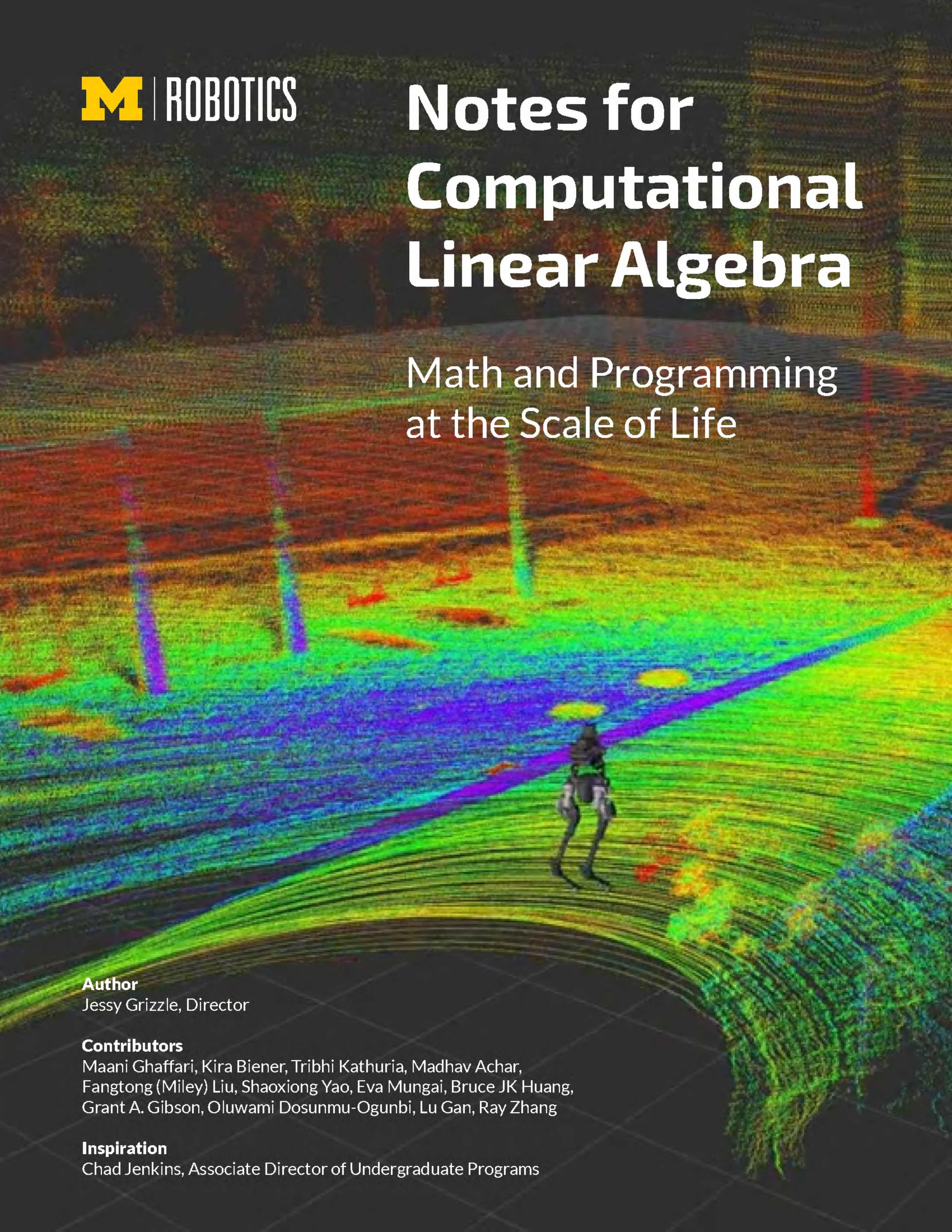


M | ROBOTICS

Notes for Computational Linear Algebra

Math and Programming at the Scale of Life

Author
Jessy Grizzle, Director

Contributors
Maani Ghaffari, Kira Biener, Tribhi Kathuria, Madhav Achar, Fangtong (Miley) Liu, Shaoxiong Yao, Eva Mungai, Bruce JK Huang, Grant A. Gibson, Oluwami Dosunmu-Ogunbi, Lu Gan, Ray Zhang

Inspiration
Chad Jenkins, Associate Director of Undergraduate Programs

Work together. Create smart machines. Serve society.

Cover design by Dan Newman, Head of Communications, Michigan Robotics

Elmer G. Gilbert Distinguished University Professor
Jerry W. and Carol L. Levin Professor of Engineering
Professor of EECS and ME

Composed between April 2020 and August 2020 for use in ROB 101, Computational Linear Algebra, a first-semester first-year course.
Revised Fall 2020 when put to the test with our Pilot Class.

First release, August 31, 2020

Contents

Preface 7

Philosophy of the Course 9

1 Introduction to Systems of Linear Equations 11
1.1 For Review on Your Own: You Have Done Algebra Before 12
1.2 Linear Systems of Equations: What can happen? 15
1.3 Naming Variables 17
1.4 A 3 x 3 Example 17
1.5 Looking Ahead 18

2 Vectors, Matrices, and Determinants 19
2.1 Scalars and Arrays 20
2.2 Row Vectors and Column Vectors 21
2.3 Remark on Brackets 21
2.4 Matrices, Rectangular and Square, and the Matrix Diagonal 22
2.5 Expressing a System of Linear Equations in terms of Vectors and Matrices 22
2.6 The Matrix Determinant 24
2.7 An Operational View of the Matrix Determinant 25
2.8 Examples of Using the Determinant 25
2.9 Looking Ahead 26

3 Triangular Systems of Equations: Forward and Back Substitution 27
3.1 Background 28
3.2 Examples of Lower Triangular (Linear) Systems 28
3.3 Determinant of a Lower Triangular Matrix 29
3.4 Lower Triangular Systems and Forward Substitution 30
3.5 Upper Triangular Systems, Upper Triangular Matrices, Determinants, and Back Substitution 31
3.6 General Cases 31
3.7 A Simple Trick with Systems of Equations: Re-arranging their Order 32
3.8 Looking Ahead 33

4 Matrix Multiplication 35
4.1 Multiplying a Row Vector by a Column Vector 36
4.2 Examples of Row and Column Partitions 37
4.3 General Case of Partitions 37
4.4 Standard Matrix Multiplication: It's All About Rows and Columns 38
4.4.1 Examples 38
4.4.2 Optional Read: General case: what is happening inside Julia 40
4.5 Multiplication by Summing over Columns and Rows 41
4.6 The Matrix View of Swapping the Order of Equations: Permutation Matrices 42
4.7 Looking Ahead 44

5 LU (Lower-Upper) Factorization 45
5.1 Recalling Forward and Back Substitution . . . 46
5.2 Recalling Matrix Multiplication in the Form of Columns Times Rows . . . 46
5.3 LU (Lower-Upper) Factorization (without row permutations) . . . 47
5.4 LU Factorization for Solving Linear Equations . . . 56
5.5 (Optional Read): Toward LU Factorization with Row Permutations . . . 57
5.6 (Optional Read): An Algorithm for LU Factorization with Row Permutations . . . 62
5.7 Looking Ahead . . . 68

6 Determinant of a Product, Matrix Inverses, and the Matrix Transpose 69
6.1 A very Useful Fact Regarding the Matrix Determinant . . . 70
6.2 Identity Matrix and Matrix Inverse . . . 71
6.3 Utility of the Matrix Inverse and its Computation . . . 73
6.4 Matrix Transpose and Symmetric Matrices . . . 74
6.5 Revisiting Permutation Matrices . . . 77
6.6 Looking Ahead . . . 77

7 The Vector Space R^n: Part 1 79
7.1 Vectors in $\mathbb{R}^n$ and Some Basic Algebra . . . 80
7.2 The Notion of Linear Combinations . . . 83
7.2.1 Linear combinations through the lens of $Ax = b$ . . . 83
7.2.2 Linear Combinations in $\mathbb{R}^n$ . . . 84
7.3 Existence of Solutions to $\mathbf{Ax} = \mathbf{b}$ . . . 86
7.4 Linear Independence of a Set of Vectors . . . 87
7.4.1 Preamble . . . 87
7.4.2 Linear Independence through the Lens of $Ax = 0$ . . . 87
7.4.3 Linear Independence in $\mathbb{R}^n$ (and why theory matters) . . . 88
7.4.4 A Pro Tip for Checking Linear Independence . . . 91
7.4.5 (Optional Read): Why the Pro Tip Works . . . 92
7.5 Number of Linearly Independent Vectors in a Set . . . 93
7.6 Attractive Test for Linear Combinations . . . 98
7.7 Existence and Uniqueness of Solutions to $Ax = b$ . . . 99
7.8 (Optional Read): LU Factorization for Symmetric Matrices . . . 101
7.9 (Optional Read): Still Another Take on LU Factorization for Checking Linear Independence . . . 105
7.10 Looking Ahead . . . 108

8 Euclidean Norm, Least Squared Error Solutions to Linear Equations, and Linear Regression 109
8.1 Norm or “Length” of a Vector . . . 110
8.2 Least Squared Error Solutions to Linear Equations . . . 111
8.3 Linear Regression or Fitting Functions to Data . . . 114
8.4 (Optional Read): How to Derive the Main Regression Formula . . . 119
8.4.1 Completing the Square for the Quadratic Formula: . . . 119
8.4.2 Completing the Square for Least Squares Solutions to Systems of Linear Equations: . . . 119
8.5 Looking Ahead . . . 120

9 The Vector Space R^n: Part 2 121
9.1 $\mathbb{R}^n$ as a Vector Space . . . 122
9.2 Subspaces of $\mathbb{R}^n$ . . . 123
9.3 Basis Vectors and Dimension . . . 129
9.4 Some Handy Matrix Facts . . . 134
9.5 Dot Product, Orthonormal Vectors, Orthogonal Matrices . . . 136
9.5.1 Dot Product or Inner Product . . . 136
9.5.2 Orthonormal Vectors . . . 138
9.5.3 Orthogonal Matrices . . . 139
9.6 Constructing Orthonormal Vectors: the Gram-Schmidt Process . . . 140
9.6.1 Building Orthogonal Vectors . . . 140
9.6.2 Gram-Schmidt Process or the Gram-Schmidt Algorithm . . . 142

9.7 QR Factorization and Solutions of Linear Equations 144
9.8 Optional Read: Modified Gram-Schmidt Algorithm 147
9.9 What to do When $\mathbf{Ax} = \mathbf{b}$ has an Infinite Number of Solutions? 149
9.10 (Optional Read): Why the Matrix Properties on Rank and Nullity are True 151
9.11 Looking Ahead 153

10 Changing Gears: Solutions of Nonlinear Equations 155
10.1 Motivation and Simple Ideas 156
10.2 Bisection 157
10.3 The Concept of a Derivative and Its Numerical Approximation 163
10.4 Newton's Method for Scalar Problems 166
10.5 Vector Valued Functions: Linear Approximations, Partial Derivatives, Jacobians, and the Gradient 170
10.5.1 Linear Approximation about a Point: Take 1 170
10.5.2 Partial Derivatives 171
10.5.3 The Jacobian and Linear Approximation of a Function about a Point 174
10.5.4 The Gradient and Linear Approximation of a Function about a Point 176
10.5.5 Summary on Partial Derivatives 178
10.6 Newton-Raphson for Vector Functions 178
10.7 (Optional Read): From the Gradient or Jacobian of a Function to its Linear Approximation 180
10.7.1 The Gradient 180
10.7.2 Expanding on Vector Notation 183
10.7.3 The Jacobian 184
10.7.4 Linear Approximation of Vector Valued Functions 186
10.8 Looking Ahead 187

11 Changing Gears Again: Basic Ideas of Optimization 189
11.1 Motivation and Basic Ideas 190
11.2 Contour Plots and the Gradient of the Cost 193
11.3 Gradient Descent 195
11.4 Extrinsic Calibration Using Gradient Descent 197
11.4.1 Introduction 197
11.4.2 Problem Setup and Initial Solution 198
11.5 Optimization as a Root Finding Problem: the Hessian 200
11.5.1 Second Derivatives 200
11.5.2 The Hessian is the Jacobian of the Transpose of the Gradient 201
11.5.3 Use of the Second Derivative and Hessian in Optimization 202
11.6 Local vs Global 207
11.7 Maximization as Minimizing the Negative of a Function 209
11.8 (Optional Read): Quadratic Programs: Our first Encounter with Constraints 209
11.9 (Optional Read): QP Solver in Julia 215
11.10(Optional Read): Optimization Packages: The Sky is the Limit 217

A Background for Machine Learning 219
A.1 Separating Hyperplanes 220
A.1.1 Lines in $\mathbb{R}^2$ as Separating Hyperplanes 220
A.1.2 Hyper Subspaces 221
A.1.3 Translations of Sets, Hyper Subspaces, and Hyperplanes 223
A.2 Orthogonal Projection 225
A.3 Signed Distance and Orthogonal Projection onto Hyperplanes 230
A.3.1 Signed Distance to a Hyperplane 230
A.3.2 Orthogonal Projection onto Linear Varieties (translations of subspaces) 231
A.4 Max-margin Classifier 233

B To Learn on Your Own (if you want to): Cool and Important Things We Omitted From our Linear Algebra Introduction 235

B.1 Complex Numbers and Complex Vectors . . . 236

B.1.1 Arithmetic of Complex Numbers: Enough to Get You By . . . 237

B.1.2 Angles of Complex Numbers and Euler's Formula: More Advanced Aspects . . . 238

B.1.3 Iterating with Complex Numbers: Background for Eigenvalues . . . 240

B.1.4 $\mathbb{C}^n$, the Space of Complex Vectors . . . 242

B.1.5 Iterating with Matrices: The Case for Eigenvalues and Eigenvectors . . . 243

B.2 Eigenvalues and Eigenvectors . . . 245

B.2.1 General Square Matrices . . . 246

B.2.2 Real Symmetric Matrices . . . 248

B.3 Positive Definite Matrices . . . 250

B.4 Singular Value Decomposition or SVD . . . 252

B.4.1 Motivation . . . 253

B.4.2 Definition and Main Theorem . . . 253

B.4.3 Numerical Linear Independence . . . 255

B.5 Linear Transformations and Matrix Representations . . . 257

B.6 Affine Transformations . . . 260

C What is an Ordinary Differential Equation? 261

C.1 Preliminaries: Expanding our Concept of an Equation . . . 262

C.2 Time in a Digital Computer is Discrete . . . 262

C.3 Digital Time may be Discrete, but We Can Make the Time Increment $\delta\mathbf{t}$ Quite Small . . . 263

C.4 Equations with Derivatives in them are called Differential Equations . . . 264

C.5 Discretization of ODEs of Higher Dimension . . . 266

C.6 Julia Code for Generating the Figures . . . 269

C.6.1 For Figures C.1 and C.2 . . . 269

C.6.2 Code for Figure C.3 . . . 269

C.6.3 Code for Figure C.4 . . . 269

C.6.4 Code for Figure C.5 . . . 270

C.6.5 Code for Figure C.6 . . . 270

D Camera and LiDAR Models for Students of Robotics 273

D.1 Pinhole Camera Model . . . 274

D.2 Preliminaries: Geometrical Transformations . . . 274

D.2.1 Homogeneous Coordinates . . . 274

D.2.2 2D Translation . . . 274

D.2.3 2D Scaling . . . 275

D.2.4 2D Rotation . . . 275

D.2.5 Other 2D Geometrical Transformations in Homogeneous Coordinates . . . 276

D.3 Pinhole Model . . . 276

D.4 Geometric and Mathematical Relations . . . 279

D.4.1 Intrinsic Matrix K . . . 279

D.4.2 Projection Matrix . . . 279

D.4.3 Extrinsic Matrix . . . 279

D.4.4 Full Projection Matrix Workspace to Camera . . . 279

D.5 Intrinsic Matrix Calibration . . . 280

D.6 Nonlinear Phenomena in Calibration . . . 280

D.7 Projection Map from LiDAR to Camera . . . 281

D.8 Projection Map . . . 281

Preface

This collection of course notes is dedicated to the group of students who were brave enough to take the pilot offering of ROB 101, Computational Linear Algebra.

ROB 101 was conceived by Prof. Chad Jenkins as one part of a complete undergraduate curriculum in Robotics. Chad and I both thank Associate Dean for Undergraduate Education, Prof. Joanna Mirecki Millunchick, for her support in the offering of this course as ROB 101.

The following remarks are adapted from an Education Proposal led by Prof. Jenkins, Dr. Mark Guzdial, Ella Atkins, and myself.

A challenge for the current undergraduate curricular structure at the University of Michigan (and elsewhere) is that the best Robotics major is a quadruple major in ME, EE, CS, and Math with a minor in Psychology. The Robotics Institute at Michigan is poised to address this challenge in revolutionary ways as it evolves toward a department. The Robotics faculty are using the opportunity of a clean slate in the area of undergraduate robotics education-—and the absolute necessity to integrate learning across a wide range of traditional disciplines—to design a new curricular system where computation, hardware, and mathematics are on an equal footing in preparing a future innovation workforce.

By integrating linear algebra, optimization, and computation in the first semester, students will experience mathematics as a means of making predictions and reasoning about experimental outcomes and robot designs. They will be prepared to encounter physics as a means to build mathematical models that describe the movement of objects, those with a palpable mass as well as electrons and waves, while grounding all of this in design. And computation? They will find it is the engine that drives discovery and the means of extracting information from data, allowing their machines to make decisions in real-time.

In addition to **ROB 101 (*Computational Linear Algebra*)** in the first year, we are planning **ROB 102 (*Graph Search for Robotics and AI*)**, which will show how computers can reason autonomously through graph search algorithms. The objective of the course is for students to implement a path planner for autonomous navigation with a given mobile robot at a known location in a known environment. The course will build towards providing a broader conceptual foundation for modeling problems as graphs and inferring solutions through search.

In **ROB 103 (*Robotic Mechanisms*)**, students will experience hands-on robotic systems design, build, and operation. The objective of the course is for students, in teams, to build an omni-drive mobile robot that can be teleoperated, as a step towards the gateway course: ROB 204. Students will learn to safely and efficiently operate modern shop tools such as 3D printers and laser cutters as well as traditional machining tools such as lathes, mills, and drill presses. Students will learn basic electronic and power systems principles including safe battery management, wiring harness design and assembly, and signal measurement for test and debugging. Students will design and build their real-world mobile robot based on application requirements from conception with CAD software through manufacturing, assembly, and test. Each student team will be given a “kit” of servos, sensors, and low-cost embedded processing components to use in their design.

The new first-year curriculum allows for major innovation in the second and third year curriculum, and the advancing of many graduate topics to much earlier places in Robotics education. We hope that you will follow our efforts in bringing this curriculum to the University of Michigan.

Jessy Grizzle
Fall Term, 2020

Philosophy of the Course

The Robotics faculty want out of the Sputnik era of educating engineers, where we are forced to pound our students with four semesters of Calculus before we are able to engage them in any interesting engineering. We seek to prepare students for the era of Information, AI, Data, and of course, Robotics. While we believe our ideas for this Linear Algebra course will work for most engineering departments, we understand that the College of Engineering needs a skunkworks to test some of our more revolutionary suggestions before turning them loose on everyone. We are proud to serve that role.

ROB 101 assumes a High School course in Algebra and no background in programming. With these entry requirements, we seek to open up mathematics and programming to everyone with the drive and skills to arrive at the University of Michigan's College of Engineering. From its inception in December 2019, the plan has always been to teach the course in a hybrid mode. We are being very intentional to design the course for inclusivity with a focus on making sure that one's zip code is not the best predictor of success. To do that, we are re-imagining the way mathematics is introduced to first-semester Y1 undergrads. We want to break the Sputnik era 4-semester calculus chain where AP credits are a huge predictor of success.

We will begin mathematics with Linear Algebra, the workhorse of modern autonomous systems, machine learning, and computer vision. We are integrating it with computation, and to make sure that students without access to high-end tools can be successful, all the programming will be run in a web browser through cloud services that are hidden from the student. If you can google at home or the library, then you can complete our programming assignments. Setting this up is a challenge, but we feel it is very important to have a platform that has equity in mind. Our plans for a hybrid mode of teaching are motivated by a desire to have students from Minority Serving Institutions join the course this fall.

The material in the course leaves out many traditional Linear Algebra topics, such as eigenvalues, eigenvectors, or how to compute determinants for matrices larger than 2×2. Sounds crazy! Good. The course focuses on solving systems of linear equations at scale, meaning hundreds or thousands of variables, and all the mathematics in the book should be implementable in HW sets in Julia. With that as a premise, we chose to focus on a few things in Linear Algebra that work well at scale, such as triangular systems of equations. This led to the conviction that LU Factorization should be introduced early and in an understandable manner, which was made possible by a magical video by Prof. Gilbert Strang (MIT). Once you understand that matrix multiplication $C = A \cdot B$ can be done by multiplying the columns of A by the rows of B and summing them up, the LU Factorization becomes completely transparent, allowing triangular matrices to rule the day. And once you focus on triangular structures, even linear independence and linear combinations become much more approachable. Of course, some basic geometry is good, and thus Gram-Schmidt is featured along with the QR Factorization. In between, least squared error solutions to linear equations and regression are treated along with other useful tools.

The book wraps up with a treatment of root finding for both scalar and vector nonlinear equations, and a users' view of optimization that highlights the role that the abstract function "$\arg\min$" is playing in modern engineering.

Readers of the book will not find any of the juicy computational tools as they are treated in the HW sets, via jupyter notebooks, and in three amazing Projects. The first project leads students through the process of building a map for robot navigation from LiDAR data collected on the UofM North Campus Grove. The main Linear Algebra concept being explored is the transformation of points and collections of points in $\mathbb{R}^3$ under rigid body transformations. The second project is built around regression and will give students insight into the power of Machine Learning. The third project will focus on the control of a planar Segway using a simplified version of Model Predictive Control. Students will experience the excitement of balancing a challenging ubiquitous mobile platform, and while doing so, will learn about ODEs and their relation to iterative discrete-time models.

Finally, we thank Prof. Steven Boyd (Stanford) for making his Y1 Applied Linear Algebra course material open source (http://vmls-book.stanford.edu/). We plan to do the same.

Jessy Grizzle and Maani Ghaffari Ann Arbor, Michigan USA

Chapter 1

Introduction to Systems of Linear Equations

Learning Objectives

- Get you going on Algebra, just in case Calculus has erased it from your mind
- Review on your own the quadratic equation.
- Set the stage for cool things to come.

Outcomes

- Refresher on the quadratic equation.
- Examples of systems of linear equations with two unknowns. Show that three things are possible when seeking a solution:
    - there is one and only one solution (one says there is a unique solution, which is shorthand for “there is a solution and it is unique”);
    - there is no solution (at all); and
    - there are an infinite number of solutions
- Notation that will allow us to have as many unknowns as we’ll need.
- Remarks that you can mostly ignore on why some numbers are called counting numbers, rational numbers, irrational numbers, or complex numbers
- Remarks on your first project.

1.1 For Review on Your Own: You Have Done Algebra Before

Quadratic Equation

$ax^2 + bx + c = 0$, where x is an *unknown variable* and *typically* a, b, and c are fixed real numbers, called *constants*.
If $a \neq 0$, the solutions to this *nonlinear algebraic equation* are

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

where the symbol ± (read, plus minus) indicates that there is one solution x corresponding to the plus sign and a second solution x corresponding to the minus sign.

The *discriminant* of a quadratic equation is $\Delta := b^2 - 4ac$. If $\Delta \geq 0$, the two solutions are *real numbers*, while if $\Delta < 0$, the two solutions are *complex numbers*.

Complex Numbers

If you have not learned complex numbers (**a**lso **k**nown **a**s (**aka**) ***imaginary numbers***) or are fuzzy on the details, let your GSI know. We may not use complex numbers at all in the graded portion of the course. We're not sure yet! We will need complex numbers when we study eigenvalues of matrices, which could happen at the end of the term, or it could be that we do not get that far. Piloting a Linear Algebra course during a pandemic has never been done before!

Example 1.1 ***(two distinct real solutions)*** *Find the roots of* $2x^2 + 8x - 10 = 0$.

Solution:

$$\begin{aligned} a &= 2, b = 8, c = -10 \\ b^2 - 4ac &= 144 > 0 \\ x &= \frac{-8 \pm \sqrt{144}}{4} \\ &= \frac{-8 \pm 12}{4} \\ &= -2 \pm 3 \end{aligned}$$

The two solutions are $x = 1$ and $x = -5$. ■

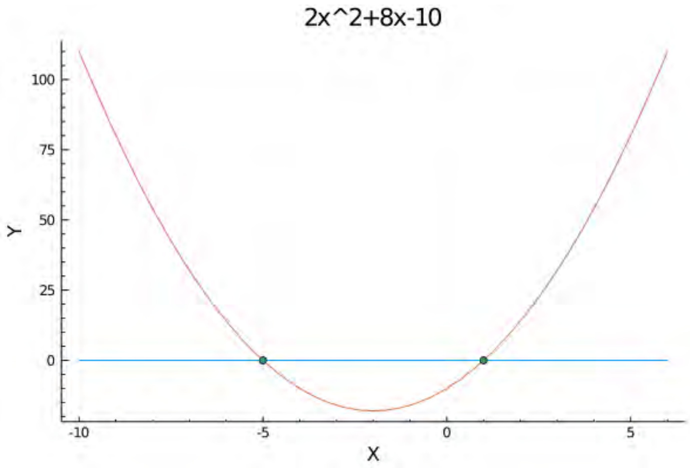


Figure 1.1: The two roots are at the intersection of the quadratic with the line $y = 0$. Why? Because, by definition, the roots are values of x where $ax^2 + bx + c$ equals zero!

Example 1.2 ***(two repeated real solutions)*** *Find the roots of* $2x^2 + 8x + 8 = 0$.

Solution:

$$\begin{aligned} a = 2, b = 8, c = 8 \\ b^2 - 4ac = 0 \\ x &= \frac{-8 \pm \sqrt{0}}{4} \\ &= \frac{-8 \pm 0}{4} \\ &= -2 \pm 0 \end{aligned}$$

The two solutions are $x = -2$ and $x = -2$. ■

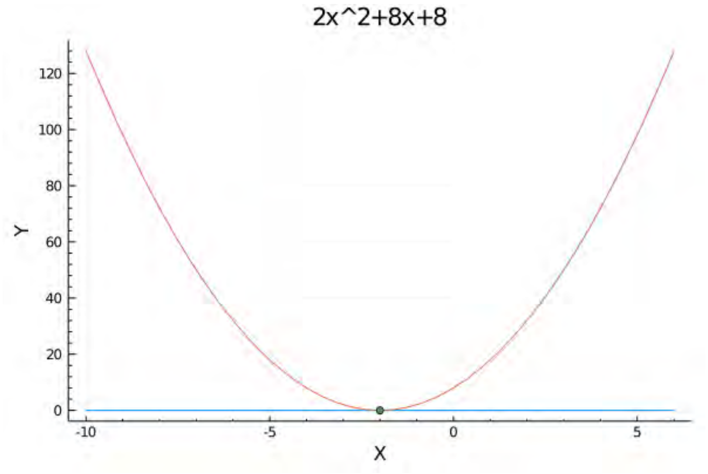


Figure 1.2: Note that there is now only a single intersection with $y = 0$ at $x = -2$. Because quadratic equations have two solutions, we say **the root at** $\mathbf{x = -2}$ **is repeated**.

Example 1.3 ***(two distinct complex solutions)*** *Find the roots of* $2x^2 + 8x + 10 = 0$.

Solution:

$$\begin{aligned} a = 2, b = 8, c = 10 \\ b^2 - 4ac = -16 \\ x &= \frac{-8 \pm \sqrt{-16}}{4} \\ &= \frac{-8 \pm \sqrt{16}\sqrt{-1}}{4} \\ &= \frac{-8 \pm 4\sqrt{-1}}{4} \\ &= -2 \pm \sqrt{-1} \\ &= -2 \pm i \end{aligned}$$

The two solutions are $x = -2 + i$ and $x = -2 - i$, where $i := \sqrt{-1}$ is an *imaginary number*. If you have not already learned complex numbers, do not sweat it. If we need them at all in ROB 101, it will be at the end of the term and we will teach complex numbers before we use them! ■

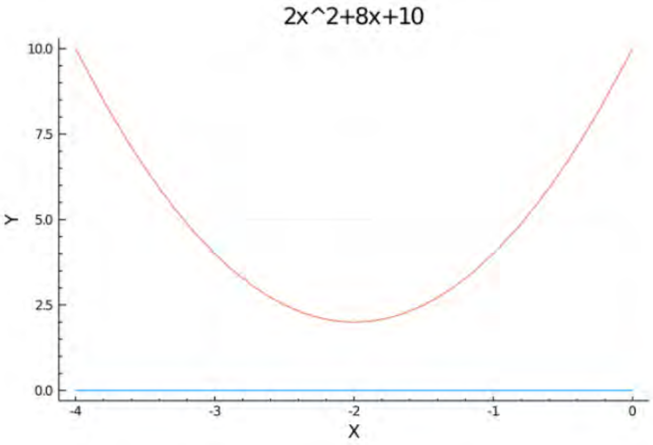


Figure 1.3: Note that there are no intersections with $y = 0$ and hence there are not any real solutions.

Remark: As in many subjects, the vocabulary in mathematics reflects the many twists and turns of history. If you are expecting mathematics to be a *pure science* in the sense that it is 100% logical, then you will often be disappointed! The Greeks and Egyptians called $1, 2, 3, \ldots$ *natural numbers* or *counting numbers* because they arose "naturally" in "counting objects": one cow, two sheep, three coins, four sacks of flour, etc. They were also comfortable with *fractions*, $\frac{m}{n}$, where both m and n were counting numbers, and hence, n could never be zero. Fractions were called *rational numbers* based on the word *ratio*. The square root of two, $\sqrt{2}$, was known to the Babylonians, Indians and Greeks. Our notion of $\sqrt{2}$ is probably thanks to Pythagoras, the Ionian Greek philosopher known to you for developing the Pythagorean Theorem relating the sides of right triangles, yes, the famous formula $a^2 + b^2 = c^2$, which gives $1^2 + 1^2 = (\sqrt{2})^2$, where the symbol $\sqrt{2}$ stands for the "quantity that when squared gives the counting number 2". It took a long time for the Greeks to accept that $\sqrt{2}$ could not be written as a *ratio of two counting numbers*. Eventually, it was proved to be *irrational*, that is, *not rational*, which means precisely that it cannot be expressed as the ratio of two counting numbers. In case you are interested, while Euclid was not the first to prove $\sqrt{2}$ was irrational, the beautiful reasoning he invented for the proof of $\sqrt{2}$ not being a rational number is still taught today. It is called *proof by contradiction*[1].

So far, so good with regards to vocabulary. But why call things involving $\sqrt{-1}$ complex numbers? Initially, mathematicians could not justify the existence of *quantities involving square roots of negative numbers.* Using them in calculations was a sign of "careless" mathematics and such numbers were treated as being *figments of one's imagination*, literally, *imaginary numbers*. Nevertheless, some (brave) mathematicians found them convenient for solving equations and others even for describing physical phenomena. Eventually, formal algebra caught up with the notion of imaginary numbers and their rules of use were rigorously justified. The name imaginary numbers stuck, however! Here is a link to a slightly simplified explanation of how mathematicians "codify" the existence of complex numbers: http://www.math.toronto.edu/mathnet/answers/imagexist.html Your instructors are unsure if they could have followed the reasoning in this document when they were at your stage of college, so do not sweat it.

Just for fun, you might want to learn more about numbers on Wikipedia https://en.wikipedia.org/wiki/Number. Were negative numbers always accepted? What about the notion of zero? **None of these remarks on numbers are required reading**. You will not be tested on the history of numbers!

To Know

- What are the counting numbers (also known as the natural numbers)?
- What are rational numbers?
- Be able to give an example of an irrational number, but of course, you are not expected to prove it is irrational.
- Later in the course (though it is not sure): what are imaginary numbers?

[1] It may seem remarkable to you that the two words "proof" and "contradiction" can coexist in logical statements. As you advance in your mathematical training, you may come across the term again. "Proof by contradcition" is not for amateurs...think about it as semi-professional-grade logic and math.

In ROB 101, the first nine Chapters focus on linear equations, hence, equations that do not have quadratic terms x^2, cubic terms x^3, nor terms with higher powers; they also do not have $\sin(x)$, $\cos(x)$, $\sqrt{x}$, or e^x or anything like that. It is perhaps hard to believe that equations with only order-one terms and constants could be interesting or important, but they are both interesting and important!

1.2 Linear Systems of Equations: What can happen?

We begin with several examples to illustrate what can happen.

Same number of equations as unknowns, and there is a unique solution:

$$\begin{aligned} x + y &= 4 \\ 2x - y &= -1 \end{aligned} \tag{1.1}$$

One way to compute a solution is to solve for x in terms of y in the first equation and then substitute that into the second equation,

$$\begin{aligned} x + y = 4 &\implies x = 4 - y \\ 2x - y = -1 &\implies 2(4 - y) - y = -1 \\ &\implies -3y = -9 \\ &\implies y = 3 \\ &\text{going back to the top} \\ x = 4 - y &\implies x = 1 \end{aligned}$$

You can try on your own solving for y in terms of x and repeating the above steps. You will obtain the same answer, namely $x = 1, y = 3$.
Another "trick" you can try, just to see if we can generate a different answer, is to add the first equation to the second, which will eliminate y,

$$\begin{aligned} x + y &= 4 \\ +\ 2x - y &= -1 \\ &\overline{\qquad\qquad} \\ 3x + 0y &= 3 \\ &\implies x = 1 \end{aligned}$$

Going back to the top and using either of the two equations

$$\begin{aligned} x + y = 4 &\implies y = 4 - x \\ &\implies y = 3 \\ &\text{or} \\ 2x - y = -1 &\implies -y = -2x - 1 \\ &\implies -y = -3 \\ &\implies y = 3 \end{aligned}$$

gives the same answer as before, namely, $x = 1, y = 3$.

In fact, the set of equations (1.1) has one, and only one, solution. In math-speak, one says the set of equations (1.1) has a *unique solution*. Often, we will stack x and y together and write the solution as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}.$$

Same number of equations as unknowns, and there is no solution:

$$\begin{aligned} x - y &= 1 \\ 2x - 2y &= -1 \end{aligned} \tag{1.2}$$

Because there are only two equations with a very nice set of numbers you might notice almost immediately that the left-hand side of the second equation is twice the left-hand side of the first equation, namely, $2x - 2y = 2(x - y)$, but when we look to the right-hand sides, $-1 \neq 2 \cdot 1$, and hence the equations (1.2) are *inconsistent*.

While the above is one way to analyze the problem, let's try to find a solution just as we did for the first set of equations,

$$\begin{aligned} x - y = 1 &\implies x = y + 1 \\ 2x - 2y = -1 &\implies 2(y+1) - 2y = -1 \\ &\implies 2 + 2y - 2y = -1 \\ &\implies 2 = -1. \end{aligned}$$

Hence, trying to solve the equations has led us to a *contradiction*, namely $2 = -1$. Perhaps if we had solved for y in terms of x, we could have found a solution? Let's try

$$\begin{aligned} x - y = 1 &\implies -y = -x + 1 \\ &\implies y = x - 1 \\ 2x - 2y = -1 &\implies 2x - 2(x-1) = -1 \\ &\implies 2x - 2x + 2 = -1 \\ &\implies 2 = -1 \end{aligned}$$

again! No matter how you manipulate the equations (1.2), while obeying "the rules of algebra" (which we have **not yet learned** for systems of linear equations), you cannot extract a sensible answer from these equations. They really are *inconsistent*.

Same number of equations as unknowns, and there are an infinite number of solutions:

$$\begin{aligned} x - y &= 1 \\ 2x - 2y &= 2 \end{aligned} \tag{1.3}$$

Because there are only two equations with a very nice set of numbers, you might notice that the left-hand side of the second equation is twice the left-hand side of the first equation, namely, $2x - 2y = 2(x - y)$, and this time, when we look to the right-hand sides, $2 = 2 \cdot 1$, and hence the two equations are actually the "same" in the sense that is one equation can be obtained from the other equation by multiplying both sides of it by a non-zero constant. We could elaborate a bit, but it's not worth it at this point. Instead, let's approach the solution just as we did in the first case.

We solve for x in terms of y in the first equation and then substitute that into the second equation,

$$\begin{aligned} x - y = 1 &\implies x = y + 1 \\ 2x - 2y = 2 &\implies 2(y+1) - 2y = 2 \\ &\implies 2y + 2 - 2y = 2 \\ &\implies 2 = 2. \end{aligned}$$

The conclusion $2 = 2$, is perfectly correct, but tells us nothing about y. In fact, we can view the value of y as an arbitrary *free parameter* and hence the solution to (1.3) is

$$x = y + 1, \ \ -\infty < y < \infty.$$

The solution can also be expressed as

$$y = x - 1, \ \ -\infty < x < \infty,$$

which perhaps inspires you to plot the solution as a line in $\mathbb{R}^2$, with slope $m = 1$ and y-intercept $b = -1$.

Summary So Far

Consider a set of two equations with two unknowns x and y

$$\begin{aligned} a_{11}x + a_{12}y &= b_1 \\ a_{21}x + a_{22}y &= b_2, \end{aligned} \tag{1.4}$$

constants $a_{11}, a_{12}, a_{21}, a_{22}$ and b_1, b_2. Depending on the values of the constants, the linear equations (1.4) can have a unique solution, no solution, or an infinity of solutions.

