demonstrates we care about the *direction* of an eigenvector rather than its length.

[BACK TO TEXT](#)

Homework 14.4 Let λ be an eigenvalue of A and let $E_\lambda(A) = \{x \in \mathbb{C}^m | Ax = \lambda x\}$ denote the set of all eigenvectors of A associated with λ (including the zero vector, which is not really considered an eigenvector). Show that this set is a (nontrivial) subspace of \mathbb{C}^m .

Answer:

- $0 \in E_\lambda(A)$: (since we explicitly include it).
- $x \in E_\lambda(A)$ implies $\alpha x \in E_\lambda(A)$: (by the last exercise).
- $x, y \in E_\lambda(A)$ implies $x + y \in E_\lambda(A)$:

$$A(x+y) = Ax + Ay = \lambda x + \lambda y = \lambda(x+y).$$

[BACK TO TEXT](#)

Homework 14.8 The eigenvalues of a diagonal matrix equal the values on its diagonal. The eigenvalues of a triangular matrix equal the values on its diagonal.

Answer: Since a diagonal matrix is a special case of a triangular matrix, it suffices to prove that the eigenvalues of a triangular matrix are the values on its diagonal.

If A is triangular, so is $A - \lambda I$. By definition, λ is an eigenvalue of A if and only if $A - \lambda I$ is singular. But a triangular matrix is singular if and only if it has a zero on its diagonal. The triangular matrix $A - \lambda I$ has a zero on its diagonal if and only if λ equals one of the diagonal elements of A .

[BACK TO TEXT](#)

Homework 14.19 Prove Lemma 14.18. Then generalize it to a result for block upper triangular matrices:

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

Answer:

Lemma 14.18 Let $A \in \mathbb{C}^{m \times m}$ be of form $A = \begin{pmatrix} A_{TL} & A_{TR} \\ 0 & A_{BR} \end{pmatrix}$. Assume that Q_{TL} and Q_{BR} are unitary “of appropriate size”. Then

$$A = \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right)^H \left(\begin{array}{c|c} Q_{TL}A_{TL}Q_{TL}^H & Q_{TL}A_{TR}Q_{BR}^H \\ \hline 0 & Q_{BR}A_{BR}Q_{BR}^H \end{array} \right) \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right).$$

Proof:

$$\begin{aligned} & \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right)^H \left(\begin{array}{c|c} Q_{TL}A_{TL}Q_{TL}^H & Q_{TL}A_{TR}Q_{BR}^H \\ \hline 0 & Q_{BR}A_{BR}Q_{BR}^H \end{array} \right) \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right) \\ &= \left(\begin{array}{c|c} Q_{TL}^H Q_{TL} A_{TL} Q_{TL}^H Q_{TL} & Q_{TL}^H Q_{TL} A_{TR} Q_{BR}^H Q_{BR} \\ \hline 0 & Q_{BR}^H Q_{BR} A_{BR} Q_{BR}^H Q_{BR} \end{array} \right) \\ &= \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right) = A. \end{aligned}$$

By simple extension $A = Q^H B Q$ where

$$Q = \left(\begin{array}{c|c|c|c} Q_{0,0} & 0 & \cdots & 0 \\ \hline 0 & Q_{1,1} & \cdots & 0 \\ \hline 0 & 0 & \ddots & \vdots \\ \hline 0 & 0 & \cdots & Q_{N-1,N-1} \end{array} \right)$$

and

$$B = \left(\begin{array}{c|c|c|c} Q_{0,0} A_{0,0} Q_{0,0}^H & Q_{0,0} A_{0,1} Q_{1,1}^H & \cdots & Q_{0,0} A_{0,N-1} Q_{N-1,N-1}^H \\ \hline 0 & Q_{1,1} A_{1,1} Q_{1,1}^H & \cdots & Q_{1,1} A_{1,N-1} Q_{N-1,N-1}^H \\ \hline 0 & 0 & \ddots & \vdots \\ \hline 0 & 0 & \cdots & Q_{N-1,N-1} A_{N-1,N-1} Q_{N-1,N-1}^H \end{array} \right).$$

◀ BACK TO TEXT

Homework 14.21 Prove Corollary 14.20. Then generalize it to a result for block upper triangular matrices.

Answer:

Corollary 14.20 Let $A \in \mathbb{C}^{m \times m}$ be of for $A = \begin{pmatrix} A_{TL} & A_{TR} \\ 0 & A_{BR} \end{pmatrix}$. Then $\Lambda(A) = \Lambda(A_{TL}) \cup \Lambda(A_{BR})$.

Proof: This follows immediately from Lemma 14.18. Let $A_{TL} = Q_{TL}T_{TL}Q_{TL}^H$ and $A_{BR} = Q_{BR}T_{BR}Q_{BR}^H$ be the Shur decompositions of the diagonal blocks. Then $\Lambda(A_{TL})$ equals the diagonal elements of T_{TL} and $\Lambda(A_{BR})$ equals the diagonal elements of T_{BR} . Now, by Lemma 14.18 the Schur decomposition of A is given by

$$\begin{aligned} A &= \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right)^H \left(\begin{array}{c|c} Q_{TL}A_{TL}Q_{TL}^H & Q_{TL}A_{TR}Q_{BR}^H \\ \hline 0 & Q_{BR}A_{BR}Q_{BR}^H \end{array} \right) \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right) \\ &= \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right)^H \underbrace{\left(\begin{array}{c|c} T_{TL} & Q_{TL}A_{TR}Q_{BR}^H \\ \hline 0 & T_{BR} \end{array} \right)}_{T_A} \left(\begin{array}{c|c} Q_{TL} & 0 \\ \hline 0 & Q_{BR} \end{array} \right). \end{aligned}$$

Hence the diagonal elements of T_A (which is upper triangular) equal the elements of $\Lambda(A)$. The set of those elements is clearly $\Lambda(A_{TL}) \cup \Lambda(A_{BR})$.

The generalization is that if

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

where the blocks on the diagonal are square, then

$$\Lambda(A) = \Lambda(A_{0,0}) \cup \Lambda(A_{1,1}) \cup \cdots \cup \Lambda(A_{N-1,N-1}).$$

◀ BACK TO TEXT

Homework 14.24 Let A be Hermitian and λ and μ be distinct eigenvalues with eigenvectors x_λ and x_μ , respectively. Then $x_\lambda^H x_\mu = 0$. (In other words, the eigenvectors of a Hermitian matrix corresponding to distinct eigenvalues are orthogonal.)

Answer: Assume that $Ax_\lambda = \lambda x_\lambda$ and $Ax_\mu = \mu x_\mu$, for nonzero vectors x_λ and x_μ and $\lambda \neq \mu$. Then

$$x_\mu^H A x_\lambda = \lambda x_\mu^H x_\lambda$$

and

$$x_\lambda^H A x_\mu = \mu x_\lambda^H x_\mu.$$

Because A is Hermitian and λ is real

$$\mu x_\lambda^H x_\mu = x_\lambda^H A x_\mu = (x_\lambda^H A x_\lambda)^H = (\lambda x_\mu^H x_\lambda)^H = \lambda x_\lambda^H x_\mu.$$

Hence

$$\mu x_\lambda^H x_\mu = \lambda x_\lambda^H x_\mu.$$

If $x_\lambda^H x_\mu \neq 0$ then $\mu = \lambda$, which is a contradiction.

◀ BACK TO TEXT

Homework 14.25 Let $A \in \mathbb{C}^{m \times m}$ be a Hermitian matrix, $A = Q\Lambda Q^H$ its Spectral Decomposition, and $A = U\Sigma V^H$ its SVD. Relate Q , U , V , Λ , and Σ .

Answer: I am going to answer this by showing how to take the Spectral decomposition of A , and turn this into the SVD of A . Observations:

- We will assume all eigenvalues are nonzero. It should be pretty obvious how to fix the below if some of them are zero.
- Q is unitary and Λ is diagonal. Thus, $A = Q\Lambda Q^H$ is the SVD *except* that the diagonal elements of Λ are not necessarily nonnegative *and* they are not ordered from largest to smallest in magnitude.
- We can fix the fact that they are not nonnegative with the following observation:

$$\begin{aligned}
A &= Q\Lambda Q^H = Q \begin{pmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{n-1} \end{pmatrix} Q^H \\
&= Q \underbrace{\begin{pmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{n-1} \end{pmatrix}}_{I} \underbrace{\begin{pmatrix} \text{sign}(\lambda_0) & 0 & \cdots & 0 \\ 0 & \text{sign}(\lambda_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \text{sign}(\lambda_{n-1}) \end{pmatrix}}_{Q^H} \underbrace{\begin{pmatrix} \text{sign}(\lambda_0) & 0 & \cdots & 0 \\ 0 & \text{sign}(\lambda_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \text{sign}(\lambda_{n-1}) \end{pmatrix}}_{Q^H} \\
&= Q \underbrace{\begin{pmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{n-1} \end{pmatrix}}_{\tilde{\Lambda}} \underbrace{\begin{pmatrix} \text{sign}(\lambda_0) & 0 & \cdots & 0 \\ 0 & \text{sign}(\lambda_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \text{sign}(\lambda_{n-1}) \end{pmatrix}}_{\tilde{Q}^H} \underbrace{\begin{pmatrix} \text{sign}(\lambda_0) & 0 & \cdots & 0 \\ 0 & \text{sign}(\lambda_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \text{sign}(\lambda_{n-1}) \end{pmatrix}}_{Q^H} \\
&= Q \tilde{\Lambda} \tilde{Q}^H.
\end{aligned}$$

Now $\tilde{\Lambda}$ has nonnegative diagonal entries and \tilde{Q} is unitary since it is the product of two unitary matrices.

- Next, we fix the fact that the entries of $\tilde{\Lambda}$ are not ordered from largest to smallest. We do so by noting that there is a permutation matrix P such that $\Sigma = P\tilde{\Lambda}P^H$ equals the diagonal matrix with the values ordered from the largest to smallest. Then

$$A = Q\tilde{\Lambda}\tilde{Q}^H$$

$$\begin{aligned}
&= Q \underbrace{P^H P}_{I} \tilde{\Lambda} \underbrace{P^H P}_{I} \tilde{Q}^H \\
&= \underbrace{Q P^H}_{U} \underbrace{P \tilde{\Lambda} P^H}_{\Sigma} \underbrace{P \tilde{Q}^H}_{V^H} \\
&= U \Sigma V^H.
\end{aligned}$$

BACK TO TEXT

Homework 14.26 Let $A \in \mathbb{C}^{m \times m}$ and $A = U \Sigma V^H$ its SVD. Relate the Spectral decompositions of $A^H A$ and AA^H to U , V , and Σ .

Answer:

$$A^H A = (U \Sigma V^H)^H U \Sigma V^H = V \Sigma U^H U \Sigma V^H = V \Sigma^2 V^H.$$

Thus, the eigenvalues of $A^H A$ equal the square of the singular values of A .

$$AA^H = U \Sigma V^H (U \Sigma V^H)^H = U \Sigma V^H V \Sigma U^H = U \Sigma^2 U^H.$$

Thus, the eigenvalues of AA^H equal the square of the singular values of A .

BACK TO TEXT

Chapter 14. Notes on the Power Method and Related Methods (Answers)

Homework 15.4 Prove Lemma 15.3.

Answer: We need to show that

- Let $y \neq 0$. Show that $\|y\|_X > 0$: Let $z = Xy$. Then $z \neq 0$ since X is nonsingular. Hence

$$\|y\|_X = \|Xy\| = \|z\| > 0.$$

- Show that if $\alpha \in \mathbb{C}$ and $y \in \mathbb{C}^m$ then $\|\alpha y\|_X = |\alpha| \|y\|_X$:

$$\|\alpha y\|_X = \|X(\alpha y)\| = \|\alpha Xy\| = |\alpha| \|Xy\| = |\alpha| \|y\|_X.$$

- Show that if $x, y \in \mathbb{C}^m$ then $\|x + y\|_X \leq \|x\|_X + \|y\|_X$:

$$\|x + y\|_X = \|X(x + y)\| = \|Xx + Xy\| \leq \|Xx\| + \|Xy\| = \|x\|_X + \|y\|_X.$$

◀ BACK TO TEXT

Homework 15.7 Assume that

$$|\lambda_0| \geq |\lambda_1| \geq \cdots \geq |\lambda_{m-2}| > |\lambda_{m-1}| > 0.$$

Show that

$$\left| \frac{1}{\lambda_{m-1}} \right| > \left| \frac{1}{\lambda_{m-2}} \right| \geq \left| \frac{1}{\lambda_{m-3}} \right| \geq \cdots \geq \left| \frac{1}{\lambda_0} \right|.$$

Answer: This follows immediately from the fact that if $\alpha > 0$ and $\beta > 0$ then

- $\alpha > \beta$ implies that $1/\beta > 1/\alpha$ and
- $\alpha \geq \beta$ implies that $1/\beta \geq 1/\alpha$.

◀ BACK TO TEXT

Homework 15.9 Prove Lemma 15.8.

Answer: If $Ax = \lambda x$ then $(A - \mu I)x = Ax - \mu x = \lambda x - \mu x = (\lambda - \mu)x$.

◀ BACK TO TEXT

Homework 15.11 Prove Lemma 15.10.

Answer:

$$A - \mu I = X \Lambda X^{-1} - \mu X X^{-1} = X(\Lambda - \mu I)X^{-1}.$$

◀ BACK TO TEXT

Chapter 16. Notes on the QR Algorithm and other Dense Eigensolvers (Answers)

Homework 16.2 Prove that in Figure 16.3, $\widehat{V}^{(k)} = V^{(k)}$, and $\widehat{A}^{(k)} = A^{(k)}$, $k = 0, \dots$

Answer: This requires a proof by induction.

- Base case: $k = 0$. We need to show that $\widehat{V}^{(0)} = V^{(0)}$ and $\widehat{A}^{(0)} = A^{(0)}$. This is trivially true.
- Inductive step: Inductive Hypothesis (IH): $\widehat{V}^{(k)} = V^{(k)}$ and $\widehat{A}^{(k)} = A^{(k)}$.

We need to show that $\widehat{V}^{(k+1)} = V^{(k+1)}$ and $\widehat{A}^{(k+1)} = A^{(k+1)}$. Notice that

$$A\widehat{V}^{(k)} = \widehat{V}^{(k+1)}\widehat{R}^{(k+1)} \quad (\text{QR factorization})$$

so that

$$\widehat{A}^{(k)} = \widehat{V}^{(k)T}A\widehat{V}^{(k)} = \widehat{V}^{(k)T}\widehat{V}^{(k+1)}\widehat{R}^{(k+1)}$$

Since $\widehat{V}^{(k)T}\widehat{V}^{(k+1)}$ this means that $\widehat{A}^{(k)} = \widehat{V}^{(k)T}\widehat{V}^{(k+1)}\widehat{R}^{(k+1)}$ is a QR factorization of $A^{(k)}$.

Now, by the I.H.,

$$A^{(k)} = \widehat{A}^{(k)}\widehat{V}^{(k)T}\widehat{V}^{(k+1)}\widehat{R}^{(k+1)}$$

and in the algorithm on the right

$$A^{(k)} = Q^{(k+1)}R^{(k+1)}.$$

By the uniqueness of the QR factorization, this means that

$$Q^{(k+1)} = \widehat{V}^{(k)T}\widehat{V}^{(k+1)}.$$

But then

$$V^{(k+1)} = V^{(k)}Q^{(k+1)} = \widehat{V}^{(k)}Q^{(k+1)} = \underbrace{\widehat{V}^{(k)}\widehat{V}^{(k)T}}_I \widehat{V}^{(k+1)} = \widehat{V}^{(k+1)}.$$

Finally,

$$\begin{aligned} \widehat{A}^{(k+1)} &= \widehat{V}^{(k+1)T}A\widehat{V}^{(k+1)} = V^{(k+1)T}AV^{(k+1)} = (V^{(k)}Q^{(k+1)})^TA(V^{(k)}Q^{(k+1)}) \\ &= Q^{(k+1)T}V^{(k)}AV^{(k)}Q^{(k+1)} = \underbrace{Q^{(k+1)T}A^{(k)}}_{R^{(k+1)}}Q^{(k+1)} = R^{(k+1)}Q^{(k+1)} = A^{(k+1)} \end{aligned}$$

- By the Principle of Mathematical Induction the result holds for all k .

 BACK TO TEXT

Homework 16.3 Prove that in Figure 16.4, $\widehat{V}^{(k)} = V^{(k)}$, and $\widehat{A}^{(k)} = A^{(k)}$, $k = 0, \dots$

Answer: This requires a proof by induction.