Limit of Hand Solutions

When there are only two equations and two unknowns, determining *by hand* if the equations have one solution, no solution, or an infinity of solutions is quite do-able. With sufficient motivation and "nice numbers", three equations and three unknowns is also not too bad. At four, it becomes super tedious and errors begin to sprout like weeds. Hence, what about 100 equations and 100 unknowns? **Our four-week goal is for you to handle systems of linear equations with hundreds of variables with confidence and ease** As you can imagine, this is where the "computational" part of ROB 101's name comes into play!

1.3 Naming Variables

If we have two variables (also called unknowns), it is natural to call them x and y. If we have three variables, naming them x, y, and z works fine. If we have 26 variables, would we start with a and go all the way to z? What if we have more variables? Well, there are 24 Greek letters, do we add them to the list? Throw in Mandarin Characters to get us to several thousand variables? Clearly, this is not a practical way to go. The only real possibility is to add counting numbers, because there are as many of them as we need, and computers understand numbers!

Welcome to $x_1, x_2, x_3, \ldots, x_n$, or $y_1, y_2, \ldots, y_m$ etc.

1.4 A 3 x 3 Example

Here is a *system of linear equations* with variables x_1, x_2, x_3.

$$\begin{aligned} x_1 + x_2 + 2x_3 &= 7 \\ 2x_1 - x_2 + x_3 &= 0.5 \\ x_1 + 4x_3 &= 7 \end{aligned} \tag{1.5}$$

The strategy for seeking a solution is the same as with two equations and two unknowns: solve for one of the variables in terms of the other variables, substitute into the remaining equations, simplify them, and repeat. We can start anywhere, so let's solve for x_1 in the bottom equation and plug that back into the two equations above it. Doing so gives us,

$$x_1 = 7 - 4x_3 \tag{1.6}$$

and then

$$\begin{aligned} \underbrace{(7-4x_3)}_{x_1} + x_2 + 2x_3 = 7 &\implies x_2 - 2x_3 = 0 \\ \underbrace{2(7-4x_3)}_{2x_1} - x_2 + x_3 = 0.5 &\implies -x_2 - 7x_3 = -13.5 \end{aligned}$$

$$\begin{aligned} x_2 - 2x_3 = 0 &\implies x_2 = 2x_3 \\ -x_2 - 7x_3 = -13.5 &\implies \underbrace{-(2x_3)}_{-x_2} - 7x_3 = -13.5 \\ &\implies -9x_3 = -13.5 \\ &\implies x_3 = 1.5 \end{aligned} \tag{1.7}$$

Plugging "$x_3 = 1.5$" into (1.6) gives

$$x_1 = 7 - 4 \cdot (1.5) = 7 - 6 = 1.$$

And then from (1.6), we have that

$$x_2 = 2x_3 = 2 \cdot (1.5) = 3.$$

Hence, the solution is

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 3.0 \\ 1.5 \end{bmatrix}.$$

Tedium and Pain $\Longrightarrow$ Motivation!

The hope is that you found that super tedious and something you'd prefer to avoid! The more tedious and painful you found it, the more motivation you will have to learn some new math that will make solving systems of linear equations a snap.

1.5 Looking Ahead

We need a straightforward way to check whether a set of equations falls into the nice case of having a unique solution or is it one of the "problematic cases" where there may be no solution at all or an infinity of solutions. To get there, we need to learn:

- what are *matrices* and *vectors*;
- how to express a system of linear equations in terms of *matrices* and *vectors*;
- what it means for a set of linear equations to be *triangular* or *square*; and
- what is the *determinant* of a matrix.

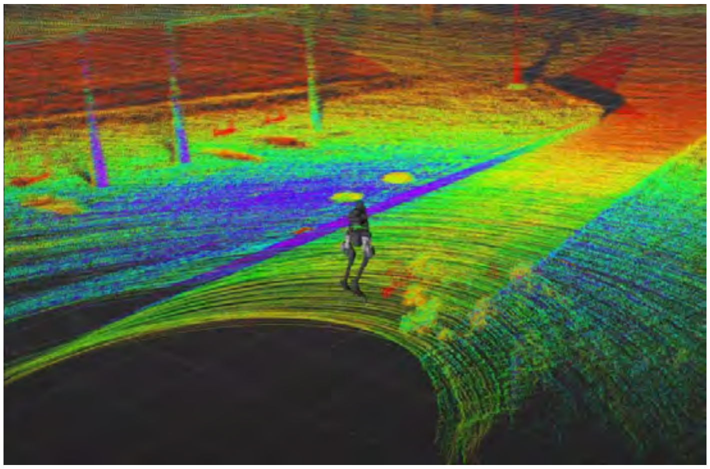


Figure 1.4: In your first project, you will learn what is a LiDAR sensor and how to process the LiDAR data to build a "map" that a robot can use for navigation. Each colored dot in the above image is a LiDAR measurement. How many are there? Ten thousand, maybe? The image is the result of combining multiple LiDAR scans into a single image. You will learn that this involves applying matrices to vectors. You will learn about coordinate frames. Right now, this seems like magic. In a few weeks, you will be comfortable enough with the Julia programming language to manipulate a dataset with 10,000 points or more. **Remember, one of our goals is mathematics and programming at the scale of life!**

Chapter 2

Vectors, Matrices, and Determinants

Learning Objectives

- Begin to understand the vocabulary of mathematics and programming
- Answer the question: why make a distinction between integers and decimal numbers.
- An introduction to the most important tools of linear algebra: vectors and matrices.
- Find out an easy way to determine when a set of linear equations has a unique answer.

Outcomes

- Scalars vs Array
- Row vectors and column vectors
- Rectangular matrices and square matrices
- Learn new mathematical notation $x := y$, which means that x is by definition equal to y. You may be more used to $x \triangleq y$.
- Using matrices and vectors to express systems of linear equations
- Determinant of a square matrix and its relation to uniqueness of solutions of systems of linear equations

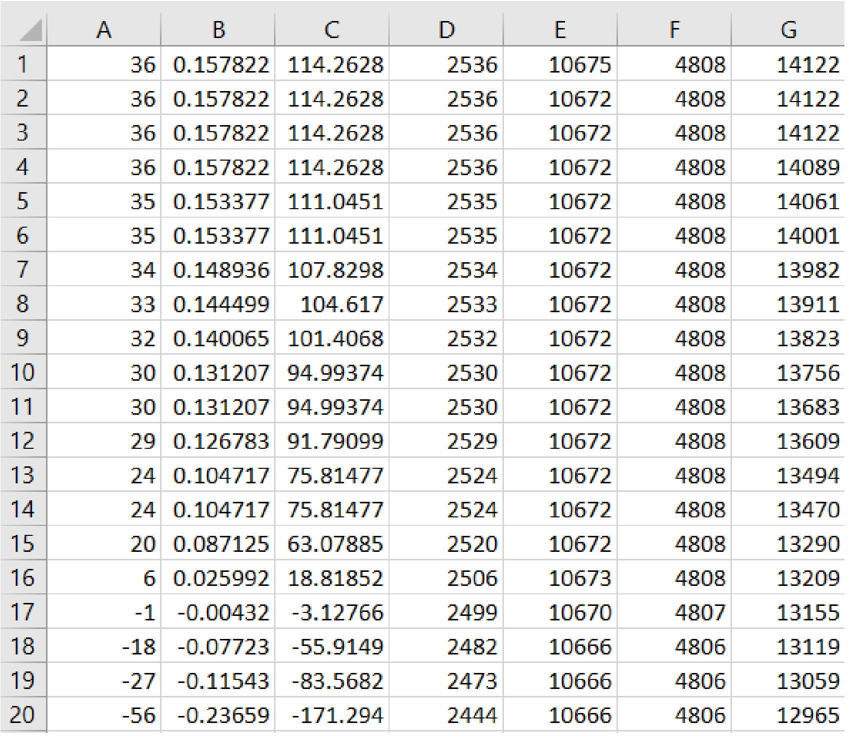
2.1 Scalars and Arrays

Scalars are simply numbers such as the ones you have been using for a long time: 25.77763, $\sqrt{17}, 10, -4, \pi$. In the Julia programming language, you will soon learn that for computational and storage efficiency, Julia differentiates between scalars that require decimal points and those that do not. Get ready for that! In ROB 101, when we do math, scalars are just numbers. When we do programming, scalars that do not require decimal points are called INTEGERS and those that do require decimal points are called FLOATING POINT NUMBERS because, with the same number of zeros and ones[1], Julia has to represent very large numbers such as 5.972×10^{24}, the mass of the earth in kilograms, and very small numbers, such as the mass of one atom of lead in kilograms $3.4406366 \times 10^{-22}$. To do that, where the decimal point appears in a list of zeros and ones has to "float", that is, it varies from number to number.

In ROB 101, we will not need to learn how computers represent numbers in *binary*. We will simply accept that computers use a different representation for numbers than we humans use. The more zeros and ones we allow in the representation of number, the more space it takes and the longer it takes to add them or multiply them, etc. The computer that provided navigational assistance for the moon landing on 20 July 1969 had a 16 bit word length, meaning its computations were based on groups of 16 binary digits (zeros and ones), called *bits*. In Julia, we'll typically use 64 zeros and ones to represent numbers:

- Int64
- Float64

Remark (can skip): In case you are wondering, ∞ (infinity) is a concept: it is NOT a number. Because it is not a number, ∞ is neither an integer nor a decimal number. The symbol ∞ is used to indicate that there is no positive upper bound for how big something can grow (i.e., it can get bigger and bigger and bigger ...), while $-\infty$ is similar to indicate that something can get more and more negative without end. In ROB 101, you will only encounter ∞ when you try to divide something by zero. Julia has been programmed to rock your world when you do that! (Just kidding).



	A	B	C	D	E	F	G
1	36	0.157822	114.2628	2536	10675	4808	14122
2	36	0.157822	114.2628	2536	10672	4808	14122
3	36	0.157822	114.2628	2536	10672	4808	14122
4	36	0.157822	114.2628	2536	10672	4808	14089
5	35	0.153377	111.0451	2535	10672	4808	14061
6	35	0.153377	111.0451	2535	10672	4808	14001
7	34	0.148936	107.8298	2534	10672	4808	13982
8	33	0.144499	104.617	2533	10672	4808	13911
9	32	0.140065	101.4068	2532	10672	4808	13823
10	30	0.131207	94.99374	2530	10672	4808	13756
11	30	0.131207	94.99374	2530	10672	4808	13683
12	29	0.126783	91.79099	2529	10672	4808	13609
13	24	0.104717	75.81477	2524	10672	4808	13494
14	24	0.104717	75.81477	2524	10672	4808	13470
15	20	0.087125	63.07885	2520	10672	4808	13290
16	6	0.025992	18.81852	2506	10673	4808	13209
17	-1	-0.00432	-3.12766	2499	10670	4807	13155
18	-18	-0.07723	-55.9149	2482	10666	4806	13119
19	-27	-0.11543	-83.5682	2473	10666	4806	13059
20	-56	-0.23659	-171.294	2444	10666	4806	12965

Figure 2.1: Here is an array of numbers from a spreadsheet. The rows are numbered while the columns are labeled with letters.

[1] Zeros and ones are sometimes called the binary language of computers. If you are interested in binary numbers and binary arithmetic, you can find many sources on the web.

Arrays are scalars that have been organized somehow into lists. The lists could be rectangular or not, they could be made of columns of numbers, or rows of numbers. If you've ever used a spreadsheet, then you have seen an array of numbers. In Fig. 2.1, columns A, D, E, F, and G have only integer numbers in the them, while columns B and C use decimal numbers. Each row has both integer numbers and decimal numbers.

2.2 Row Vectors and Column Vectors

For us, a *vector* is a finite ordered list of numbers or of unknowns. In Julia, a vector is a special case of an *array*. For example

$$v = \begin{bmatrix} 1.1 \\ -3 \\ 44.7 \end{bmatrix} \tag{2.1}$$

is a vector of *length* three; sometimes we may call it a *3-vector*. By *ordered* we mean the list has a first element $v_1 = 1.1$, a second element $v_2 = -3$, and a third element $v_3 = 44.7$, and we also mean that if we change the order in which we list the elements, we (usually) obtain a different vector. For example, the vector

$$w = \begin{bmatrix} 1.1 \\ 44.7 \\ -3 \end{bmatrix} \tag{2.2}$$

is not equal to the vector v, even though they are made of the same set of three numbers.

Vectors written from "top" to "bottom" as in (2.1) and (2.2) are called *column* vectors. It is also useful to write vectors from "left" to "right" as in

$$v^{\text{row}} = \begin{bmatrix} 1.1 & -3 & 44.7 \end{bmatrix} \tag{2.3}$$

and we call them *row* vectors. What we said about the word "ordered" applies equally well to row vectors in that the row vector v has a first element $v_1^{\text{row}} = 1.1$, a second element $v_2^{\text{row}} = -3.0$, and a third element $v_3^{\text{row}} = 44.7$. Moreover, if we change the order in which we list the elements, we (usually) obtain a different row vector. Here, we were super careful and added the superscript $^{\text{row}}$ to v^{row} to clearly distinguish the symbol v being used for the column vector (2.1) from the row vector (2.3). Normally, we will not be that fussy with the notation, BUT, you must be aware that column vectors and row vectors are different "animals" except in the case that they have length one, as in $v = [v_1]$ is both a row vector and a column vector.

A general length n column vector is written like this,

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}, \tag{2.4}$$

while a general length n row vector is written like this

$$v = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}. \tag{2.5}$$

This general notation allows for the case that $n = 1$, yielding

$$v = [v_1],$$

which as we noted above, is both a row vector and a column vector.

2.3 Remark on Brackets

Usually, in this course and in most books, *square brackets* [ ] are used to enclose vectors, but that is mainly *a matter of taste* as you can also use *parentheses* ( ) as in

$$v = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} \tag{2.6}$$

and

$$v = \begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix}. \tag{2.7}$$

In most programming languages, and Julia is no exception, you can only use square brackets! The choice of symbols, words, and their allowed arrangements in a language is called *syntax*. In a programming language, syntax is typically much more restrictive than in a written version of a spoken language. At some point, you may appreciate that throwing errors for bad syntax helps us to reduce *ambiguity* and *bugs* when we program.

2.4 Matrices, Rectangular and Square, and the Matrix Diagonal

Matrices are generalizations of vectors that allow multiple columns and rows, where each row must have the same number of elements[2]. Here is a 3×2 matrix,

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \tag{2.8}$$

meaning it has three rows and two columns, while here is a 2×3 matrix,

$$A = \begin{bmatrix} 1.2 & -2.6 & 11.7 \\ 3.1 & \frac{11}{7} & 0.0 \end{bmatrix}, \tag{2.9}$$

meaning it has two rows and three columns. It is customary to call a $1 \times n$ matrix a row vector and an $n \times 1$ matrix a column vector, but it is perfectly fine to call them matrices too!

A general $n \times m$ matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}, \tag{2.10}$$

is said to be *rectangular of size* $n \times m$ (one reads this as "n by m"), and when $n = m$, we naturally say that the $n \times n$ matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \tag{2.11}$$

is *square*. We note that a_{ij}, the ij-element of A, lies on the intersection of the i-th row and the j-th column.

Definition: The **diagonal** of the square matrix A in (2.11) is

$$\operatorname{diag}(A) = \begin{bmatrix} a_{11} & a_{22} & \ldots & a_{nn} \end{bmatrix} \tag{2.12}$$

What we are calling the diagonal is sometimes called the *main diagonal of a matrix*. We now highlight the diagonal in red to help those with a "visual memory"

$$A = \begin{bmatrix} {\color{red}a_{11}} & a_{12} & \cdots & a_{1n} \\ a_{21} & {\color{red}a_{22}} & \cdots & a_{2n} \\ \vdots & \vdots & {\color{red}\ddots} & \vdots \\ a_{n1} & a_{n2} & \cdots & {\color{red}a_{nn}} \end{bmatrix} \iff \operatorname{diag}(A) = \begin{bmatrix} {\color{red}a_{11}} & {\color{red}a_{22}} & {\color{red}\ldots} & {\color{red}a_{nn}} \end{bmatrix}.$$

While it is possible to define the diagonal of general rectangular matrices, we will not do so at this time.

2.5 Expressing a System of Linear Equations in terms of Vectors and Matrices

Before we even talk about "matrix-vector multiplication", we can address the task of writing a system of linear equations in "matrix-vector form". In the beginning, we will diligently follow the notation $Ax = b$, so let's see how we can identify a matrix A, a column vector x, and a column vector b.

[2] Equivalently, each column has the same number of elements.

Example 2.1 *Express the System of Linear Equations in Matrix Form:*

$$\begin{aligned} x_1 + x_2 &= 4 \\ 2x_1 - x_2 &= -1. \end{aligned} \tag{2.13}$$

Solution: When your instructors look at this equation, they see two unknowns x_1 and x_2 and *coefficients*[3] associated with them on the left-hand and right-hand sides of the two equations.

We can use the unknowns to define a column vector of length two, the four coefficients multiplying the unknowns to build a 2×2 matrix A, and the two coefficients on the right-hand side to define another column vector of length two,

$$\begin{aligned} x_1 + x_2 &= 4 \\ 2x_1 - x_2 &= -1 \end{aligned} \iff \underbrace{\begin{bmatrix} 1 & 1 \\ 2 & -1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 4 \\ -1 \end{bmatrix}}_{b}. \tag{2.14}$$

■

Example 2.2 *Express the System of Linear Equations in Matrix Form:*

$$\begin{aligned} x_1 - x_2 &= 1 \\ 2x_1 - 2x_2 &= -1. \end{aligned} \tag{2.15}$$

Solution: We now jump straight into it. We form the 2×1 vector x of unknowns as before and place the coefficients by rows into the 2×2 matrix A,

$$\begin{aligned} x_1 - x_2 &= 1 \\ 2x_1 - 2x_2 &= -1 \end{aligned} \iff \underbrace{\begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 1 \\ -1 \end{bmatrix}}_{b}. \tag{2.16}$$

■

Example 2.3 *Express the System of Linear Equations in Matrix Form:*

$$\begin{aligned} x_1 + x_2 + 2x_3 &= 7 \\ 2x_1 - x_2 + x_3 &= 0.5 \\ x_1 + 4x_3 &= 7 \end{aligned} \tag{2.17}$$

Solution: Note that we treat "any missing coefficients" (as in the "missing x_2" in the third equation) as being zeros! This is super important to remember.

$$\begin{aligned} x_1 + x_2 + 2x_3 &= 7 \\ 2x_1 - x_2 + x_3 &= 0.5 \\ x_1 + 4x_3 &= 7 \end{aligned} \iff \underbrace{\begin{bmatrix} 1 & 1 & 2 \\ 2 & -1 & 1 \\ 1 & \boxed{0} & 4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 7 \\ 0.5 \\ 7 \end{bmatrix}}_{b} \tag{2.18}$$

■

How to Handle "Missing" Variables or Coefficients

A zero in row i and column j of a matrix corresponds to the variable x_j being absent from the i-th equation

$$\underbrace{\begin{bmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \\ 0 & -1 & 4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 7 \\ 0.5 \\ 7 \end{bmatrix}}_{b} \iff \begin{aligned} 2x_3 &= 7 \\ 2x_1 &= 0.5 \\ -x_2 + 4x_3 &= 7 \end{aligned} \iff \underbrace{\begin{aligned} 0x_1 + 0x_2 + 2x_3 &= 7 \\ 2x_1 + 0x_2 + 0x_3 &= 0.5 \\ 0x_1 - x_2 + 4x_3 &= 7 \end{aligned}}_{\text{we'd never write it like this}} \tag{2.19}$$

[3] Coefficients in this case are the numbers multiplying x_1 and x_2. An equations typically has variables (unknowns) and coefficients (numbers) that multiply the unknowns or that determine what the equation is supposed to equal, as in $3x_1 + 4x_2 = 7$.

2.6 The Matrix Determinant

One of the more difficult topics in *Linear Algebra* is the notion of the determinant of a matrix. Most people who claim to understand it are wrong. We are going to give you an "operational view" of the matrix determinant, which means we'll tell you enough that you can use it, but not enough that you understand it! If you are fine with this, please skip to Chap. 2.7. If you are troubled by the idea of an "operational understanding" of something, then please continue reading this section.

What? Operational view!! Yes, that may sound strange to you, but how many of you think you understand what the *real numbers* are? You know they include numbers like these[4],

$$\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$$
$$\ldots, -\frac{2}{3}, -\frac{1}{2}, -\frac{1}{3}, 0, \frac{1}{8}, \frac{11}{16}, \frac{41}{7}, \ldots$$
$$\ldots, -e^{2\pi}, -\sqrt[3]{111}, -\varphi, 0, e, \pi, \sqrt{17}, \ldots$$

and you can manipulate them just fine, and perhaps you even know decimal approximations for some of them, such as

$$\sqrt{2} \approx 1.4142$$

However, unless you have taken a course in Real Analysis (here at Michigan, Math 451), you really have no idea how to define the real numbers[5]. One would say that you have an "operational view of the real numbers": you know how to use the concept but are a bit fuzzy on their definition!

Let's dig into a layered, operational "definition" of a determinant.

1. The determinant is a function[6] that maps a square matrix to a real number.
2. The determinant of a 1×1 matrix is the scalar value that defines the matrix.
3. The determinant of the 2×2 square matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is

   $$\det(A) := ad - bc.$$

   We will mention this formula again in the next section and ask you to memorize it. It is equally as important as the famous quadratic formula!

   Remark: Note that we wrote $\det(A) := ad - bc$ instead of $\det(A) \triangleq ad - bc$. Why, because the colon sign on the LEFT SIDE of the equals sign, :=, allows us to specify that we are defining $\det(A)$ and you are supposed to already know that a, b, c, d are elements of A. We could also have written it as $ad - bc =: \det(A)$, but we'll not do that very often, if at all, in ROB 101.
4. The determinant of a 3×3 matrix is

   $$\det\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}\right) := a \cdot \det\left(\begin{bmatrix} e & f \\ h & i \end{bmatrix}\right) - b \cdot \det\left(\begin{bmatrix} d & f \\ g & i \end{bmatrix}\right) + c \cdot \det\left(\begin{bmatrix} d & e \\ g & h \end{bmatrix}\right). \tag{2.20}$$

   In ROB 101, you do not need to memorize this formula. We may play with it in HW a bit using Julia.
5. Equation (2.20) gives an expression for the determinant of a 3×3 matrix as the sum of three terms that rely on the determinant of 2×2 matrices. In a similar manner, you can "boot strap" yourself to the determinant of an $n \times n$ matrix as a sum of n terms involving the determinant of $(n-1) \times (n-1)$ matrices! Each determinant of an $(n-1) \times (n-1)$ matrix is a sum of $(n-1)$ terms involving the determinants of $(n-2) \times (n-2)$ matrices Joy! Joy! Is this totally clear? Maybe not so much! Yes, now you see why we want an operational view.

[4] The "Golden Ratio" is denoted by φ and is equal to $\frac{1+\sqrt{5}}{2}$. Knowing this will not help at all in ROB 101!

[5] Wow, four more years of mathematical training before you even know what real numbers are! Don't fret too much about it, your instructor bets that fewer than 5% of the faculty in the College of Engineering have taken a course in Real Analysis and yet they are perfectly functional engineers; you can be too.

[6] Recall that a function $f : S_1 \to S_2$ is a rule that maps each element of the set S_1 to one element of the set S_2. The function is "det", S_1 is the set of $n \times n$ matrices and S_2 is the set of real numbers.

6. **Optional and for sure, skip on your first read: Khan Academy** Here are the (Tiny URLs) of two videos by the Khan Academy,
   - https://tinyurl.com/gqt93l3 works a 3×3 example based on (2.20).
   - https://tinyurl.com/zhlwb5v shows an alternative way to compute the determinant of a 3×3 matrix.
   - **In ROB 101**, you do not need to use either of these methods.

2.7 An Operational View of the Matrix Determinant

> What you want and need to know about the determinant function: the first four points are extra important.

Fact 1 The determinant of a square matrix A is a real number, denoted $\det(A)$.

Fact 2 A square system of linear equations (i.e., n equations and n unknowns), $Ax = b$, has a unique solution x for any $n \times 1$ vector b if, and only if, $\det(A) \neq 0$.

Fact 3 When $\det(A) = 0$, the set of linear equations $Ax = b$ may have either no solution or an infinite number of solutions. To determine which case applies (and to be clear, only one case can apply), depends on how "b is related to A", which is another way of saying that we will have to address this later in the course.

Fact 4 The determinant of the 2×2 square matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is

$$\det(A) := ad - bc.$$

Fact 5 In written mathematics, but not in programming, you will also encounter the notation $\det(A) = |A|$, so that

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} := ad - bc.$$

Fact 6 The determinant is only defined for square matrices; see Fact 1.

Fact 7 In Julia, once you state that you are using the linear algebra package, that is `using LinearAlgebra`, the command is det(A) , if A is already defined, or, for example, det([1 2 3; 4 5 6; 7 8 9])

Fact 8 We will learn later how to compute by hand the determinant of $n \times n$ matrices with special structure, but for general matrices, you only need to know how to *compute by hand the determinant of* a 2×2 matrix. **In HW, we may use the formula given in** (2.20) **but it will not be part of any quiz or exam**.

2.8 Examples of Using the Determinant

We reuse some previous examples and add in a few more to illustrate Fact 2 of Sec. 2.7

Example 2.4 *Check for existence and uniqueness of solutions:*

$$\begin{aligned} x_1 + x_2 &= 4 \\ 2x_1 - x_2 &= -1 \end{aligned} \iff \underbrace{\begin{bmatrix} 1 & 1 \\ 2 & -1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 4 \\ -1 \end{bmatrix}}_{b}. \tag{2.21}$$

Solution: We compute $\det(A) = (1) \cdot (-1) - (1) \cdot (2) = -3 \neq 0$ and conclude that (2.21) has a unique solution. ■

Example 2.5 *Check for existence and uniqueness of solutions:*

$$\begin{aligned} x_1 - x_2 &= 1 \\ 2x_1 - 2x_2 &= -1 \end{aligned} \iff \underbrace{\begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 1 \\ -1 \end{bmatrix}}_{b} \tag{2.22}$$

Solution: We compute $\det(A) = (1) \cdot (-2) - (-1) \cdot (2) = 0$. We therefore conclude that (2.22) does not have a unique solution. In fact, it may have either no solution at all or it may have an infinite number of solutions. At this point in the course, we do not have the tools to determine which case applies here without grinding through the equations. Referring back to (1.2), where we did grind through the equations, we know that this system of linear equations has no solution. ■

Example 2.6 *Check for existence and uniqueness of solutions:*

$$\begin{aligned} x_1 - x_2 &= 1 \\ 2x_1 - 2x_2 &= 2 \end{aligned} \iff \underbrace{\begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 1 \\ 2 \end{bmatrix}}_{b} \tag{2.23}$$

Solution: We compute $\det(A) = (1) \cdot (-2) - (1) \cdot (2) = 0$. We therefore conclude that (2.23) does not have a unique solution. In fact, it may have either no solution at all or it may have an infinite number of solutions. At this point in the course, we do not have the tools to determine which case applies here without grinding through the equations. Referring back to (1.3), we know that this system of linear equations has an infinite number of solutions. ■

Example 2.7 *Check for existence and uniqueness of solutions:*

$$\begin{aligned} x_1 + x_2 + 2x_3 &= 7 \\ 2x_1 - x_2 + x_3 &= 0.5 \\ x_1 + 4x_3 &= 7 \end{aligned} \iff \underbrace{\begin{bmatrix} 1 & 1 & 2 \\ 2 & -1 & 1 \\ 1 & 0 & 4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 7 \\ 0.5 \\ 7 \end{bmatrix}}_{b} \tag{2.24}$$

Solution: Using Julia, we compute $\det(A) = 9 \neq 0$. We therefore conclude that (2.24) has a unique solution for x. ■

Example 2.8 *Check for existence and uniqueness of solutions in an example where the unknowns are not in the "correct" order and we have a bunch of "missing coefficients":*

$$\begin{aligned} x_1 + x_2 + 2x_3 &= 7 \\ -x_2 + x_3 + 2x_1 &= 0.5 \\ x_1 + 4x_3 &= 7 \\ x_4 + 2x_3 - 5x_5 - 11 &= 0 \\ -4x_2 + 12x_4 &= 0 \end{aligned} \iff \underbrace{\begin{bmatrix} 1 & 1 & 2 & 0 & 0 \\ 2 & -1 & 1 & 0 & 0 \\ 1 & 0 & 4 & 0 & 0 \\ 0 & 0 & 2 & 1 & -5 \\ 0 & -4 & 0 & 12 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 7 \\ 0.5 \\ 7 \\ 11 \\ 0 \end{bmatrix}}_{b} \tag{2.25}$$

Solution: Using Julia, one computes $\det(A) = -540 \neq 0$. We therefore conclude that (2.25) has a unique solution. Grinding through the equations would have been no fun at all! ■

2.9 Looking Ahead

We want to solve very large sets of linear equations, say more than 100 variables. We could teach you the matrix inverse command in Julia, but then you'd understand nothing. In the next chapter, we will begin exploring how to solve problems with special structure. Soon after that, we'll show how to reduce all linear systems of n-equations in n-variables to being solvable with "special strucutre". To get there we need to understand:

- what are *square and triangular matrices*;
- what is *forward* and *back substitution*;
- how to *factor* a square matrix as the product of two triangular matrices; and
- what does it mean to multiply two matrices.

Chapter 3

Triangular Systems of Equations: Forward and Back Substitution

Learning Objectives

- Equations that have special structure are often much easier to solve
- Some examples to show this.

Outcomes

- Recognize triangular systems of linear equations and distinguish those with a unique answer.
- Learn that the determinant of a square triangular matrix is the product of the terms on its diagonal.
- How to use forward and back substitution.
- Swapping rows of equations and permutation matrices.

3.1 Background

- A system of linear equations is **square** if it has the same number of equations as unknowns.
- A square system of linear equations $Ax = b$, has a unique solution x for any $n \times 1$ vector b if, and only if, the $n \times n$ matrix A satisfies $\det(A) \neq 0$.
- Computing the determinant of a general square matrix is tedious.
- The diagonal of the matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

is

$$\operatorname{diag}(A) = \begin{bmatrix} a_{11} & a_{22} & \ldots & a_{nn} \end{bmatrix}.$$

3.2 Examples of Lower Triangular (Linear) Systems

This is an example of a square system of linear equations that is *Lower Triangular*

$$\begin{aligned} 3x_1 &= 6 \\ 2x_1 - x_2 &= -2 \\ x_1 - 2x_2 + 3x_3 &= 2. \end{aligned} \tag{3.1}$$

When we write the system as $Ax = b$, in the lower triangular case we have

$$\begin{aligned} 3x_1 &= 6 \\ 2x_1 - x_2 &= -2 \\ x_1 - 2x_2 + 3x_3 &= 2 \end{aligned} \iff \underbrace{\begin{bmatrix} 3 & 0 & 0 \\ 2 & -1 & 0 \\ 1 & -2 & 3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ -2 \\ 2 \end{bmatrix}}_{b}. \tag{3.2}$$

where we note that all terms "above" the diagonal of the matrix A are zero. More precisely, the condition is $a_{ij} = 0$ for all $j > i$. Such matrices are called *lower-triangular*.

Here are two more examples of square lower triangular systems

$$\begin{aligned} 3x_1 &= 6 \\ 2x_1 - x_2 &= 0 \end{aligned} \iff \underbrace{\begin{bmatrix} 3 & 0 \\ 2 & -1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ 0 \end{bmatrix}}_{b} \tag{3.3}$$

$$\begin{aligned} 3x_1 &= 6 \\ 2x_2 &= -2 \\ x_1 - 2x_2 &= 2 \\ x_1 - x_3 + 2x_4 &= 10. \end{aligned} \iff \underbrace{\begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & -2 & 0 & 0 \\ 1 & 0 & -1 & 2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ -2 \\ 2 \\ 10 \end{bmatrix}}_{b} \tag{3.4}$$

The following systems of linear equations are square, but they are not lower triangular

$$\begin{aligned} 3x_1 &= 6 \\ 2x_1 - x_2 - \boxed{x_3} &= -2 \\ x_1 - 2x_2 + 3x_3 &= 2. \end{aligned} \iff \underbrace{\begin{bmatrix} 3 & 0 & 0 \\ 2 & -1 & \boxed{-1} \\ 1 & -2 & 3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ -2 \\ 2 \end{bmatrix}}_{b} \tag{3.5}$$

$$\begin{aligned} 3x_1 + \boxed{3x_2} &= 6 \\ 2x_1 - 2x_2 &= 0. \end{aligned} \iff \underbrace{\begin{bmatrix} 3 & \boxed{3} \\ 2 & -1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ 0 \end{bmatrix}}_{b} \tag{3.6}$$

The key ingredients of a square lower triangular system are

- The unknowns are ordered, as in $\begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}$ or $\begin{pmatrix} u & v & w & x & y & z \end{pmatrix}$
- The first equation only involves the first unknown.
- The second equation involves only the first two unknowns
- More generally, the i-th equation involves only the first i unknowns.
- The coefficients a_{ij} on or below the diagonal can be zero or non-zero.

From a matrix point of view, the condition is, all terms above the diagonal are zero. What does this mean? It means that A looks like this,

$$A = \begin{bmatrix} \mathbf{a_{11}} & 0 & 0 & 0 & \cdots & 0 & 0 \\ a_{21} & \mathbf{a_{22}} & 0 & 0 & \cdots & 0 & 0 \\ a_{31} & a_{32} & \mathbf{a_{33}} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 & 0 \\ \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ a_{(n-1)1} & a_{(n-1)2} & a_{(n-1)3} & a_{(n-1)4} & \cdots & \mathbf{a_{(n-1)(n-1)}} & 0 \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{n(n-1)} & \mathbf{a_{nn}} \end{bmatrix}, \tag{3.7}$$

namely, $a_{ij} = 0$ for $j > i$. Note that the diagonal is in bold and the terms above the diagonal are in red. The matrices in (3.2) through (3.4) are *lower triangular*, while the matrices in (3.5) and (3.6) are not lower triangular.

3.3 Determinant of a Lower Triangular Matrix

Highly Useful Fact: The matrix determinant of a square lower triangular matrix is equal to the product of the elements on the diagonal. For the matrix A in (3.7), its determinant is

$$\det(A) = a_{11} \cdot a_{22} \cdot \ldots \cdot a_{nn}. \tag{3.8}$$

Hence, for lower triangular matrices of size 10×10 or so, the determinant can be computed by inspection! Let's do a few:

$$\det \left(\begin{bmatrix} 3 & 0 & 0 \\ 2 & -1 & 0 \\ 1 & -2 & 3 \end{bmatrix} \right) = 3 \cdot (-1)3 = -9 \neq 0.$$

Hence, the system of equations (3.2) has a unique solution.

Let's now use the alternative notation for the determinant of a matrix,

$$\begin{vmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & -2 & 0 & 0 \\ 1 & 0 & -1 & 2 \end{vmatrix} = 3 \cdot 2 \cdot 0 \cdot 2 = 0.$$

Hence, the system of equations (3.4) is one of the problem cases: it may have no solution or it may have an infinite number of solutions.

Remark: For a square lower triangular matrix, the matrix determinant is nonzero if, and only if, all of the elements on the diagonal of the matrix are non-zero. Equivalently, for a square lower triangular matrix, the matrix determinant is zero if, and only if, at least one of the elements on the diagonal of the matrix is zero. In other words, we do not really need to multiply them out to check for $\det(A) \neq 0$, the condition for uniqueness of solutions of square systems of linear equations.

3.4 Lower Triangular Systems and Forward Substitution

We develop a solution to a lower triangular system by starting at the top and working our way to the bottom via a method called **forward substitution**. As an example, we use (3.2), which for convenience, we repeat here:

$$\begin{aligned} 3x_1 &= 6 \\ 2x_1 - x_2 &= -2 \\ x_1 - 2x_2 + 3x_3 &= 2 \end{aligned} \iff \underbrace{\begin{bmatrix} 3 & 0 & 0 \\ 2 & -1 & 0 \\ 1 & -2 & 3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ -2 \\ 2 \end{bmatrix}}_{b}. \tag{3.9}$$

Because we have ordered the variables as $\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}$, we isolate x_1 in the first equation, x_2 in the second equation, and x_3 in the third equation by moving the other variables to the right-hand side,

$$\begin{aligned} 3x_1 &= 6 \\ -x_2 &= -2 - 2x_1 \\ 3x_3 &= 2 - x_1 + 2x_2. \end{aligned} \tag{3.10}$$

You can see how the first equation is trivial, and so is the second one, once the first is solved, etc. Next, we can make the coefficients of the leading variables all equal to one by dividing through by the coefficients that multiply them, yielding

$$\begin{aligned} x_1 &= \frac{1}{3}6 = 2 \\ x_2 &= -\left[-2 - 2x_1\right] = 2 + 2x_1 \\ x_3 &= \frac{1}{3}\left[2 - x_1 + 2x_2\right]. \end{aligned} \tag{3.11}$$

Next, we substitute in, working from top to bottom:

$$\begin{aligned} x_1 &= \frac{1}{3}6 = 2 \\ x_2 &= 2 + 2x_1 = 6 \\ x_3 &= \frac{1}{3}\left[2 - x_1 + 2x_2\right] = \frac{1}{3}\left[12\right] = 4, \end{aligned} \tag{3.12}$$

The method is called **forward substitution** because once we have solved x_1, we take its value forward to the next equation where we solve for x_2, and once we have solved for x_1 and x_2, we take their values forward to the next equation where we solve for x_3. I am guessing that you see the pattern.