- Base case: $k = 0$. We need to show that $\widehat{V}^{(0)} = V^{(0)}$ and $\widehat{A}^{(0)} = A^{(0)}$. This is trivially true.

- Inductive step: Inductive Hypothesis (IH): $\widehat{V}^{(k)} = V^{(k)}$ and $\widehat{A}^{(k)} = A^{(k)}$.

We need to show that $\widehat{V}^{(k+1)} = V^{(k+1)}$ and $\widehat{A}^{(k+1)} = A^{(k+1)}$.

The proof of the inductive step is a minor modification of the last proof *except* that we need to show that $\widehat{\mu}_k = \mu_k$:

$$\begin{aligned}\widehat{\mu}_k &= \widehat{v}_{n-1}^{(k)T} A \widehat{v}_{n-1}^{(k)} (V^{(k)} e_{n-1})^T A (V^{(k)} e_{n-1}) = e_{n-1} V^{(k)T} A V^{(k)} e_{n-1} \\ &= \underbrace{e_{n-1} \widehat{A}^{(k)} e_{n-1}}_{\text{by I.H.}} = e_{n-1} A^{(k)} e_{n-1} = \alpha_{n-1, n-1}^{(k)} = \mu_k\end{aligned}$$

- By the Principle of Mathematical Induction the result holds for all k .

BACK TO TEXT

Homework 16.5 Prove the above theorem.

Answer: I believe we already proved this in “Notes on Eigenvalues and Eigenvectors”.

BACK TO TEXT

Homework 16.9 Give all the details of the above proof.

Answer: Assume that q_1, \dots, q_k and the column indexed with $k - 1$ of B have been shown to be uniquely determined under the stated assumptions. We now show that then q_{k+1} and the column indexed by k of B are uniquely determined. (This is the inductive step in the proof.) Then

$$Aq_k = \beta_{0,k}q_0 + \beta_{1,k}q_1 + \dots + \beta_{k,k}q_k + \beta_{k+1,k}q_{k+1}.$$

We can determine $\beta_{0,k}$ through $\beta_{k,k}$ by observing that

$$q_j^T A q_k = \beta_{j,k}$$

for $j = 0, \dots, k$. Then

$$\beta_{k+1,k}q_{k+1} = Aq_k - (\beta_{0,k}q_0 + \beta_{1,k}q_1 + \dots + \beta_{k,k}q_k) = \tilde{q}_{k+1}.$$

Since it is assumed that $\beta_{k+1,k} > 0$, it can be determined as

$$\beta_{k+1,k} = \|\tilde{q}_{k+1}\|_2$$

and then

$$q_{k+1} = \tilde{q}_{k+1}/\beta_{k+1,k}.$$

This way, the columns of Q and B can be determined, one-by-one.

BACK TO TEXT

Homework 16.11 In the above discussion, show that $\alpha_{11}^2 + 2\alpha_{31}^2 + \alpha_{33}^2 = \hat{\alpha}_{11}^2 + \hat{\alpha}_{33}^2$.

Answer: If $\hat{A}_{31} = J_{31}A_{31}J_{31}^T$ then $\|\hat{A}_{31}\|_F^2 = \|A_{31}\|_F^2$ because multiplying on the left and/or the right by a unitary matrix preserves the Frobenius norm of a matrix. Hence

$$\left\| \begin{pmatrix} \alpha_{11} & \alpha_{31} \\ \alpha_{31} & \alpha_{33} \end{pmatrix} \right\|_F^2 = \left\| \begin{pmatrix} \hat{\alpha}_{11} & 0 \\ 0 & \hat{\alpha}_{33} \end{pmatrix} \right\|_F^2.$$

But

$$\left\| \begin{pmatrix} \alpha_{11} & \alpha_{31} \\ \alpha_{31} & \alpha_{33} \end{pmatrix} \right\|_F^2 = \alpha_{11}^2 + 2\alpha_{31}^2 + \alpha_{33}^2$$

and

$$\left\| \begin{pmatrix} \hat{\alpha}_{11} & 0 \\ 0 & \hat{\alpha}_{33} \end{pmatrix} \right\|_F^2 = \hat{\alpha}_{11}^2 + \hat{\alpha}_{33}^2,$$

which proves the result.

[BACK TO TEXT](#)

Homework 16.14 Give the approximate total cost for reducing a nonsymmetric $A \in \mathbb{R}^{n \times n}$ to upper-Hessenberg form.

Answer: To prove this, assume that in the current iteration of the algorithm A_{00} is $k \times k$. The update in the current iteration is then

$[u_{21}, \tau_1, a_{21}] := \text{HouseV}(a_{21})$	lower order cost, ignore
$y_{01} := A_{02}u_{21}$	$2k(n - k - 1)$ flops since A_{02} is $k \times (n - k - 1)$.
$A_{02} := A_{02} - \frac{1}{\tau}y_{01}u_{21}^T$	$2k(n - k - 1)$ flops since A_{02} is $k \times (n - k - 1)$.
$\psi_{11} := a_{12}^T u_{21}$	lower order cost, ignore
$a_{12}^T := a_{12}^T + \frac{\psi_{11}}{\tau}u_{21}^T$	lower order cost, ignore
$y_{21} := A_{22}u_{21}$	$2(n - k - 1)^2$ flops since A_{22} is $(n - k - 1) \times (n - k - 1)$.
$\beta := u_{21}^T y_{21} / 2$	lower order cost, ignore
$z_{21} := (y_{21} - \beta u_{21} / \tau_1) / \tau_1$	lower order cost, ignore
$w_{21} := (A_{22}^T u_{21} - \beta u_{21} / \tau) / \tau$	$2(n - k - 1)^2$ flops since A_{22} is $(n - k - 1) \times (n - k - 1)$ and the significant cost is in the matrix-vector multiply.
$A_{22} := A_{22} - (u_{21} w_{21}^T + z_{21} u_{21}^T)$	$2 \times 2k(n - k - 1)$ flops since A_{22} is $(n - k - 1) \times (n - k - 1)$, which is updated by two rank-1 updates.

Thus, the total cost in flops is, approximately,

$$\begin{aligned}
& \sum_{k=0}^{n-1} [2k(n-k-1) + 2k(n-k-1) + 2(n-k-1)^2 + 2(n-k-1)^2 + 2 \times 2(n-k-1)^2] \\
&= \sum_{k=0}^{n-1} [4k(n-k-1) + 4(n-k-1)^2] + \sum_{k=0}^{n-1} 4(n-k-1)^2 \\
&= \sum_{k=0}^{n-1} \underbrace{[4k(n-k-1) + 4(n-k-1)^2]}_{= 4(k+n-k-1)(n-k-1)} + 4(n-k-1)^2 \\
&\quad = 4(n-1)(n-k-1) \\
&\quad \approx 4n(n-k-1) \\
&\approx 4n \sum_{k=0}^{n-1} (n-k-1) + 4 \sum_{k=0}^{n-1} (n-k-1)^2 = 4n \sum_{j=0}^{n-1} j + 4 \sum_{j=0}^{n-1} j^2 \\
&\approx 4n \frac{n(n-1)}{2} + 4 \int_{j=0}^n j^2 dj \approx 2n^3 + \frac{4}{3}n^3 = \frac{10}{3}n^3.
\end{aligned}$$

Thus, the cost is (approximately)

$$\frac{10}{3}n^3 \text{ flops.}$$

 BACK TO TEXT

Algorithm: $A := \text{LDLT}(A)$

$$\text{Partition } A \rightarrow \left(\begin{array}{c|c} A_{TL} & * \\ \hline A_{BL} & A_{BR} \end{array} \right)$$

where A_{TL} is 0×0

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

Repartition

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

where α_{11} is 1×1

Option 1:	Option 2:	Option 3:
$l_{21} := a_{21}/\alpha_{11}$	$a_{21} := a_{21}/\alpha_{11}$	
$A_{22} := A_{22} - l_{21}a_{21}^T$ (updating lower triangle)	$A_{22} := A_{22} - \alpha_{11}a_{21}a_{21}^T$ (updating lower triangle)	$A_{22} := A_{22} - \frac{1}{\alpha_{11}}a_{21}a_{21}^T$ (updating lower triangle)
$a_{21} := l_{21}$		$a_{21} := a_{21}/\alpha_{11}$

Continue with

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

endwhile

Figure 17.11: Unblocked algorithm for computing $A \rightarrow LDL^T$, overwriting A .