When can forward substitution go wrong? Well first of all, we only use it for lower triangular systems of linear equations. If the diagonal of the matrix corresponding to the system of linear equations has a zero on it, then the matrix determinant is zero ($\implies$ no solution or an infinite number of solutions) and forward substitution would lead us to **divide by zero, which we know is a major error in mathematics**.

As an example we take (3.4), which we repeat here

$$\begin{aligned} 3x_1 &= 6 \\ 2x_2 &= -2 \\ x_1 - 2x_2 &= 2 \\ x_1 - x_3 + 2x_4 &= 10. \end{aligned} \iff \underbrace{\begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & -2 & \boxed{0} & 0 \\ 1 & 0 & -1 & 2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ -2 \\ 2 \\ 10 \end{bmatrix}}_{b} \tag{3.13}$$

When we try to isolate x_3 in the third equation we have a problem. To make it more obvious, we make x_3 explicit with its corresponding coefficient of zero

$$\left.\begin{aligned} 3x_1 &= 6 \\ 2x_2 &= -2 \\ x_1 - 2x_2 + \boxed{0x_3} &= 2 \\ x_1 - x_3 + 2x_4 &= 10 \end{aligned}\right\} \Longrightarrow \left.\begin{aligned} 3x_1 &= 6 \\ 2x_2 &= -2 \\ \boxed{0x_3} &= 2 - (x_1 - 2x_2) \\ 2x_4 &= 10 - (x_1 - x_3) \end{aligned}\right\} \overset{???}{\Longrightarrow} \begin{aligned} x_1 &= 2 \\ x_2 &= -1 \\ x_3 &= \boxed{\frac{1}{0}} \cdot (-2) \;\; \text{?!? Divide by zero: not allowed!} \\ x_4 &= \frac{1}{4}(10 - 2 + \infty) \;\; \text{?!?!? } \textbf{Wrong!} \end{aligned} \tag{3.14}$$

3.5 Upper Triangular Systems, Upper Triangular Matrices, Determinants, and Back Substitution

This is an example of a square **upper triangular system of equations** and its corresponding **upper triangular matrix**

$$\begin{aligned} x_1 + 3x_2 + 2x_3 &= 6 \\ 2x_2 + x_3 &= -2 \\ 3x_3 &= 4, \end{aligned} \quad \Longleftrightarrow \quad \underbrace{\begin{bmatrix} 1 & 3 & 2 \\ 0 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ -2 \\ 4 \end{bmatrix}}_{b}. \tag{3.15}$$

We note that all of the coefficients of A *below the diagonal* are zero; that is $a_{ij} = 0$ for $i > j$.

Highly Useful Fact: The matrix determinant of a square upper triangular matrix is equal to the product of the elements on the diagonal. For the matrix A in (3.15), its determinant is

$$\det(A) = 1 \cdot 2 \cdot 3 = 6. \tag{3.16}$$

Hence, we know the system of equations has a unique solution.

We develop a solution by starting at the bottom and work our way to the top via **back substitution**. We first solve for the leading variables, and here, we do it in one step

$$\begin{aligned} x_1 &= 6 - (3x_2 + 2x_3) \\ x_2 &= \frac{1}{2}(-2 - x_3) \\ x_3 &= \frac{4}{3}, \end{aligned} \tag{3.17}$$

but of course, you can do it in two steps as we did for lower triangular systems. Next, we do back substitution, from bottom to top, sequentially plugging in numbers from the previous equations

$$\begin{array}{llll} x_1 & = 6 - (3x_2 + 2x_3) & = 6 - (3 \cdot (-\frac{5}{3}) + 2 \cdot \frac{4}{3}) & = \frac{18}{3} - \frac{7}{3} = \frac{11}{3} = 3\frac{2}{3} \\ x_2 & = \frac{1}{2} \cdot (-2 - x_3) & = \frac{1}{2} \cdot \left(-2 - \frac{4}{3}\right) & = \frac{1}{2} \cdot \left(-\frac{10}{3}\right) = -\frac{5}{3} = -1\frac{2}{3} \\ x_3 & = \frac{4}{3}. & & \end{array} \tag{3.18}$$

3.6 General Cases

The general form of a lower triangular system with a non-zero determinant is

$$\begin{aligned} a_{11}x_1 &= b_1 \quad (a_{11} \neq 0) \\ a_{21}x_1 + a_{22}x_2 &= b_2 \quad (a_{22} \neq 0) \\ \vdots &= \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n \quad (a_{nn} \neq 0) \end{aligned} \tag{3.19}$$

and the solution proceeds from top to bottom, like this

$$
\begin{aligned}
x_1 &= \frac{b_1}{a_{11}} \quad (a_{11} \neq 0) \\
x_2 &= \frac{b_2 - a_{21}x_1}{a_{22}} \quad (a_{22} \neq 0) \\
\vdots &= \vdots \\
x_n &= \frac{b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}}{a_{nn}} \quad (a_{nn} \neq 0).
\end{aligned}
\tag{3.20}
$$

The general form of an upper triangular system with a non-zero determinant is

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \quad (a_{11} \neq 0) \\
a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \quad (a_{22} \neq 0) \\
a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \quad (a_{33} \neq 0) \\
\vdots &= \vdots \\
a_{nn}x_n &= b_n \quad (a_{nn} \neq 0),
\end{aligned}
\tag{3.21}
$$

and the solution proceeds from bottom to top, like this,

$$
\begin{aligned}
x_1 &= \frac{b_1 - a_{12}x_2 - \cdots - a_{1n}x_n}{a_{11}} && (a_{11} \neq 0) \\
x_2 &= \frac{b_2 - a_{23}x_3 - \cdots - a_{2n}x_n}{a_{22}} && (a_{22} \neq 0) \\
\vdots &= \vdots && \vdots \\
x_{n-1} &= \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}} && (a_{n-1,n-1} \neq 0) \\
x_n &= \frac{b_n}{a_{nn}} && (a_{nn} \neq 0),
\end{aligned}
\tag{3.22}
$$

For those of you with a "visual memory", here is a graphical representation for upper and lower triangular matrices

$$
\text{(lower triangular)} \begin{bmatrix} \times & & & & \\ \times & \times & & \mathbf{0} & \\ \times & \times & \times & & \\ \times & \times & \times & \times & \\ \times & \times & \times & \times & \times \end{bmatrix} \qquad \begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & \mathbf{0} & & \times & \times \\ & & & & \times \end{bmatrix} \text{(upper triangular).} \tag{3.23}
$$

Lower: everything above the diagonal is zero. **Upper:** everything below the diagonal is zero. For us to be able to solve the equations for arbitrary values b on the right-hand side of $Ax = b$, we need the elements on the diagonal to be non-zero.

Remark: In HW you will develop Julia code to solve triangular systems of equations, whether upper or lower triangular. Your data will be given to you sometimes as systems of equations written out as formulas and sometimes directly as matrices. We'll also have you upload some BIG matrices and give it a go!

3.7 A Simple Trick with Systems of Equations: Re-arranging their Order

This system of equations is neither upper triangular nor lower triangular

$$
\begin{aligned}
3x_1 &= 6 \\
x_1 - 2x_2 + 3x_3 &= 2 \\
2x_1 - x_2 &= -2.
\end{aligned}
\tag{3.24}
$$

We can simply re-arrange the order of the equations to arrive at

$$\begin{aligned} 3x_1 &= 6 \\ 2x_1 - x_2 &= -2 \\ x_1 - 2x_2 + 3x_3 &= 2, \end{aligned} \tag{3.25}$$

which happens to be lower triangular. We still have the same equations and hence their solution has not changed. The equations have just been re-arranged to make the process of computing their solution fall into a nice pattern that we already know how to handle.

This is a really useful trick in mathematics: re-arranging a set of equations without changing the answer. Right here, it may seem kind of trivial, but when we look at this in terms of matrices in the next Chapter, we get a cool piece of insight.

3.8 Looking Ahead

Once again, our goal is to solve very large sets of linear equations, say more than 100 variables. In this chapter, we saw how to solve problems with triangular structure. Our next major way point is to reduce all linear systems of n-equations in n-variables to being solvable with forward substitution and back substitution. To get there we need to understand:

- the standard way to multiply two matrices
- a little known way to multiply two matrices
- how to *factor* a square matrix as the product of two triangular matrices

Chapter 4

Matrix Multiplication

Learning Objectives

- How to partition matrices into rows and columns
- How to multiply two matrices
- How to swap rows of a matrix

Outcomes

- Multiplying a row vector by a column vector.
- Recognizing the rows and columns of a matrix
- Standard definition of matrix multiplication $A \cdot B$ using the rows of A and the columns of B
- Size restrictions when multiplying matrices
- Examples that work and those that don't because the sizes are wrong
- Introduce a second way of computing the product of two matrices using the columns of A and the rows of B. This will later be used to compute the LU decomposition in a very simple way.
- Permutation matrices

4.1 Multiplying a Row Vector by a Column Vector

Let $a^{\rm row} = \begin{bmatrix} a_1 & a_2 & \cdots & a_k \end{bmatrix}$ be a row vector with k elements and let $b^{\rm col} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$ be a column vector with the same number of elements as $a^{\rm row}$. The **product** of $a^{\rm row}$ and $b^{\rm col}$ is defined[1] as

$$a^{\rm row} \cdot b^{\rm col} := \sum_{i=1}^{k} a_i b_i := a_1 b_1 + a_2 b_2 + \cdots + a_k b_k. \tag{4.1}$$

For many, the following visual representation is more understandable,

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_k \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} := a_1 b_1 + a_2 b_2 + \cdots + a_k b_k. \tag{4.2}$$

Remarks on Multiplying a Row Vector Times a Column Vector

- It is very important to observe that the two vectors MUST have the same number of elements for their product to be defined.
- While (4.1) and (4.2) are equivalent (meaning they represent the same mathematical information), the formula in (4.1) is more convenient when writing a program and the visual representation in (4.2) is more convenient for hand computations.
- At this point, a column vector "times" a row vector is not defined. When we do make sense of it, it will not be equal to $\sum_{i=1}^{k} a_i b_i$.

Example 4.1 *Let $a^{\rm row} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ be a row vector with $k = 3$ elements and let $b^{\rm col} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$ be a column vector with $k = 3$ elements. Perform their multiplication if it makes sense.*

Solution: Because they have the same number of elements, we can form their product and we compute

$$a^{\rm row} \cdot b^{\rm col} := \sum_{i=1}^{3} a_i b_i = (1)(4) + (2)(5) + (3)(6) = 32,$$

or we can look at it in the form

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = (1)(4) + (2)(5) + (3)(6) = 32.$$

■

Example 4.2 *Let $a^{\rm row} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ be a row vector with $k = 3$ elements and let $b^{\rm col} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$ be a column vector with $k = 2$ elements. Perform their multiplication if it makes sense.*

Solution: Because they do NOT have the same number of elements, we cannot form their product.

$$a^{\rm row} \cdot b^{\rm col} := \text{undefined},$$

or

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 5 \end{bmatrix} = \text{error!}$$

■

[1] If you have not seen the "summation" symbol before, here are some examples: $\sum_{i=1}^{3} i := 1 + 2 + 3$, $\sum_{i=5}^{8} i := 5 + 6 + 7 + 8$, and $\sum_{k=1}^{n} k^2 := 1 + (2)^2 + (3)^2 + \cdots + (n)^2$.

Example 4.3 *Let* $a^{\rm row} = \begin{bmatrix} 2 & -3 & -1 & 11 \end{bmatrix}$ *be a row vector with* $k = 4$ *elements and let* $b^{\rm col} = \begin{bmatrix} 3 \\ 5 \\ -1 \\ -2 \end{bmatrix}$ *be a column vector with* $k = 4$ *elements. Perform their multiplication if it makes sense.*

Solution: Because they have the same number of elements, we can form their product and we compute

$$a^{\rm row} \cdot b^{\rm col} := \sum_{i=1}^{4} a_i b_i = (2)(3) + (-3)(5) + (-1)(-1) + (11)(-2) = -30,$$

or, equivalently, we write it like this

$$\begin{bmatrix} 2 & -3 & -1 & 11 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 5 \\ -1 \\ -2 \end{bmatrix} = (2)(3) + (-3)(5) + (-1)(-1) + (11)(-2) = -30.$$

■

4.2 Examples of Row and Column Partitions

Let $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ be a 2×3 matrix. Then a **partition** of A **into rows is**

$$\begin{bmatrix} a_1^{\rm row} \\ a_2^{\rm row} \end{bmatrix} = \begin{bmatrix} \boxed{1 \;\; 2 \;\; 3} \\ \\ \boxed{4 \;\; 5 \;\; 6} \end{bmatrix}, \quad \text{that is,} \quad \begin{aligned} a_1^{\rm row} &= [1 \;\; 2 \;\; 3] \\ a_2^{\rm row} &= [4 \;\; 5 \;\; 6]. \end{aligned}$$

We note that $a_1^{\rm row}$ and $a_2^{\rm row}$ are row vectors of size 1×3; they have the same number of entries as A has columns.

A partition of $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ **into columns is**

$$\begin{bmatrix} a_1^{\rm col} & a_2^{\rm col} & a_3^{\rm col} \end{bmatrix} = \left[\boxed{\begin{matrix} 1 \\ 4 \end{matrix}} \;\; \boxed{\begin{matrix} 2 \\ 5 \end{matrix}} \;\; \boxed{\begin{matrix} 3 \\ 6 \end{matrix}}\right], \quad \text{that is,} \quad a_1^{\rm col} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \; a_2^{\rm col} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}, \; a_3^{\rm col} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

We note that $a_1^{\rm col}$, $a_2^{\rm col}$, and $a_3^{\rm col}$ are column vectors of size 2×1; they have the same number of entries as A has rows.

4.3 General Case of Partitions

Let A be an $n \times m$ matrix. We recall that n is the number of rows in A, m is the number of columns, and a_{ij} is the notation for the ij element of A, that is, its value on the i-th row and j-th column. **A partition of** A **into rows** is

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} = \begin{bmatrix} a_1^{\rm row} \\ a_2^{\rm row} \\ \vdots \\ a_n^{\rm row} \end{bmatrix} = \begin{bmatrix} \boxed{a_{11} \;\; a_{12} \;\; \cdots \;\; a_{1m}} \\ \boxed{a_{21} \;\; a_{22} \;\; \cdots \;\; a_{2m}} \\ \vdots \\ \boxed{a_{n1} \;\; a_{n2} \;\; \cdots \;\; a_{nm}} \end{bmatrix}.$$

That is, the i-th row is the $1 \times m$ row vector

$$a_i^{\rm row} = [a_{i1} \;\; a_{i2} \;\; \cdots \;\; a_{im}],$$

where i varies from 1 to n.

A partition of A into columns is

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} = \begin{bmatrix} a_1^{\rm col} & a_2^{\rm col} & \cdots & a_m^{\rm col} \end{bmatrix} = \left[\boxed{\begin{matrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{matrix}} \; \boxed{\begin{matrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{matrix}} \; \cdots \; \boxed{\begin{matrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{matrix}} \right].$$

That is, the j-th column is the $n \times 1$ column vector

$$a_j^{\rm col} = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{nj} \end{bmatrix},$$

where j varies from 1 to m.

4.4 Standard Matrix Multiplication: It's All About Rows and Columns

Let A be an $n \times k$ matrix, meaning it has n rows and k columns, and let B be a $k \times m$ matrix, so that it has k rows and m columns. When the number of columns of the first matrix A equals the number of rows of the second matrix B, the **matrix product** of A and B is defined and results in an $n \times m$ matrix:

$$[n \times k \;\; \text{matrix}] \cdot [k \times m \;\; \text{matrix}] = [n \times m \;\; \text{matrix}].$$

The values of n, m, and k can be any integers greater than or equal to one. In particular, they can all be different numbers.

- $[4 \times 2 \;\; \text{matrix}] \cdot [2 \times 5 \;\; \text{matrix}] = [4 \times 5 \;\; \text{matrix}]$.
- $[1 \times 11 \;\; \text{matrix}] \cdot [11 \times 1 \;\; \text{matrix}] = [1 \times 1 \;\; \text{matrix}]$, which in Julia is different than a scalar.
- $[4 \times 4 \;\; \text{matrix}] \cdot [4 \times 4 \;\; \text{matrix}] = [4 \times 4 \;\; \text{matrix}]$.
- $[4 \times 3 \;\; \text{matrix}] \cdot [4 \times 4 \;\; \text{matrix}] =$ **undefined**.

Matrix multiplication using rows of A and columns of B

The standard way of doing **matrix multiplication** $A \cdot B$ involves multiplying the rows of A with the columns of B. We do some small examples before giving the general formula. **We will observe that the order in which we multiply two matrices is very important: in general, A $\cdot$ B $\neq$ B $\cdot$ A even when A and B are square matrices of the same size.**

4.4.1 Examples

Example 4.4 *We consider a 2×2 matrix A and a 2×1 matrix B, where we partition A by rows and B by columns. We note that the number of columns of A matches the number of rows of B so their product $A \cdot B$ is supposed to be defined. Let's do it!*

Solution:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a_1^{\rm row} \\ a_2^{\rm row} \end{bmatrix} = \begin{bmatrix} \boxed{1 \;\; 2} \\ \boxed{3 \;\; 4} \end{bmatrix} \text{ and } B = \begin{bmatrix} 5 \\ 6 \end{bmatrix} = \begin{bmatrix} b_1^{\rm col} \end{bmatrix} = \left[\boxed{\begin{matrix} 5 \\ 6 \end{matrix}} \right]. \tag{4.3}$$

The matrix product of A and B is

$$A \cdot B = \begin{bmatrix} a_1^{\rm row} \\ a_2^{\rm row} \end{bmatrix} \cdot \begin{bmatrix} b_1^{\rm col} \end{bmatrix} := \begin{bmatrix} a_1^{\rm row} \cdot b_1^{\rm col} \\ a_2^{\rm row} \cdot b_1^{\rm col} \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix} \tag{4.4}$$

because

$$a_1^{\text{row}} \cdot b_1^{\text{col}} = \begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix} = 5 + 12 = 17$$
$$a_2^{\text{row}} \cdot b_1^{\text{col}} = \begin{bmatrix} 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix} = 15 + 24 = 39.$$

A more visual way to do the multiplication is like this,

$$A \cdot B = \begin{bmatrix} \boxed{1\ \ 2} \\ \boxed{3\ \ 4} \end{bmatrix} \cdot \begin{bmatrix} \boxed{\begin{matrix} 5 \\ 6 \end{matrix}} \end{bmatrix} = \begin{bmatrix} (1)(5) + (2)(6) \\ (3)(5) + (4)(6) \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix} \tag{4.5}$$

■

Remarks

- A 2×2 matrix times a 2×1 matrix yields a 2×1 matrix.
- Let's observe that the operations for computing the matrix product boil down to performing the products of row vectors and column vectors of equal lengths! This is a general fact, as we will see.
- Let's also recall that we do not know how to form the product of a row vector with a column vector when their lengths are not equal.
- The number elements in $a_\bullet^{\text{row}}$ is equal to the number of columns of A, while the number of elements in $b_\bullet^{\text{col}}$ is equal to the number of rows of B. This is why the number of columns of A must equal the number of rows of B for their matrix product $A \cdot B$ to be defined.

Example 4.5 *We reuse A and B above and ask if we can form the matrix product in the order $B \cdot A$.*

Solution: We have that the first matrix B is 2×1 and the second matrix A is 2×2. The number of columns of the first matrix does not match the number of rows of the second matrix, and hence the product cannot be defined in this direction. ■

Example 4.6 *We consider a 2×2 matrix A and a 2×2 matrix B. We note that the number of columns of A matches the number of rows of B so their product $A \cdot B$ is defined. We also note that the number columns of B matches the number of rows of A, so the product $B \cdot A$ is also defined. This is a general property of square matrices A and B of the same size: their matrix product can be performed in either order.* ***We will note, however, that* $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{B} \cdot \mathbf{A}$*. Hence, when multiplying matrices, the order matters!***

Solution:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a_1^{\text{row}} \\ a_2^{\text{row}} \end{bmatrix} = \begin{bmatrix} \boxed{1\ \ 2} \\ \boxed{3\ \ 4} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 5 & -2 \\ 6 & 1 \end{bmatrix} = \begin{bmatrix} b_1^{\text{col}} & b_2^{\text{col}} \end{bmatrix} = \begin{bmatrix} \boxed{\begin{matrix} 5 \\ 6 \end{matrix}} & \boxed{\begin{matrix} -2 \\ 1 \end{matrix}} \end{bmatrix}. \tag{4.6}$$

The matrix product of A and B is

$$A \cdot B = \begin{bmatrix} \boxed{1\ \ 2} \\ \boxed{3\ \ 4} \end{bmatrix} \cdot \begin{bmatrix} \boxed{\begin{matrix} 5 \\ 6 \end{matrix}} & \boxed{\begin{matrix} -2 \\ 1 \end{matrix}} \end{bmatrix} = \begin{bmatrix} (1)(5) + (2)(6) & (1)(-2) + (2)(1) \\ (3)(5) + (4)(6) & (3)(-2) + (4)(1) \end{bmatrix} = \begin{bmatrix} 11 & 0 \\ 39 & -2 \end{bmatrix}. \tag{4.7}$$

The matrix product of B and A is

$$B \cdot A = \begin{bmatrix} \boxed{5\ -2} \\ \boxed{6\ \ \ 1} \end{bmatrix} \cdot \begin{bmatrix} \boxed{\begin{matrix} 1 \\ 3 \end{matrix}} & \boxed{\begin{matrix} 2 \\ 4 \end{matrix}} \end{bmatrix} = \begin{bmatrix} (5)(1) + (-2)(3) & (5)(2) + (-2)(4) \\ (6)(1) + (1)(3) & (6)(2) + (1)(4) \end{bmatrix} = \begin{bmatrix} -1 & 2 \\ 9 & 16 \end{bmatrix}. \tag{4.8}$$

■

Order Matters When Multiplying Matrices

- We note that $\begin{bmatrix} 11 & 0 \\ 39 & -2 \end{bmatrix} = A \cdot B \neq B \cdot A = \begin{bmatrix} -1 & 2 \\ 9 & 16 \end{bmatrix}$. The order in which matrices are multiplied matters. This is very different from the order of two real numbers, such as π and $\sqrt{2}$, which you can multiply in either order and you always get the same answer!
- $A \cdot B := \begin{bmatrix} a_1^{\rm row} \cdot b_1^{\rm col} & a_1^{\rm row} \cdot b_2^{\rm col} \\ a_2^{\rm row} \cdot b_1^{\rm col} & a_2^{\rm row} \cdot b_2^{\rm col} \end{bmatrix}$, where you can find $a_i^{\rm row}$ and $b_j^{\rm col}$ highlighted in (4.6).
- $B \cdot A := \begin{bmatrix} b_1^{\rm row} \cdot a_1^{\rm col} & b_1^{\rm row} \cdot a_2^{\rm col} \\ b_2^{\rm row} \cdot a_1^{\rm col} & b_2^{\rm row} \cdot a_2^{\rm col} \end{bmatrix}$, where you can find $b_i^{\rm row}$ and $a_j^{\rm col}$ highlighted in (4.8).

Example 4.7 *For the given* 3×2 *matrix* A *and* 2×2 *matrix* B*, compute* $A \cdot B$ *and* $B \cdot A$ *if the given multiplications make sense.*

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \text{ and } B = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}.$$

Solution: We note that the number of columns of A equals the number of rows of B so their product $A \cdot B$ is defined. We also note that the number columns of B does not equal the number of rows of A, so the product $B \cdot A$ is not defined.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} a_1^{\rm row} \\ a_2^{\rm row} \\ a_3^{\rm row} \end{bmatrix} = \begin{bmatrix} \boxed{1\ 2} \\ \boxed{3\ 4} \\ \boxed{5\ 6} \end{bmatrix} \text{ and } B = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} b_1^{\rm col} & b_2^{\rm col} \end{bmatrix} = \begin{bmatrix} \boxed{\begin{matrix} 3 \\ 2 \end{matrix}} & \boxed{\begin{matrix} 4 \\ 1 \end{matrix}} \end{bmatrix}$$

By now, you may have a favorite method, and using it, you should compute that

$$A \cdot B = \begin{bmatrix} 7 & 6 \\ 17 & 16 \\ 27 & 26 \end{bmatrix}$$

■

Remark: When doing calculations by hand, 3×3 matrices are about as big as you ever really want to do. In Julia, "the sky is the limit". The following section is to help us understand what Julia is doing when it multiplies two matrices with the command `A * B`.

4.4.2 Optional Read: General case: what is happening inside Julia

We partition the $n \times k$ matrix A into rows and the $k \times m$ matrix B into columns, as in

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix} = \begin{bmatrix} a_1^{\rm row} \\ a_2^{\rm row} \\ \vdots \\ a_n^{\rm row} \end{bmatrix} = \begin{bmatrix} \boxed{a_{11}\ a_{12}\ \cdots\ a_{1k}} \\ \boxed{a_{21}\ a_{22}\ \cdots\ a_{2k}} \\ \vdots \\ \boxed{a_{n1}\ a_{n2}\ \cdots\ a_{nk}} \end{bmatrix} \tag{4.9}$$

and

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k1} & b_{k2} & \cdots & b_{km} \end{bmatrix} = \begin{bmatrix} b_1^{\rm col} & b_2^{\rm col} & \cdots & b_m^{\rm col} \end{bmatrix} = \begin{bmatrix} \boxed{\begin{matrix} b_{11} \\ b_{21} \\ \vdots \\ b_{k1} \end{matrix}} & \boxed{\begin{matrix} b_{12} \\ b_{22} \\ \vdots \\ b_{k2} \end{matrix}} & \cdots & \boxed{\begin{matrix} b_{1m} \\ b_{2m} \\ \vdots \\ b_{km} \end{matrix}} \end{bmatrix}, \tag{4.10}$$

then

$$A \cdot B := \begin{bmatrix} a_1^{\rm row} \cdot b_1^{\rm col} & a_1^{\rm row} \cdot b_2^{\rm col} & \cdots & a_1^{\rm row} \cdot b_m^{\rm col} \\ a_2^{\rm row} \cdot b_1^{\rm col} & a_2^{\rm row} \cdot b_2^{\rm col} & \cdots & a_2^{\rm row} \cdot b_m^{\rm col} \\ \vdots & \vdots & \ddots & \vdots \\ a_n^{\rm row} \cdot b_1^{\rm col} & a_n^{\rm row} \cdot b_2^{\rm col} & \cdots & a_n^{\rm row} \cdot b_m^{\rm col} \end{bmatrix}. \tag{4.11}$$

Another way to see the pattern is like this

$$A\cdot B := \begin{bmatrix} \boxed{a_{11}\ \ a_{12}\ \ \cdots\ \ a_{1k}} \\ \boxed{a_{21}\ \ a_{22}\ \ \cdots\ \ a_{2k}} \\ \vdots \\ \boxed{a_{n1}\ \ a_{n2}\ \ \cdots\ \ a_{nk}} \end{bmatrix} \cdot \begin{bmatrix} \boxed{\begin{matrix} b_{11} \\ b_{21} \\ \vdots \\ b_{k1} \end{matrix}} & \boxed{\begin{matrix} b_{12} \\ b_{22} \\ \vdots \\ b_{k2} \end{matrix}} & \cdots & \boxed{\begin{matrix} b_{1m} \\ b_{2m} \\ \vdots \\ b_{km} \end{matrix}} \end{bmatrix} = \begin{bmatrix} \sum\limits_{i=1}^{k} a_{1i}b_{i1} & \sum\limits_{i=1}^{k} a_{1i}b_{i2} & \cdots & \sum\limits_{i=1}^{k} a_{1i}b_{im} \\ \sum\limits_{i=1}^{k} a_{2i}b_{i1} & \sum\limits_{i=1}^{k} a_{2i}b_{i2} & \cdots & \sum\limits_{i=1}^{k} a_{2i}b_{im} \\ \vdots & \vdots & \ddots & \vdots \\ \sum\limits_{i=1}^{k} a_{ni}b_{i1} & \sum\limits_{i=1}^{k} a_{ni}b_{i2} & \cdots & \sum\limits_{i=1}^{k} a_{ni}b_{im} \end{bmatrix}. \tag{4.12}$$

Bottom Line on Standard Multiplication

In the standard way of defining matrix multiplication, the ij-entry of $A\cdot B$ is obtained by multiplying the i-th row of A by the j-th column of B (whenever the multiplication makes sense, meaning that the number of columns of A equals the number of rows of B). We will not use the following notation on a regular basis, but some of you make like it; if we let $[A\cdot B]_{ij}$ denote the ij-element of the matrix, $A\cdot B$, then

$$[A\cdot B]_{ij} := a_i^{\text{row}} \cdot b_j^{\text{col}}.$$

4.5 Multiplication by Summing over Columns and Rows

Rock your World: an Alternative Formula for Matrix Multiplication

- We now introduce an alternative way to compute the product of two matrices. It gives the same answer as the "standard method". **Very few people use or even know this definition because, for hand calculations, it takes more time to write out the individual steps.** ROB 101, however, is about computational linear algebra, and hence we ignore such trivial concerns as what is best for doing hand calculations!
- We will see shortly that understanding this alternative way of matrix multiplication will allow us to solve almost any system of linear equations via a combination of forward and back substitution.
- Instead of solving one hard system of linear equations, we will find the solution by solving two triangular systems of linear equations, one upper triangular and one lower triangular!

We reconsider two matrices A and B from Example 4.7, but this time we partition A into columns and B into rows

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} a_1^{\text{col}} & a_2^{\text{col}} \end{bmatrix} = \begin{bmatrix} \boxed{\begin{matrix} 1 \\ 3 \\ 5 \end{matrix}} & \boxed{\begin{matrix} 2 \\ 4 \\ 6 \end{matrix}} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} b_1^{\text{row}} \\ b_2^{\text{row}} \end{bmatrix} = \begin{bmatrix} \boxed{3\ \ 4} \\ \boxed{2\ \ 1} \end{bmatrix}.$$

What happens when we form the sum of the columns of A times the rows of B,

$$a_1^{\text{col}} b_1^{\text{row}} + a_2^{\text{col}} b_2^{\text{row}} = \textbf{?}$$

We first observe that the columns of A are 3×1 while the rows of B are 1×2. Hence, their product with the usual rules of matrix multiplication will be 3×2, the same size as $A\cdot B$. That seems interesting. We do the two multiplications and obtain

$$a_1^{\text{col}} \cdot b_1^{\text{row}} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \cdot \begin{bmatrix} 3 & 4 \end{bmatrix} = \begin{bmatrix} (1)\cdot(3) & (1)\cdot(4) \\ (3)\cdot(3) & (3)\cdot(4) \\ (5)\cdot(3) & (5)\cdot(4) \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 9 & 12 \\ 15 & 20 \end{bmatrix}$$

$$a_2^{\text{col}} \cdot b_2^{\text{row}} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} (2)\cdot(2) & (2)\cdot(1) \\ (4)\cdot(2) & (4)\cdot(1) \\ (6)\cdot(2) & (6)\cdot(1) \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 8 & 4 \\ 12 & 6 \end{bmatrix}$$

and hence their sum is

$$a_1^{\rm col} b_1^{\rm row} + a_2^{\rm col} b_2^{\rm row} = \begin{bmatrix} 3 & 4 \\ 9 & 12 \\ 15 & 20 \end{bmatrix} + \begin{bmatrix} 4 & 2 \\ 8 & 4 \\ 12 & 6 \end{bmatrix} = \begin{bmatrix} 7 & 6 \\ 17 & 16 \\ 27 & 26 \end{bmatrix} = A \cdot B$$

So while the bookkeeping is a bit different, the individual computations are easier than in the standard way of performing matrix multiplication, and perhaps they are also easier to get right. You're free to do your matrix multiplications as you wish. This particular method has a theoretical benefit that we will uncover shortly. Keep in mind that not so many people think of matrix multiplication as "sums of columns times rows" and you might confuse friends and other instructors when you talk about it in this manner. In fact, you may be told that you are wrong and that no one ever does it that way, even if it is correct.

General Case of Matrix Multiplication using Columns of A and Rows of B

Suppose that A is $n \times k$ and B is $k \times m$ so that the two matrices are compatible for matrix multiplication. Then

$$A \cdot B = \sum_{i=1}^{k} a_i^{\rm col} \cdot b_i^{\rm row},$$

the "sum of the columns of A multiplied by the rows of B". A more precise way to say it would be "the sum over i of the i-th column of A times the i-th row of B." In HW, we'll play with this idea enough that it will become ingrained into your subconscious!

4.6 The Matrix View of Swapping the Order of Equations: Permutation Matrices

Back in Sec. 3.7, we made a brief remark on swapping rows of equations to put them into a nicer form. For simplicity, we recall the equations again here.

This system of equations is neither upper triangular nor lower triangular

$$\begin{aligned} 3x_1 &= 6 \\ x_1 - 2x_2 + 3x_3 &= 2 \\ 2x_1 - x_2 &= -2, \end{aligned} \tag{4.13}$$

but if we simply re-arrange the order of the equations, we arrive at the lower triangular equations

$$\begin{aligned} 3x_1 &= 6 \\ 2x_1 - x_2 &= -2 \\ x_1 - 2x_2 + 3x_3 &= 2. \end{aligned} \tag{4.14}$$

As we noted before, we still have the same equations and hence their solution has not changed. The equations have just been re-arranged to make the process of computing their solution fall into a nice pattern that we already know how to handle.

We now write out the matrix equations for the "unfortunately ordered" equations (4.13) and then the "nicely" re-arranged system of equations (4.14)

$$\underbrace{\begin{bmatrix} 3 & 0 & 0 \\ 1 & -2 & 3 \\ 2 & -1 & 0 \end{bmatrix}}_{A_O} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ 2 \\ -2 \end{bmatrix}}_{b_O} \qquad \underbrace{\begin{bmatrix} 3 & 0 & 0 \\ 2 & -1 & 0 \\ 1 & -2 & 3 \end{bmatrix}}_{A_L} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 6 \\ -2 \\ 2 \end{bmatrix}}_{b_L}.$$

We see the second and third rows are swapped when we compare A_O to A_L and b_O to b_L. The swapping of rows can be accomplished by multiplying A_O and b_O on the left by

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

Indeed, we check that

$$P \cdot A_O = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 1 & -2 & 3 \\ 2 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 0 \\ 2 & -1 & 0 \\ 1 & -2 & 3 \end{bmatrix} = A_L$$

and

$$P \cdot b_O = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 6 \\ 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 6 \\ -2 \\ 2 \end{bmatrix} = b_L.$$

P is called a permutation matrix.

Key Insight on Swapping Rows

We note that the permutation matrix P is constructed from the 3×3 identity matrix I by swapping its second and third rows, exactly the rows we wanted to swap in A_O and b_O. This observation works in general.

For example, suppose we want to do the following rearrangement

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \leftrightarrow \begin{bmatrix} 4 \\ 2 \\ 5 \\ 1 \\ 3 \end{bmatrix}, \tag{4.15}$$

where rows one and four are swapped and three and five are swapped. We have shown the arrow as going back and forth ($\leftrightarrow$) because applying the swap twice results in the original arrangement. To see the resulting structure at a matrix level, we put the 5×5 identity matrix on the left and the permutation matrix P on the right

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \leftrightarrow P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

so that it is apparent that P is just a re-ordering of the rows of I. Moreover, if we want to go from

$$\begin{bmatrix} 4 \\ 2 \\ 5 \\ 1 \\ 3 \end{bmatrix} \leftrightarrow \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix},$$

we can apply the matrix, P, once again. Indeed, you are invited to check in Julia that $P \cdot P = I$.

The kind of permutation matrix we have introduced here, where permuting the rows twice results in the original ordering of the rows, is a special case of a more general kind of permutation. While we do not need the more general matrices in ROB 101, we feel you need to know that they exist.

Heads Up! The above is only half the story on permutation matrices.

Not needed in ROB 101: A more general rearrangement of rows would be something like this

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \rightarrow \begin{bmatrix} 4 \\ 2 \\ 1 \\ 5 \\ 3 \end{bmatrix}, \tag{4.16}$$

where applying the rearrangement twice does not result in the original order. **We do not really need this level of generality in ROB 101. So you can stop reading here, if you wish.**

To see how this looks at the matrix level, we put the 5×5 identity matrix on the left and the permutation matrix P on the right

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

P is still just a re-ordering of the rows of I, but it is no longer true that $P \cdot P = I$.

Permutation Matrices, the Full Story

In general, matrices that consist of all ones and zeros, with each row and column having a single one, are called **permutation matrices**.

In Chapter 6.4, we'll cover the matrix transpose, $P^\top$. Indeed, you are invited to look ahead and see what that is about. If you do so, you can check in Julia that $P \neq P^\top$ (we'll say that the matrix is not symmetric) and $P \cdot P \neq I$, but that $P^\top \cdot P = I$.

In Chapter 6.2, we introduce the matrix inverse, denoted P^{-1}. In general, inverses of matrices do not exist and they are hard to compute. For permutation matrices, $P^{-1} = P^\top$. They automatically satisfy $P^\top \cdot P = P \cdot P^\top = I$.

4.7 Looking Ahead

Once again, our goal is to solve very large sets of linear equations, say more than 100 variables. In the previous chapter, we saw how to solve problems with triangular structure. In this Chapter, we learned how to multiply two matrices.