Chapter 17. Notes on the Method of Relatively Robust Representations (Answers)

Homework 17.1 Modify the algorithm in Figure 17.1 so that it computes the LDL^T factorization. (Fill in Figure 17.3.)

Answer:

Three possible answers are given in Figure 17.11.

 **BACK TO TEXT**

Homework 17.2 Modify the algorithm in Figure 17.2 so that it computes the LDL^T factorization of a tridiagonal matrix. (Fill in Figure 17.4.) What is the approximate cost, in floating point operations, of

Algorithm: $A := \text{LDLT_TRI}(A)$

Partition $A \rightarrow \begin{pmatrix} A_{FF} & * & * \\ \alpha_{MFE}^T & \alpha_{MM} & * \\ 0 & \alpha_{LME}e_F & A_{LL} \end{pmatrix}$

where A_{LL} is 0×0

while $m(A_{FF}) < m(A)$ **do**

Repartition

$$\begin{pmatrix} A_{FF} & * & * \\ \alpha_{MFE}^T & \alpha_{MM} & * \\ 0 & \alpha_{LME}e_F & A_{LL} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & * & * & | \\ \alpha_{10}e_L^T & \alpha_{11} & * & * \\ 0 & \alpha_{21} & \alpha_{22} & * \\ 0 & 0 & \alpha_{32}e_F & A_{33} \end{pmatrix}$$

where α_{22} is a scalar

Option 1:

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

$$\alpha_{22} := \alpha_{22} - \lambda_{21}\alpha_{21}$$

(updating lower triangle)

$$\alpha_{21} := \lambda_{21}$$

Option 2:

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

$$\alpha_{22} := \alpha_{22} - \alpha_{11}\alpha_{21}^2$$

(updating lower triangle)

Option 3:

$$\alpha_{22} := \alpha_{22} - \frac{1}{\alpha_{11}}\alpha_{21}^2$$

(updating lower triangle)

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

Continue with

$$\begin{pmatrix} A_{FF} & * & * \\ \alpha_{MFE}^T & \alpha_{MM} & * \\ 0 & \alpha_{LME}e_F & A_{LL} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & * & * & | \\ \alpha_{10}e_L^T & \alpha_{11} & * & * \\ 0 & \alpha_{21} & \alpha_{22} & * \\ 0 & 0 & \alpha_{32}e_F & A_{33} \end{pmatrix}$$

endwhile

Figure 17.12: Algorithm for computing the the LDL^T factorization of a tridiagonal matrix.

computing the LDL^T factorization of a tridiagonal matrix? Count a divide, multiply, and add/subtract as a floating point operation each. Show how you came up with the algorithm, similar to how we derived the algorithm for the tridiagonal Cholesky factorization.

Answer:

Three possible algorithms are given in Figure 17.12.

The key insight is to recognize that, relative to the algorithm in Figure 17.11, $a_{21} = \begin{pmatrix} \alpha_{21} \\ 0 \end{pmatrix}$ so that,

for example, $a_{21} := a_{21}/\alpha_{11}$ becomes

$$\begin{pmatrix} \overline{\alpha_{21}} \\ 0 \end{pmatrix} := \begin{pmatrix} \overline{\alpha_{21}} \\ 0 \end{pmatrix} / \alpha_{11} = \begin{pmatrix} \overline{\alpha_{21}/\alpha_{11}} \\ 0 \end{pmatrix}.$$

Then, an update like $A_{22} := A_{22} - \alpha_{11}a_{21}a_{21}^T$ becomes

$$\begin{aligned} \begin{pmatrix} \alpha_{22} & \star \\ \hline \alpha_{32}e_F & A_{33} \end{pmatrix} &:= \begin{pmatrix} \alpha_{22} & \star \\ \hline \alpha_{32}e_F & A_{33} \end{pmatrix} - \alpha_{11} \begin{pmatrix} \overline{\alpha_{21}} \\ 0 \end{pmatrix} \begin{pmatrix} \overline{\alpha_{21}} \\ 0 \end{pmatrix}^T \\ &= \begin{pmatrix} \alpha_{22} - \alpha_{11}\alpha_{21}^2 & \star \\ \hline \alpha_{32}e_F & A_{33} \end{pmatrix}. \end{aligned}$$

BACK TO TEXT

Homework 17.3 Derive an algorithm that, given an indefinite matrix A , computes $A = UDU^T$. Overwrite only the upper triangular part of A . (Fill in Figure 17.5.) Show how you came up with the algorithm, similar to how we derived the algorithm for LDL^T .

Answer: Three possible algorithms are given in Figure 17.13.

Partition

$$A \rightarrow \begin{pmatrix} A_{00} & a_{01} \\ \hline a_{01}^T & \alpha_{11} \end{pmatrix} U \rightarrow \begin{pmatrix} U_{00} & u_{01} \\ \hline 0 & 1 \end{pmatrix}, \text{ and } D \rightarrow \begin{pmatrix} D_{00} & 0 \\ \hline 0 & \delta_1 \end{pmatrix}$$

Then

$$\begin{aligned} \begin{pmatrix} A_{00} & a_{01} \\ \hline a_{01}^T & \alpha_{11} \end{pmatrix} &= \begin{pmatrix} U_{00} & u_{01} \\ \hline 0 & 1 \end{pmatrix} \begin{pmatrix} D_{00} & 0 \\ \hline 0 & \delta_1 \end{pmatrix} \begin{pmatrix} U_{00} & u_{01} \\ \hline 0 & 1 \end{pmatrix}^T \\ &= \begin{pmatrix} U_{00}D_{00} & u_{01}\delta_1 \\ \hline 0 & \delta_1 \end{pmatrix} \begin{pmatrix} U_{00}^T & 0 \\ \hline u_{01}^T & 1 \end{pmatrix} \\ &= \begin{pmatrix} U_{00}D_{00}U_{00}^T + \delta_1 u_{01}u_{01}^T & u_{01}\delta_1 \\ \hline * & \delta_1 \end{pmatrix} \end{aligned}$$

This suggests the following algorithm for overwriting the strictly upper triangular part of A with the strictly upper triangular part of U and the diagonal of A with D :

- Partition $A \rightarrow \begin{pmatrix} A_{00} & a_{01} \\ \hline a_{01}^T & \alpha_{11} \end{pmatrix}$.
- $\alpha_{11} := \delta_{11} = \alpha_{11}$ (no-op).

Algorithm: $A := \text{UDU}^T(A)$

Partition $A \rightarrow \begin{pmatrix} A_{TL} & | & A_{TR} \\ \hline * & | & A_{BR} \end{pmatrix}$

where A_{BR} is 0×0

while $m(A_{BR}) < m(A)$ **do**

Repartition

$$\begin{pmatrix} A_{TL} & | & A_{TR} \\ \hline * & | & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & | & a_{01} & | & A_{02} \\ \hline * & | & \alpha_{11} & | & a_{12}^T \\ \hline * & | & * & | & A_{22} \end{pmatrix}$$

where α_{11} is 1×1

Option 1:

$$u_{01} := a_{01}/\alpha_{11}$$

$$A_{00} := A_{00} - u_{01}a_{01}^T$$

(updating upper triangle)

$$a_{01} := u_{01}$$

Option 2:

$$u_{01} := a_{01}/\alpha_{11}$$

$$A_{00} := A_{00} - \alpha_{11}a_{01}a_{01}^T$$

(updating upper triangle)

Option 3:

$$A_{00} := A_{00} - \frac{1}{\alpha_{11}}a_{01}a_{01}^T$$

(updating upper triangle)

$$a_{01} := a_{01}/\alpha_{11}$$

Continue with

$$\begin{pmatrix} A_{TL} & | & A_{TR} \\ \hline * & | & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & | & a_{01} & | & A_{02} \\ \hline * & | & \alpha_{11} & | & a_{12}^T \\ \hline * & | & * & | & A_{22} \end{pmatrix}$$

endwhile

Figure 17.13: Algorithm for computing the the UDU^T factorization of a tridiagonal matrix.

- Compute $u_{01} := u_{01}/\alpha_{11}$.
- Update $A_{22} := A_{22} - u_{01}a_{01}^T$ (updating only the upper triangular part).
- $a_{01} := u_{01}$.
- Continue with computing $A_{00} \rightarrow U_{00}D_{00}L_{00}^T$.