Let's use our knowledge and see what happens when we multiply a lower triangular matrix times an upper triangular matrix. We define two matrices by L and U for lower and upper triangular, respectively,

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -2 & 1 \end{bmatrix} \text{ and } U = \begin{bmatrix} 3 & 3 & 2 \\ 0 & 6 & 1 \\ 0 & 0 & -3 \end{bmatrix} \implies L \cdot U = \begin{bmatrix} 3 & 3 & 2 \\ -6 & 0 & -3 \\ 9 & -3 & 1 \end{bmatrix} =: A.$$

Hence the product of a lower triangular matrix and an upper triangular matrix seems to be a "general matrix" A, meaning it has no particular structure.

Question: Is it possible to go backwards? That is, starting with a general matrix A, it is (always) possible to write it as the product of a lower triangular matrix and an upper triangular matrix? And if it is even possible, is it useful? What's the trick?

Answers: Almost always. Very! Our alternative way of doing matrix multiplication.

Chapter 5

LU (Lower-Upper) Factorization

Learning Objectives

- How to reduce a hard problem to two much easier problems
- The concept of "factoring" a matrix into a product of two simpler matrices that are in turn useful for solving systems of linear equations.

Outcomes

- Our first encounter in lecture with an explicit algorithm
- Learn how to do a *special case* of the LU decomposition, where L is a lower triangular matrix and U is an upper triangular matrix.
- Use the LU decomposition to solve linear equations
- More advanced: what we missed in our first pass at LU factorization: a (row) permutation matrix.

5.1 Recalling Forward and Back Substitution

For a lower triangular system of equations with non-zero leading coefficients, such as

$$\begin{aligned} 3x_1 &= 6 \\ 2x_1 - x_2 &= -2 \\ x_1 - 2x_2 + 3x_3 &= 2, \end{aligned} \iff \begin{bmatrix} 3 & 0 & 0 \\ 2 & -1 & 0 \\ 1 & -2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -2 \\ 2 \end{bmatrix} \tag{5.1}$$

we can find a solution via forward substitution,

$$\begin{aligned} x_1 &= \frac{1}{3}6 = 2 \\ x_2 &= -\left[-2 - 2x_1\right] \\ x_3 &= \frac{1}{3}\left[2 - x_1 + 2x_2\right], \end{aligned} \implies \begin{cases} x_1 = \frac{1}{3}6 = 2 \\ x_2 = 2 + 2x_1 = 2 + 2(2) = 6 \\ x_3 = \frac{1}{3}\left[2 - x_1 + 2x_2\right] = \frac{1}{3}\left[2 - (2) + 2(6)\right] = \frac{12}{3} = 4. \end{cases} \tag{5.2}$$

On the other hand, for an upper triangular system of equations with non-zero leading coefficients, such as

$$\begin{aligned} x_1 + 3x_2 + 2x_3 &= 6 \\ 2x_2 + x_3 &= -2 \\ 3x_3 &= 4, \end{aligned} \iff \begin{bmatrix} 1 & 3 & 2 \\ 0 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -2 \\ 4 \end{bmatrix} \tag{5.3}$$

we can find a solution by back substitution,

$$\begin{aligned} x_1 &= 6 - (3x_2 + 2x_3) \\ x_2 &= \frac{1}{2}\left(-2 - x_3\right) \\ x_3 &= \frac{4}{3}, \end{aligned} \implies \begin{cases} x_1 = 6 - (3x_2 + 2x_3) = 6 - (\frac{3}{3} + \frac{8}{3}) = \frac{18}{3} - \frac{11}{3} = 2\frac{1}{3} \\ x_2 = \frac{1}{2}\left(-2 - x_3\right) = \frac{1}{2}\left(-2 - \frac{4}{3}\right) = \frac{1}{2}\left(\frac{2}{3}\right) = \frac{1}{3} \\ x_3 = \frac{4}{3}. \end{cases} \tag{5.4}$$

5.2 Recalling Matrix Multiplication in the Form of Columns Times Rows

Suppose that A is $n \times k$ and B is $k \times m$ so that the two matrices are compatible for matrix multiplication. Then

$$A \cdot B = \sum_{i=1}^{k} a_i^{\text{col}} \cdot b_i^{\text{row}},$$

the "sum of the columns of A multiplied by the rows of B".

Example 5.1 *Form the matrix product of* $A = \begin{bmatrix} 1 & 0 \\ 3 & 4 \end{bmatrix}$ *and* $B = \begin{bmatrix} 5 & 2 \\ 0 & -1 \end{bmatrix}$.

Solution

$$A = \begin{bmatrix} 1 & 0 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a_1^{\text{col}} & a_2^{\text{col}} \end{bmatrix} = \left[\boxed{\begin{matrix} 1 \\ 3 \end{matrix}}\ \ \boxed{\begin{matrix} 0 \\ 4 \end{matrix}}\right] \text{ and } B = \begin{bmatrix} 5 & 2 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} b_1^{\text{row}} \\ b_2^{\text{row}} \end{bmatrix} = \begin{bmatrix} \boxed{5 \quad 2} \\ \boxed{0 \ -1} \end{bmatrix}$$

$$a_1^{\text{col}} b_1^{\text{row}} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 5 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 15 & 6 \end{bmatrix}$$

$$a_2^{\text{col}} b_2^{\text{row}} = \begin{bmatrix} 0 \\ 4 \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & -4 \end{bmatrix}$$

and the matrix product is

$$A \cdot B = a_1^{\text{col}} b_1^{\text{row}} + a_2^{\text{col}} b_2^{\text{row}} = \begin{bmatrix} 5 & 2 \\ 15 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -4 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 15 & 2 \end{bmatrix}$$

■

5.3 LU (Lower-Upper) Factorization (without row permutations)

As a lead in to our main topic, note that in Example 5.1, A is a lower triangular matrix, B is an upper triangular matrix, while their product $A \cdot B$ is neither. Can this process be reversed? That is, given a generic square matrix, can we factor it as the product of a lower-triangular matrix and an upper-triangular matrix? And even if we can do such a factorization, would it be helpful?

We'll delay an explicit answer to the question of utility because you have a sense already that triangular matrices make your life easier! Let's consider the factorization question for

$$M = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix}.$$

We define C_1 and R_1 to be the first column of M and the first row of M, respectively, that is

$$C_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ and } R_1 = \begin{bmatrix} 1 & 4 & 5 \end{bmatrix}.$$

Then

$$\begin{aligned} M - C_1 \cdot R_1 &= \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 4 & 5 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} - \begin{bmatrix} 1 & 4 & 5 \\ 2 & 8 & 10 \\ 3 & 12 & 15 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 43 \end{bmatrix} \end{aligned}$$

Oh! We have taken a 3 × 3 matrix and essentially made it into a 2 × 2 matrix!! Can we do this again?

We define C_2 and R_2 to be the second column and second row of $M - C_1 \cdot R_1$, respectively, that is

$$C_2 = \begin{bmatrix} 0 \\ 1 \\ 6 \end{bmatrix} \text{ and } R_2 = \begin{bmatrix} 0 & 1 & 7 \end{bmatrix}.$$

Then we compute that

$$\begin{aligned} (M - C_1 \cdot R_1) - C_2 \cdot R_2 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 43 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 6 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 7 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 43 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 42 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Oh! Now we are essentially down to a 1 × 1 matrix!! We note that if we define C_3 and R_3 to be the third column and third row of $M - C_1 \cdot R_1 - C_2 \cdot R_2$, respectively, that is

$$C_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ and } R_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix},$$

then

$$C_3 \cdot R_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and hence,

$$M = C_1 \cdot R_1 + C_2 \cdot R_2 + C_3 \cdot R_3 = \left[\begin{array}{ccc} C_1 & C_2 & C_3 \end{array}\right] \cdot \left[\begin{array}{c} R_1 \\ R_2 \\ R_3 \end{array}\right].$$

Moreover,

- $L := \left[\begin{array}{ccc} C_1 & C_2 & C_3 \end{array}\right] = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 6 & 1 \end{array}\right]$ is **lower triangular**,

- $U := \left[\begin{array}{c} R_1 \\ R_2 \\ R_3 \end{array}\right] = \left[\begin{array}{ccc} 1 & 4 & 5 \\ 0 & 1 & 7 \\ 0 & 0 & 1 \end{array}\right]$ is **upper triangular**, and

- $M = L \cdot U$, the product of a lower triangular matrix and an upper triangular matrix.

Secret Sauce of LU Factorization

Is that all there is to LU Factorization? No, there's a bit more to it. The real algorithm has a "normalization step". To motivate it we take a matrix M that has something other than a one in its a_{11}-entry and naively form "C_1" as the first column of the matrix and "R_1" as the first row of the matrix. Then

$$\underbrace{\left[\begin{array}{cc} 2 & 3 \\ 4 & 5 \end{array}\right]}_{M} - \underbrace{\left[\begin{array}{c} 2 \\ 4 \end{array}\right]}_{\text{"}C_1\text{"}} \cdot \underbrace{\left[\begin{array}{cc} 2 & 3 \end{array}\right]}_{\text{"}R_1\text{"}} = \underbrace{\left[\begin{array}{cc} 2 & 3 \\ 4 & 5 \end{array}\right]}_{M} - \underbrace{\left[\begin{array}{cc} 4 & 6 \\ 8 & 12 \end{array}\right]}_{\text{"}C_1 \cdot R_1\text{"}} = \left[\begin{array}{cc} -2 & -3 \\ -4 & -7 \end{array}\right],$$

which does NOT result in a matrix with zeros in its leading column and row! However, if we keep R_1 as the first row of M but this time, we form C_1 from the first column of M normalized by its first entry, then we can make it work. Watch!

$$\underbrace{\left[\begin{array}{cc} 2 & 3 \\ 4 & 5 \end{array}\right]}_{M} - \underbrace{\left[\begin{array}{c} {}^2\!/_2 \\ {}^4\!/_2 \end{array}\right]}_{C_1} \cdot \underbrace{\left[\begin{array}{cc} 2 & 3 \end{array}\right]}_{R_1} = \underbrace{\left[\begin{array}{cc} 2 & 3 \\ 4 & 5 \end{array}\right]}_{M} - \underbrace{\left[\begin{array}{c} 1 \\ 2 \end{array}\right]}_{C_1} \cdot \underbrace{\left[\begin{array}{cc} 2 & 3 \end{array}\right]}_{R_1} = \underbrace{\left[\begin{array}{cc} 2 & 3 \\ 4 & 5 \end{array}\right]}_{M} - \underbrace{\left[\begin{array}{cc} 2 & 3 \\ 4 & 6 \end{array}\right]}_{C_1 \cdot R_1} = \left[\begin{array}{cc} 0 & 0 \\ 0 & -1 \end{array}\right].$$

Warning: When we do the normalization, we could get into trouble with a divide by zero!

More generally, if we assume that $\mathbf{a_{11}} \neq 0$,

$$\left[\begin{array}{cccc} \mathbf{a_{11}} & \mathbf{a_{12}} & \cdots & \mathbf{a_{1n}} \\ \mathbf{a_{21}} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a_{n1}} & a_{n2} & \cdots & a_{nn} \end{array}\right] - \left[\begin{array}{c} \mathbf{1.0} \\ \mathbf{a_{21}}/\mathbf{a_{11}} \\ \vdots \\ \mathbf{a_{n1}}/\mathbf{a_{11}} \end{array}\right] \cdot \left[\begin{array}{cccc} \mathbf{a_{11}} & \mathbf{a_{12}} & \cdots & \mathbf{a_{1n}} \end{array}\right] = \left[\begin{array}{cccc} \mathbf{a_{11}} & \mathbf{a_{12}} & \cdots & \mathbf{a_{1n}} \\ \mathbf{a_{21}} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a_{n1}} & a_{n2} & \cdots & a_{nn} \end{array}\right] - \left[\begin{array}{cccc} \mathbf{a_{11}} & \mathbf{a_{12}} & \cdots & \mathbf{a_{1n}} \\ \mathbf{a_{21}} & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a_{n1}} & * & \cdots & * \end{array}\right]$$

$$= \left[\begin{array}{cccc} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & * & \cdots & * \end{array}\right]$$

LU Factorization

Algorithm 1: LU-Factorization (without permutations)

```
Result: For M an $n \times n$ square matrix, with $n \geq 2$, find $L$ and $U$ such that $M = LU$

initialization:
Temp=copy(M);
L = Array{Float64,2}(undef, n, 0)    # L=[] Empty matrix
U = Array{Float64,2}(undef, 0, n)    # U=[] Empty matrix
Flag=1;
end initialization:

for k=1:n
C=Temp[:,k]; # k-th column
pivot=C[k];
R=Temp[k,:]'; # k-th row, which in Julia, requires a transpose
# Alternative R=Temp[k:k,:]  # also works in Julia for k-th row
if ! isapprox(pivot, 0, atol = 1E-8) # check for zero then
    C=C/pivot; # normalize so that $k$-th entry is equal to 1.0
    Temp=Temp-C*R;
    L=[L C]; # Build the lower-triangular matrix by columns
    U=[U;R]; # Build the upper-triangular matrix by rows;
else
    Flag=0;
    println("Matrix requires row permutations")
    println("Step where algorithm failed is k= $k")
    break   # Jump out of the for loop and terminate the algorithm
end

return L, U
```

The algorithm is easy to do by hand for very small matrices. In Julia, you can write code that will do the LU factorization of a 1000×1000 matrix in a few seconds. The `lu` function in the LinearAlgebra package of Julia can do it in a few milliseconds! The exact command is

```julia
using LinearAlgebra
L, U = lu(M, Val(false))
```

to return L and U **without permutations**. If you leave off " `Val(false)`", then the algorithm uses row permutations and you should call it as

```julia
using LinearAlgebra
F = lu(M)
L=F.L
U=F.U
P=F.P
```

Example 5.2 **(2×2 *Matrix*)** *Perform the LU Factorization of* $M = \begin{bmatrix} 5 & 2 \\ 15 & 2 \end{bmatrix}$, *a* 2×2 *square matrix.*

Solution: We'll do every step, just as you would program it up.

We *initialize* our algorithm by

$$\text{Temp} := M = \begin{bmatrix} 5 & 2 \\ 15 & 2 \end{bmatrix}, L := [\text{empty matrix}], \text{ and } U := [\text{empty matrix}]$$

k=1: We start with C as the first column of Temp scaled by the $k = 1$ entry of C, and thus[1]

$$\begin{aligned} C &= \begin{bmatrix} 5 \\ 15 \end{bmatrix} \\ \text{pivot} &= C[1] = 5 \\ C &= C./\text{pivot} \\ &= \begin{bmatrix} 5 \\ 15 \end{bmatrix} \cdot \frac{1}{5} \\ &= \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \end{aligned}$$

while R is the first row of Temp

$$R = \begin{bmatrix} 5 & 2 \end{bmatrix}.$$

We finish up the first step by defining

$$\begin{aligned} \text{Temp} :=& \text{Temp} - C \cdot R \\ =& \begin{bmatrix} 5 & 2 \\ 15 & 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 5 & 2 \end{bmatrix} \\ =& \begin{bmatrix} 5 & 2 \\ 15 & 2 \end{bmatrix} - \begin{bmatrix} 5 & 2 \\ 15 & 6 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 \\ 0 & -4 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} L =& [L, C] = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \\ U =& [U; R] = \begin{bmatrix} 5 & 2 \end{bmatrix} \end{aligned}$$

Why LU Factorization Works

We observe that L and U were defined so that their product exactly matched the first column and row of M. Hence, when we subtract them from M, we are left with a problem that is effectively one dimension lower, meaning, in this case, the non-zero part is 1×1.

While you may be able to finish the factorization by "inspection", we will do the complete algorithm just like you would do in a program.

k=2: C is the second column of Temp scaled by its $k = 2$ entry,

$$\begin{aligned} C &= \begin{bmatrix} 0 \\ -4 \end{bmatrix} \\ \text{pivot} &= C[2] = -4 \\ C &= \begin{bmatrix} 0 \\ -4 \end{bmatrix} ./\text{pivot} \\ &= \begin{bmatrix} 0 \\ -4 \end{bmatrix} \cdot \frac{-1}{4} \\ &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \end{aligned}$$

while R is the second row of Temp,

$$R = \begin{bmatrix} 0 & -4 \end{bmatrix}.$$

[1] In HW solutions or exams, it is fine to skip a step and write down L directly, however, it is then more difficult to give you a lot of partial credit.

$$\begin{aligned}\text{Temp} :=& \text{Temp} - C\cdot R\\ =& \begin{bmatrix} 0 & 0\\ 0 & -4\end{bmatrix} - \begin{bmatrix} 0\\ 1\end{bmatrix}\cdot \begin{bmatrix} 0 & -4\end{bmatrix}\\ =& \begin{bmatrix} 0 & 0\\ 0 & -4\end{bmatrix} - \begin{bmatrix} 0 & 0\\ 0 & -4\end{bmatrix}\\ =& \begin{bmatrix} 0 & 0\\ 0 & 0\end{bmatrix}\end{aligned}$$

$$\begin{aligned} L =& [L, C] = \begin{bmatrix} 1 & 0\\ 3 & 1\end{bmatrix} = \text{ lower triangular matrix}\\ U =& [U; R] = \begin{bmatrix} 5 & 2\\ 0 & -4\end{bmatrix} = \text{ upper triangular matrix}\end{aligned}$$

Let’s remind ourselves why this works:

The matrix product $L\cdot U$ is equal to the sum of the columns of L times the rows of U. The columns and rows were iteratively designed to remove columns and rows from M. Indeed,

$$\begin{aligned}\text{Temp} :=& M - L\cdot U\\ =& \begin{bmatrix} 5 & 2\\ 15 & 2\end{bmatrix} - \begin{bmatrix} 1 & 0\\ 3 & 1\end{bmatrix}\cdot \begin{bmatrix} 5 & 2\\ 0 & -4\end{bmatrix}\\ =& \begin{bmatrix} 5 & 2\\ 15 & 2\end{bmatrix} - \left(\begin{bmatrix} 1\\ 3\end{bmatrix}\cdot \begin{bmatrix} 5 & 2\end{bmatrix} + \begin{bmatrix} 0\\ 1\end{bmatrix}\cdot \begin{bmatrix} 0 & -4\end{bmatrix}\right)\\ =& \begin{bmatrix} 5 & 2\\ 15 & 2\end{bmatrix} - \left(\begin{bmatrix} 5 & 2\\ 15 & 6\end{bmatrix} + \begin{bmatrix} 0 & 0\\ 0 & -4\end{bmatrix}\right)\\ =& \begin{bmatrix} 5 & 2\\ 15 & 2\end{bmatrix} - \begin{bmatrix} 5 & 2\\ 15 & 6\end{bmatrix} - \begin{bmatrix} 0 & 0\\ 0 & -4\end{bmatrix}\\ =& \left(\begin{bmatrix} 5 & 2\\ 15 & 2\end{bmatrix} - \begin{bmatrix} 5 & 2\\ 15 & 6\end{bmatrix}\right) - \begin{bmatrix} 0 & 0\\ 0 & -4\end{bmatrix}\\ =& \left(\begin{bmatrix} 0 & 0\\ 0 & -4\end{bmatrix}\right) - \begin{bmatrix} 0 & 0\\ 0 & -4\end{bmatrix}\\ =& \begin{bmatrix} 0 & 0\\ 0 & 0\end{bmatrix}\end{aligned}$$

Hence,

$$\underbrace{\begin{bmatrix} 5 & 2\\ 15 & 2\end{bmatrix}}_{M} = \underbrace{\begin{bmatrix} 1 & 0\\ 3 & 1\end{bmatrix}}_{L}\cdot \underbrace{\begin{bmatrix} 5 & 2\\ 0 & -4\end{bmatrix}}_{U}$$

■

Example 5.3 **(3 × 3 *Matrix*)** *Perform the LU Factorization of* $M = \begin{bmatrix} -2 & -4 & -6\\ -2 & 1 & -4\\ -2 & 11 & -4\end{bmatrix}$*, a* 3×3 *square matrix.*

Solution: We *initialize* our algorithm by

$$\text{Temp} := M = \begin{bmatrix} -2 & -4 & -6\\ -2 & 1 & -4\\ -2 & 11 & -4\end{bmatrix}, L := \text{[empty matrix]}, \text{ and } U := \text{[empty matrix]}$$

k=1: We start with C as first column of Temp normalized by $C[1]$ so that its first entry is one

$$\begin{aligned} C &= \begin{bmatrix} -2 \\ -2 \\ -2 \end{bmatrix} \\ \text{pivot} &= C[1] = -2 \\ C &= C./\text{pivot} \\ &= \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \end{aligned}$$

while R is the first row of Temp

$$R = \begin{bmatrix} -2 & -4 & -6 \end{bmatrix}.$$

We finish up the first step by defining

$$\begin{aligned} \mathrm{Temp} :=& \mathrm{Temp} - C \cdot R \\ =& \begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} -2 & -4 & -6 \end{bmatrix} \\ =& \begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix} - \begin{bmatrix} -2 & -4 & -6 \\ -2 & -4 & -6 \\ -2 & -4 & -6 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 2 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} L =& [L, C] = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\ U =& [U; R] = \begin{bmatrix} -2 & -4 & -6 \end{bmatrix} \end{aligned}$$

Reminder of Why LU Factorization Works

We observe that L and U have been constructed so that their product exactly matches the first column and first row of M. Hence, when we subtract $L \cdot U$ from M, we are left with a problem that is effectively one dimension lower, meaning, in this case, the non-zero part is 2×2. Your wily instructor has arranged for the 2×2 matrix that remains to be an example that we have already worked!

Nevertheless, we will complete the algorithmic steps just like you would do in Julia.

k=2: C is the second column of Temp scaled by its $k = 2$ entry,

$$\begin{aligned} C &= \begin{bmatrix} 0 \\ 5 \\ 15 \end{bmatrix} \\ \text{pivot} &= C[2] = 5 \\ C &= \begin{bmatrix} 0 \\ 5 \\ 15 \end{bmatrix} ./\text{pivot} \\ &= \begin{bmatrix} 0 \\ 5 \\ 15 \end{bmatrix} \cdot \frac{1}{5} \\ &= \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}, \end{aligned}$$

while R is the second row of Temp,

$$R = \begin{bmatrix} 0 & 5 & 2 \end{bmatrix}.$$

$$\begin{aligned} \mathrm{Temp} :=& \mathrm{Temp} - C \cdot R \\ =& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 2 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 0 & 5 & 2 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 6 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} L =& [L, C] = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 3 \end{bmatrix} \\ U =& [U; R] = \begin{bmatrix} -2 & -4 & -6 \\ 0 & 5 & 2 \end{bmatrix} \end{aligned}$$

Another Reminder:

We observe that L and U have been constructed so that their product exactly matches the first two columns and first two rows of M. Hence, when we subtract $L \cdot U$ from M, we are left with a problem that is effectively two dimensions lower, meaning, in this case, the non-zero part is 1×1.

As before, we do the complete algorithm just like you would do in Julia.

k=3: C is the third column of Temp scaled by its $k = 3$ entry,

$$\begin{aligned} C &= \begin{bmatrix} 0 \\ 0 \\ -4 \end{bmatrix} \\ \text{pivot} &= C[3] = -4 \\ C &= \begin{bmatrix} 0 \\ 0 \\ -4 \end{bmatrix} ./\text{pivot} \\ &= \begin{bmatrix} 0 \\ 0 \\ -4 \end{bmatrix} \cdot \frac{1}{-4} \\ &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \end{aligned}$$

while R is the third row of Temp,

$$R = \begin{bmatrix} 0 & 0 & -4 \end{bmatrix}.$$

$$\begin{aligned} \mathrm{Temp} :=& \mathrm{Temp} - C \cdot R \\ =& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -4 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} L =& [L, C] = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix} = \text{ lower triangular matrix} \\ U =& [U; R] = \begin{bmatrix} -2 & -4 & -6 \\ 0 & 5 & 2 \\ 0 & 0 & -4 \end{bmatrix} = \text{ upper triangular matrix} \end{aligned}$$

Let’s remind ourselves why this works:

As before,

$$\begin{aligned}
\text{Temp} :=& M - L \cdot U \\
=& \begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} -2 & -4 & -6 \\ 0 & 5 & 2 \\ 0 & 0 & -4 \end{bmatrix} \\
=& \begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix} - \left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} -2 & -4 & -6 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 0 & 5 & 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -4 \end{bmatrix} \right) \\
=& \begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix} - \left(\begin{bmatrix} -2 & -4 & -6 \\ -2 & -4 & -6 \\ -2 & -4 & -6 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} \right) \\
=& \begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix} - \begin{bmatrix} -2 & -4 & -6 \\ -2 & -4 & -6 \\ -2 & -4 & -6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} \\
=& \left(\begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix} - \begin{bmatrix} -2 & -4 & -6 \\ -2 & -4 & -6 \\ -2 & -4 & -6 \end{bmatrix} \right) - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} \\
=& \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 2 \\ 0 & 15 & 6 \end{bmatrix} \right) - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} \\
=& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{bmatrix} \\
=& \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\end{aligned}$$

Hence,

$$\underbrace{\begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix}}_{M} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} -2 & -4 & -6 \\ 0 & 5 & 2 \\ 0 & 0 & -4 \end{bmatrix}}_{U}$$

■

5.4 LU Factorization for Solving Linear Equations

Solving $Ax = b$ via LU

We seek to solve the system of linear equations $Ax = b$. Suppose that we can factor $A = L \cdot U$, where U is upper triangular and L is lower triangular. Hence, we are solving the equation

$$L \cdot Ux = b. \tag{5.5}$$

If we define $Ux = y$, then (5.5) becomes two equations

$$Ly = b \tag{5.6}$$

$$Ux = y. \tag{5.7}$$

Our solution strategy is to solve (5.6) by forward substitution, and then, once we have y in hand, we solve (5.7) by back substitution to find x, the solution to (5.5).

Major Important Fact

This process may seem a bit long and tedious when doing it by hand. On a computer, it is a snap. Once you have your Julia functions created, you'll see that this scales to big problems in a very nice way.

Example 5.4 *Use LU Factorization to solve the system of linear equations*

$$\underbrace{\begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 2 \\ 3 \\ -7 \end{bmatrix}}_{b}. \tag{5.8}$$

Solution: From our hard work above, we know that

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} -2 & -4 & -6 \\ 0 & 5 & 2 \\ 0 & 0 & -4 \end{bmatrix}}_{U} \tag{5.9}$$

We first solve, using forward substitution,

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} 2 \\ 3 \\ -7 \end{bmatrix}}_{b} \implies \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ -12 \end{bmatrix}.$$

And then we use this result to solve, using back substitution,

$$\underbrace{\begin{bmatrix} -2 & -4 & -6 \\ 0 & 5 & 2 \\ 0 & 0 & -4 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 2 \\ 1 \\ -12 \end{bmatrix}}_{y} \implies \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -8 \\ -1 \\ 3 \end{bmatrix}.$$

■

The next section develops the LU Factorization with row Permutations. Why is that needed? Some matrices do not have an LU Factorization without row permutations. If we to attempt to compute an LU Factorization for

$$A = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

with our current algorithm, we'd define

$$C = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \cdot \frac{1}{0} = \text{ undefined }, \quad R = \begin{bmatrix} 0 & 1 \end{bmatrix},$$

so we are stuck. If we permute rows one and two however, we'd have

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$P \cdot A = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix}$$

which is already factored because we can define

$$L = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$U = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix},$$

giving us $P \cdot A = L \cdot U$.

Solving $Ax = b$ via LU with Row Permutations

We once again seek to solve the system of linear equations $Ax = b$. Suppose this time we can factor $P \cdot A = L \cdot U$, where P is a permutation matrix, U is upper triangular and L is lower triangular.

A beautiful property of permutation matrices is that $P^\top \cdot P = P \cdot P^\top = I$. Hence we have

$$Ax = b \iff P \cdot Ax = P \cdot b \iff L \cdot Ux = P \cdot b.$$

Hence, the only difference from our previous result is that P is no longer the identity matrix. Hence, we are solving the equation

$$L \cdot Ux = P \cdot b. \tag{5.10}$$

If we define $Ux = y$, then (5.10) again becomes two equations

$$Ly = P \cdot b \tag{5.11}$$
$$Ux = y. \tag{5.12}$$

Our solution strategy is to solve (5.11) by forward substitution, and then, once we have y in hand, we solve (5.7) by back substitution to find x, the solution to (5.10).

5.5 (Optional Read): Toward LU Factorization with Row Permutations

Are there Cases where our Simplified LU Factorization Process Fails?

Yes. They are not that hard to handle, but as your first introduction to LU factorization, you do not have to learn all of the nitty-gritty details. You've done well to get this far. You may wish to use the following in a project, in some other course, or just to understand what the Julia `lu` function is doing.

Case 1 (easiest): C becomes an entire column of zeros Suppose we have $M = \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -1 & 6 \end{bmatrix}$. To start off the process, we define

$$C = \begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix} \cdot \frac{1}{2} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}, \quad R = \begin{bmatrix} 2 & -1 & 2 \end{bmatrix},$$

and arrive at

$$\begin{aligned} Temp = M - RC &= \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -1 & 6 \end{bmatrix} - \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 6 \\ 2 & -1 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 0 & 4 \end{bmatrix} \end{aligned}$$

The next step would be

$$C = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \cdot \frac{1}{0} = \begin{bmatrix} \text{undefined} \\ \text{undefined} \\ \text{undefined} \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 & 3 \end{bmatrix},$$

leading us to attempt a divide by zero. The solution is to define

$$C = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \text{ while maintaining } R = \begin{bmatrix} 0 & 0 & 3 \end{bmatrix}.$$

With this definition,

$$\begin{aligned} Temp - RC &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 0 & 4 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 3 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 0 & 4 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \end{bmatrix}. \end{aligned}$$

We've seen this last step before and note that we have

$$C = \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix} \cdot \frac{1}{4} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 & 4 \end{bmatrix}.$$

Assembling all the pieces, we have

$$\underbrace{\begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -1 & 6 \end{bmatrix}}_{M} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} 2 & 1 & -2 \\ 0 & 0 & 3 \\ 0 & 0 & 4 \end{bmatrix}}_{U},$$

and we note that U has a zero on its diagonal[2].

In general, at the k-th step, if C becomes a column of all zeros, set its k-th entry to one and define R as usual.

Case 2 (requires row permutation): C is not all zeros, but the pivot value is zero. In this case, we need to swap rows in the matrix $Temp$ so that the new C has a non-zero pivot value. This introduces a permutation matrix, which you can review in Chapter 4.6. Keeping track of the row permutations can be a bit tricky, which is why we left this to the end.

Suppose we have $M = \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -3 & 6 \end{bmatrix}$. We initialize the process by $L = [\ ]$, $U = [\ ]$, and $\text{Temp} = M$.

[2] Hence, $\det(U) = 0$, and if we are trying to solve $Ux = y$, we are then in the problematic case of having either no solution or an infinite number of solutions.

Step $k = 1$ We have

$$C = \begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix} \cdot \frac{1}{2} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}, \quad R = \begin{bmatrix} 2 & -1 & 2 \end{bmatrix},$$

and

$$\begin{aligned} \text{Temp} &= \text{Temp} - CR \\ &= \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -3 & 6 \end{bmatrix} - \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 6 \\ 2 & -1 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & -2 & 4 \end{bmatrix}. \end{aligned}$$

We then update L and U by

$$\begin{aligned} L = [L, C] &= \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \\ U = [U; R] &= \begin{bmatrix} 2 & -1 & 2 \end{bmatrix} \end{aligned}$$

Step $k = 2$ The problem appears when we attempt to define

$$C = \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix} \cdot \frac{1}{C[2]} = \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix} \cdot \frac{1}{0},$$

which sets off the divide by zero warning because we have $\text{pivot} = 0$. The solution is to permute the second and third rows of M so that at step $k = 2$, Temp will become

$$\text{Temp}_{\text{new}} := \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2 & 4 \\ 0 & 0 & 3 \end{bmatrix}.$$

We could do this by going all the way back to the very beginning and starting the factorization over after doing the row permutation on M, but that is not very efficient. **It turns out that all we really have to do is permute the two rows in the current value of** Temp ***AND*** **the corresponding rows in** L**.** Nothing has to be done U.

We'll first refresh our memory on permutation matrices. Then we'll write down the full LU factorization algorithm with row permutations and do an example that has both of these new steps in them.

Row Permutation Matrices Consider

$$M_a = \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -3 & 6 \end{bmatrix}.$$

And suppose we want to swap rows one and three to obtain

$$M_b = \begin{bmatrix} 2 & -3 & 6 \\ 6 & -3 & 9 \\ 2 & -1 & 2 \end{bmatrix}$$

Then this is the result of pre-multiplying M_a by the permutation matrix

$$I := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \to \text{swap first and third rows} \to P_a := \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix},$$

which we can verify by

$$P_a \cdot M_a = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -3 & 6 \end{bmatrix} = \begin{bmatrix} 2 & -3 & 6 \\ 6 & -3 & 9 \\ 2 & -1 & 2 \end{bmatrix} = M_b$$

In fact, with our "second way" of doing matrix multiplication as the sum over columns times rows, the above product can be written out as

$$\begin{aligned} P_a \cdot M_a &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -3 & 6 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -1 & 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 6 & -3 & 9 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & -3 & 6 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & -1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 6 & -3 & 9 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 2 & -3 & 6 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 2 & -3 & 6 \\ 6 & -3 & 9 \\ 2 & -1 & 2 \end{bmatrix} \\ &= M_b \end{aligned}$$

It is fine to stop here, but if you want more practice with permutation matrices, then continue reading.

Example 5.5 *Let's now swap rows one and two of M_b to obtain*

$$M_c = \begin{bmatrix} 6 & -3 & 9 \\ 2 & -3 & 6 \\ 2 & -1 & 2 \end{bmatrix}.$$

Find the permutation matrix P_b such that $M_c = P_b \cdot M_b$ and the permutation matrix P_c such that $M_c = P_c \cdot M_a$. Verify that $P_c = P_b \cdot P_a$. This will help you to understand matrix multiplication.

Solution: To find the permutation matrix P_b

$$I := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \text{swap first and second rows} \rightarrow P_b = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

which we can verify by

$$P_b \cdot M_b = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -3 & 6 \\ 6 & -3 & 9 \\ 2 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 6 & -3 & 9 \\ 2 & -3 & 6 \\ 2 & -1 & 2 \end{bmatrix} = M_c$$

To find the permutation matrix P_c It is observed that M_c can be obtained by swapping the first with the last two rows of M_a.

$$I := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \text{swap first and last two rows} \rightarrow P_c = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix},$$

which we can verify by

$$P_c \cdot M_a = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & -1 & 2 \\ 6 & -3 & 9 \\ 2 & -3 & 6 \end{bmatrix} = \begin{bmatrix} 6 & -3 & 9 \\ 2 & -3 & 6 \\ 2 & -1 & 2 \end{bmatrix} = M_c$$

We verify $P_c = P_b \times P_a$ by

$$P_b \cdot P_a = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = P_c$$

■

How General Can LU Go?

All of our examples have been done for M a square matrix. LU even works for M rectangular. We'll discuss this a bit later in the course.