This algorithm will complete as long as $\delta_{11} \neq 0$,

BACK TO TEXT

Homework 17.4 Derive an algorithm that, given an indefinite tridiagonal matrix A , computes $A = UDU^T$. Overwrite only the upper triangular part of A . (Fill in Figure 17.6.) Show how you came up with the algorithm, similar to how we derived the algorithm for LDL^T .

Answer: Three possible algorithms are given in Figure 17.14.

The key insight is to recognize that, relative to the algorithm in Figure 17.13, $\alpha_{11} = \alpha_{22}$ and $a_{10} = \begin{pmatrix} 0 \\ \alpha_{12} \end{pmatrix}$ so that, for example, $a_{10} := a_{10}/\alpha_{11}$ becomes

$$\begin{pmatrix} 0 \\ \alpha_{12} \end{pmatrix} := \begin{pmatrix} 0 \\ \alpha_{12} \end{pmatrix} / \alpha_{22} = \begin{pmatrix} 0 \\ \alpha_{12}/\alpha_{22} \end{pmatrix}.$$

Then, an update like $A_{00} := A_{00} - \alpha_{11}a_{01}a_{01}^T$ becomes

$$\begin{aligned} \left(\begin{array}{c|c} A_{00} & \alpha_{01}e_L \\ \hline * & \alpha_{11} \end{array} \right) &:= \left(\begin{array}{c|c} A_{00} & \alpha_{01}e_L \\ \hline * & \alpha_{11} \end{array} \right) - \alpha_{22} \begin{pmatrix} 0 \\ \alpha_{12} \end{pmatrix} \begin{pmatrix} 0 \\ \alpha_{12} \end{pmatrix}^T \\ &= \left(\begin{array}{c|c} A_{00} & \alpha_{01}e_L \\ \hline * & \alpha_{11} - \alpha_{12}^2 \end{array} \right). \end{aligned}$$

BACK TO TEXT

Homework 17.5 Show that, provided ϕ_1 is chosen appropriately,

$$\begin{aligned} \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & \phi_1 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right)^T \\ = \left(\begin{array}{c|c|c} A_{00} & \alpha_{01}e_L & 0 \\ \hline \alpha_{01}e_L^T & \alpha_{11} & \alpha_{21}e_F^T \\ \hline 0 & \alpha_{21}e_F & A_{22} \end{array} \right). \end{aligned}$$

Algorithm: $A := \text{UDU}^T \text{.TRI}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c|c} A_{FF} & \alpha_{FME}e_L^T & 0 \\ \hline * & \alpha_{MM} & \alpha_{MLE}e_F^T \\ \hline * & * & A_{LL} \end{array} \right)$

where A_{FF} is 0×0

while $m(A_{LL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c|c} A_{FF} & \alpha_{FME}e_L & 0 \\ \hline * & \alpha_{MM} & \alpha_{MLE}e_F^T \\ \hline * & * & A_{LL} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c|c} A_{00} & \alpha_{01}e_L & 0 & 0 \\ \hline * & \alpha_{11} & \alpha_{12} & 0 \\ \hline * & * & \alpha_{22} & \alpha_{23}e_F^T \\ \hline * & * & * & A_{33} \end{array} \right)$$

where

Option 1:	Option 2:	Option 3:
$v_{12} := \alpha_{12}/\alpha_{22}$	$\alpha_{12} := \alpha_{12}/\alpha_{22}$	
$\alpha_{11} := \alpha_{11} - v_{12}\alpha_{12}^T$ (updating upper triangle)	$\alpha_{11} := \alpha_{11} - \alpha_{22}\alpha_{12}^2$ (updating upper triangle)	$\alpha_{11} := \alpha_{11} - \frac{1}{\alpha_{22}}\alpha_{12}^2$ (updating upper triangle)
$\alpha_{12} := v_{12}$		$\alpha_{12} := \alpha_{12}/\alpha_{22}$

Continue with

$$\left(\begin{array}{c|c|c} A_{FF} & \alpha_{FME}e_L & 0 \\ \hline * & \alpha_{MM} & \alpha_{MLE}e_F^T \\ \hline * & * & A_{LL} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c|c} A_{00} & \alpha_{01}e_L & 0 & 0 \\ \hline * & \alpha_{11} & \alpha_{12} & 0 \\ \hline * & * & \alpha_{22} & \alpha_{23}e_F^T \\ \hline * & * & * & A_{33} \end{array} \right)$$

endwhile

Figure 17.14: Algorithm for computing the the UDU^T factorization of a tridiagonal matrix.

(Hint: multiply out $A = LDL^T$ and $A = UEU^T$ with the partitioned matrices first. Then multiply out the above. Compare and match...) How should ϕ_1 be chosen? What is the cost of computing the twisted factorization given that you have already computed the LDL^T and UDU^T factorizations? A "Big O" estimate is sufficient. Be sure to take into account what $e_L^T D_{00} e_L$ and $e_F^T E_{22} e_F$ equal in your cost estimate.

Answer:

$$\begin{aligned}
& \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & 0 \\ \hline 0 & \lambda_{21}e_F & L_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & \delta_1 & 0 \\ \hline 0 & 0 & D_{22} \end{array} \right) \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & 0 \\ \hline 0 & \lambda_{21}e_F & L_{22} \end{array} \right)^T \\
= & \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & 0 \\ \hline 0 & \lambda_{21}e_F & L_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & \delta_1 & 0 \\ \hline 0 & 0 & D_{22} \end{array} \right) \left(\begin{array}{c|c|c} L_{00}^T & 0 & \lambda_{10}e_L \\ \hline 0 & 1 & \lambda_{21}e_F^T \\ \hline 0 & 0 & L_{22}^T \end{array} \right) \\
= & \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & 0 \\ \hline 0 & \lambda_{21}e_F & L_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00}L_{00}^T & \lambda_{10}D_{00}e_L & 0 \\ \hline 0 & \delta_1 & \lambda_{21}\delta_1e_F^T \\ \hline 0 & 0 & D_{22}L_{22}^T \end{array} \right) \\
= & \left(\begin{array}{c|c|c} L_{00}D_{00}L_{00}^T & \lambda_{10}L_{00}D_{00}e_L & 0 \\ \hline \lambda_{10}e_L^T D_{00}L_{00}^T & \lambda_{10}^2 e_L^T D_{00}e_L + \delta_1 & \lambda_{21}\delta_1e_F^T \\ \hline 0 & \lambda_{21}\delta_1e_F & \lambda_{21}^2 \delta_1e_F e_F^T + L_{22}D_{22}L_{22}^T \end{array} \right) = \left(\begin{array}{c|c|c} A_{00} & \alpha_{01}e_L & 0 \\ \hline \alpha_{01}e_L^T & \alpha_{11} & \alpha_{21}e_F^T \\ \hline 0 & \alpha_{21}e_F & A_{22} \end{array} \right). \quad (17.2)
\end{aligned}$$

and

$$\begin{aligned}
& \left(\begin{array}{c|c|c} U_{00} & v_{01}e_L & 0 \\ \hline 0 & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} E_{00} & 0 & 0 \\ \hline 0 & \varepsilon_1 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \left(\begin{array}{c|c|c} U_{00} & v_{01}e_L & 0 \\ \hline 0 & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right)^T \\
= & \left(\begin{array}{c|c|c} U_{00} & v_{01}e_L & 0 \\ \hline 0 & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} E_{00} & 0 & 0 \\ \hline 0 & \varepsilon_1 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \left(\begin{array}{c|c|c} U_{00}^T & 0 & 0 \\ \hline v_{01}e_L^T & 1 & 0 \\ \hline 0 & v_{21}e_F & U_{22}^T \end{array} \right) \\
= & \left(\begin{array}{c|c|c} U_{00} & v_{01}e_L & 0 \\ \hline 0 & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} E_{00}U_{00}^T & 0 & 0 \\ \hline \varepsilon_1 v_{01}e_L^T & \varepsilon_1 & 0 \\ \hline 0 & v_{21}E_{22}e_F & E_{22}U_{22}^T \end{array} \right) \\
= & \left(\begin{array}{c|c|c} U_{00}E_{00}U_{00}^T + v_{01}^2 \varepsilon_1 e_L e_L^T & v_{10}\varepsilon_1 e_L & 0 \\ \hline v_{01}\varepsilon_1 e_L^T & \varepsilon_1 + v_{01}^2 e_F^T E_{22}e_F & \varepsilon_1 v_{21}e_F^T E_{22}U_{22}^T \\ \hline 0 & v_{21}U_{22}^T E_{22}e_F & U_{22}E_{22}U_{22}^T \end{array} \right) = \left(\begin{array}{c|c|c} A_{00} & \alpha_{01}e_L & 0 \\ \hline \alpha_{01}e_L^T & \alpha_{11} & \alpha_{21}e_F^T \\ \hline 0 & \alpha_{21}e_F & A_{22} \end{array} \right). \quad (17.3)
\end{aligned}$$

Finally,

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & \phi_1 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right)^T$$

$$\begin{aligned}
&= \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & \phi_1 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \left(\begin{array}{c|c|c} L_{00}^T & \lambda_{10}e_L & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & v_{21}e_F & U_{22}^T \end{array} \right) \\
&= \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00}L_{00}^T & \lambda_{10}D_{00}e_L & 0 \\ \hline 0 & \phi_1 & 0 \\ \hline 0 & v_{21}E_{22}e_F & E_{22}U_{22}^T \end{array} \right) \\
&= \left(\begin{array}{c|c|c} L_{00}D_{00}L_{00}^T & \lambda_{10}L_{00}D_{00}e_L & 0 \\ \hline \lambda_{10}e_L^TD_{00}L_{00}^T & \lambda_{10}^2e_L^TD_{00}e_L + \phi_1 + v_{21}^2e_F^TE_{22}e_F & v_{21}e_F^TE_{22}U_{22}^T \\ \hline 0 & v_{21}U_{22}E_{22}e_F & U_{22}E_{22}U_{22}^T \end{array} \right) = \left(\begin{array}{c|c|c} A_{00} & \alpha_{01}e_L & 0 \\ \hline \alpha_{01}e_L^T & \alpha_{11} & \alpha_{21}e_F^T \\ \hline 0 & \alpha_{21}e_F & A_{22} \end{array} \right). \quad (17.4)
\end{aligned}$$

The equivalence of the submatrices highlighted in yellow in (17.2) and (??) justify the submatrices highlighted in yellow in (17.4).

The equivalence of the submatrices highlighted in grey in (17.2), (??), and (17.4) tell us that

$$\begin{aligned}
\alpha_{11} &= \lambda_{10}^2e_L^TD_{00}e_L + \delta_1 \\
\alpha_{11} &= v_{01}^2e_F^TE_{22}e_F + \varepsilon_1 \\
\alpha_{11} &= \lambda_{10}^2e_L^TD_{00}e_L + \phi_1 + v_{21}^2e_F^TE_{22}e_F.
\end{aligned}$$

or

$$\begin{aligned}
\alpha_{11} - \delta_1 &= \lambda_{10}^2e_L^TD_{00}e_L \\
\alpha_{11} - \varepsilon_1 &= v_{01}^2e_F^TE_{22}e_F
\end{aligned}$$

so that

$$\alpha_{11} = (\alpha_{11} - \delta_1) + \phi_1 + (\alpha_{11} - \varepsilon_1).$$

Solving this for ϕ_1 yields

$$\phi_1 = \delta_1 + \varepsilon_1 - \alpha_{11}.$$

Notice that, given the factorizations $A = LDL^T$ and $A = UEU^T$ the cost of computing the twisted factorization is $O(1)$.

 BACK TO TEXT

Homework 17.6 Compute x_0 , χ_1 , and x_2 so that

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \underbrace{\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right)^T}_{\text{Hint: } \begin{pmatrix} 0 \\ \hline 1 \\ \hline 0 \end{pmatrix}} \begin{pmatrix} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ \hline 0 \\ \hline 0 \end{pmatrix},$$

where $x = \begin{pmatrix} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{pmatrix}$ is not a zero vector. What is the cost of this computation, given that L_{00} and U_{22} have special structure?

Answer: Choose x_0, χ_1 , and x_2 so that

$$\left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right)^T \begin{pmatrix} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ \hline 1 \\ \hline 0 \end{pmatrix}$$

or, equivalently,

$$\left(\begin{array}{c|c|c} L_{00}^T & \lambda_{10}e_L & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & v_{21}e_F & U_{22}^T \end{array} \right) \begin{pmatrix} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ \hline 1 \\ \hline 0 \end{pmatrix}$$

We conclude that $\chi_1 = 1$ and

$$\begin{aligned} L_{00}^T x_0 &= -\lambda_{10}e_L \\ U_{22}^T x_2 &= -v_{21}e_F \end{aligned}$$

so that x_0 and x_2 can be computed via solves with a bidiagonal upper triangular matrix (L_{00}^T) and bidiagonal lower triangular matrix (U_{22}^T), respectively.

Then

$$\begin{aligned} & \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right)^T \begin{pmatrix} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{pmatrix} \\ &= \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \left(\begin{array}{c|c|c} D_{00} & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & E_{22} \end{array} \right) \begin{pmatrix} 0 \\ \hline 1 \\ \hline 0 \end{pmatrix} \\ &= \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline \lambda_{10}e_L^T & 1 & v_{21}e_F^T \\ \hline 0 & 0 & U_{22} \end{array} \right) \begin{pmatrix} 0 \\ \hline 0 \\ \hline 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \hline 0 \\ \hline 0 \end{pmatrix}, \end{aligned}$$

Now, consider solving $L_{00}^T x_0 = -\lambda_{10} e_L$:

$$\begin{pmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -\lambda_{10} \end{pmatrix}$$

A moment of reflection shows that if L_{00} is $k \times k$, then solving $L_{00}^T x_0 = -\lambda_{10} e_L$ requires $O(k)$ flops. Similarly, since then U_{22} is then $(n-k-1) \times (n-k-1)$, then solving $U_{22}^T x_2 = -v_{21} e_F$ requires $O(n-k-1)$ flops. Thus, computing x requires $O(n)$ flops.

Thus,

- Given tridiagonal \hat{A} computing all eigenvalues requires $O(n^2)$ computation. (We have not discussed this in detail...)
- Then, for each eigenvalue
 - Computing $A := \hat{A} - \lambda I$ requires $O(n)$ flops.
 - Factoring $A = LDL^T$ requires $O(n)$ flops.
 - Factoring $A = UEU^T$ requires $O(n)$ flops.
 - Computing *all* ϕ_1 so that the smallest can be chosen requires $O(n)$ flops.
 - Computing the eigenvector of \hat{A} associated with λ from the twisted factorization requires $O(n)$ computation.

Thus, computing all eigenvalues and eigenvectors of a tridiagonal matrix via this method requires $O(n^2)$ computation. This is *much* better than computing these via the tridiagonal QR algorithm.

 BACK TO TEXT

Chapter 18. Notes on Computing the SVD (Answers)

Homework 18.2 If $A = U\Sigma V^T$ is the SVD of A then $A^T A = V\Sigma^2 V^T$ is the Spectral Decomposition of $A^T A$.

 [BACK TO TEXT](#)

Homework 18.3 Homework 18.4 *Give the approximate total cost for reducing $A \in \mathbb{R}^{n \times n}$ to bidiagonal form.*

 [BACK TO TEXT](#)

Appendix **A**

How to Download

Videos associated with these notes can be viewed in one of three ways:

-  [YouTube](#) links to the video uploaded to YouTube.
-  [Download from UT Box](#) links to the video uploaded to UT-Austin's "UT Box" File Sharing Service".
-  [View After Local Download](#) links to the video downloaded to your own computer in directory Video within the same directory in which you stored this document. You can download videos by first linking on  [Download from UT Box](#). Alternatively, visit the [UT Box directory](#) in which the videos are stored and download some or all.

Appendix **B**

LAFF Routines (FLAME@lab)

Figure B summarizes the most important routines that are part of the `laff` FLAME@lab (MATLAB) library used in these materials.