5.6 (Optional Read): An Algorithm for LU Factorization with Row Permutations

LU Factorization of Square Matrices

Algorithm 2: LU-Factorization (with permutations)

```
Result: For M an $n \times n$ square matrix, with $n \geq 2$, find $L$, $U$, and $P$ such that $PM = LU$

initialization:
Temp=copy(M);
L = Array{Float64,2}(undef, n, 0)   # L=[] Empty matrix
U = Array{Float64,2}(undef, 0, n)   # U=[] Empty matrix
P= I_n # $n \times n$ identity matrix
eps= 1e-16 # estimate of machine epsilon
Kappa = 10 #How small is too small for you?
end of initialization:

for  k=1:n
C=Temp[:,k]; # k-th column
R=Temp[k,:]; # k-th row
if max(abs(C)) <= Kappa*eps # (check for C all zeros) then
    C=0*C # Set tiny values to zero
    C[k]=1.0
    Temp=Temp-C*R;
    L=[L,C] # Build the lower-triangular matrix by columns
    U=[U;R] # Build the upper-triangular matrix by rows;
else
    # We know C has at least one non-zero quantity
    # We'll bring its largest entry to the top; # while this is overkill, it helps with numerical aspects of the algorithm
    # Bringing biggest to top will always avoid divide by zero
    # It's enough to bring ANY non-zero value to the top
    #
    nrow=argmax(abs(C)) # find the index where C is "biggest"
    P[[k,nrow],:]=P[[nrow,k],:] # permute (i.e., swap) rows of P
    Temp[[k,nrow],:]=Temp[[nrow,k],:] # do same for Temp
    # if L is non-empty, also swap its rows
    if k > 1 then
        L[[k,nrow],:]=L[[nrow,k],:]
    end
    C=Temp[:,k]; # k-th column
    pivot = C[k]
    C=C/pivot #divide all elements of C by the pivot
    R=Temp[k,:] # k-th row
    Temp=Temp-C*R;
    L=[L C] # Build the lower-triangular matrix by columns
    U=[U;R] # Build the upper-triangular matrix by rows
end
return L, U, P
```

Example 5.6 ***We do it by hand so that you can see all the steps.)*** $M = \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & 2 & 8 \\ 3 & -1 & -1 & 20 \end{bmatrix}$, *a* 4×4 *square matrix.*

Solution: We *initialize* our algorithm by

$$\text{Temp} := M = \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & 2 & 8 \\ 3 & -1 & -1 & 20 \end{bmatrix}, P := I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, L := \text{[empty matrix]}, \text{ and } U := \text{[empty matrix]}$$

k=1: We start with C as first column of Temp normalized by $C[1]$ so that its first entry is one

$$\begin{aligned} C &= \begin{bmatrix} 6 \\ 3 \\ 3 \\ 3 \end{bmatrix} \\ \text{nrow} &= \text{argmax}\,(\text{abs}\,(C)) = 1 \\ P &= P \\ \text{Temp} &= \text{Temp} \\ C &= \text{Temp}\,[:,1] = C \\ \text{pivot} &= C[1] = 6 \\ C &= C./\text{pivot} \\ &= \begin{bmatrix} 1 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}, \end{aligned}$$

while R is the first row of Temp

$$R = \begin{bmatrix} 6 & -2 & 4 & 4 \end{bmatrix}.$$

We finish up the first step by defining

$$\begin{aligned} \text{Temp} :=& \text{Temp} - C \cdot R \\ =& \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & 2 & 8 \\ 3 & -1 & -1 & 20 \end{bmatrix} - \begin{bmatrix} 1 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \cdot \begin{bmatrix} 6 & -2 & 4 & 4 \end{bmatrix} \\ =& \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & 2 & 8 \\ 3 & -1 & -1 & 20 \end{bmatrix} - \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 2 & 2 \\ 3 & -1 & 2 & 2 \\ 3 & -1 & 2 & 2 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & -3 & 18 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} L =& [L, C] = \begin{bmatrix} 1 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \\ U =& [U; R] = \begin{bmatrix} 6 & -2 & 4 & 4 \end{bmatrix} \\ P =& \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

We observe that L and U have been constructed so that their product exactly matches the first two columns of M. Hence, when we subtract $L \cdot U$ from M, we are left with a problem that the second column of temp becomes an entire column of zeros, meaning, in this case, the non-zero part is 3×2.

k=2: C is the second column of Temp and it is all zeros,

$$\begin{aligned} C[2] &= 1 \\ C &= \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \end{aligned}$$

while R is the second row of Temp,

$$R = \begin{bmatrix} 0 & 0 & 6 & 8 \end{bmatrix}.$$

$$\begin{aligned} \mathrm{Temp} :=& \mathrm{Temp} - C \cdot R \\ =& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & -3 & 18 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 6 & 8 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & -3 & 18 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ =& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & -3 & 18 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} L =& [L, C] = \begin{bmatrix} 1 & 0 \\ 0.5 & 1 \\ 0.5 & 0 \\ 0.5 & 0 \end{bmatrix} \\ U =& [U; R] = \begin{bmatrix} 6 & -2 & 4 & 4 \\ 0 & 0 & 6 & 8 \end{bmatrix} \\ P =& \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

When we subtract $L \cdot U$ from temp, we are left with a problem that is effectively two dimensions lower, meaning, in this case, the non-zero part is 2×2 and we are making good progress on the LU factorization! Moreover, we observe that the third column of Temp is not all zeros but the pivot value is zero. Hence, row permutation is required.

k=3: C is the third column of Temp and has 0 pivot,

$$
\begin{aligned}
C &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ -3 \end{bmatrix} \\
\text{nrow} &= \text{argmax}\left(\text{abs}\left(C\right)\right) = 4 \\
P[[3,4],:] = P[[4,3],:] &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\
\text{Temp}[[3,4],:] = \text{Temp}[[4,3],:] &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 6 \end{bmatrix} \\
L[[3,4],:] = L[[4,3],:] &= \begin{bmatrix} 1 & 0 \\ 0.5 & 1 \\ 0.5 & 0 \\ 0.5 & 0 \end{bmatrix} \\
C = \text{Temp}[:,3] &= \begin{bmatrix} 0 \\ 0 \\ -3 \\ 0 \end{bmatrix} \\
\text{pivot} &= C[3] = -3 \\
C &= C./\text{pivot} \\
&= \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},
\end{aligned}
$$

while R is the third row of Temp,

$$
R = \begin{bmatrix} 0 & 0 & -3 & 18 \end{bmatrix}.
$$

$$
\begin{aligned}
\text{Temp} :=& \text{Temp} - C \cdot R \\
=& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 6 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -3 & 18 \end{bmatrix} \\
=& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
=& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}
\end{aligned}
$$

$$\begin{aligned}L =&[L, C] = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0 & 1 \\ 0.5 & 0 & 0 \end{bmatrix}\\ U =&[U; R] = \begin{bmatrix} 6 & -2 & 4 & 4 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & -3 & 18 \end{bmatrix}\\ P =& \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}\end{aligned}$$

k=4: C is the fourth column of Temp scaled by its $k = 4$ entry,

$$\begin{aligned} C =& \begin{bmatrix} 0 \\ 0 \\ 0 \\ 6 \end{bmatrix}\\ \text{nrow} =& \text{ argmax}\,(\text{abs}\,(C)) = 4\\ P =& P\\ \text{Temp} =& \text{ Temp}\\ L =& L\\ C =& \text{ Temp}[:, 4] = C\\ \text{pivot} =& C[4] = 6\\ C =& C./\text{pivot}\\ =& \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},\end{aligned}$$

while R is the fourth row of Temp,

$$R = \begin{bmatrix} 0 & 0 & 0 & 6 \end{bmatrix}.$$

$$\begin{aligned}\text{Temp} :=& \text{Temp} - C \cdot R\\ =& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 6 \end{bmatrix}\\ =& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}\\ =& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}\end{aligned}$$

$$L = [L, C] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix} = \text{ lower triangular matrix}$$

$$U = [U; R] = \begin{bmatrix} 6 & -2 & 4 & 4 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 6 \end{bmatrix} = \text{ upper triangular matrix}$$

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

Let's remind ourselves why this works: As before,

$$\begin{aligned}
\text{Temp} :=& P \cdot M - L \cdot U \\
=& \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & 2 & 8 \\ 3 & -1 & -1 & 20 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 6 & -2 & 4 & 4 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 6 \end{bmatrix} \\
=& \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & -1 & 20 \\ 3 & -1 & 2 & 8 \end{bmatrix} - \left(\begin{bmatrix} 1 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \cdot \begin{bmatrix} 6 & -2 & 4 & 4 \end{bmatrix} + \ldots + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 6 \end{bmatrix} \right) \\
=& \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & -1 & 20 \\ 3 & -1 & 2 & 8 \end{bmatrix} - \left(\begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 2 & 2 \\ 3 & -1 & 2 & 2 \\ 3 & -1 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} \right) \\
=& \left(\begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & -1 & 20 \\ 3 & -1 & 2 & 8 \end{bmatrix} - \begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 2 & 2 \\ 3 & -1 & 2 & 2 \\ 3 & -1 & 2 & 2 \end{bmatrix} \right) - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} \\
=& \left(\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right) - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} \\
=& \left(\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right) - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} \\
=& \left(\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} \right) \\
=& \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}$$

Hence,

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{P} \cdot \underbrace{\begin{bmatrix} 6 & -2 & 4 & 4 \\ 3 & -1 & 8 & 10 \\ 3 & -1 & 2 & 8 \\ 3 & -1 & -1 & 20 \end{bmatrix}}_{M} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 0 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} 6 & -2 & 4 & 4 \\ 0 & 0 & 6 & 8 \\ 0 & 0 & -3 & 18 \\ 0 & 0 & 0 & 6 \end{bmatrix}}_{U}$$

■

Disclaimer

This is your book's author speaking: We will never do this by hand again. Don't even ask. The good thing for you, I will not ask you do to something that I would not do myself! (Thank you, Miley.)

Solving $\mathrm{Ax} = \mathrm{b}$ when A is square and $\mathrm{P} \cdot \mathrm{A} = \mathrm{L} \cdot \mathrm{U}$

So, how does the permutation matrix change how one solves $Ax = b$? Because **permutation matrices are invertible**, the main logic is

$$Ax = b \iff P \cdot Ax = P \cdot b \iff L \cdot Ux = P \cdot b, \tag{5.13}$$

where we have used $P \cdot A = L \cdot U$. Hence, (5.6) and (5.7) are modified as follows: if we define $Ux = y$ as before, then (5.13) becomes two equations

$$Ly = P \cdot b \tag{5.14}$$

$$Ux = y. \tag{5.15}$$

The solution strategy is to solve (5.14) by forward substitution, and then, once we have y in hand, we solve (5.15) by back substitution to find x, the solution to (5.13).

5.7 Looking Ahead

Once you have programmed in Julia

(a) algorithms for forward and back substitution (methods for solving triangular systems of equations), and

(b) the LU Factorization Algorithm,

you will be well placed for solving systems of linear equations of very large dimension. In the next chapter, we will fill in some of the more standard results that students learn in linear algebra, where the focus is on solving "drill problems" and not "real engineering problems". In your later courses, most instructors will simply assume you know the more standard material and that you have no clue about the more nuanced approach to problem solving that we have developed in ROB 101.

Chapter 6

Determinant of a Product, Matrix Inverses, and the Matrix Transpose

Learning Objectives

- Fill in some gaps that we left during our sprint to an effective means for solving large systems of linear equations.

Outcomes

- Whenever two square matrices A and B can be multiplied, it is true that $\det(A \cdot B) = \det(A) \cdot \det(B)$
- You will learn what it means to “invert a matrix,” and you will understand that you rarely want to actually compute a matrix inverse!
- If $ad - bc \neq 0$, then $\left[\begin{array}{cc} a & b \\ c & d \end{array}\right]^{-1} = \frac{1}{ad-bc}\left[\begin{array}{cc} d & -b \\ -c & a \end{array}\right]$.
- Moreover, this may be the only matrix inverse you really want to compute explicitly, unless a matrix has special structure.
- Matrix transpose takes columns of one matrix into the rows of another.

From time to time, we will start taking some material from Kuttler 2017, A First Course in Linear Algebra, Page(s) 68-69. The authors have generously made it an open-source resource for the STEM community. The book is posted on our Canvas site.

6.1 A very Useful Fact Regarding the Matrix Determinant

Out of Kutler's entire chapter on $\det(A)$, we take the following highly useful fact,

Let A and B be two $n \times n$ matrices. Then

$$\det(AB) = \det(A)\det(B)$$

In order to find the determinant of a product of matrices, we can simply take the product of the determinants.

Now suppose that we have done the LU factorization of a square matrix A. Then, using the fact that the determinant of a product of two square matrices is the product their respective determinants, we have that

$$\det(A) = \det(L \cdot U) = \det(L) \cdot \det(U).$$

Because L and U are triangular matrices, each of their determinants is given by the product of the diagonal elements. Hence, we have a way of computing the determinant for square matrices of arbitrary size.

Example Going back to Chap. 5, specifically, (5.8) and (5.9), we have that

$$\underbrace{\begin{bmatrix} -2 & -4 & -6 \\ -2 & 1 & -4 \\ -2 & 11 & -4 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 3 & 1 \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} -2 & -4 & -6 \\ 0 & 5 & 2 \\ 0 & 0 & -4 \end{bmatrix}}_{U}$$

Hence,

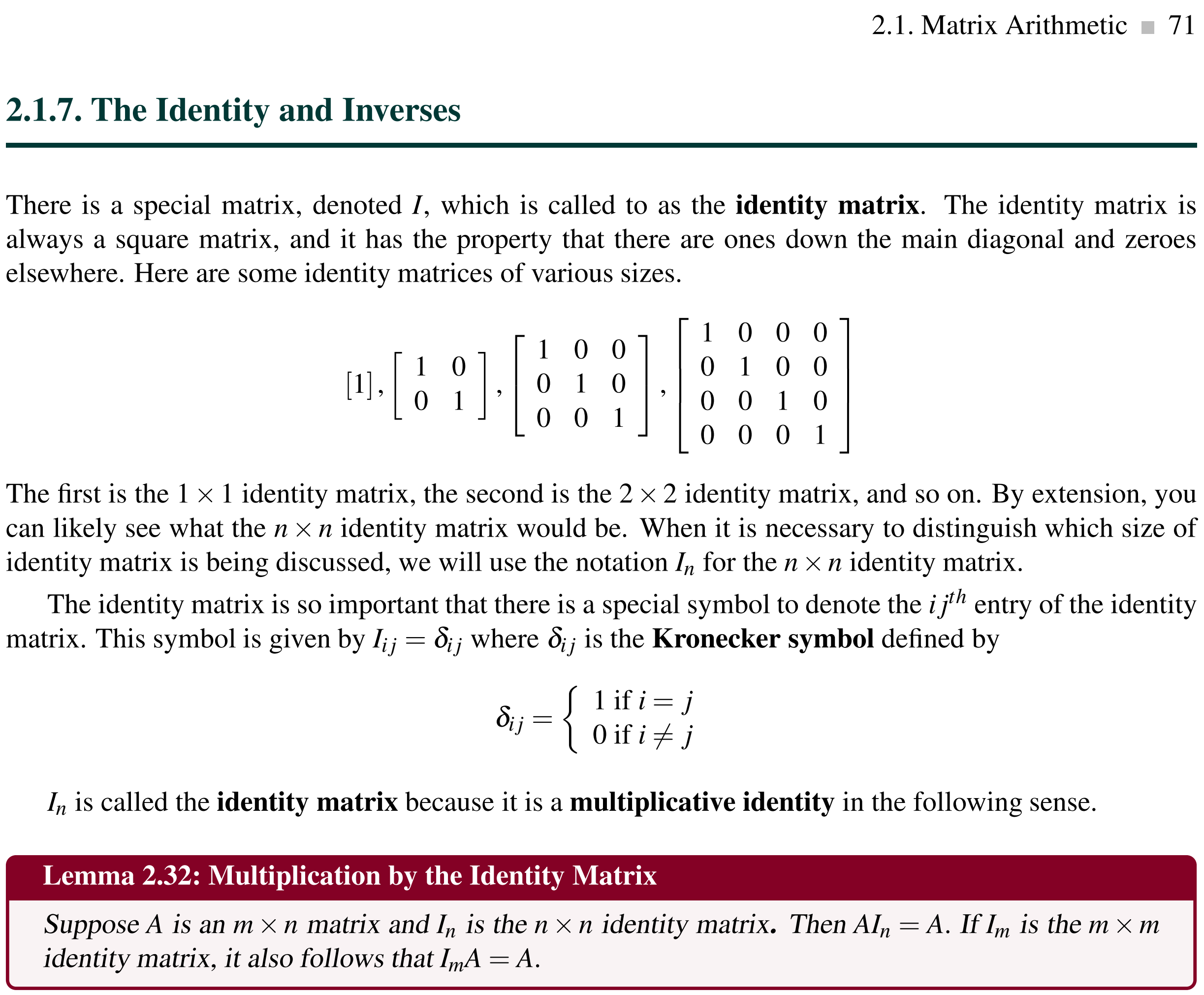
$$\det(A) = \underbrace{(1)\cdot(1)\cdot(1)}_{\det(L)} \cdot \underbrace{(-2)\cdot(5)\cdot(-4)}_{\det(U)} = 40.$$

Once we know that $\det(A \cdot B) = \det(A) \cdot \det(B)$, we immediately obtain from it

$$\boxed{\det(A \cdot B \cdot C) = \det(A) \cdot \det(B) \cdot \det(C)}$$

because $\det(A \cdot B \cdot C) = \det((A \cdot B) \cdot C) = \det(A \cdot B) \cdot \det(C) = \det(A) \cdot \det(B) \cdot \det(C)$. This formula extends to any product of $n \times n$ matrices.

6.2 Identity Matrix and Matrix Inverse



2.1. Matrix Arithmetic ■ 71

2.1.7. The Identity and Inverses

There is a special matrix, denoted I, which is called to as the **identity matrix**. The identity matrix is always a square matrix, and it has the property that there are ones down the main diagonal and zeroes elsewhere. Here are some identity matrices of various sizes.

$$[1], \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The first is the 1×1 identity matrix, the second is the 2×2 identity matrix, and so on. By extension, you can likely see what the $n \times n$ identity matrix would be. When it is necessary to distinguish which size of identity matrix is being discussed, we will use the notation I_n for the $n \times n$ identity matrix.

The identity matrix is so important that there is a special symbol to denote the ij^{th} entry of the identity matrix. This symbol is given by $I_{ij} = \delta_{ij}$ where δ_{ij} is the **Kronecker symbol** defined by

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

I_n is called the **identity matrix** because it is a **multiplicative identity** in the following sense.

Lemma 2.32: Multiplication by the Identity Matrix

Suppose A is an $m \times n$ matrix and I_n is the $n \times n$ identity matrix. Then $AI_n = A$. If I_m is the $m \times m$ identity matrix, it also follows that $I_mA = A$.

Optional Read: Suppose that A is 2×3. To see why $I_2A = A$ and $AI_3 = A$, just apply our "second definition" of matrix multiplication.

$$\begin{aligned}
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} a_{21} & a_{22} & a_{23} \end{bmatrix} \\
&= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \\
&= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}
\end{aligned}$$

and

$$\begin{aligned}
\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} &= \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} a_{12} \\ a_{22} \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & 0 \\ 0 & a_{22} & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & a_{13} \\ 0 & 0 & a_{23} \end{bmatrix} \\
&= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}
\end{aligned}$$

We now define the matrix operation which in some ways plays the role of division.

Definition 2.33: The Inverse of a Matrix

A square $n \times n$ matrix A is said to have an ***inverse*** *A^{-1} if and only if*

$$AA^{-1} = A^{-1}A = I_n$$

In this case, the matrix A is called ***invertible****.*

Example: Consider $A = \begin{bmatrix} 4 & 2 \\ 5 & 3 \end{bmatrix}$. Let's check if $B = \frac{1}{2}\begin{bmatrix} 3 & -2 \\ -5 & 4 \end{bmatrix}$ is in fact an inverse for A. According to the definition, we need to check that $A \cdot B = B \cdot A = I_2$. Well,

$$A \cdot B = \begin{bmatrix} 4 & 2 \\ 5 & 3 \end{bmatrix} \cdot \left(\frac{1}{2}\begin{bmatrix} 3 & -2 \\ -5 & 4 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} (4 \cdot 3 - 2 \cdot 5) & (4 \cdot (-2) + 2 \cdot 4) \\ (5 \cdot 3 - 3 \cdot 5) & (5 \cdot (-2) + 3 \cdot 4) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and, because we passed the first part of the test, we do the second part

$$B \cdot A = \left(\frac{1}{2}\begin{bmatrix} 3 & -2 \\ -5 & 4 \end{bmatrix}\right) \cdot \begin{bmatrix} 4 & 2 \\ 5 & 3 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} (3 \cdot 4 - 2 \cdot 5) & (3 \cdot 2 - 2 \cdot 3) \\ ((-5) \cdot 4 + 4 \cdot 5) & ((-5) \cdot 2 + 4 \cdot 3) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and hence we conclude that $B = A^{-1}$.

Most Important Matrix Inverse for ROB 101

Consider $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and suppose that $\det(A) = a \cdot d - b \cdot c \neq 0$. Then,

$$A^{-1} = \frac{1}{a \cdot d - b \cdot c}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{\det(A)}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Applying the above formula for the inverse of a 2×2 matrix immediately gives that

$$\begin{bmatrix} 4 & 2 \\ 5 & 3 \end{bmatrix}^{-1} = \frac{1}{2}\begin{bmatrix} 3 & -2 \\ -5 & 4 \end{bmatrix}.$$

Remark: If a matrix has an inverse, it is unique (that is, there is only one of them). If A and B are both $n \times n$, then

$$(A \cdot B = B \cdot A = I_n) \iff B = A^{-1}.$$

Moreover, for square matrices with **real (or complex) elements**, you do not have to check both $A \cdot B = I$ and $B \cdot A = I$ to conclude that $B = A^{-1}$. **It is enough to check one of them because**

$$(A \cdot B = I) \iff (B \cdot A = I) \iff B = A^{-1}.$$

For more general notions of numbers, one does have to check both sides.

Important Facts for the Matrix Determinant:

- Suppose that A is $n \times n$. Because the "determinant of a product is the product of the determinants", we have that

$$1 = \det(I_n) = \det(A \cdot A^{-1}) = \det(A) \cdot \det(A^{-1})$$

- It follows that if A has an inverse, then $\det(A) \neq 0$ and $\det(A^{-1}) = \frac{1}{\det(A)}$
- The other way around is also true: if $\det(A) \neq 0$, then it has an inverse (one also says that A^{-1} exists). Putting these facts together gives the next result.

Major Important Fact

An $n \times n$ matrix A is invertible if, and only if, $\det(A) \neq 0$.

Another useful fact about matrix inverses is that if A and B are both $n \times n$ and invertible, then their product is also invertible and

$$\boxed{(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}.}$$

Note that the order is swapped when you compute the inverse. To see why this is true, we note that

$$(A \cdot B) \cdot (B^{-1} \cdot A^{-1}) = A \cdot (B \cdot B^{-1}) \cdot A^{-1} = A \cdot (I) \cdot A^{-1} = A \cdot A^{-1} = I.$$

Hence, $(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$ **and NOT** $A^{-1} \cdot B^{-1}$**.**

LU and Matrix Inverses

A consequence of this is that if A is invertible and $A = L \cdot U$ is the LU factorization of A, then

$$A^{-1} = U^{-1} \cdot L^{-1}.$$

While we will not spend time on it in ROB 101, it is relatively simple to compute the inverse of a triangular matrix whose determinant is non-zero.

6.3 Utility of the Matrix Inverse and its Computation

The primary use of the matrix inverse is that it provides a **closed-form solution** to linear systems of equations. Suppose that A is square and invertible, then

$$\boxed{Ax = b \iff x = A^{-1} \cdot b.} \tag{6.1}$$

While it is a beautiful thing to write down the closed-form solution given in (6.1) **as the answer to** $Ax = b$**, one should rarely use it in computations.** It is much better to solve $Ax = b$ by factoring $A = L \cdot U$ and using back and forward substitution, than to first compute A^{-1} and then multiply A^{-1} and b. Later in ROB 101, we'll learn another good method called the QR factorization.

We (your instructors) know the above sounds bizarre to you! You've been told that $Ax = b$ has a unique solution for x if, and only if, $\det(A) \neq 0$, and you know that A^{-1} exists if, and only if, $\det(A) \neq 0$. Hence, logic tells you know that $x = A^{-1}b$ if, and only if, $\det(A) \neq 0$. So what gives?

Theory vs Reality

$x = A^{-1}b$ if, and only if, $\det(A) \neq 0$ is true in the world of perfect arithmetic, but not in the approximate arithmetic done by a computer, or by a hand calculator, for that matter. The problem is that the determinant of a matrix can be "very nice", meaning its absolute value is near 1.0, while, from a numerical point of view, the matrix is "barely invertible".

Example 1 Consider $A = \begin{bmatrix} 1 & 10^{-15} \\ 10^{10} & 1 \end{bmatrix}$. For this small example, we can see that A has a huge number and a tiny number in it, but imagine that A is the result of an intermediate computation in your algorithm and hence you'd never look at it, or the matrix is so large, you would not notice such numbers. If you want to check whether A is invertible or not, you find that $\det(A) = 1 - 10^{-5} = 0.99999$, which is very close to 1.0, and thus the determinant has given us no hint that A has crazy numbers of vastly different sizes in it.

Example 2 Consider a 3×3 matrix

$$A = \begin{bmatrix} 100.0000 & 90.0000 & -49.0000 \\ 90.0000 & 81.0010 & 5.4900 \\ 100.0000 & 90.0010 & 59.0100 \end{bmatrix}. \tag{6.2}$$

We compute the determinant and check that it is not close to zero. Indeed, $\det(A) = 0.90100$ (correct to more than ten decimal points[1]), and then bravely, we use Julia to compute the inverse, yielding

$$A^{-1} = \begin{bmatrix} -178.9000 & -10,789.0666 & 9,889.0666 \\ 198.7791 & 11,987.7913 & -10,987.981 \\ -110.9878 & -0.1110 & 0.1110 \end{bmatrix}.$$

As a contrast, we compute $A = L \cdot U$, the LU factorization (without permutation), yielding

$$L = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.9 & 1.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix} \qquad U = \begin{bmatrix} 100.000 & 90.000 & -49.000 \\ 0.000 & \boxed{-0.001} & 99.000 \\ 0.000 & 0.000 & 9.010 \end{bmatrix}.$$

We see that U has a small number on the diagonal, and hence, if we do back substitution to solve $Ux = y$, for example, as part of solving

$$Ax = b \iff L \cdot Ux = b \iff (Ly = b \text{ and } Ux = y)$$

we know in advance that we'll be dividing by -0.001.

Moreover, we see the diagonal of L is $[1.0, 1.0, 1.0]$, and hence $\det(L) = 1$. The diagonal of U is $[100.0, -0.001, 9.01]$, and hence $\det(U) = 0.901$, and we realize that we ended with a number close to 1.0 in magnitude by multiplying a large number and a small number.

Theory vs Reality

The value of $|\det(A)|$ (magnitude of the determinant of A) is a poor predictor of whether or not A^{-1} has very large and very small elements in it, and hence poses numerical challenges for its computation. For typical HW "drill" problems, you rarely have to worry about this. However, for "real" engineering problems, where a typical dimension (size) of A may be 50 or more, then **please please please avoid the computation of the matrix inverse whenever you can!**

The LU factorization is a more reliable predictor of numerical problems that may be encountered when computing A^{-1}. But once you have the LU factorization, you must ask yourself, do you even need A^{-1}? In the majority of cases, the answer is NO!

6.4 Matrix Transpose and Symmetric Matrices

The transpose takes the rows of a matrix and turns them into the columns of a new matrix, as in

$$\begin{bmatrix} \boxed{a_{11} \;\; a_{12} \;\; \cdots \;\; a_{1m}} \\ \boxed{a_{21} \;\; a_{22} \;\; \cdots \;\; a_{2m}} \\ \vdots \\ \boxed{a_{n1} \;\; a_{n2} \;\; \cdots \;\; a_{nm}} \end{bmatrix}^{\top} := \begin{bmatrix} \boxed{\begin{matrix} a_{11} \\ a_{12} \\ \vdots \\ a_{1n} \end{matrix}} & \boxed{\begin{matrix} a_{21} \\ a_{22} \\ \vdots \\ a_{2n} \end{matrix}} & \cdots & \boxed{\begin{matrix} a_{m1} \\ a_{m2} \\ \vdots \\ a_{mn} \end{matrix}} \end{bmatrix}.$$

[1] $\det(A) - 0.9010 = -1.7 \times 10^{-11}$.

Equivalently, you can view the matrix transpose as taking each column of one matrix and laying the elements out as rows in a new matrix

$$\left[\boxed{\begin{matrix}a_{11}\\a_{12}\\\vdots\\a_{1n}\end{matrix}}\ \boxed{\begin{matrix}a_{21}\\a_{22}\\\vdots\\a_{2n}\end{matrix}}\ \cdots\ \boxed{\begin{matrix}a_{m1}\\a_{m2}\\\vdots\\a_{mn}\end{matrix}}\right]^{\top} := \begin{bmatrix}\boxed{a_{11}\ a_{12}\ \cdots\ a_{1m}}\\ \boxed{a_{21}\ a_{22}\ \cdots\ a_{2m}}\\ \vdots\\ \boxed{a_{n1}\ a_{n2}\ \cdots\ a_{nm}}\end{bmatrix}.$$

Definition 2.26: The Transpose of a Matrix

Let A be an $m \times n$ matrix. Then A^T, the ***transpose*** *of A, denotes the $n \times m$ matrix given by*

$$A^T = \left[a_{ij}\right]^T = \left[a_{ji}\right]$$

The (i,j)-entry of A becomes the (j,i)-entry of A^T.

Consider the following example.

Example 2.27: The Transpose of a Matrix

Calculate A^T for the following matrix

$$A = \begin{bmatrix} 1 & 2 & -6 \\ 3 & 5 & 4 \end{bmatrix}$$

Solution. By Definition 2.26, we know that for $A = \left[a_{ij}\right]$, $A^T = \left[a_{ji}\right]$. In other words, we switch the row and column location of each entry. The $(1,2)$-entry becomes the $(2,1)$-entry.

Thus,

$$A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ -6 & 4 \end{bmatrix}$$

Notice that A is a 2×3 matrix, while A^T is a 3×2 matrix. ♠

The transpose of a matrix has the following important properties .

Lemma 2.28: Properties of the Transpose of a Matrix

Let A be an $m \times n$ matrix, B an $n \times p$ matrix, and r and s scalars. Then

1. $\left(A^T\right)^T = A$
2. $(AB)^T = B^T A^T$
3. $(rA + sB)^T = rA^T + sB^T$

Definition 2.29: Symmetric and Skew Symmetric Matrices

*An $n \times n$ matrix A is said to be **symmetric** if $A = A^T$. It is said to be **skew symmetric** if $A = -A^T$.*

We will explore these definitions in the following examples.

Example 2.30: Symmetric Matrices

Let

$$A = \left[\begin{array}{rrr} 2 & 1 & 3 \\ 1 & 5 & -3 \\ 3 & -3 & 7 \end{array} \right]$$

Use Definition 2.29 to show that A is symmetric.

Solution. By Definition 2.29, we need to show that $A = A^T$. Now, using Definition 2.26,

$$A^T = \left[\begin{array}{rrr} 2 & 1 & 3 \\ 1 & 5 & -3 \\ 3 & -3 & 7 \end{array} \right]$$

Hence, $A = A^T$, so A is symmetric. ♠

Example 2.31: A Skew Symmetric Matrix

Let

$$A = \left[\begin{array}{rrr} 0 & 1 & 3 \\ -1 & 0 & 2 \\ -3 & -2 & 0 \end{array} \right]$$

Show that A is skew symmetric.

Solution. By Definition 2.29,

$$A^T = \left[\begin{array}{rrr} 0 & -1 & -3 \\ 1 & 0 & -2 \\ 3 & 2 & 0 \end{array} \right]$$

You can see that each entry of A^T is equal to -1 times the same entry of A. Hence, $A^T = -A$ and so by Definition 2.29, A is skew symmetric. ♠

6.5 Revisiting Permutation Matrices

In Chap. 4.6, we jumped ahead and introduced permutation matrices before properly treating the matrix transpose and the matrix inverse. We'll set things right here. Let's recall the definition first.

Permutation Matrices

Matrices that consist of all ones and zeros, with each row and each column having a single one, are called permutation matrices.

Here is one of the permutations we considered before where applying the permutation twice does not undo the swapping of rows,

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \to \begin{bmatrix} 4 \\ 2 \\ 1 \\ 5 \\ 3 \end{bmatrix}. \tag{6.3}$$

As we did before, we put the 5×5 identity matrix on the left and the corresponding permutation matrix P on the right

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \leftrightarrow P = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \leftrightarrow \begin{bmatrix} 3 \to 1 \\ 2 \to 2 \\ 5 \to 3 \\ 1 \to 4 \\ 4 \to 5 \end{bmatrix}.$$

P is still just a re-ordering of the rows of I. You can check that P is not symmetric, that is, $P^\top \neq P$, but that $P^\top \cdot P = P \cdot P^\top = I$. Hence, computing the inverse of a permutation matrix is a snap!

Here are two snippets of Julia code that show how to take the desired permutation of rows and build the permutation matrix from the identify matrix. In the first method, you have to build the mapping from the rows of the identify matrix to rows of the permutation matrix.

```
using LinearAlgebra
Indices1=[3, 2, 5, 1, 4]
Identity=zeros(5,5)+I
P=Identity[Indices1,:]
```

In this second method, we use the desired permutation of the indices directly. This actually builds the transpose of the permutation matrix. The best way to understand how this works is to play around with a few examples!

```
#Altenrative, which is easier to build
using LinearAlgebra
Indices2=[4, 2, 1, 5, 3]
Identity=zeros(5,5)+I
P=Identity[Indices2,:]'
```

6.6 Looking Ahead

You have now seen the most important material for solving systems of linear equations. You have been cautioned about the **"theory vs practice gap"** when it comes to computing matrix inverses. In fact, you were told, with a mix of seriousness and joking, that "your instructors don't hang out with people who compute matrix inverses." The purpose of the statement is to drive home the fact that computing a matrix inverse is a numerically delicate operation.

Our next task is to develop a deeper understanding of vectors and special sets built out of vectors, called "vector spaces." We'll learn about linear independence, basis vectors, dimension, as well as how to measure the size of a vector and the distance between two vectors. **This knowledge will allow us to analyze systems of linear equations $Ax = b$ that do not have an exact solution!** Right now, this might seem bizarre: if you already know there is not an exact solution, what could you possibly be analyzing? We will be seeking an approximate answer that minimizes the error in the solution, where the error is defined as

$$e := Ax - b.$$

We seek a value of x that makes e as "small as possible in magnitude". **In Robotics as in life, most interesting problems do not have exact answers. The goal is then to find approximate answers that are good enough!**

Chapter 7

The Vector Space R^{n}: Part 1

Learning Objectives

- Instead of working with individual vectors, we will work with a collection of vectors.
- Our first encounter with some of the essential concepts in Linear Algebra that go beyond systems of equations.

Outcomes

- Vectors as n-tuples of real numbers
- $\mathbb{R}^n$ as the collection of all n-tuples of real numbers
- Linear combinations of vectors
- Linear independence of vectors
- Relation of these concepts to the existence and uniqueness of solutions to $Ax = b$.
- How the LU Factorization makes it very straightforward to check the linear independence of a set of vectors, and how it makes it reasonably straightforward to check if one vector is a linear combination of a set of vectors.

7.1 Vectors in $\mathbb{R}^n$ and Some Basic Algebra

An n-tuple is a fancy name for an ordered list of n numbers, $(x_1, x_2, \dots, x_n)$. If this sounds a lot like a vector, then good, because **we will systematically identify**

$$\mathbb{R}^n := \{(x_1, x_2, \dots, x_n) \mid x_i \in R, 1 \le i \le n\} \tag{7.1}$$

with column vectors. In other words, for us,

$$\mathbb{R}^n := \{(x_1, x_2, \dots, x_n) \mid x_i \in R, 1 \le i \le n\} \iff \left\{ \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \middle| \; x_i \in R, 1 \le i \le n \right\} =: \mathbb{R}^n. \tag{7.2}$$

The choice of identifying n-tuples of numbers with column vectors instead of row vectors is completely arbitrary, and yet, it is very common.

Columns of Matrices are Vectors and Vice Versa

Suppose that A is an $n \times m$ matrix, then its columns are vectors in $\mathbb{R}^n$ and conversely, given vectors in $\mathbb{R}^n$, we can stack them together and form a matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} = \begin{bmatrix} a_1^{\rm col} & a_2^{\rm col} & \cdots & a_m^{\rm col} \end{bmatrix} \iff a_j^{\rm col} := \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{nj} \end{bmatrix} \in \mathbb{R}^n, 1 \le j \le n \tag{7.3}$$

Example 7.1 *The following are all vectors in* $\mathbb{R}^4$

$$u = \begin{bmatrix} 1 \\ -2 \\ \pi \\ \sqrt{17} \end{bmatrix}, \; v = \begin{bmatrix} 4.1 \\ -1.1 \\ 0.0 \\ 0.0 \end{bmatrix}. \; w = \begin{bmatrix} 10^3 \\ 0 \\ 0.7 \\ 1.0 \end{bmatrix}.$$

Use them to make a 4×3 *matrix.*

Solution:

$$A = \begin{bmatrix} 1 & 4.1 & 10^3 \\ -2 & -1.1 & 0.0 \\ \pi & 0.0 & 0.7 \\ \sqrt{17} & 0.0 & 1.0 \end{bmatrix}.$$

■

Example 7.2 *The matrix* A *is a* 2×4*. Extract its columns to form vectors in* $\mathbb{R}^2$.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}.$$

Solution:

$$a_1^{\rm col} = \begin{bmatrix} 1 \\ 5 \end{bmatrix} \in \mathbb{R}^2, \; a_2^{\rm col} = \begin{bmatrix} 2 \\ 6 \end{bmatrix} \in \mathbb{R}^2, \; a_3^{\rm col} = \begin{bmatrix} 3 \\ 7 \end{bmatrix} \in \mathbb{R}^2, \; a_4^{\rm col} = \begin{bmatrix} 4 \\ 8 \end{bmatrix} \in \mathbb{R}^2.$$

■

For many applications, we go back and forth between columns of matrices and vectors. In such cases, we are really treating vectors as *ordered lists of numbers*, where *ordered* simply means there is a first element, a second element, etc., just as we did way back in Chapter 8. In other applications, we think of vectors as points in space. This is especially common with $\mathbb{R}^2$, the two-dimensional plane, or $\mathbb{R}^3$, the three-dimensional world in which we live, but if you've heard of Einstein's "space-time continuum", the space in which his theory of general relativity lives, then you know about four-dimensional space!

Some Books use Different Notation for Vectors in $\mathbb{R}^n$

For $\mathbb{R}^n := \{(x_1, x_2, \cdots, x_n) \mid x_j \in \mathbb{R}, j = 1, 2, \cdots, n\}$, the current way to denote a vector is

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

is called a **vector**. The real numbers x_j are called the **components** of x or the **entries** of x.

You will find many textbooks that place arrows over vectors, as in

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

The "arrow notation" for vectors is falling out of favor. **When your instructors write out vectors, they will not use the "arrow" notation.** This actually helps to make your mathematics look closer to your programming, which is ALWAYS a good thing, but it is not the primary motivation.