Operation Abbrev.	Definition	Function	Approx. cost	
			flops	memops
Vector-vector operations				
Copy (COPY)	$y := x$	$y = \text{laff_copy}(\mathbf{x}, \mathbf{y})$	0	$2n$
Vector scaling (SCAL)	$x := \alpha x$	$\mathbf{x} = \text{laff_scal}(\alpha, \mathbf{x})$	n	$2n$
Vector scaling (SCAL)	$x := x/\alpha$	$\mathbf{x} = \text{laff_invscale}(\alpha, \mathbf{x})$	n	$2n$
Scaled addition (AXPY)	$y := \alpha x + y$	$\mathbf{y} = \text{laff_axpy}(\alpha, \mathbf{x}, \mathbf{y})$	$2n$	$3n$
Dot product (DOT)	$\alpha := \mathbf{x}^T \mathbf{y}$	$\alpha = \text{laff_dot}(\mathbf{x}, \mathbf{y})$	$2n$	$2n$
Dot product (DOTS)	$\alpha := \mathbf{x}^T \mathbf{y} + \alpha$	$\alpha = \text{laff_dots}(\mathbf{x}, \mathbf{y}, \alpha)$	$2n$	$2n$
Length (NORM2)	$\alpha := \ x\ _2$	$\alpha = \text{laff_norm2}(\mathbf{x})$	$2n$	n
Matrix-vector operations				
General matrix-vector multiplication (GEMV)	$y := \alpha A \mathbf{x} + \beta \mathbf{y}$	$\mathbf{y} = \text{laff_gemm}(\text{'No transpose'}, \alpha, \mathbf{A}, \mathbf{x}, \beta, \mathbf{y})$	$2mn$	mn
Rank-1 update (GER)	$A := \alpha \mathbf{x} \mathbf{x}^T + A$	$\mathbf{y} = \text{laff_gemm}(\text{'Transpose'}, \alpha, \mathbf{A}, \mathbf{x}, \beta, \mathbf{y})$	$2mn$	mn
Triangular matrix solve (TRSV)	$b := L^{-1} \mathbf{b}$, $b := U^{-1} \mathbf{b}$ $b := L^{-T} \mathbf{b}$, $b := U^{-T} \mathbf{b}$	example: $\mathbf{b} = \text{laff_trsv}(\text{'Upper triangular'}, \text{'No transpose'}, \text{'Nonunit diagonal'}, \mathbf{U}, \mathbf{b})$	$2mn$	mn
Triangular matrix-vector multiply (TRMV)	$x := L \mathbf{x}$, $x := U \mathbf{x}$ $x := L^T \mathbf{x}$, $x := U^T \mathbf{x}$	example: $\mathbf{x} = \text{laff_trmv}(\text{'Upper triangular'}, \text{'No transpose'}, \text{'Nonunit diagonal'}, \mathbf{U}, \mathbf{x})$	n^2	$n^2/2$
Matrix-matrix operations				
General matrix-matrix multiplication (GEMM)	$C := \alpha AB + \beta C$ $C := \alpha A^T B + \beta C$ $C := \alpha AB^T + \beta C$ $C := \alpha A^T B^T + \beta C$	example: $\mathbf{C} = \text{laff_gemm}(\text{'Transpose'}, \text{'No transpose'}, \alpha, \mathbf{A}, \mathbf{B}, \beta, \mathbf{C})$	$2mnk$	$2mn + mk + nk$
Triangular solve with MRHs (TRSM)	$B := \alpha L^{-1} B$ $B := \alpha U^{-T} B$ $B := \alpha B L^{-1}$ $B := \alpha B U^{-T}$	example: $\mathbf{B} = \text{laff_trsm}(\text{'Left'}, \text{'Lower triangular'}, \text{'No transpose'}, \text{'Nonunit diagonal'}, \alpha, \mathbf{U}, \mathbf{B})$	$m^2 n$ $m^2 n$ $m^2 n$ $m^2 n$	$m^2 + mn$ $m^2 + mn$ $m^2 + mn$ $m^2 + mn$

Bibliography

- [1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide (third ed.)*. SIAM, Philadelphia, PA, USA, 1999.
- [2] E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.
- [3] Paolo Bientinesi. *Mechanical Derivation and Systematic Analysis of Correct Linear Algebra Algorithms*. PhD thesis, Department of Computer Sciences, The University of Texas, 2006. Technical Report TR-06-46. September 2006.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [4] Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Soft.*, 31(1):1–26, March 2005.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [5] Paolo Bientinesi, Brian Gunter, and Robert A. van de Geijn. Families of algorithms related to the inversion of a symmetric positive definite matrix. *ACM Trans. Math. Softw.*, 35(1):3:1–3:22, July 2008.
- [6] Paolo Bientinesi, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. Representing linear algebra algorithms in code: The FLAME APIs. *ACM Trans. Math. Soft.*, 31(1):27–59, March 2005.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [7] Paolo Bientinesi and Robert A. van de Geijn. The science of deriving stability analyses. FLAME Working Note #33. Technical Report AICES-2008-2, Aachen Institute for Computational Engineering Sciences, RWTH Aachen, November 2008.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [8] Paolo Bientinesi and Robert A. van de Geijn. Goal-oriented and modular stability analysis. *SIAM J. Matrix Anal. & Appl.*, 32(1):286–308, 2011.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [9] Paolo Bientinesi and Robert A. van de Geijn. Goal-oriented and modular stability analysis. *SIAM J. Matrix Anal. Appl.*, 32(1):286–308, March 2011.

Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.

We suggest you read FLAME Working Note #33 for more details.

- [10] Christian Bischof and Charles Van Loan. The WY representation for products of Householder matrices. *SIAM J. Sci. Stat. Comput.*, 8(1):s2–s13, Jan. 1987.
- [11] Basic linear algebra subprograms technical forum standard. *International Journal of High Performance Applications and Supercomputing*, 16(1), 2002.
- [12] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.
- [13] I. S. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, Computer Science Division, University of California, Berkeley, California, May 1997. Available as UC Berkeley Technical Report No. UCB//CSD-97-971.
- [14] I. S. Dhillon. Reliable computation of the condition number of a tridiagonal matrix in $O(n)$ time. *SIAM J. Matrix Anal. Appl.*, 19(3):776–796, July 1998.
- [15] I. S. Dhillon and B. N. Parlett. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Lin. Alg. Appl.*, 387:1–28, August 2004.
- [16] Inderjit S. Dhillon, Beresford N. Parlett, and Christof Vömel. The design and implementation of the MRRR algorithm. *ACM Trans. Math. Soft.*, 32(4):533–560, December 2006.
- [17] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users’ Guide*. SIAM, Philadelphia, 1979.
- [18] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.
- [19] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.
- [20] Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. SIAM, Philadelphia, PA, 1991.
- [21] Jack J. Dongarra, Sven J. Hammarling, and Danny C. Sorensen. Block reduction of matrices to condensed forms for eigenvalue computations. *Journal of Computational and Applied Mathematics*, 27, 1989.
- [22] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.
- [23] Kazushige Goto and Robert van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Soft.*, 34(3):12:1–12:25, May 2008.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [24] John A. Gunnels. *A Systematic Approach to the Design and Analysis of Parallel Dense Linear Algebra Algorithms*. PhD thesis, Department of Computer Sciences, The University of Texas, December 2001.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.

- [25] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment. *ACM Trans. Math. Soft.*, 27(4):422–455, December 2001.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [26] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.
- [27] C. G. J. Jacobi. Über ein leichtes Verfahren, die in der Theorie der Säkular-störungen vorkommenden Gleichungen numerisch aufzulösen. *Crelle's Journal*, 30:51–94, 1846.
- [28] Thierry Joffrain, Tze Meng Low, Enrique S. Quintana-Ortí, Robert van de Geijn, and Field G. Van Zee. Accumulating Householder transformations, revisited. *ACM Trans. Math. Softw.*, 32(2):169–179, June 2006.
- [29] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5(3):308–323, Sept. 1979.
- [30] 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, 2014.
Download from <http://www.ulaff.net>.
- [31] Jack Poulson, Bryan Marker, Robert A. van de Geijn, Jeff R. Hammond, and Nichols A. Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Trans. Math. Softw.*, 39(2):13:1–13:24, February 2013.
- [32] C. Puglisi. Modification of the Householder method based on the compact WY representation. *SIAM J. Sci. Stat. Comput.*, 13:723–726, 1992.
- [33] Enrique S. Quintana, Gregorio Quintana, Xiaobai Sun, and Robert van de Geijn. A note on parallel matrix inversion. *SIAM J. Sci. Comput.*, 22(5):1762–1771, 2001.
- [34] Gregorio Quintana-Ortí and Robert van de Geijn. Improving the performance of reduction to Hessenberg form. *ACM Trans. Math. Softw.*, 32(2):180–194, June 2006.
- [35] Robert Schreiber and Charles Van Loan. A storage-efficient WY representation for products of Householder transformations. *SIAM J. Sci. Stat. Comput.*, 10(1):53–57, Jan. 1989.
- [36] G. W. Stewart. *Matrix Algorithms Volume 1: Basic Decompositions*. SIAM, Philadelphia, PA, USA, 1998.
- [37] G. W. Stewart. *Matrix Algorithms Volume II: Eigensystems*. SIAM, Philadelphia, PA, USA, 2001.
- [38] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM, 1997.
- [39] Robert van de Geijn and Kazushige Goto. *Encyclopedia of Parallel Computing*, chapter BLAS (Basic Linear Algebra Subprograms), pages Part 2, 157–164. Springer, 2011.
- [40] Robert A. van de Geijn and Enrique S. Quintana-Ortí. *The Science of Programming Matrix Computations*. www.lulu.com/contents/contents/1911788/, 2008.
- [41] Field G. Van Zee. libflame: *The Complete Reference*. www.lulu.com, 2012.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.