Definitions: Consider two vectors $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$. We define their vector sum by

$$x + y = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} := \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix},$$

that is, we sum their respective components or entries. Let α be a real number. Then we define

$$\alpha x := \begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{bmatrix},$$

that is, to multiply a vector by a real number, we multiply each of the components of the vector. ■

With these definitions, two vectors x and y are **equal** if, and only if, they have the same components,

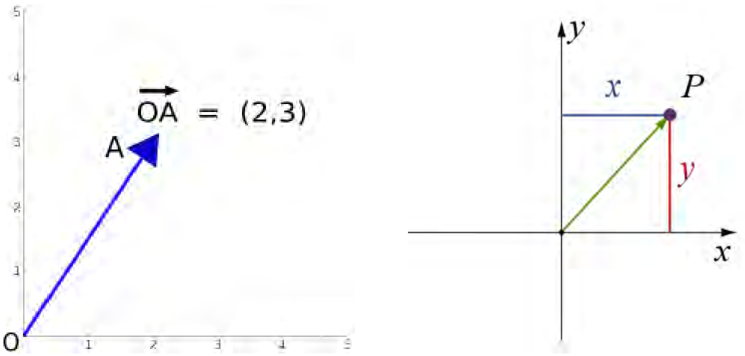
$$x := \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} =: y \iff x_j = y_j \text{ for all } 1 \le j \le n.$$

Said another way,

$$x = y \iff x - y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0_{n \times 1},$$

the zero vector in $\mathbb{R}^n$.

Remark: Normally, the zero vector in $\mathbb{R}^n$ is simply denoted by 0. From time to time, to help our learning, we'll be very explicit and write it as $0_{n\times 1}$ in order to emphasize its dimension!



(a)

(b)

Figure 7.1: Images of the xy-plane with Cartesian coordinates x and y, courtesy of the WikiMedia Commons. (a) shows a vector drawn to the **point** $(2,3)$ and labeled suggestively as a vector from the origin O to point A. (b) shows a **point** P represented with coordinates (x,y), also with a vector drawn to it. These are both fine interpretations of vectors. In ROB 101, points and vectors are the same thing: an ordered list of numbers.

Figure 7.1 shows two geometric depictions of vectors in 2D, one emphasizing a vector as a directed line segment from the origin O to a point A located at $(2,3)$, while the second is a directed line segment from the origin O to a point P located at (x,y). Both of these are fine interpretations of vectors. In $\mathbb{R}^2$, graphical depictions of vectors are relatively straightforward to understand. Figure 7.2a illustrates so-called unit vectors along the xyz-axes in $\mathbb{R}^3$, while Fig. 7.2b shows a point in $\mathbb{R}^3$ with Cartesian coordinates (x,y,z). These depictions of vectors and points in $\mathbb{R}^3$ are fairly simple, but it is not hard to give other examples in $\mathbb{R}^3$ that are harder to "piciture" as a 2D-image, which is all we can show in a sheet of paper. So yes, it easy to get lost in 3D-space and if you have a hard time "imagining" (that is, representing visually in your head for the purpose of building intuition) objects in 3D, you are not alone. Anyone want to try $\mathbb{R}^{27}$, that is, 27D-space?

In ROB 101, points in $real^n$ are vectors. We treat points and vectors as being different ways of representing the same mathematical object: an ordered list of numbers. Don't get hung up on the geometric aspect of vectors. In Physics, Dynamics, and Electromagnetics courses, you'll be challenged to "imagine" vectors in 3D. Deal with it there. For now, it is OK to think of vectors as lists of numbers that obey some basic algebraic relationships and nothing more. Besides, when dealing with a vector that has hundreds of components, what else can you do?

Y
$\hat{j}$
o
$\hat{k}$
$\hat{i}$
X
Z
Unit vector
Z
(x,y,z)
O
z
x
y
Y
X

(a)

(b)

Figure 7.2: Images of 3D space with Cartesian coordinates x, y, and z, courtesy of the WikiMedia Commons. (a) shows an uncommon orientation of the 3d space, but still using xyz-coordiantes! (b) shows a more common orientation of the xyz-axes along with a point represented with coordinates (x,y,z). These are both fine interpretations of 3D space. In ROB 101, we do not rely on your ability to imagine vectors in nD space for $n > 2$.

3 Special Vectors in $\mathbb{R}^3$

Figure 7.2a defines "unit" vectors $\widehat{i}$, $\widehat{j}$, and $\widehat{k}$. You will see these a lot in Physics. The more accepted mathematical notation in Linear Algebra is

$$e_1 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \; e_2 := \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \text{ and } e_3 := \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Properties of Vector Addition and Scalar Times Vector Multiplication

1. Addition is commutative: For any $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$,

$$x + y = y + x.$$

2. Addition is associative: For any $x \in \mathbb{R}^n$, $y \in \mathbb{R}^n$ and $z \in \mathbb{R}^n$,

$$(x + y) + z = x + (y + z)\,.$$

3. Scalar multiplication is associative: For any $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$, and any $x \in \mathbb{R}^n$ in

$$\alpha\,(\beta x) = (\alpha\beta)\,x.$$

4. Scalar multiplication is distributive: For any $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$ and for any $x \in \mathbb{R}^n$, $y \in \mathbb{R}^n$,

$$(\alpha + \beta)x = \alpha x + \beta x,$$

and

$$\alpha\,(x + y) = \alpha x + \alpha y.$$

All of these properties follow from the corresponding properties for real numbers. Hence, there is no real reason to make a big deal about them. You've been adding vectors and multiplying them by constants in Julia for several weeks now!

7.2 The Notion of Linear Combinations

Sums of vectors times scalars are called linear combinations. We'll introduce the concept in two ways: first by studying what Ax really means in terms of the columns of A and the components of x, and in a second pass, a more "pure" or "abstract" defintion in $\mathbb{R}^n$.

7.2.1 Linear combinations through the lens of $Ax = b$

Let's consider $Ax = b$ and see if we truly understand the product of an $n \times m$ matrix A and an $m \times 1$ column vector x, and then the equation $Ax = b$! We write

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \text{ and } \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}. \tag{7.4}$$

Then, using our column times row method for matrix multiplication, we have

$$Ax = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} x_1 + \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} x_2 + \cdots + \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix} x_m. \tag{7.5}$$

If the above is not clicking for you, go back and look at our second method for matrix multiplication in Chapter 4.

The next step is to move the x_i in front of the column vectors of A, as in

$$Ax = x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} + \cdots + x_m \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix}.$$

Because we are used to thinking of the x_i as variables or unknowns, let's substitute in a "numerical" value, such as α_i. Of course, this has not changed anything, but it might look different to you,

$$A \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix} = \alpha_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} + \alpha_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} + \cdots + \alpha_m \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix}.$$

Linear Combination of the Columns of A

Definition The following sum of scalars times vectors,

$$\alpha_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} + \alpha_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} + \cdots + \alpha_m \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix}, \tag{7.6}$$

is called a **linear combination** of the columns of A. When we set $A\alpha = b$, and turn it around as $b = A\alpha$, we arrive at an important **Fact:** a vector $\alpha \in \mathbb{R}^m$ is a solution to $Ax = b$ if, and only if

$$b = \alpha_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} + \alpha_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} + \cdots + \alpha_m \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix}, \tag{7.7}$$

that is, b can be expressed as a linear combination of the columns of A. We will revisit this later, but (7.7) is a primary motivation for introducing the notion of linear combinations.

7.2.2 Linear Combinations in $\mathbb{R}^n$

Linear Combination

A vector $v \in \mathbb{R}^n$ is a **linear combination of** $\{u_1, u_2, \ldots, u_m\} \subset \mathbb{R}^n$ if there exist real numbers $\alpha_1, \alpha_2, \ldots, \alpha_m$ such that

$$v = \alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_m u_m. \tag{7.8}$$

Example 7.3 *Because*

$$\underbrace{\begin{bmatrix} -3 \\ -5 \\ -7 \end{bmatrix}}_{v} = 2 \underbrace{\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}}_{u_1} -9 \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{u_2},$$

we have that v is a linear combination of $\{u_1, u_2\}$.

When you are given the coefficients, it is easy to observe that a given vector is a linear combination of other vectors. But when you have to check if such coefficients exist or do not exist, then it's just a wee bit more challenging!

Example 7.4 *Is the vector* $v = \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix}$ *a linear combination of*

$$u_1 = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \text{ and } u_2 = \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix}?$$

Solution: The vector v is a linear combination of the vectors u_1 and u_2 if, and only if, there exist real numbers a_1 and a_2 such that

$$a_1 u_1 + a_2 u_2 = v. \tag{7.9}$$

What we have done here is apply[1] **the definition of a linear combination.** So, we write down the linear equations corresponding to (7.9), namely

$$\begin{gathered} a_1 \begin{bmatrix} 3 \\ 1 \\ -1 \end{bmatrix} + a_2 \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix} \\ \Updownarrow \\ \begin{bmatrix} 3 & 2 \\ 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix} \end{gathered} \tag{7.10}$$

and see if we can find a solution for the **unknowns** a_1 and a_2! What makes this look hard is that we have a non-square (rectangular) system of linear equations, namely, three equations and two unknowns.

What are the three equations? They may look more familiar to you if we write them out like this,

$$\begin{aligned} 3a_1 + 2a_2 &= 4 \\ a_1 - 2a_2 &= 4 \\ -a_1 + a_2 &= 4. \end{aligned}$$

A systematic way to approach such problems is to observe that if a_1 and a_2 are to simultaneously satisfy all three equations, then they must necessarily satisfy any two of them. Moreover, if you find that the solution to the resulting square system of two equations and two unknowns is unique, then two things are possible:

- the solution of the smaller square system of equations also satisfies the equation(s) you did not use, giving you a solution to the full set of equations, or
- the solution of the smaller square system of equations does not satisfy the equation(s) you did not use, telling you that the full set of equations is inconsistent and does not have a solution.

In our case, if we remove the last equation, we have

$$\begin{bmatrix} 3 & 2 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \tag{7.11}$$

and we observe that the matrix multiplying the vector of unknowns has determinant -8, and thus (7.11) has a unique solution. A bit of matrix magic with our formula for a 2 × 2 inverse gives

$$a_1 = 2 \text{ and } a_2 = -1.$$

Do these values satisfy the equation we did not use, namely, the last row of (7.11),

$$\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 4 \end{bmatrix}?$$

[1] In the beginning, many of us are confused about how to go about solving a problem. Sound advice is to start with the definitions at your disposal. Then try to turn the definition into a set of equations to be solved.

Clearly not, because $-1a_1 + a_2 = -3 \neq 4$, and **hence, v is NOT a linear combination of u_1 and u_2.**

The key point is that a_1 and a_2 must satisfy all three equations in (7.10). It does not matter in what order we use the equations when seeking a solution. To illustrate that, we'll first solve the last two equations,

$$\begin{aligned} a_1 - 2a_2 &= 4 \\ -a_1 + a_2 &= 4, \end{aligned}$$

which yields, $a_1 = -12$ and $a_2 = -8$. We then check if these values satisfy the first equation

$$3a_1 + 2a_2 = 4 \iff -36 - 16 = 4 \iff -52 = 4,$$

which is false, and therefore, v is NOT a linear combination of u_1 and u_2.

Rework the problem: Let's keep u_1 and u_2 as given and change v to

$$\tilde{v} = \begin{bmatrix} 0 \\ -8 \\ 5 \end{bmatrix}.$$

Applying the above strategy to the equations

$$\begin{aligned} 3a_1 + 2a_2 &= 0 \\ a_1 - 2a_2 &= -8 \\ -a_1 + a_2 &= 5 \end{aligned}$$

yields $a_1 = -2$ and $a_2 = 3$, as you can verify by direct substitution. **Hence, $\tilde{v}$ is a linear combination of u_1 and u_2.** ■

Remark: Checking whether a given vector can be written as a linear combination of other vectors always comes down to solving a system of linear equations. If that system is square with a non-zero determinant, then solving the system of equations is cake. Otherwise, solving the equations by hand is mostly painful. **We will need to develop a better way to check if a vector is, or is not, a linear combination of other vectors!**

7.3 Existence of Solutions to Ax = b

Because it is so important, we now revisit the relation of linear combinations to the existence of solutions to systems of linear equations. We want to make sure you did NOT miss something important when reading Chapter 7.2.1. If you are confident of your knowledge, then feel free to move on to the next section. Otherwise, please read on!

Let A be an $n \times m$ matrix, not necessarily square, as in (7.3). Recall that columns of A and vectors in $\mathbb{R}^n$ are the same thing. We write out $Ax = b$ is all its glory,

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{b}. \tag{7.12}$$

Existence of Solutions

The equation $Ax = b$ has a solution if, and only if, b can be written as a linear combination of the columns of A.

The following is more or less a proof: Suppose that $\bar{x}$ satisfies $A\bar{x} = b$, which is the same thing as saying $\bar{x}$ is a solution of $Ax = b$. Then, doing the indicated multiplication of $A\bar{x}$ via our now favorite method, the columns of A times the rows of the vector $\bar{x}$, which are its scalar entries $\bar{x}_i$, yields

$$\begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} \bar{x}_1 + \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} \bar{x}_2 + \cdots + \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix} \bar{x}_m = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{7.13}$$

Exchanging the two sides of the equal sign and moving the scalars $\bar{x}_i$ to the front of the vectors give

$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \bar{x}_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} + \bar{x}_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} + \cdots + \bar{x}_m \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix}; \tag{7.14}$$

in other words, b is a linear combination of the columns of A. The other way around works as well: if we manage to write

$$b = c_1 a_1^{\text{col}} + c_2 a_2^{\text{col}} + \cdots c_m a_m^{\text{col}}$$

for some real numbers $c_i \in \mathbb{R}$, then $\bar{x} = \begin{bmatrix} c_1 & c_2 & \cdots & c_m \end{bmatrix}^\top$ satisfies $A\bar{x} = b$, and hence it is a solution to $Ax = b$.

Recall that $a_j^{\text{col}} := \begin{bmatrix} a_{ij} & a_{2j} & \cdots & a_{nj} \end{bmatrix}^\top, 1 \leq j \leq m.$

A weakness of the above result is that we do not yet have an effective means for checking whether or not a given vector is a linear combination of a specified set of vectors. We will solve this problem too, a little bit later in the Chapter. All we have done so far is indicate that the concept of a linear combination is related to the existence of a solution to a corresponding system of linear equations. **The result is mainly conceptual in that we have not yet provided a practical way to check this in code. That will come, though! Long Live the LU Factorization!**

7.4 Linear Independence of a Set of Vectors

7.4.1 Preamble

The concept of *linear independence*, or its logical opposite, the concept of *linear dependence*, is one of the most important ideas in all of linear algebra. In the beginning, most of us struggle with two things: (1) why is the property of linear independence so important; and (2), how do you test for it? In this section, we highlight the importance of linear independence of vectors by relating it to uniqueness of solutions to $\mathbf{Ax} = \mathbf{0}$ and eventually, to uniqueness of solutions to $\mathbf{Ax} = \mathbf{b}$. We'll also show that our friend, the LU Factorization, makes testing for linear independence very straightforward.

7.4.2 Linear Independence through the Lens of $Ax = 0$

let A be an $n \times m$ matrix. We say that $x \in \mathbb{R}^m$ is a **nontrivial solution** to $Ax = 0_{n\times 1}$ if

- $Ax = 0_{n\times 1}$ (x is a solution), and
- $x \neq 0_{m\times 1}$ (x is not the zero vector in $\mathbb{R}^m$).

Because $x = 0$ is always a solution to $Ax = 0$, for it to be the **unique solution**, there must not be a non-trivial solution to $Ax = 0$.

Why are making such a big deal about this? Because uniqueness is a desirable property and we want to learn how to check for it even in the case of rectangular systems of equations[2]. Our immediate goal is to see what this means in terms of the columns of A, just as we did when introducing the notion of a linear combination.

[2] Recall, when A is square, uniqueness is equivalent to $\det(A) \neq 0$.

To see what it means to have "non-trivial solutions" to $Ax = 0_{n\times 1}$, once again, we replace $x \in \mathbb{R}^m$ by a vector $\alpha \in \mathbb{R}^m$ and write

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \text{ and } \alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}. \tag{7.15}$$

Then, using our column times row method for matrix multiplication, and re-arranging terms a bit as we did in Chapter 7.2.1,

$$\begin{aligned} A\alpha &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix} \\ &= \alpha_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} + \alpha_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} + \cdots + \alpha_m \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix}. \end{aligned} \tag{7.16}$$

Hence, $A\alpha = 0_{n\times 1}$ if, and only if,

$$\alpha_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} + \alpha_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} + \cdots + \alpha_m \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{n\times 1}. \tag{7.17}$$

Our takeaway is, $\alpha = 0_{m\times 1}$ is the unique solution to $A\alpha = 0$ if, and only if, the only way we can add up the columns of A and obtain the zero vector in $\mathbb{R}^n$ is with

$$\alpha_1 = 0, \alpha_2 = 0, \dots, \alpha_m = 0.$$

Or equivalently, $A\alpha = 0$ has a non-trivial solution $\alpha \neq 0_{m\times 1}$, if, and only if, there exists real numbers $\alpha_1, \alpha_2, \dots, \alpha_m$, **NOT ALL ZERO**, such that (7.17) is satisfied.

7.4.3 Linear Independence in $\mathbb{R}^n$ (and why theory matters)

The definition of linear independence of an "abstract" set of vectors in $\mathbb{R}^n$ is 100% motivated by the study above on the uniqueness of solutions to $Ax = 0$. Of course, vectors in $\mathbb{R}^n$ are columns of $n \times m$ matrices, so who's to say what is abstract and what is not!

Linear Independence of a Set of Vectors

The set of vectors $\{v_1, v_2, ..., v_m\} \subset \mathbb{R}^n$ is **linearly dependent** if there exist real numbers $\alpha_1, \alpha_2, \dots, \alpha_m$ **NOT ALL ZERO** yielding a linear combination of vectors that adds up to the zero vector,

$$\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_m v_m = 0. \tag{7.18}$$

On the other hand, the vectors $\{v_1, v_2, ..., v_m\}$ are **linearly independent** if the **only** real numbers $\alpha_1, \alpha_2, \dots, \alpha_m$ yielding a linear combination of vectors that adds up to the zero vector,

$$\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_m v_m = 0, \tag{7.19}$$

are $\boldsymbol{\alpha_1 = 0, \alpha_2 = 0, \dots, \alpha_m = 0}$.

Concise Definition of Linear Independence:

$$\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_m v_m = 0_{n\times 1} \iff \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Example 7.5 *By applying the definition, determine if the set of vectors*

$$v_1 = \begin{bmatrix} \sqrt{2} \\ 0 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 4 \\ 7 \\ 0 \end{bmatrix}, v_3 = \begin{bmatrix} 3 \\ 1 \\ -1 \end{bmatrix}$$

is linearly independent or dependent.

Solution: We form the linear combination and do the indicated multiplications and additions

$$\alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = \alpha_1 \begin{bmatrix} \sqrt{2} \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 4 \\ 7 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 3 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}\,\alpha_1 + 4\,\alpha_2 + 3\,\alpha_3 \\ 7\,\alpha_2 + \alpha_3 \\ -1\,\alpha_3 \end{bmatrix}. \tag{7.20}$$

Setting the right hand side of (7.20) to the zero vector yields

$$\begin{bmatrix} \sqrt{2}\,\alpha_1 + 4\,\alpha_2 + 3\,\alpha_3 \\ 7\,\alpha_2 + \alpha_3 \\ -1\,\alpha_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{7.21}$$

This is one of our friendly triangular systems of linear equations which we can solve via back substitution. We see immediately that the only solution to the bottom equation is $\alpha_3 = 0$, the only solution to the middle equation is then $\alpha_2 = 0$, and finally, the only solution to the top equation is $\alpha_1 = 0$. Hence, the only solution to

$$\alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = 0$$

is $\alpha_1 = 0$, $\alpha_2 = 0$, and $\alpha_3 = 0$, and hence the set of vectors $\{v_1, v_2, v_3\}$ is linearly independent.

The above analysis becomes even more clear when we write (7.21) as

$$\begin{bmatrix} \sqrt{2} & 4 & 3 \\ 0 & 7 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{7.22}$$

■

Example 7.6 *By applying the definition, determine if the set of vectors*

$$v_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, v_2 = \begin{bmatrix} 1 \\ -2 \\ -4 \end{bmatrix}$$

is linearly independent or dependent.

Solution: We seek to determine if there are non-zero coefficients α_1 and α_2 resulting in a linear combination that forms the zero vector in $\mathbb{R}^3$,

$$\alpha_1 v_1 + \alpha_2 v_2 = \alpha_1 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 \\ -2 \\ -4 \end{bmatrix} = \begin{bmatrix} \alpha_1 + \alpha_2 \\ 2\alpha_1 - 2\alpha_2 \\ 3\alpha_1 - 4\alpha_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{7.23}$$

We observe that we have three equations in two unknowns and no obvious triangular structure to help us! What do we do? Well, if we can take any two of the three equations and show that the only solution is the trivial solution, $\alpha_1 = 0$ and $\alpha_2 = 0$, then we are done, because the trivial solution will always satisfy the remaining equation. Please check this reasoning out for yourself.

We arbitrarily group the equations into the first two equations and then the last equation, as follows

$$\alpha_1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{7.24}$$

$$\alpha_1 \begin{bmatrix} 3 \end{bmatrix} + \alpha_2 \begin{bmatrix} -4 \end{bmatrix} = 0. \tag{7.25}$$

We can rewrite (7.24) in the form $A\alpha = b$

$$\underbrace{\begin{bmatrix} 1 & 1 \\ 2 & -2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}}_{\alpha} = \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{b}. \tag{7.26}$$

We note that $\det(A) = (1)(-2) - (1)(2) = -4 \neq 0$, and hence (7.26) has a unique solution. Because $\alpha_1 = 0$ and $\alpha_2 = 0$ is a solution and the solution is unique, we know that there cannot exist a different set of non-zero α_1 and α_2 that also solve the equation. We therefore conclude that the vectors $\{v_1, v_2\}$ are linearly independent.

Remark: Let's step back and see what is going on here. We have three equations and two unknowns. The trivial solution is always a solution to the full set of equations. We wonder if there is any other solution. If we make a choice of two of the equations, so that we have a more manageable square system of equations, and then find that those two equations constrain the solution to being the trivial solution, we are done! (Why, because the trivial solution will automatically satisfy the remaining equation and it is then the only solution that satisfies all three equations.) ■

That was a lot of work! It does not seem like it will scale to bigger sets of vectors. We'll do one more example to convince you that we are in dire need of a **Pro Tip!**

Example 7.7 *By applying the definition, determine if the vectors*

$$v_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ -2 \\ 4 \\ 5 \end{bmatrix}, v_3 = \begin{bmatrix} 2 \\ 6 \\ 2 \\ -3 \end{bmatrix}$$

are linearly independent or dependent.

Solution: We form the linear combination and set it equal to zero,

$$\alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = \alpha_1 \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ -2 \\ 4 \\ 5 \end{bmatrix} + \alpha_3 \begin{bmatrix} 2 \\ 6 \\ 2 \\ -3 \end{bmatrix} = \begin{bmatrix} \alpha_1 + 2\,\alpha_3 \\ 2\alpha_1 - 2\alpha_2 + 6\alpha_3 \\ 3\alpha_1 + 4\alpha_2 + 2\alpha_3 \\ \alpha_1 + 5\alpha_2 - 3\alpha_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{7.27}$$

We must check whether or not there are *non-trivial* solutions to (7.27), where non-trivial means at least one of the coefficients α_1, α_2, or α_3 is non-zero.

We observe that we have four equations in three unknowns. Similar to Example 7.6, we could select any three of the four equations, solve those three, and then see if that solution is compatible with the remaining equation. Once again, this will be a lot of work. We'll save ourselves the effort and let you verify that $\alpha_1 = 2, \alpha_2 = -1, \alpha_3 = -1$ is a non-trivial solution to (7.27) and hence the vectors v_1, v_2, v_3 are linearly dependent. **Where is that Pro Tip?** ■

7.4.4 A Pro Tip for Checking Linear Independence

Pro-tip! Linear Independence in a Nutshell

Consider the vectors in $\mathbb{R}^n$,

$$\left\{ v_1 = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix}, v_2 = \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix}, ..., v_m = \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix} \right\},$$

and use them as the columns of a matrix that we call A,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}. \tag{7.28}$$

The following statements are equivalent:

- The set of vectors $\{v_1, v_2, \ldots, v_m\}$ is linearly independent.
- The $m \times m$ matrix $A^\top \cdot A$ is invertible.
- $\det(A^\top \cdot A) \neq 0$.
- For any LU Factorization $P \cdot (A^\top \cdot A) = L \cdot U$ of $A^\top A$, the $m \times m$ upper triangular matrix U has no zeros on its diagonal.

We'll prove the Pro Tip shortly. For now, we'll focus on how it makes our lives so much easier in terms of checking linear independence.

What if you really want to know a set of non-trivial coefficients such that $A\alpha = 0$? Instead of solving $A\alpha = 0$, you can solve the triangular system of equations, $U\alpha = 0$. It is emphasized that we only do the additional work of solving $U \cdot \alpha = 0$ if we want to find the explicit coefficients $\alpha_1, \alpha_2, \ldots, \alpha_m$ such that

$$\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_m v_m = 0.$$

Many times we do not really need the coefficients. In HW and Quizzes, we'll tell you if you need to find the coefficients or whether a YES vs NO answer on linear independence is acceptable.

Example 7.8 *We apply the Pro Tip to Example 7.5*

Solution: We use the vectors $v_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ and $v_2 = \begin{bmatrix} 1 \\ -2 \\ -4 \end{bmatrix}$ as the columns of a matrix $A := [v_1 \;\; v_2]$, so that

$$A = \begin{bmatrix} 1 & 1 \\ 2 & -2 \\ 3 & -4 \end{bmatrix}.$$

We compute that

$$A^\top \cdot A = \begin{bmatrix} 14.0 & -15.0 \\ -15.0 & 21.0 \end{bmatrix}.$$

Because it is 2×2, we can compute its determinant easily and obtain

$$\det(A^\top \cdot A) = 69.0 \neq 0,$$

and hence the vectors $\{v_1, v_2\}$ are linearly independent. ■

Example 7.9 *We apply the Pro Tip to Example 7.6.*

Solution: We use the vectors

$$v_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ -2 \\ 4 \\ 5 \end{bmatrix}, v_3 = \begin{bmatrix} 2 \\ 6 \\ 2 \\ -3 \end{bmatrix}$$

and form the matrix

$$A := \begin{bmatrix} 1 & 0 & 2 \\ 2 & -2 & 6 \\ 3 & 4 & 2 \\ 1 & 5 & -3 \end{bmatrix}.$$

We go to Julia and compute that

$$A^\top \cdot A = \begin{bmatrix} 15.0 & 13.0 & 17.0 \\ 13.0 & 45.0 & -19.0 \\ 17.0 & -19.0 & 53.0 \end{bmatrix},$$

and that its LU Factorization is $P \cdot \left(A^\top \cdot A\right) = L \cdot U$, where

$$P = \begin{bmatrix} 0.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \end{bmatrix}, \; L = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.8 & 1.0 & 0.0 \\ 0.9 & 0.5 & 1.0 \end{bmatrix}, \text{ and } U = \begin{bmatrix} 17.0 & -19.0 & 53.0 \\ 0.0 & 59.5 & -59.5 \\ 0.0 & 0.0 & \boxed{-0.0} \end{bmatrix}.$$

We observe that U has a zero on its diagonal and hence the set $\{v_1, v_2, v_3\}$ is linearly dependent. ■

Yeah, that Pro Tip on Linear Independence is certainly worth using! Can something like it be used for testing whether a given vector is (or is not) a linear combination of another set of vectors? The answer is YES! And we'll get to that after we show why our current Pro Tip works.

7.4.5 (Optional Read): Why the Pro Tip Works

We first consider a vector in $\mathbb{R}^n$ that we denote as

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

We next note that

$$y^\top y = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = (y_1)^2 + (y_2)^2 + \cdots + (y_n)^2.$$

From this, we deduce the useful fact that

$$y = 0_{n \times 1} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \iff y^\top \cdot y = 0.$$

To determine linear independence or dependence, we are looking to exclude (or find) non-trivial solutions to $A\alpha = 0$, where

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix},$$

and $A = \begin{bmatrix} v_1 & v_2 & \cdots & v_m \end{bmatrix}$, the matrix formed from our set of vectors $\{v_1, v_2, \cdots, v_m\}$. Motivated by this, we let

$$y = A\alpha.$$

We then note the following chain of implications[3]

$$(A\alpha = 0) \implies \left(A^\top \cdot A\alpha = 0\right) \implies \left(\alpha^\top A^\top \cdot A\alpha = 0\right) \implies \left((A\alpha)^\top \cdot (A\alpha) = 0\right) \implies (A\alpha = 0)\,,$$

where the last implication follows from $y = 0_{n\times 1} \iff y^\top y = 0$.

From logic, we know that when we have

$$(a) \implies (b) \implies (c) \implies (d) \implies (a),$$

a chain of implications that begins and ends with the same proposition, then we deduce that

$$(a) \iff (b) \iff (c) \iff (d).$$

In our case, we only interested in $(a) \iff (b)$, that is,

$$\boxed{A\alpha = 0 \iff \left(A^\top A\right)\alpha = 0.} \tag{7.29}$$

We next note that the matrix $A^\top A$ is $m \times m$, because it is the product of $m \times n$ and $n \times m$ matrices, $A^\top$ and A, respectively. Hence, the equation

$$\left(A^\top A\right)\alpha = 0 \tag{7.30}$$

has a unique solution if, and only if, $\det(A^\top A) \neq 0$.

Now, why are we done? If $\alpha = 0_{m\times 1}$ is the ONLY solution to (7.30), then it is also the only solution to $A\bar{\alpha} = 0$, and we deduce that the columns of A are linearly independent. If $\alpha = 0_{m\times 1}$ is not a unique solution to (7.30), then there exists a non-zero vector $\bar{\alpha} \in \mathbb{R}^m$ that is also a solution to (7.30), meaning that $\left(A^\top A\right)\bar{\alpha} = 0$. But we know from (7.29) that this also means that $\bar{\alpha} \neq 0$ is a solution of $A\bar{\alpha} = 0$, and hence the columns of A are linearly dependent. ■

Don't miss seeing the forest for the trees!

Don't let the last details of the proof distract you too much. The main steps of the Pro Tip are

- [rectangular system of equations] $A\alpha = 0 \iff A^\top \cdot A\alpha = 0$ [square system of equations].
- The square system of equations $A^\top \cdot A\alpha = 0$ has a unique solution of $\alpha = 0$, the 0-vector in $\mathbb{R}^m$, if, and only if, $\det(A^\top \cdot A) \neq 0$.
- Hence, $A\alpha = 0$ has a unique solution of $\alpha = 0$, the 0-vector in $\mathbb{R}^m$, if, and only if, $\det(A^\top \cdot A) \neq 0$.
- Our final results is, $A\alpha = 0$ has a unique solution of $\alpha = 0$, the 0-vector in $\mathbb{R}^m$, if, and only if, the columns of A are linearly independent, where the last implication uses the **definition of linear independence**.

7.5 Number of Linearly Independent Vectors in a Set

Consider a finite set of vectors in $\mathbb{R}^n$, such as $\{v_1, \ldots, v_m\}$. Is there an intelligent way to talk about the maximum number of linearly independent vectors in the set? That is, the number of vectors that would remain in we discard the fewest vectors so that the remaining vectors are linearly independent?

In fact there is. As illustrated in Fig. 7.3, we can start from left to right and ask, is the set $\{v_1\}$ linearly independent? If it is, keep v_1 and if not, discard it (meaning, in this case, v_1 was the zero vector). For the sake of argument, let's suppose that $v_1 \neq 0$ and hence we keep it. Next, we ask, is the set $\{v_1, v_2\}$ linearly independent? If not, then v_2 is a linear combination of v_1 and we discard it, otherwise, we keep it. For the sake of argument, let's suppose that v_2 is a linear combination of v_1 and hence we discard it. We next ask, is the set $\{v_1, v_3\}$ linearly independent? Let's say it is, and then we would ask if the set $\{v_1, v_3, v_4\}$ is linearly independent, etc.

[3] The second implication follows from the first by multiplying on the left by $A^\top$. The third implication follows from the second by multiplying on the left by $\alpha^\top$. The fourth implication follows from the third by recognizing that $\alpha^\top A^\top = (A\alpha)^\top$. The last implication uses our fact that $y = 0 \iff y^\top \cdot y = 0$.

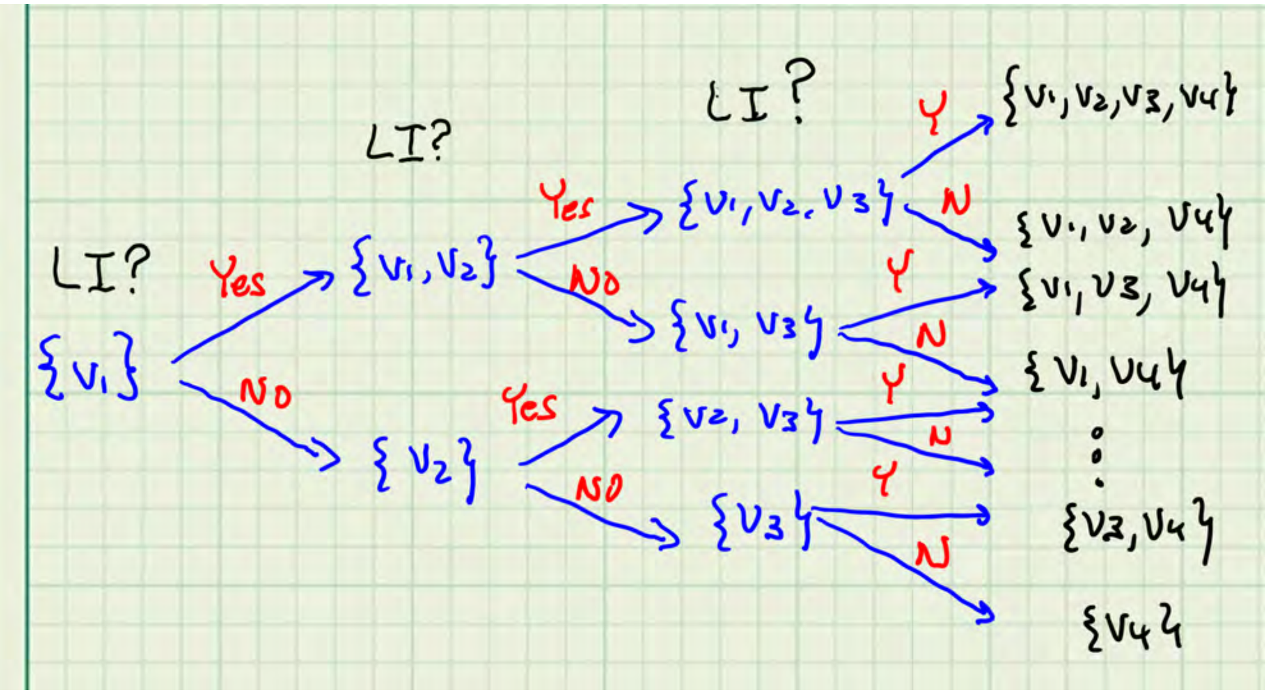


Figure 7.3: Checking linear independence from left to right. You could also start from the right and go to the left, or you could start in the middle and proceed to the two ends. You just need to do an organized search of the vectors!

In the end, we have built the largest set of linearly independent vectors from the given set and we can ask, how many elements does it contain?

Number of Linearly Independent Vectors in a Finite Set

The following statements are equivalent:

- One can divide the set of vectors in $\mathbb{R}^n$

$$v_1 = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix}, v_2 = \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix}, ..., v_m = \begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix} \tag{7.31}$$

  into k linearly independent vectors and $m - k$ vectors that are linearly dependent on them.

- The matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \tag{7.32}$$

  has k linearly independent columns and $m - k$ columns that are linearly dependent on them.

- Let $P \cdot \left(A^\top A\right) = L \cdot U$ be an LU Factorization of $A^\top A$. Then U has k linearly independent columns and $m - k$ dependent columns. Because U is triangular, as in Example 7.5, checking linear independence is much easier than for the original matrix A.

Example 7.10 *How many columns of the matrix*

$$A = \begin{bmatrix} -0.2 & -0.2 & -0.4 & 0.3 & 0.3 \\ 0.3 & 1.0 & -0.1 & -1.1 & -1.7 \\ 0.7 & -1.9 & 1.5 & -0.0 & -3.0 \\ 0.9 & -1.0 & -0.7 & 0.6 & -1.8 \\ -0.5 & 0.8 & -1.1 & -0.5 & -0.5 \\ -2.0 & -0.9 & -0.5 & 0.2 & 0.3 \\ -1.0 & 0.6 & 0.7 & -0.9 & 0.2 \end{bmatrix}_{7\times 5} \tag{7.33}$$

are linearly independent? Doing this as shown in Fig. 7.3 would be painful.