- [42] Field G. Van Zee, Ernie Chan, Robert van de Geijn, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí. The libflame library for dense matrix computations. *IEEE Computation in Science & Engineering*, 11(6):56–62, 2009.
- [43] Field G. Van Zee and Robert A. van de Geijn. BLIS: A framework for rapid instantiation of BLAS functionality. *ACM Trans. Math. Soft.*, 2015. To appear.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [44] Field G. Van Zee, Robert A. van de Geijn, and Gregorio Quintana-Ortí. Restructuring the tridiagonal and bidiagonal QR algorithms for performance. *ACM Trans. Math. Soft.*, 40(3):18:1–18:34, April 2014.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [45] Field G. Van Zee, Robert A. van de Geijn, Gregorio Quintana-Ortí, and G. Joseph Elizondo. Families of algorithms for reducing a matrix to condensed form. *ACM Trans. Math. Soft.*, 39(1), 2012.
Download from <http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html>.
- [46] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Stat. Comput.*, 9(1):152–163, 1988.
- [47] David S. Watkins. *Fundamentals of Matrix Computations, 3rd Edition*. Wiley, third edition, 2010.
- [48] Stephen J. Wright. A collection of problems for which Gaussian elimination with partial pivoting is unstable. *SIAM J. Sci. Comput.*, 14(1):231–238, 1993.

Index

- $\Gamma B30D \cdot \Gamma B30D_1$ -norm
 - vector, 40
- $\Gamma B30D \cdot \Gamma B30D_2$ -norm
 - vector, 38
- $\Gamma B30D \cdot \Gamma B30D_\infty$ -norm
 - vector, 40
- 1-norm
 - vector, 40
- 2-norm
 - vector, 38
- absolute value, 13, 38
- axpy, 18
 - cost, 18, 20
- blocked matrix-matrix multiplication, 32
- bordered Cholesky factorization algorithm, 265
- CGS, 88
- Cholesky factorization
 - bordered algorithm, 265
 - other algorithm, 265
- Classical Gram-Schmidt, 88
- code skeleton, 106
- complete pivoting
 - LU factorization, 228
- complex conjugate, 13
- complex scalar
 - absolute value, 13
- condition number, 82
 - estimation, 249–253
- conjugate, 13
 - complex, 13
 - matrix, 13
 - scalar, 13
 - vector, 13
- dot, 19
- dot product, 19
- ”dot” product, 20
- eigenpair
 - definition, 278
- eigenvalue
 - (, 277
 -), 283
 - definition, 278
- eigenvector
 - (, 277
 -), 283
 - definition, 278
- Euclidean length
 - see vector 2-norm, 38
- FLAME
 - API, 103–109
 - notation, 104
- FLAME API, 103–109
- FLAME notation, 21
- floating point operations, 17
- flop, 17
- Gauss transform, 216–218
- Gaussian elimination, 211–247
- gemm
 - algorithm
 - by columns, 28
 - by rows, 29
 - Variant 1, 28
 - Variant 2, 29
 - Variant 3, 31
 - via rank-1 updates, 31
 - gemm, 26
 - cost, 28

gemv
 cost, 21
 gemv, 20
 ger
 cost, 25
 ger, 23
 Gram-Schmidt
 Classical, 88
 cost, 99
 Modified, 93
 Gram-Schmidt orthogonalization
 implementation, 104
 Gram-Schmidt QR factorization, 85–100
 Hermitian transpose
 vector, 14
 Householder QR factorization, 111
 blocked, 127–138
 Householder transformation, 115–118
 Housev(\cdot), 118
 HQR(\cdot), 122
 infinity-norm
 vector, 40
 inner product, 19
 inverse
 Moore-Penrose generalized, 78
 pseudo, 78
 laff
 routines, 441–445
 LAFF Notes, xii
 laff operations, 444
 linear system solve, 229
 triangular, 229–232
 low-rank approximation, 78
 lower triangular solve, 229–231
 LU decomposition, 215
 LU factorization, 211–247
 complete pivoting, 228
 cost, 218–219
 existence, 215
 existence proof, 226–228
 partial pivoting, 219–226
 algorithm, 221
 LU factorization:derivation, 215–216
 MAC, 32

Matlab, 104
 matrix
 condition number, 82
 conjugate, 13
 inversion, 247–249
 low-rank approximation, 78
 norms(), 41
 norms), 45
 orthogonal, 53–83
 orthogormal, 64
 permutation, 219
 spectrum, 278
 transpose, 13–16
 unitary, 64
 matrix-matrix multiplication, 26
 blocked, 32
 cost, 28
 element-by-element, 26
 via matrix-vector multiplications, 27
 via rank-1 updates, 30
 via row-vector times matrix multiplications, 29
 matrix-matrix operation
 gemm, 26
 matrix-matrix multiplication, 26
 matrix-matrix operations, 444
 matrix-matrix product, 26
 matrix-vector multiplication, 20
 algorithm, 22
 by columns, 22
 by rows, 22
 cost, 21
 via axpy, 22
 via dot, 22
 matrix-vector operation, 20
 gemv, 20
 ger, 23
 matrix-vector multiplication, 20
 rank-1 update, 23
 matrix-vector operations, 444
 matrix-vector product, 20
 MGS, 93
 Modified Gram-Schmidt, 93
 Moore-Penrose generalized inverse, 78
 multiplication
 matrix-matrix, 26
 blocked, 32

element-by-element, 26
 via matrix-vector multiplications, 27
 via rank-1 updates, 30
 via row-vector times matrix multiplications,
 29
 matrix-vector, 20
 cost, 21
 multiply-accumulate, 32

norms
 matrix(), 41
 matrix), 45
 vector(), 38
 vector), 41

notation
 FLAME, 21

Octave, 104

operations
 laff, 444
 matrix-matrix, 444
 matrix-vector, 444

orthonormal basis, 85

outer product, 23

partial pivoting
 LU factorization, 219–226

permutation
 matrix, 219

preface, x

product
 dot, 19
 inner, 19
 matrix-matrix, 26
 matrix-vector, 20
 outer, 23

projection
 onto column space, 77

pseudo inverse, 78

QR factorization, 90
 Gram-Schmidt, 85–100
 Rank Revealing, 143

Rank Revealing QR factorization, 143

rank-1 update, 23
 algorithm, 25
 by columns, 25

 by rows, 25
 cost, 25
 via axpy, 25
 via dot, 25

reduced Singular Value Decomposition, 73

reduced SVD, 73

Reflector, 115

reflector, 115

scal, 16

sca
 cost, 17

scaled vector addition, 18
 cost, 18, 20

scaling
 vector
 cost, 17

singular value, 69

Singular Value Decomposition, 53–83
 geometric interpretation, 69
 reduced, 73
 theorem, 69

solve
 triangular, 229–232

Spark webpage, 104

spectral radius
 definition, 278

spectrum
 definition, 278

SVD
 reduced, 73

Taxi-cab norm
 see vector 1-norm, 40

transpose, 13–16
 matrix, 14
 vector, 13

triangular solve, 229
 lower, 229–231
 upper, 232

triangular system solve, 232

upper triangular solve, 232

vector
 1-norm
 see 1-norm, vector, 40

2-norm
see 2-norm, vector, 38

complex conjugate, 13

conjugate, 13

Hermitian transpose, 14

infinity-norm
see infinity-norm, vector, 40

length
see vector 2-norm, 38

norms(), 38

norms), 41

operations, 444

orthogonal, 64

orthonormal, 64

perpendicular, 64

scaling
cost, 17

transpose, 13

vector addition
scaled, 18

vector length
see vector 2-norm, 38

vector-vector operation, 16
axpy, 18
dot, 19
scal, 16

vector-vector operations, 444