Solution: We turn to Julia and perform the LU Factorization: $\mathbf{P} \cdot (\mathbf{A}^\top \cdot \mathbf{A}) = \mathbf{L} \cdot \mathbf{U}$, for which we only report the upper triangular matrix

$$U = \begin{bmatrix} 6.680 & -1.090 & 1.320 & 0.900 & -4.840 \\ 0.000 & 7.282 & -1.965 & -2.733 & 4.400 \\ 0.000 & 0.000 & 4.069 & -1.525 & -0.506 \\ 0.000 & 0.000 & 0.000 & 3.124 & 9.371 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix}_{5\times 5} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 & u_5 \end{bmatrix}_{5\times 5}, \tag{7.34}$$

where we have labeled its columns as $u_1 \dots u_5$. Working from left to right with the columns of U, we have that because $u_1 \neq 0_{5\times 1}$, the set $\{u_1\}$ is linearly independent. We next check the set the set $\{u_1, u_2\}$. To emphasize the beauty of the triangular structure in U, we check if there exist non-trivial solutions to

$$\alpha_1 u_1 + \alpha_2 u_2 = \begin{bmatrix} 6.680 & -1.090 \\ 0.000 & 7.282 \\ 0.000 & 0.000 \\ 0.000 & 0.000 \\ 0.000 & 0.000 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}.$$

There answer is clearly no[4], the unique solution is $\alpha_1 = 0$ and $\alpha_2 = 0$, and thus $\{u_1, u_2\}$ is linearly independent. The same reasoning works for $\{u_1, u_2, u_3\}$ and $\{u_1, u_2, u_3, u_4\}$. Indeed, we could have jumped straight to the set of vectors $\{u_1, u_2, u_3, u_4\}$ because checking its linear independence comes down to looking for solutions to

$$\begin{bmatrix} \boxed{6.680} & -1.090 & 1.320 & 0.900 \\ 0.000 & \boxed{7.282} & -1.965 & -2.733 \\ 0.000 & 0.000 & \boxed{4.069} & -1.525 \\ 0.000 & 0.000 & 0.000 & \boxed{3.124} \\ 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}.$$

Ignoring the final row of zeros, we really have a square triangular system with a non-zero diagonal, hence the trivial solution $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0$ is, in fact, the unique solution, proving linear independence.

What about $\{u_1, u_2, u_3, u_4, u_5\}$? The answer is no, and to see this we note that

$$\begin{gathered} \alpha_1 u_1 + \alpha_2 u_2 + \alpha_3 u_3 + \alpha_4 u_4 + \alpha_5 u_5 = 0_{5\times 1} \\ \Updownarrow \\ \alpha_1 u_1 + \alpha_2 u_2 + \alpha_3 u_3 + \alpha_4 u_4 = -\alpha_5 u_5 \end{gathered}$$

Writing down this equation yields

$$\begin{bmatrix} \boxed{6.680} & -1.090 & 1.320 & 0.900 \\ 0.000 & \boxed{7.282} & -1.965 & -2.733 \\ 0.000 & 0.000 & \boxed{4.069} & -1.525 \\ 0.000 & 0.000 & 0.000 & \boxed{3.124} \\ 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = - \begin{bmatrix} -4.840 \\ 4.400 \\ -0.506 \\ 9.371 \\ 0.000 \end{bmatrix} \alpha_5.$$

[4] There are three rows of zeros. Hence, starting from the second row, back substitution provides that the unique answer is zero

If we set $\alpha_5 = -1$, for example, and once again ignore the bottom row of zeros because they do not affect the solution of the equations, we can solve the resulting triangular system of equations for α_1 through α_4, giving us a non-trivial solution to $\alpha_1 u_1 + \cdots \alpha_5 u_5 = 0$. Hence, $\{u_1, u_2, u_3, u_4, u_5\}$ is linearly dependent. ■

Yes, the triangular structure of U is very helpful, but it still requires a lot of work to check for solutions. Is there anything like our Pro-Tip for linear independence that we can apply for counting the maximum number of linear independent vectors?

Semi-Pro Tip: When Is it Enough to Look at the Diagonal of U?

Let $P \cdot \left(A^\top A\right) = L \cdot U$ be an LU Factorization of $A^\top A$, and define k as above to be the number of linearly independent columns of U. When is k equal to the number of non-zero elements on the diagonal of U?

- If the diagonal of U has no zero elements, then $k = m$ and all columns of A are linearly independent.
- If the diagonal of U has one zero element, then $k = m - 1$ and one can select $k = m - 1$ linearly independent columns from A.
- If the diagonal of U has two or more zero elements, then additional computations are necessary. For example, if the rows of U that have zeros on their diagonal element are identically zero, then k equals the number of non-zero elements on the diagonal of U.

Remark: The root of the extra complication in counting independent vectors with this method lies in what is called a non-trivial Jordan structure, something that is taught in advanced linear algebra courses. When U has more than one zero on its diagonal, then, theoretically, it may have a non-trivial Jordan structure. In physical data, non-trivial Jordan structures are exceedingly rare; in fact, they have "probability zero" of occurring.

Bottom line: We can often count the number of linearly independent vectors by doing an LU Factorization. An alternative geometric technique will be available to us when we study the Gram-Schmidt Algorithm.

Example 7.11 *We revisit Example 7.10: how many columns of the matrix*

$$A = \begin{bmatrix} -0.2 & -0.2 & -0.4 & 0.3 & 0.3 \\ 0.3 & 1.0 & -0.1 & -1.1 & -1.7 \\ 0.7 & -1.9 & 1.5 & -0.0 & -3.0 \\ 0.9 & -1.0 & -0.7 & 0.6 & -1.8 \\ -0.5 & 0.8 & -1.1 & -0.5 & -0.5 \\ -2.0 & -0.9 & -0.5 & 0.2 & 0.3 \\ -1.0 & 0.6 & 0.7 & -0.9 & 0.2 \end{bmatrix}_{7\times 5} \tag{7.35}$$

are linearly independent?

Solution: We turn to Julia and perform the LU Factorization: $\mathbf{P} \cdot (\mathbf{A}^\top \cdot \mathbf{A}) = \mathbf{L} \cdot \mathbf{U}$, for which we first report the diagonal of U

$$\text{diag}(U) = \begin{bmatrix} 6.680 & 7.282 & 4.069 & 3.124 & 0.000 \end{bmatrix}_{1\times 5} \tag{7.36}$$

Because there is a single zero on the diagonal of U and four non-zero elements, we conclude that A has four linearly independent columns. We do not need to look at U, but for completeness, we show it anyway,

$$U = \begin{bmatrix} 6.680 & -1.090 & 1.320 & 0.900 & -4.840 \\ 0.000 & 7.282 & -1.965 & -2.733 & 4.400 \\ 0.000 & 0.000 & 4.069 & -1.525 & -0.506 \\ 0.000 & 0.000 & 0.000 & 3.124 & 9.371 \\ 0.000 & 0.000 & 0.000 & 0.000 & \boxed{0.000} \end{bmatrix}_{5\times 5} \tag{7.37}$$

■

Example 7.12 *How many columns of the matrix*

$$A = \begin{bmatrix} -0.2 & -0.2 & -0.4 & 0.3 & 0.3 & -0.5 \\ 0.3 & 1.0 & -0.1 & -1.1 & -1.7 & 0.1 \\ 0.7 & -1.9 & 1.5 & -0.0 & -3.0 & 0.3 \\ 0.9 & -1.0 & -0.7 & 0.6 & -1.8 & -0.2 \\ -0.5 & 0.8 & -1.1 & -0.5 & -0.5 & -1.3 \\ -2.0 & -0.9 & -0.5 & 0.2 & 0.3 & -3.2 \\ -1.0 & 0.6 & 0.7 & -0.9 & 0.2 & -0.6 \end{bmatrix}_{7\times 6} \tag{7.38}$$

are linearly independent?

Solution: We turn to Julia and perform the LU Factorization: $\mathbf{P}\cdot(\mathbf{A}^\top\cdot\mathbf{A}) = \mathbf{L}\cdot\mathbf{U}$, for which we first report the diagonal of U

$$\text{diag}(U) = \begin{bmatrix} 7.810 & 7.643 & 3.751 & 3.682 & 0.000 & 0.000 \end{bmatrix}_{1\times 6} \tag{7.39}$$

which has four non-zero elements. However, because more than one element of the diagonal is zero, we need to look at the rows of U having zero elements on their diagonal. We report all of U

$$U = \begin{bmatrix} \boxed{7.810} & 1.310 & 3.390 & 0.170 & -1.290 & 12.680 \\ 0.000 & \boxed{7.643} & -1.707 & -2.856 & 5.010 & 3.080 \\ 0.000 & 0.000 & \boxed{3.751} & -1.536 & -0.858 & 2.215 \\ 0.000 & 0.000 & 0.000 & \boxed{3.682} & 11.047 & 3.682 \\ 0.000 & 0.000 & 0.000 & 0.000 & \boxed{0.000} & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & \boxed{0.000} \end{bmatrix}_{6\times 6} \tag{7.40}$$

Because the rows of U with a zero on the diagonal are identically zero, we conclude that A has four linearly independent columns. You have to admit, it's pretty cool to the analysis in this manner. ■

Looking Ahead: We will later have a very nice name for the number of linearly independent vectors in a set of vectors: the **dimension of span**$\{v_1, v_2, \ldots, v_m\}$ For now, we're just counting the number of linearly independent vectors.

Remark: Why does the **Semi-Pro Tip** hold? For the benefit of future instructors, here are some remarks that are not meant to be understandable to students of this book: *think about it as Penn and Teller speaking in code to let a fellow magician know that they are on to their trick!* From the proof of our previous Pro Tip, we know that the null space of A is equal to the null space of $A^\top A$. Because both P and L are invertible, we also know that a vector v is in the null space of $A^\top \cdot A$ if, and only if, it is in the null space of U, so their respective null spaces are of equal dimension. It follows that $A^\top \cdot A$ has a zero eigenvalue if, and only if U has a zero eigenvalue. Because $A^\top A$ is symmetric, we know that its eigenvalues have equal geometric and algebraic multiplicities. However, this property does not necessarily carry over to U: its zero eigenvalue can have a non-trivial Jordan form, and hence strictly smaller geometric multiplicity than algebraic multiplicity. The diagonal of U is only checking the algebraic multiplicity. But if if whenever there is a zero on the diagonal the corresponding row of U consists only of zeros, then we can verify that the geometric multiplicity of the zero eigenvalue is equal to the number of zeros on the diagonal of U, giving us a nice test for checking the dimension of the null space of U, and in turn, the rank of A. Yeah! ■

Remark: (Optional Read) For those with a driving sense of curiosity or simply a high tolerance for pain, here is an example of a matrix A where the number of non-zero elements on the diagonal of U does not predict the number of linearly independent columns of A. Consider

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

A is symmetric, meaning that $A^\top = A$. Moreover, you can check that $A^\top \cdot A = A$. The following is a valid LU Factorization with row permutations

$$\underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{P} \cdot \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{A^\top \cdot A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} \boxed{0} & 1 & 0 \\ 0 & \boxed{0} & 0 \\ 0 & 0 & \boxed{0} \end{bmatrix}}_{U}.$$

We see that U has one linearly independent column (the same as A), while $\mathrm{diag}(U)$ has three zeros. Hence, the number of non-zero elements on the diagonal of U does not always correspond to the number of linearly independent columns of A and U. For the magicians, we note that the eigenvalue zero of U has algebraic multiplicity three and geometric multiplicity two.

7.6 Attractive Test for Linear Combinations

Semi-Pro Tip! Linear Combination or Not?

Fact: A vector $v_0 \in \mathbb{R}^n$ can be written as a linear combination of $\{v_1, \ldots, v_m\} \subset \mathbb{R}^n$ if, and only if, the set $\{v_0, v_1, \ldots, v_m\}$ has the same number of linearly independent vectors as $\{v_1, \ldots, v_m\}$.

Applying this to determining if a linear system of equations $Ax = b$ has a solution, we first define $A_\mathrm{e} := [A \;\; b]$ by appending b to the columns of A. Then we do the corresponding LU Factorizations

- $P \cdot \left(A^\top A\right) = L \cdot U$
- $P_\mathrm{e} \cdot \left(A_\mathrm{e}^\top A_\mathrm{e}\right) = L_\mathrm{e} \cdot U_\mathrm{e}$.

Fact: $Ax = b$ has a solution if, and only if, U and U_e have the same number of linearly independent columns. Most of the time, this is equivalent to their diagonals having the same number of non-zero elements.

Example 7.13 *We consider a rectangular system that is on the edge of what we would like to analyze by hand,*

$$\underbrace{\begin{bmatrix} 1.0 & 1.0 & 5.0 \\ 0.0 & 2.0 & 6.0 \\ 0.0 & 3.0 & 7.0 \\ 0.0 & 4.0 & 8.0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 9.0 \\ 10.0 \\ 11.0 \\ 12.0 \end{bmatrix}}_{b}. \tag{7.41}$$

Does it have a solution?

Solution: We seek to determine if (7.41) will have a solution. We do the indicated LU Factorizations

$$P \cdot \left(A^\top A\right) = L \cdot U \;\; \text{and} \;\; P_\mathrm{e} \cdot \left([A \;\; b]^\top [A \;\; b]\right) = L_\mathrm{e} \cdot U_\mathrm{e}$$

and report the diagonals of U and U_e as row vectors

$$\mathrm{diag}(U) = \begin{bmatrix} 5.0 & 16.0 & -1.2 \end{bmatrix}_{1\times 3} \tag{7.42}$$

$$\mathrm{diag}(U_e) = \begin{bmatrix} 9.0 & 17.8 & -1.2 & 0.0 \end{bmatrix}_{1\times 4}. \tag{7.43}$$

Based on $\mathrm{diag}(U)$ and $\mathrm{diag}(U_e)$, we see that U and U_e have the same number of linearly independent columns. Hence, b is a linear combination of the columns of A and therefore (7.41) has a solution. In fact, one can compute a solution is

$$x = \begin{bmatrix} 0.0 \\ -1.0 \\ 2.0 \end{bmatrix}. \tag{7.44}$$

We now change the vector b to

$$\begin{bmatrix} 9.0 \\ 10.0 \\ 11.0 \\ \sqrt{2} \end{bmatrix},$$

and repeat the analysis, giving

$$\text{diag}(U) = \begin{bmatrix} 5.0 & 16.0 & -1.2 \end{bmatrix}_{1\times 3} \tag{7.45}$$

$$\text{diag}(U_e) = \begin{bmatrix} 9.0 & 32.4 & -3.1 & 2.0 \end{bmatrix}_{1\times 4}. \tag{7.46}$$

This time, the analysis shows that $[A \;\; b]$ has one more linearly independent column that A, and hence b is linearly independent of the columns of A. We conclude, therefore, that the new system of linear equations does not have a solution! ■

7.7 Existence and Uniqueness of Solutions to $Ax = b$

In Chapter 7.3, we showed that $Ax = b$ has a solution if, and only if, b is a linear combination of the columns of A. Here, we will show that uniqueness of a solution follows from the columns of A being linearly independent. **When both of these conditions hold, we have existence and uniqueness of solutions.**

Let A be an $n \times m$ matrix, not necessarily square, and consider

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{b}. \tag{7.47}$$

Suppose that $\overline{x}$ satisfies $A\overline{x} = b$, which is the same thing as saying $\overline{x}$ is a solution of $Ax = b$. Suppose also that $\overline{\overline{x}}$ satisfies $A\overline{\overline{x}} = b$. We give conditions that guarantee $\overline{\overline{x}} = \overline{x}$.

If both $\overline{x}$ and $\overline{\overline{x}}$ satisfy $Ax = b$, we have that

$$A\overline{\overline{x}} - A\overline{x} = A\left(\overline{\overline{x}} - \overline{x}\right) = b - b = 0.$$

We define $\alpha := \overline{\overline{x}} - \overline{x}$ and write out its components as

$$\alpha := \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

Expanding the expression $A\alpha = 0$ in terms of the columns of A and the components of the vector α gives

$$\alpha_1 \underbrace{\begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix}}_{a_1^{\text{col}}} + \alpha_2 \underbrace{\begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix}}_{a_2^{\text{col}}} + \cdots + \alpha_m \underbrace{\begin{bmatrix} a_{1m} \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix}}_{a_m^{\text{col}}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{7.48}$$

We know that $\alpha_1 = 0, \alpha_2 = 0, \ldots, \alpha_m = 0$ is the unique solution to

$$\alpha_1 a_1^{\text{col}} + \alpha_2 a_2^{\text{col}} + \ldots \alpha_m a_m^{\text{col}} = 0,$$

if and only if, the vectors $\{a_1^{\text{col}}, a_2^{\text{col}}, \ldots, a_m^{\text{col}}\}$ are linearly independent. Hence, if $Ax = b$ has a solution, it will be unique if, and only if, the columns of A are linearly independent.

Existence and Uniqueness of Solutions to $Ax = b$

The following two statements are equivalent

(a) The system of linear equations $Ax = b$ has a solution, and, it is unique.

(b) b is a linear combination of the columns of A, and, the columns of A are linearly independent.

Remark: If b is not a linear combination of the columns of A, then $Ax = b$ does not have a solution. If the columns of A are not linearly independent, then if $Ax = b$ has one solution, it also has an infinite number of solutions.

Remark: Please return to Chapter 1.2 and re-work the examples using this knowledge. Our new results do not even require the system of equations to be square, and moreover, thanks to our Pro and Semi-Pro Tips, our results are not limited to a small number of equations that are amenable to hand calculations!

Redoing Example 7.13 with the above knowledge, we see that x in (7.44) is the unique solution to the system of linear equations (7.41).

Example 7.14 *Let's analyze an example that most of us could not get right if done by hand! We consider $Ax = b$, where*

$$A = \begin{bmatrix} -0.2 & -0.2 & -0.4 & 0.3 & 0.3 \\ 0.3 & 1.0 & -0.1 & -1.1 & -1.7 \\ 0.7 & -1.9 & 1.5 & -0.0 & -3.0 \\ 0.9 & -1.0 & -0.7 & 0.6 & -1.8 \\ -0.5 & 0.8 & -1.1 & -0.5 & -0.5 \\ -2.0 & -0.9 & -0.5 & 0.2 & 0.3 \\ -1.0 & 0.6 & 0.7 & -0.9 & 0.2 \end{bmatrix}_{7\times 5} \quad \text{and} \quad b = \begin{bmatrix} -0.5 \\ 0.1 \\ 0.3 \\ -0.2 \\ -1.3 \\ -3.2 \\ -0.6 \end{bmatrix}_{7\times 1} \tag{7.49}$$

Does it have a solution? If it does, is it unique?

Solution: We form $A^\top A$ and compute in Julia its LU Factorization (with permutations) and report the diagonal of U as a row vector

$$\mathrm{diag}(U) = \begin{bmatrix} 6.680 & 7.282 & 4.069 & 3.124 & 0.000 \end{bmatrix}_{5\times 1} \tag{7.50}$$

There is a single zero on the diagonal and four non-zero elements. Thus we know that only four of the five columns of A are linearly independent. Hence, if there does exist a solution, it will not be unique.

Next, we form $A_e := [A \;\; b]$ and compute in Julia the LU Factorization (with permutations) of $A_e^\top A_e$. We report the diagonal of U_e written as a row vector

$$\mathrm{diag}(U_e) = \begin{bmatrix} 7.810 & 7.643 & 3.751 & 3.682 & 0.000 & 0.000 \end{bmatrix}_{6\times 1} \tag{7.51}$$

Because there is more than one zero on the diagonal, we need to look at the rows of U_e having zero elements on their diagonal. We report all of U_e

$$U_e = \begin{bmatrix} 7.810 & 1.310 & 3.390 & 0.170 & -1.290 & 12.680 \\ 0.000 & 7.643 & -1.707 & -2.856 & 5.010 & 3.080 \\ 0.000 & 0.000 & 3.751 & -1.536 & -0.858 & 2.215 \\ 0.000 & 0.000 & 0.000 & 3.682 & 11.047 & 3.682 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix}_{6\times 6} \tag{7.52}$$

We see that it has two entire rows of zeros and hence U_e has $6 - 2 = 4$ linearly independent columns, the same as A. We deduce that b is a linear combination of the columns of A. Hence, the system of linear equations (7.49) has a solution and it is not unique. Therefore, it has an infinite number of solutions!

We select a different right hand side for (7.49) and report the key result of its LU Factorization analysis,

$$\tilde{b} = \begin{bmatrix} 0.6 \\ 0.4 \\ 0.9 \\ 1.0 \\ 0.8 \\ 0.8 \\ 0.9 \end{bmatrix} \implies \tilde{U}_e = \begin{bmatrix} 6.7 & 7.3 & 4.1 & 3.1 & 0.0 & 1.0 \end{bmatrix}_{1\times 6}. \tag{7.53}$$

This time $\operatorname{diag}(\tilde{U}_e)$ has a single zero and five non-zero entries, where as $\operatorname{diag}(U)$ had four non-zero entries. Hence, $\tilde{b}$ is not a linear combination of the columns of A, and the system of equations, with this new right hand side, does not have a solution. ■

7.8 (Optional Read): LU Factorization for Symmetric Matrices

Recall that a matrix M is symmetric if $M^\top = M$. For such matrices, we seek to refine our way of computing the LU Factorization so that at each stage of the reduction, $\text{Temp}_{k+1} = \text{Temp}_k - C_k \cdot R_k$, we preserve the symmetry of the matrix. We will discover a condition that will allow us to write

$$P \cdot M \cdot P^\top = L \cdot D \cdot L^\top, \tag{7.54}$$

where P is a permutation matrix, L is uni-lower triangular, and D is a diagonal matrix. When we can accomplish such a factorization, it will follow that the number of linearly independent columns of M is precisely equal to the number of non-zero elements on the diagonal of D. A useful application of this result would then be counting the number of linearly independent columns in a rectangular matrix A by counting the number of linearly independent columns of $A^\top A$, which is always symmetric.

To build up our intuition, consider a 3 × 3 symmetric matrix

$$\text{Temp}_1 := M = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}, \tag{7.55}$$

which has six parameters. If $a \neq 0$, our first step in constructing the LU Factorization is to define

$$\tilde{C}_1 = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \frac{1}{a} \text{ and } \tilde{R}_1 = \begin{bmatrix} a & b & c \end{bmatrix}, \tag{7.56}$$

where we are using a tilde over the extracted columns and rows to distinguish them from a slightly different definition we will give shortly. The symmetry in $\tilde{C}_1$ and $\tilde{R}_1$ is clearly visible and it comes from the symmetry in M.

We can take better advantage of the symmetry in M if we define instead

$$C_1 = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \frac{1}{a},\ D_{11} = a \text{ and } R_1 = C_1^\top. \tag{7.57}$$

It follows that

$$C_1 \cdot D_{11} \cdot C_1^\top = C_1 \cdot D_{11} \cdot R_1 = \tilde{C}_1 \cdot \tilde{R}_1. \tag{7.58}$$

A nice property of $C_1 \cdot D_{11} \cdot C_1^\top$ is that it is clearly symmetric. Also, if we subtract one symmetric matrix from another, the result is another symmetric matrix, so the symmetry property propagates. In our case, we'd have something like

$$\text{Temp}_2 := M - C_1 \cdot D_{11} \cdot C_1^\top = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix} - \begin{bmatrix} a & b & c \\ b & \frac{b^2}{a} & \frac{bc}{a} \\ c & \frac{bc}{a} & \frac{c^2}{a} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \alpha & \beta \\ 0 & \beta & \gamma \end{bmatrix}. \tag{7.59}$$

We let the reader fill in the values of α, β, and γ because all we care is that Temp_2 is symmetric and we can repeat the process.

When doing the LU Factorization, there were two other cases to consider. One is when there is an entire column of zeros, as in

$$\text{Temp}_1 := M = \begin{bmatrix} 0 & 0 & 0 \\ 0 & d & e \\ 0 & e & f \end{bmatrix}, \tag{7.60}$$

so that $a = b = c = 0$. Because of the symmetry in Temp_1, if we have an entire column of zeros, we also have an entire row of zeros! When we had entire column of zeros, we defined

$$\tilde{C}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ and } \tilde{R}_1 = \begin{bmatrix} a & b & c \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}, \tag{7.61}$$

We can take better advantage of the symmetry if we instead define

$$C_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \; D_{11} = 0 \text{ and } R_1 = C_1^\top. \tag{7.62}$$

It still follows that

$$C_1 \cdot D_{11} \cdot C_1^\top = C_1 \cdot D_{11} \cdot R_1 = \tilde{C}_1 \cdot \tilde{R}_1, \tag{7.63}$$

and $\text{Temp}_2 := M - C_1 \cdot D_{11} \cdot C_1^\top$ is once again symmetric, with the first column and first row identically zero, so the process continues. So far, no permutation matrices have shown up!

A permutation of rows is required when our matrix has a zero in the entry that is supposed to define the pivot, such as here,

$$\text{Temp}_1 := M = \begin{bmatrix} 0 & b & c \\ b & d & e \\ c & e & f \end{bmatrix}, \tag{7.64}$$

where $a = 0$. If $b \neq 0$, we would swap the first and second rows to arrive at

$$P \cdot \text{Temp}_1 := \begin{bmatrix} b & d & e \\ 0 & b & c \\ c & e & f \end{bmatrix}, \tag{7.65}$$

where to swap the first two rows, we have

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{7.66}$$

But, in (7.64), we have destroyed the symmetry of our matrix!

Destroyed symmetry!

A key observation is that we can **RESTORE SYMMETRY** if we also swap the first and second columns, like this,

$$\begin{bmatrix} d & b & e \\ b & 0 & c \\ e & c & f \end{bmatrix}, \tag{7.67}$$

but oops, we no longer have $b \neq 0$ at the pivot position. We've moved d, an element of the diagonal, to the pivot position. If we had instead swapped rows one and three, followed by a swap of columns one and three, we'd end up with f, another element of the diagonal, at the pivot position! Hence, if preserving symmetry is our goal, we need to look at the diagonal of $\mathrm{Temp_k}$ for a non-zero element, and then to a double swap, two rows and two columns, to move it to the pivot position and continue the algorithm. The swapping of the columns is achieved by multiplying $\mathrm{Temp_k}$ on the right (instead of the left, which we do for row swapping), and the required permutation matrix can be shown to be the transpose of the matrix used to do the row swapping! Go to Julia and play round with it. It'll be fun!

If we arrive at a step where $\mathbf{diag(Temp_k)}$ is all zeros, then the algorithm fails and we cannot factor M as in (7.54). A matrix that will lead to immediate failure is

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{7.68}$$

On real data, such failures will be rare, and when they do occur, one could build into the algorithm that it switches to a factorization of the form $P \cdot M \cdot Q = L \cdot U$. The permutation matrix Q is included so that operations done up to the point of failure do not have to be discarded.

We give the resulting algorithm and then revisit a few of our previous examples. So that a novice user can play around and check whether or not $Q = P^\top$ and $U = D \cdot L^\top$, we compute the additional matrices. You are free to remove them.

```julia
# LU Factorization for Symmetric Matrices
# P M P' = L D L', L unitriangular, D diagonal, P permutation matrix
#
function luJWGSymD(M::Array{<:Number, 2})
    epsilon=1e-12; K=100
    if isapprox(norm(M-M'),0, atol=epsilon)
        # M is indeed symmetric
        n, m = size(M)
        Areduced = deepcopy(M)
        L = Array{Float64,2}(undef, n, 0)
        # Could remove U for efficiency
        U = Array{Float64,2}(undef, 0, n)
        P=zeros(n,n) + I
        # Could remove Q for efficiency
        Q=zeros(n,n) + I
        # could make D a vector for efficiency
        D=zeros(n,n)
        for i = 1:n
            C = Areduced[:,i] # i-th column
            R = C' # i-th row
            if maximum(abs.(C)) <= K*epsilon #column of zeros
                C=0.0*C; C[i]=1.0; R = 0.0*R
                U=[U;R];
                L=[L C];
                D[i,i]=0.0
                Areduced=Areduced-C*R;
            else # move the biggest entry to the pivot position
                ii=argmax( abs.(diag(Areduced)) );
                nrow=ii[1]
                P[[i,nrow],:]=P[[nrow,i],:];
                Q[:,[i,nrow]]=Q[:,[nrow,i]];
                Areduced[[i,nrow],:]=Areduced[[nrow,i],:];
                Areduced[:,[i,nrow]]=Areduced[:,[nrow,i]];
                if i>1
                    L[[i,nrow],:] = L[[nrow,i],:];
                    U[:,[i,nrow]] = U[:,[nrow,i]];
                    d1=D[i,i];d2=D[nrow,nrow]
                    D[i,i]=d2;D[nrow,nrow]=d1
                end
                C = Areduced[:,i] # i-th column
                R = C' # i-th row
                pivot = C[i];
                if isapprox(pivot,0, atol=epsilon)
                    println("Diagonal is all zeros at step $i")
                    println("A symmetric factorization PMP' = L D L' is not possible.")
                    return -1.0
                    break
                else
                    D[i,i]=pivot
                    C=C/pivot #normalize all entires by C[i]
                    U=[U;R];  # could remove U for efficiency
                    L=[L C];
                    Areduced=Areduced-C*R;
                end
                Areduced[:,i]=0*Areduced[:,i]; Areduced[i,:]=0*Areduced[i,:];
          end
    end
        # U=D*L'; Q=P'; # All matrices included for pedagogical reasons
        return L, U, P, Q, D
    else
        println("Matrix is not symmetric")
        return 0.0
    end
end
```

Example 7.15 *We revisit Example 7.14 and the existence and uniqueness of solutions to $Ax = b$, where*

$$A = \begin{bmatrix} -0.2 & -0.2 & -0.4 & 0.3 & 0.3 \\ 0.3 & 1.0 & -0.1 & -1.1 & -1.7 \\ 0.7 & -1.9 & 1.5 & -0.0 & -3.0 \\ 0.9 & -1.0 & -0.7 & 0.6 & -1.8 \\ -0.5 & 0.8 & -1.1 & -0.5 & -0.5 \\ -2.0 & -0.9 & -0.5 & 0.2 & 0.3 \\ -1.0 & 0.6 & 0.7 & -0.9 & 0.2 \end{bmatrix}_{7\times 5} \quad \textit{and} \quad b = \begin{bmatrix} -0.5 \\ 0.1 \\ 0.3 \\ -0.2 \\ -1.3 \\ -3.2 \\ -0.6 \end{bmatrix}_{7\times 1} \tag{7.69}$$

Solution: We define $A_e = [A \;\; b]$ and compute in Julia the following Symmetric LU Factorizations

$$P \cdot (A^\top \cdot A) \cdot P^\top = L \cdot D \cdot L^\top$$
$$P_e \cdot (A_e^\top \cdot A_e) \cdot P_e^\top = L_e \cdot D_e \cdot L_e^\top.$$

We report the diagonals of D and D_e

$$\mathrm{diag}(D) = \begin{bmatrix} 15.6000 & 5.7333 & 5.1312 & 4.0419 & 0.0000 \end{bmatrix}$$
$$\mathrm{diag}(D_e) = \begin{bmatrix} 15.6000 & 12.573 & 5.4928 & 3.0609 & 0.0000 & 0.0000 \end{bmatrix}.$$

Because $\operatorname{diag}(D)$ and $\operatorname{diag}(D_e)$ have the same number of non-zero elements, we conclude that b is a linear combination of the columns of A and hence a solution exists. Because $\operatorname{diag}(D)$ has a zero element, the columns of A are not linearly independent, and hence the solution is not unique. The beauty of an LU Factorization adapted to symmetric matrices is that we do not need to look at anything other than the diagonal of D, because, by construction, D is a diagonal matrix! ■

7.9 (Optional Read): Still Another Take on LU Factorization for Checking Linear Independence

The objective in this section is to revisit how to check linear independence or dependence of a set of vectors by working directly with the matrix A instead of passing to $A^\top A$. To do this, we need to generalize several notions from square matrices to rectangular matrices. For some students, this will be interesting material and for others, not so much.

The **diagonal** of an $n \times m$ rectangular matrix A is the set of elements

$$\operatorname{diag}(A) := \begin{bmatrix} a_{11} & a_{22} & \cdots & a_{kk} \end{bmatrix}$$

where $k := \min\{n, m\}$, the smaller of the number of rows and the number of columns.

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}}_{k=m \text{ because } m<n} \quad \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2n} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & \cdots & a_{nm} \end{bmatrix}}_{k=n \text{ because } n<m}$$

A matrix is **upper triangular** if everything below the diagonal is zero. Below, we show the structure for **Tall Matrices**, which have more rows than columns, and **Wide Matrices**, which have more columns than rows.

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ 0 & a_{22} & a_{23} & \cdots & a_{2m} \\ 0 & 0 & a_{33} & \cdots & a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{mm} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}}_{\textbf{tall matrix because } \mathbf{n > m}} \quad \underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & \cdots & a_{1m} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} & \cdots & a_{2m} \\ 0 & 0 & a_{33} & \cdots & \cdots & \cdots & a_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} & \cdots & a_{nm} \end{bmatrix}}_{\textbf{wide matrix because } \mathbf{m > n}}$$

A matrix is **lower triangular** if everything above the diagonal is zero

$$\underbrace{\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}}_{\text{tall matrix because } n>m} \quad \underbrace{\begin{bmatrix} a_{11} & 0 & \cdots & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & \cdots & 0 \end{bmatrix}}_{\text{wide matrix because } m>n}$$

A triangular matrix is **unitriangular** if all of the diagonal elements are 1.0

As before, we assume that A is an $n \times m$ matrix and we define $k = \min\{n, m\}$. Our algorithm for the LU Factorization works perfectly well for non-square matrices. It will produce a (row) permutation matrix P of size $n \times n$, a lower unitriangular matrix L of size $n \times k$ and an upper triangular matrix U of size $k \times m$ such that

$$P \cdot A = L \cdot U. \tag{7.70}$$

Recalling that $P^\top P = I_n$, we have that

$$A = P^\top \cdot L \cdot U. \tag{7.71}$$

Example 7.16 *To make the discussion concrete, let's take the vectors from Example 7.7 and put them into a matrix*

$$v_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ -2 \\ 4 \\ 5 \end{bmatrix}, v_3 = \begin{bmatrix} 2 \\ 6 \\ 2 \\ -3 \end{bmatrix} \iff A = \begin{bmatrix} 1 & 0 & 2 \\ 2 & -2 & 6 \\ 3 & 4 & 2 \\ 1 & 5 & -3 \end{bmatrix}$$

and check whether they are linearly independent or dependent.

Solution We recall that to check for linear dependence, we seek a vector

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$

such that $A \cdot \alpha = 0$ and at least one component of α is non-zero. If the only solution is $\alpha = 0$, then the vectors are linearly independent.

We do the LU Factorization and obtain

$$P = P^\top = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}, \; L = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ \frac{2.0}{3} & 1.0 & 0.0 \\ \frac{1.0}{3} & \frac{2.0}{7} & 1.0 \\ \frac{1.0}{3} & -\frac{11.0}{14} & 0.0 \end{bmatrix} \; U = \begin{bmatrix} 3.0 & 4.0 & 2.0 \\ 0.0 & -\frac{14}{3} & \frac{14}{3} \\ 0.0 & 0.0 & \boxed{0.0} \end{bmatrix} \tag{7.72}$$

where the diagonal terms have been highlighted in L and U. We see that U has a zero on the diagonal. Moreover, using back substitution, we see that

$$\alpha = \begin{bmatrix} -2.0 \\ 1.0 \\ 1.0 \end{bmatrix},$$

satisfies $U \cdot \alpha = 0$, and hence,

$$A \cdot \alpha = \underbrace{\left[P^\top \cdot L \cdot U\right]}_{A} \cdot \alpha = \left[P^\top \cdot L\right] \cdot \underbrace{U \cdot \alpha}_{0.0} = 0.0$$

We conclude that the vectors are linearly dependent. ■

For the next example, we reuse the LU Factorization above, except we replace the zero on the diagonal of U with a non-zero number, say 1.0.

Example 7.17

$$P = P^\top = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}, \; L = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ \frac{2.0}{3} & 1.0 & 0.0 \\ \frac{1.0}{3} & \frac{2.0}{7} & 1.0 \\ \frac{1.0}{3} & -\frac{11.0}{14} & 0.0 \end{bmatrix} \; U = \begin{bmatrix} 3.0 & 4.0 & 2.0 \\ 0.0 & -\frac{14}{3} & \frac{14}{3} \\ 0.0 & 0.0 & \boxed{\mathit{1.0}} \end{bmatrix}. \tag{7.73}$$

It follows that

$$A = P^\top \cdot L \cdot U = \begin{bmatrix} 1 & 0 & \boxed{\mathit{3}} \\ 2 & -2 & 6 \\ 3 & 4 & 2 \\ 1 & 5 & -3 \end{bmatrix},$$

with only one number changing, and it is highlighted in red. Determine if the columns of A are linearly independent or dependent.

Solution This time, all of the elements on the diagonal of U are non-zero. When we seek a solution to $U \cdot \alpha = 0$, back substitution quickly tells us that the only solution is $\alpha = 0$. **Are we done? Can we conclude that the columns of A are linearly independent?** It turns out the answer is YES. For those who are interested, we show why this is true. On a first reading of these notes, you can skip to the next highlighted text.

Let's define $\beta := U \cdot \alpha$ and $\gamma := L \cdot \beta$, so that

$$\beta := \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \text{ and } \gamma := \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix};$$

note that the dimensions follow from L being 4×3, U being 3×3, and α being 3×1. With these definitions, we have

$$A \cdot \alpha = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \iff P^\top \cdot \gamma = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

But we know that $P^\top$ simply re-orders the rows of γ, and thus

$$P^\top \cdot \gamma = 0 \iff \gamma = 0.$$

We go back one more step then and we see that

$$\gamma = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \iff L \cdot \beta = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

But L is lower unitriangular and forward substitution quickly shows that

$$L \cdot \beta = 0 \iff \beta = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Hence, it suffices to check $U \cdot \alpha = 0$ to determine whether or not the columns of A are linearly independent or dependent. ■

Checking Linear Independence or Dependence

Suppose that we have done the LU Factorization of an $n \times m$ matrix A so that

$$P \cdot A = L \cdot U.$$

- For all values of $n \geq 1$ and $m \geq 1$, the columns of A are linearly independent if, and only if, the columns of U are linearly independent. Moreover, for $\alpha \in \mathbb{R}^n$,

$$A\alpha = 0 \iff U\alpha = 0.$$

- For the special cases of **tall matrices**, that is, when $n > m$ (strictly more rows than columns), and for square matrices, $n = m$, **we have a shortcut:** the columns of A are linearly independent if, and only if, all the entries on the diagonal of U are non-zero.

- Hence, for tall and square matrices, as soon as we have a zero on the diagonal of U, we can say that the columns of A are linearly dependent. We only do the additional work of solving $U \cdot \alpha = 0$ if we want to find the explicit coefficients $\alpha_1, \alpha_2, \ldots, \alpha_m$ such that

$$\alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_m u_m = 0.$$

  Many times we do not really need the coefficients. In HW and Quizzes, we'll tell you if you need to find the coefficients or whether a YES vs NO answer is acceptable.

- In the case of **wide matrices**, that is, when $m > n$ (strictly more columns than rows), **we have another shortcut:** the columns of A are ALWAYS linearly dependent.

7.10 Looking Ahead

In the next Chapter, we show how to measure the "length" of a vector and use that to formulate the problem of finding approximate solutions to linear systems of equations that do not have exact solutions. At first blush, this seems like a strange thing to do. In real engineering, however, the data we collect is never exact. The data is perturbed by various sources of errors, from imprecision in our instruments, to fact that experiments are hard to repeat! To get around this, we take many more measurements than we need, which gives us sets of what we call "overdetermined equations," which means more equations than unknown variables. We seek to "smooth" or "average" out the measurement errors by finding approximate solutions so that $e := Ax - b$, the error in the "equation," is small. We measure the "length of the error vector" by a function called the "Euclidean norm."

Chapter 8

Euclidean Norm, Least Squared Error Solutions to Linear Equations, and Linear Regression

Learning Objectives

- Learn one way to assign a notion of length to a vector
- The concept of finding approximate solutions to $Ax = b$ when an exact solution does not exist and why this is extremely useful in engineering.

Outcomes

- Euclidean norm and its properties
- If $Ax = b$ does not have a solution, then for any $x \in \mathbb{R}^n$, the vector $Ax - b$ is never zero. We will call $e := Ax - b$ the error vector and search for the value of x that minimizes the norm of the error vector.
- An application of this idea is Linear Regression, one of the "super powers" of Linear Algebra: fitting functions to data.

8.1 Norm or “Length” of a Vector

Definition: The norm of a vector in $\mathbb{R}^n$

Let $v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$ be a vector in $\mathbb{R}^n$. The **Euclidean norm** of v, denoted $||v||$, is defined as

$$||v|| := \sqrt{(v_1)^2 + (v_2)^2 + \cdots + (v_n)^2} \tag{8.1}$$

Example: The length of the vector $v = \begin{bmatrix} \sqrt{2} \\ -1 \\ 5 \end{bmatrix}$ is

$$||v|| := \sqrt{(\sqrt{2})^2 + (-1)^2 + (5)^2} = \sqrt{2+1+25} = \sqrt{28} = \sqrt{4 \cdot 7} = 2\sqrt{7} \approx 5.29.$$

Properties of the Norm of a vector

Let’s get used to term **norm of a vector**, which is the correct mathematical term for the length of a vector. There are actually many different ways of defining a notion of “length” to a vector. The particular norm defined in Definition 4.14 is the “Euclidean norm.” All norms satisfy the following properties

- For all vectors $v \in \mathbb{R}^n$, $||v|| \geq 0$ and moreover, $||v|| = 0 \iff v = 0$.
- For any real number $\alpha \in \mathbb{R}$ and vector $v \in \mathbb{R}^n$,
$$||\alpha v|| = |\alpha| \cdot ||v||.$$
- For any pair of vectors v and w in $\mathbb{R}^n$,
$$||v + w|| \leq ||v|| + ||w||.$$

The first property says that v has norm zero if, and only if, v is the zero vector. It seems pretty obvious for our notion of norm and it is!

For the second property, we note that we have to take the absolute value of the constant when we “factor it out” of the norm. This is because $\sqrt{a^2} = |a|$ and NOT a when $a < 0$. Of course, when $a \geq 0$, $\sqrt{a^2} = a$.

The third property is often called the **triangle inequality**. It says that the norm of a sum of vectors is upper bounded by the sum of the norms of the individual vectors. Another way to say this is, “the length of $v + w$ can never be strictly larger than the length of v plus the length of w.” What happens if you have $||v - w||$? Well,

$$\begin{aligned} ||v - w|| &= ||v + (-w)|| \\ &\leq ||v|| + || - w|| \\ &= ||v|| + | - 1| \cdot ||w|| \\ &= ||v|| + ||w||. \end{aligned}$$

Hence, a minus sign changes nothing!

Example We'll take three vectors in $\mathbb{R}^4$ and check the "triangle inequality."

$$u = \begin{bmatrix} 2 \\ -1 \\ 5 \\ 2 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 3 \end{bmatrix}, w = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 4 \end{bmatrix}$$

We first compute the norms of the three vectors

$$||u|| = \sqrt{34} \approx 5.83, ||v|| = \sqrt{14} \approx 3.74, ||w|| = \sqrt{17} \approx 4.12$$

and we then form a few sums against which to test the triangle inequality

$$u + v = \begin{bmatrix} 3 \\ 1 \\ 5 \\ 5 \end{bmatrix}, u + v + w = \begin{bmatrix} 3 \\ 2 \\ 5 \\ 9 \end{bmatrix}, v + w = \begin{bmatrix} 1 \\ 3 \\ 0 \\ 7 \end{bmatrix}.$$

Then we can check that

$$\begin{aligned} ||u + v|| &= \sqrt{60} \approx 7.75 \leq 5.8 + 3.7 \leq ||u|| + ||v|| \\ ||u + v + w|| &= \sqrt{119} \approx 10.91 \leq 7.7 + 4.1 \leq ||u + v|| + ||w|| \\ ||u + v + w|| &= \sqrt{119} \approx 10.91 \leq 5.8 + 3.7 + 4.1 \leq ||u|| + ||v|| + ||w|| \end{aligned}$$

8.2 Least Squared Error Solutions to Linear Equations

In this section, we tackle the problem of providing a notion of "best approximate solution" to a system of linear equations that does not have an exact solution. If the system of equations does have an exact solution, our approximate solution will be identical to it. Hence, our development can be viewed as an alternative means of defining solutions to linear equations.

Consider a system of linear equations $Ax = b$ and define the vector

$$e(x) := Ax - b \tag{8.2}$$

as the **error in the solution** for a given value of x. Normally, we'll simply write $e := Ax - b$, but in (8.2), we are emphasizing that the error really is a function of x. Hence, as we vary x, the "length" of $e(x)$ can become bigger or smaller.

If the system of equations $Ax = b$ has a solution, then it is possible to make the error zero. In the previous section, we introduced the Euclidean norm as a means of measuring the "length" of a vector. It is traditional when posing the best approximation problem to use the square of the norm instead of the norm itself, which means we are simply removing the square root operation in the formula for a norm.

Least Squared Error Solution: Consider a linear system of equations expressed in matrix form $Ax - b$, where A is $n \times m$, x is $m \times 1$ and b is $n \times 1$. For a given value of $x \in \mathbb{R}^m$, define the error as in (8.2), an $n \times 1$ vector. The norm squared of the error vector $e(x)$ is then

$$||e(x)||^2 := \sum_{i=1}^{n} (e_i(x))^2 = e(x)^\top e(x) = (Ax - b)^\top (Ax - b) = ||Ax - b||^2. \tag{8.3}$$

We note that $||e(x)||^2 \geq 0$ for any $x \in \mathbb{R}^m$ and hence zero is a lower bound on the norm squared of the error vector. A value $x^* \in \mathbb{R}^m$ is a **Least Squared Error Solution** to $Ax = b$ if it satisfies

$$||e(x^*)||^2 = \min_{x \in \mathbb{R}^m} ||Ax - b||^2 \tag{8.4}$$

If such an $x^* \in \mathbb{R}^m$ exists and is unique, we will write it as

$$x^* := \underset{x \in \mathbb{R}^m}{\arg\min} \, ||Ax - b||^2. \tag{8.5}$$

With this notation, the value of x that minimizes the error in the solution is what is returned by the function $\arg\min$, while the minimum value of the error is what is returned by the function $\min$,

- $x^* = \arg\min_{x \in \mathbb{R}^m} ||Ax - b||^2$ is the value of x the achieves the minimum value of the squared norm of the error, $||Ax - b||^2$, while
- $||e(x^*)||^2 = ||Ax^* - b||^2 = \min_{x \in \mathbb{R}^m} ||Ax - b||^2$ is the minimum value of the "squared approximation error".

If $Ax = b$ has a solution, then

$$\min_{x \in \mathbb{R}^m} ||Ax - b||^2 = 0,$$

because the norm squared error cannot be negative; indeed, zero is the smallest possible value. Hence, while we are most interested in cases where $Ax = b$ does not have a solution, if it does have a unique solution $\bar{x}$ such that $A\bar{x} = b$, then

$$\bar{x} = x^* := \underset{x \in \mathbb{R}^m}{\arg\min} \, ||Ax - b||^2.$$

We recall that uniqueness of solutions to $Ax = b$ is tied to the columns of A being linearly independent. Let's also observe that if $Ax = b$, then

- multiplying both sides of $Ax = b$ by $A^\top$ gives

$$A^\top \cdot Ax = A^\top \cdot b.$$

- The indicated matrix multiplications are well defined because $A^\top$ is $m \times n$, A is $n \times m$, and b is $n \times 1$.
- $A^\top \cdot A$ is therefore square and $m \times m$.
- $A^\top \cdot A$ is symmetric because

$$\left(A^\top \cdot A\right)^\top = A^\top \cdot \left(A^\top\right)^\top = A^\top \cdot A$$

because $\left(A^\top\right)^\top = A$.

Least Squares Solutions to Linear Equations

Here are the main results on solutions to $Ax = b$ that minimize the squared error $||Ax - b||^2$.

(a) $A^\top A$ is invertible if, and only if, the columns of A are linearly independent.

(b) If $A^\top A$ is invertible, then there is a unique vector $x^* \in \mathbb{R}^m$ achieving $\min_{x \in \mathbb{R}^m} ||Ax - b||^2$ and it satisfies the equation

$$\left(A^\top A\right) x^* = A^\top b. \tag{8.6}$$

(c) Therefore, if $A^\top A$ is invertible,

$$x^* = (A^\top A)^{-1} A^\top b \iff x^* = \underset{x \in \mathbb{R}^m}{\arg\min} ||Ax - b||^2 \iff \left(A^\top A\right) x^* = A^\top b. \tag{8.7}$$

As you might guess by now, your instructors prefer that for large systems of equations, you solve (8.6) to obtain the least squares solution and avoid doing the inverse. For small systems, we'll cut you some slack.

Useful Remark: Suppose that A is a "tall matrix" (more rows than columns) and suppose that $Ax = b$ has a solution (hence, b is a linear combination of the columns of A). Then, if the columns of A are linearly independent, you can compute the solution using (8.6) and the squared error will be zero, meaning x^* really is a solution to the equation because

$$\text{the squared error is zero} \iff ||Ax^* - b||^2 = 0 \iff ||Ax^* - b|| = 0 \iff Ax^* - b = 0 \iff Ax^* = b.$$

Try it on your own in Julia!

Example 8.1 *Let's consider a system of linear equations, with more equations than unknowns. The extra equations provide more conditions that a solution must satisfy, making non-existence of a solution a common occurrence! We take*

$$\underbrace{\begin{bmatrix} 1.0 & 1.0 \\ 2.0 & 1.0 \\ 4.0 & 1.0 \\ 5.0 & 1.0 \\ 7.0 & 1.0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 4 \\ 8 \\ 10 \\ 12 \\ 18 \end{bmatrix}}_{b}, \tag{8.8}$$

and note that the columns of A are linearly independent. If a regular solution exists, find it. If not, then a least squared solution will be fine.

Solution: We'll compute a least squared error solution to the equations, and then we'll evaluate the error; if the error is zero, we'll also have an exact solution to (8.8). We compute

$$A^\top \cdot A = \begin{bmatrix} 95.0 & 19.0 \\ 19.0 & 5.0 \end{bmatrix}, \; A^\top \cdot b = \begin{bmatrix} 246.0 \\ 52.0 \end{bmatrix} \implies x^* = \begin{bmatrix} 2.12 \\ 2.33 \end{bmatrix}$$

For the record, $\det(A^\top \cdot A) = 114$. We also compute

$$e^* := Ax^* - b = \begin{bmatrix} 0.456 \\ -1.421 \\ 0.825 \\ 0.947 \\ -0.807 \end{bmatrix}, \; ||e|| = 2.111, \text{ and } ||e||^2 = 4.456$$

Because the "smallest" error vector $e^* = Ax^* - b$ is non-zero, we now know that (8.8) does not have an exact solution. ■

The set of linear equations (8.8) and its solution look rather ho hum. They are actually anything but boring. Figure 8.1 shows that the equations and their solution correspond to fitting a line through data, while minimizing the "fitting error"! In the next section, we will develop this idea thoroughly and give you a hint of some of the things you can do with least squared error solutions to linear equations.

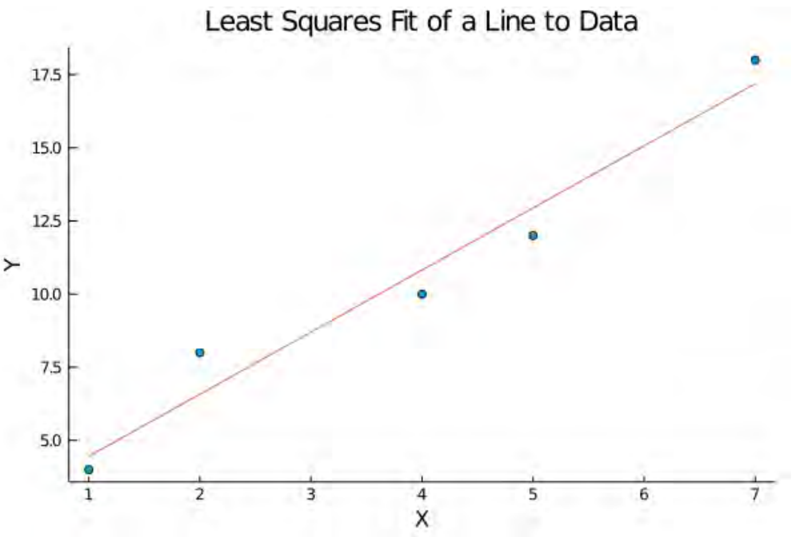


Figure 8.1: The physical basis for the numbers in (8.8), which is really about fitting a line to data with minimum squared error! The dots are the (x, y) values of the raw (measured) data, while the line is a model that summarizes the data in the form $y = mx + b$. One value of the model is that you can now predict values of y for values of x that do not correspond to direct measurements.

8.3 Linear Regression or Fitting Functions to Data

The goal of this section is explain how to fit functions to data. The following is a prototypical problem: given the data shown in Table 8.1, which is also plotted in Fig. 8.2, find a function that explains the data.

It is clear that the data do NOT lie exactly on a straight line. How can we **approximately** fit a straight line to the data? In particular, how can we find a function that minimizes a meaningful sense of fitting error to the data?

Table 8.1: Data for our first fitting problem.

i	x_i	y_i
1	1	4
2	2	8
3	4	10
4	5	12
5	7	18

Let's suppose that we wish to fit a linear model $\hat{y} = mx + b$ to the data, which has been plotted in Fig. 8.2. We set up the linear equations

$$y_i = mx_i + b = \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix}, \quad 1 \le i \le N,$$

where N is the number of data points (five in the case of Table 8.1), and write it out in matrix form

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & 1 \\ x_N & 1 \end{bmatrix}}_{\Phi} \cdot \underbrace{\begin{bmatrix} m \\ b \end{bmatrix}}_{\alpha}, \tag{8.9}$$

where Y is the vector of y-data, Φ is called the **regressor matrix** and α is the vector of **unknown coefficients** that parameterize the model.

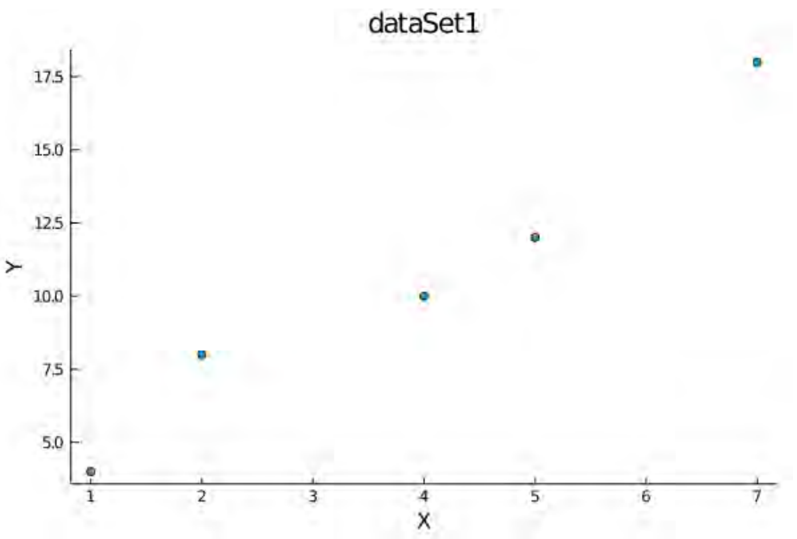


Figure 8.2: Plot of data in Table 8.1

From the data in Table 8.1, the matrices are

$$Y = \begin{bmatrix} 4 \\ 8 \\ 10 \\ 12 \\ 18 \end{bmatrix}, \ \Phi = \begin{bmatrix} 1.0 & 1.0 \\ 2.0 & 1.0 \\ 4.0 & 1.0 \\ 5.0 & 1.0 \\ 7.0 & 1.0 \end{bmatrix}, \text{ and } \alpha = \begin{bmatrix} m \\ b \end{bmatrix}.$$

Y is the vector of "measured" y-values. α is the vector of unknown coefficients that we seek to estimate. Φ, the regressor matrix, is defined so that the i-th row of $Y = \Phi\alpha$ corresponds to $y_i = mx_i + b$.

The fitting error will be $e_i = y_i - (mx_i + b)$, which when written as a vector gives

$$\underbrace{\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{bmatrix}}_{e} = \underbrace{\begin{bmatrix} 4 \\ 8 \\ 10 \\ 12 \\ 18 \end{bmatrix}}_{Y} - \underbrace{\begin{bmatrix} 1.0 & 1.0 \\ 2.0 & 1.0 \\ 4.0 & 1.0 \\ 5.0 & 1.0 \\ 7.0 & 1.0 \end{bmatrix}}_{\Phi} \cdot \underbrace{\begin{bmatrix} m \\ b \end{bmatrix}}_{\alpha},$$

that is, $e := Y - \Phi\alpha$. We propose to chose the coefficients in α so as to minimize the total squared error

$$E_{tot} = \sum_{i=1}^{5} (e_i)^2 = e^\top e = ||e||^2 = ||Y - \Phi\alpha||^2.$$

Least Squares Fit to Data also called Linear Regression

From Chapter 8.2, if the columns of Φ are linearly independent, or equivalently, $\Phi^\top \Phi$ is invertible, then the following are equivalent

$$\alpha^* = \left(\Phi^\top \Phi\right)^{-1} \Phi^\top Y \iff \alpha^* = \underset{\alpha}{\arg\min}\, ||Y - \Phi\alpha||^2 \iff \left(\Phi^\top \Phi\right) \alpha^* = \Phi^\top Y. \tag{8.10}$$

The associated equations are formulated and solved in the Julia code given below. The plot of the fit is given in Fig. 8.1. You can generate your own plots too!

```
using Plots
gr()

```

```
# Given data
X=[1  2  4  5  7]'
Y=[4  8  10  12  18]'

# Scatter plot
scatter(X,Y)
plot!(xlabel="X", ylabel="Y", title="dataSet1", leg=false)
plot!(fmt = :png)

# Build the regressor matrix
Phi=[X ones(5,1)]
@show Phi

Phi = [1.0 1.0; 2.0 1.0; 4.0 1.0; 5.0 1.0; 7.0 1.0]
5ÃŮ2 Array{Float64,2}:
 1.0  1.0
 2.0  1.0
 4.0  1.0
 5.0  1.0
 7.0  1.0

 # Take a shortcut to finding alpha
 # because the problem is small
 alphaStar=inv(Phi'*Phi)*Phi'*Y
 2ÃŮ1 Array{Float64,2}:
 2.1228070175438596
 2.3333333333333375

# Extract the physically meaningful parameters
m=alphaStar[1]
b=alphaStar[2]

# Build the line for plotting
XX=[1, 7]
YY=m*XX.+b
scatter(X,Y)
plot!(xlabel="X", ylabel="Y", title="Least Squares Fit of a Line to Data", leg=false)
plot!(fmt = :png)

# Plot the line over the scatter plot of the data
plot!(XX,YY)
```

The above shows you how to formulate a least squares fit to data by working through THE CLASSIC EXAMPLE, fitting a line to data by minimizing the total squared error (square of the norm of the error vector). We'll do another example so that you understand that you are not limited to fitting "linear functions" to data.

Table 8.2: Data for our second fitting problem.

i	x_i	y_i
1	0	1.0
2	0.25	1.0
3	0.5	1.5
4	0.75	2.0
5	1.0	3.0
6	1.25	4.25
7	1.5	5.5
8	1.75	7.0
9	2.0	10.0

Example 8.2 *Our method of fitting a line to data is more general than it might seem. Consider the data in Table 8.2, which has been plotted in Fig. 8.3. It sure doesn't seem like we should fit a line to it. How about a quadratic?*

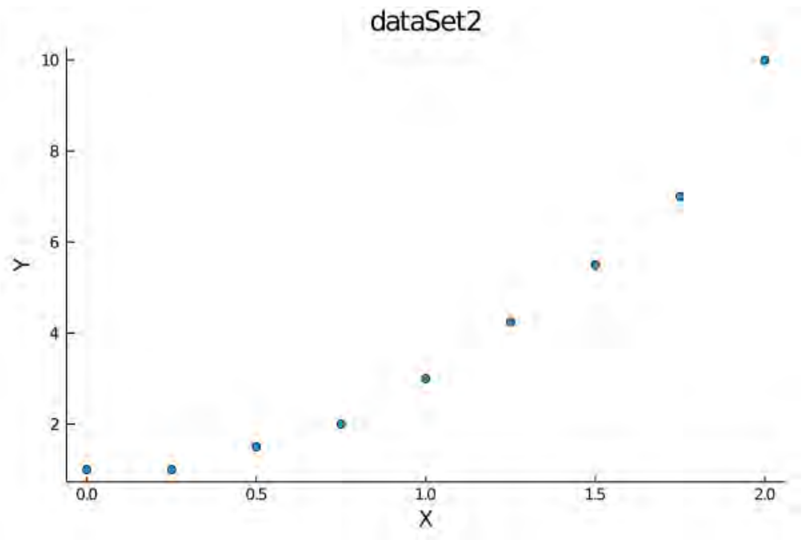


Figure 8.3: Scatter plot of the data in Table 8.2. The curve looks nonlinear!

Solution: Let's choose a model of the form

$$y = c_0 + c_1 x + c_2 x^2 = \begin{bmatrix} 1 & x & x^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}.$$

Note that even though the model is nonlinear in x, it is linear in the unknown coefficients $c_0,\ c_1,\ c_2$. This is what is important!!! Just as before, define $\hat{y}_i = c_0 + c_1 x_i + c_2 x_i^2$, the i-th term of the error vector is then

$$e_i := y_i - \hat{y}_i = y_i - (c_0 + c_1 x_i + c_2 x_i^2)$$

and the total squared error is

$$E_{tot} = \sum_{i=1}^{N} e_i^2.$$

Writing out the equation $y_i = c_0 + c_1 x_i + c_2 x_i^2$, $i = 1, \cdots, N$ in matrix form yields

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} 1 & x_1 & (x_1)^2 \\ 1 & x_2 & (x_2)^2 \\ \vdots & \vdots & \\ 1 & x_N & (x_N)^2 \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}}_{\alpha},$$

which gives us the equation $Y = \Phi\alpha$. We plug in our numbers and check that $\det(\Phi^\top \cdot \Phi) = 40.6 \neq 0$. The resulting fit is given in Fig. 8.4. The Julia code is also given.

```
# Data set
dataSet2=[
1  0.0    1.0
2  0.25   1.0
3  0.5    1.5
4  0.75   2.0
5  1.0    3.0
6  1.25   4.25
7  1.5    5.5
8  1.75   7.0
9  2.0   10.0]

# Extract relevant data
X=dataSet2[:,2]
Y=dataSet2[:,3]
```

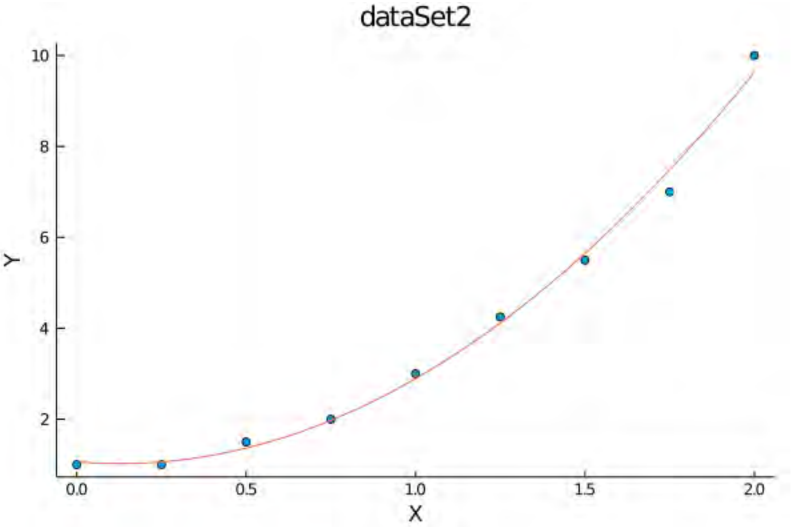


Figure 8.4: A least squares fit of a quadratic curve, $\hat{y} = c_0 + c_1 x + c_2 x^2$, to data.

```
# Look at data to see what kind of curve it may be
using Plots
gr()
scatter(X,Y)
plot!(xlabel="X", ylabel="Y", title="dataSet2", leg=false)
plot!(fmt = :png)

# Build the regressor matrix
Phi=[ones(9,1) X  X.^2]
9 x 3 Array{Float64,2}:
 1.0  0.0   0.0
 1.0  0.25  0.0625
 1.0  0.5   0.25
 1.0  0.75  0.5625
 1.0  1.0   1.0
 1.0  1.25  1.5625
 1.0  1.5   2.25
 1.0  1.75  3.0625
 1.0  2.0   4.0

# Solve Regression Problem
alphaStar=inv(Phi’*Phi)*Phi’*Y

# Plot the curve: first way to do it
Yhat=Phi*alphaStar
plot!(X,Yhat)

# Plot the curve with more points in x so that it is smoother
temp=1:200
XX=vec(temp/100.0);
N=length(XX)
PHI=[ones(N,1) XX XX.^2]
#YY=c0.+ c1.*XX + c2.*(XX).^2;
YY=PHI*alphaStar

# Plot used in the notes
scatter(X,Y)
plot!(xlabel="X", ylabel="Y", title="dataSet2", leg=false)
plot!(fmt = :png)
plot!(XX,YY)
```

In HW and in Project 2, we’ll explore more refined aspects of regressing functions and surfaces to data. We’ll actually divide the data set into two pieces: one to be used for fitting the data and a second portion of the data reserved for checking the quality of the fit. We will be looking to see that the fit to data thta was not used in the regression problem is comparable to the fit produced by the regression algorithm. The idea is that the second piece of data better represents how your regressed function (or surface) will work in the real world. **If you have heard of Machine Learning, these ideas are very close to current practice when "training" Supervised Machine Learning Algorithms.**

■

8.4 (Optional Read): How to Derive the Main Regression Formula

The main method is "completing the square". You probably learned that in High School when you studied the quadratic formula. We recall first the derivation of the quadratic formula via "completing the square" and then give the main steps for a "vector-matrix version of completing the square."

8.4.1 Completing the Square for the Quadratic Formula:

We assume that $a \neq 0$ and that all coefficients are real numbers. The main "trick" it so recall that $(x+d)^2 = x^2 + 2dx + d^2$. Hence, if you see something like $x^2 + 2dx$, you can complete the square by adding and subtracting d^2 to obtain $x^2 + 2dx = (x+d)^2 - d^2$. Below, we do this for $d = \frac{b}{a}$.

$$\begin{aligned}
ax^2 + bx + c &= 0 \\
&\Updownarrow \\
x^2 + \frac{b}{a}x + \frac{c}{a} &= 0 \\
&\Updownarrow \\
\left(x + \frac{b}{2a}\right)^2 - \left(\frac{b}{2a}\right)^2 + \frac{c}{a} &= \quad \text{(the square was completed here!)} \\
&\Updownarrow \\
\left(x + \frac{b}{2a}\right)^2 &= \left(\frac{b}{2a}\right)^2 - \frac{c}{a} \\
&\Updownarrow \\
\left(x + \frac{b}{2a}\right)^2 &= \frac{b^2 - 4ac}{4a^2} \quad \text{(a perfect square)} \\
&\Updownarrow \\
\left(x + \frac{b}{2a}\right) &= \pm\sqrt{\frac{b^2 - 4ac}{4a^2}} \quad \text{(note the plus-minus sign)} \\
&\Updownarrow \\
\left(x + \frac{b}{2a}\right) &= \pm\frac{\sqrt{b^2 - 4ac}}{2a} \quad \text{(the rest is ``algebra'')} \\
&\Updownarrow \\
x &= -\frac{b}{2a} \pm \frac{\sqrt{b^2 - 4ac}}{2a} \\
&\Updownarrow \\
x &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\end{aligned}$$

That's a lot of steps. You probably forgot how it went, right? We had to refresh our own thoughts on completing the square.

8.4.2 Completing the Square for Least Squares Solutions to Systems of Linear Equations:

Consider the set of equations $Ax = b$ and suppose that the columns of A are linearly independent, or equivalently, that $A^\top A$ is invertible (i.e., $\det(A^\top A) \neq 0$). Then the value of x that minimizes the error satisfies

$$A^\top A x^* = A^\top b.$$

Sketch:

$$\begin{aligned} ||Ax-b||^2 &= (Ax-b)^\top (Ax-b) \\ &= x^\top A^\top A x - x^\top A^\top B - b^\top A x + b^\top b \end{aligned} \tag{8.11}$$

This is where completing the square comes in. It is much easier to do if you already know that answer from other techniques! In Linear Algebra, an expression of the form

$$\boxed{\left(A^\top A x - A^\top b\right)^\top \left(A^\top A\right)^{-1} \left(A^\top A x - A^\top b\right)}$$

is the equivalent of a perfect square. We expand all the terms

$$\begin{aligned} \left(A^\top A x - A^\top b\right)^\top \left(A^\top A\right)^{-1} \left(A^\top A x - A^\top b\right) &= \left(x^\top A^\top A - b^\top A\right) \left(A^\top A\right)^{-1} \left(A^\top A x - A^\top b\right) \\ &= \left(x^\top - b^\top A \left(A^\top A\right)^{-1}\right) \left(A^\top A x - A^\top b\right) \\ &= x^\top A^\top A x - x^\top A^\top b - b^\top A x + b^\top A \left(A^\top A\right)^{-1} A^\top b \end{aligned}$$

and then relate them to the equation we are trying to minimize.

Substituting this result into (8.11) gives

$$\begin{aligned} ||Ax-b||^2 &= (Ax-b)^\top (Ax-b) \\ &= x^\top A^\top A x - x^\top A^\top b - b^\top A x + b^\top b \\ &= x^\top A^\top A x - x^\top A^\top b - b^\top A x + b^\top b + b^\top A \left(A^\top A\right)^{-1} A^\top b - b^\top A \left(A^\top A\right)^{-1} A^\top b \\ &= \left(A^\top A x - A^\top b\right)^\top \left(A^\top A\right)^{-1} \left(A^\top A x - A^\top b\right) + b^\top b - b^\top A \left(A^\top A\right)^{-1} A^\top b \end{aligned}$$

Here is the *coup de grâce*: We note that $b^\top b - b^\top A \left(A^\top A\right)^{-1} A^\top b$ does not depend on x. Hence, the x^* that minimizes (8.11) is that same as the x^* that minimizes

$$\left(A^\top A x - A^\top b\right)^\top \left(A^\top A\right)^{-1} \left(A^\top A x - A^\top b\right).$$

Therefore, the solution is

$$\left(A^\top A x^* - A^\top b\right) = 0 \iff A^\top A x^* = A^\top b.$$

Remark: Uniqueness follows because $A^\top A$ is invertible. A more complete derivation would use properties of positive definite matrices that we have not yet introduced.

8.5 Looking Ahead

The next Chapter will conclude our introduction to Computational Linear Algebra. We want to leave space for covering some non-linear topics in Chapters 10 and 11, topics that are very computational in nature and super useful in engineering practice.

Our final Chapter Linear Algebra Chapter will build on our notions of linear combinations and liner independence to introduce the notion of basis vectors, which is a minimal set of vectors that can be used to generate all other vectors in a vector space by linear combinations. They should have been called generating vectors! The material is a bit technical for a first course, but somehow essential. After wading through the technical stuff, we'll be on the proverbial Easy Street. We will introduce a tool, called a dot product, that allows us to study the notion of "orthogonal vectors" (also called perpendicular vectors) in $\mathbb{R}^n$ for any $n \geq 2$. You probably recall "right angles" from planar geometry? Don't worry, we will not be doing "similar triangles" or the "side-angle-side theorem". What we will be doing is more on the level of difficulty of the Pythagorean Theorem, $a^2 + b^2 = c^2$. We'll introduce an algorithm that you will want to program up in Julia for constructing orthogonal vectors from a set of linearly independent vectors. And from such vectors, we build matrices that have the amazing property that their inverse is equal to their transpose! We know, it seems too good to be true. And we'll learn how to write any matrix A as the product of one of these magical matrices and an upper triangular matrix.

Chapter 9

The Vector Space R^{n}: Part 2

Learning Objectives

- A second encounter with some of the essential concepts in Linear Algebra.
- A more abstract view of $\mathbb{R}^n$ as a vector spcae.

Outcomes

- What is a vector space, a subspace, and the span of a set of vectors.
- Range, column span, and null space of a matrix.
- A basis for a subspace is a set of linear independent vectors, from which , using linear combinations, you can generate all vectors in the subspace.
- Handy matrix properties dealing with rank and nullity.
- The dot product and orthogonality.
- Gram Schmidt process for generating a basis consisting of orthogonal vectors
- Orthogonal matrices: they have the magical property that their inverse is the matrix transpose.
- The QR Factorization is the most numerically robust method for solving systems of linear equations.
- Our recommended "pipeline" (CS-speak for method) for solving $Ax = b$.

9.1 $\mathbb{R}^n$ as a Vector Space

Let's recall that an n-tuple is a fancy name for an ordered list of n numbers, $(x_1, x_2, \dots, x_n)$ and that we typically identify them with column vectors, as in

$$(x_1, x_2, \dots, x_n) \longleftrightarrow \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Moreover, we identify $\mathbb{R}^n$ with the set of all n-column vectors with real entries

$$\{(x_1, x_2, \dots, x_n) \mid x_i \in R, 1 \le i \le n\} \iff \mathbb{R}^n \iff \left\{ \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \middle| \, x_i \in R, 1 \le i \le n \right\}$$

Finally, the choice of identifying n-tuples of numbers with column vectors instead of row vectors is completely arbitrary, and yet, we have to choose one or the other when doing computations, and the most common choice is to use column vectors.

Two absolutely key properties of vectors in $\mathbb{R}^n$ is that we know how to add them and obtain another vector in $\mathbb{R}^n$, namely

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} := \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}, \tag{9.1}$$

and we know how to multiply a scalar times a vector and obtain another vector in $\mathbb{R}^n$, namely

$$\alpha \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} := \begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{bmatrix}. \tag{9.2}$$

Equation (9.1) says that the sum of two vectors is DEFINED by the sum of their respective components using the definition of addition of real numbers. Equation (9.2) says that the product of a scalar and a vector is DEFINED by the multiplication of the real numbers constituting the components of the vector by the scalar, which is another real number. It is emphasized that the vector operations are defined in terms the elementary operations of addition and multiplication of real numbers.

Now, (9.1) **and** (9.2) **are special cases of linear combinations. In fact, the following statements are equivalent (means, one holds if, and only if, the other holds)**

(a) For all real numbers α and β, and all vectors x and y in $\mathbb{R}^n$

$$\alpha \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \alpha x_1 + \beta y_1 \\ \alpha x_2 + \beta y_2 \\ \vdots \\ \alpha x_n + \beta y_n \end{bmatrix} \tag{9.3}$$

(b) Both (9.1) and (9.2) hold individually.

Remark: To see the equivalency, note that if in (9.3), you take $\alpha = \beta = 1$, you obtain (9.1), while if you take $\beta = 0$, you obtain (9.2). To go the other way around, we observe that, by (9.2),

$$\alpha \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{bmatrix} + \begin{bmatrix} \beta y_1 \\ \beta y_2 \\ \vdots \\ \beta y_n \end{bmatrix},$$

and that by (9.1),

$$\begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{bmatrix} + \begin{bmatrix} \beta y_1 \\ \beta y_2 \\ \vdots \\ \beta y_n \end{bmatrix} = \begin{bmatrix} \alpha x_1 + \beta y_1 \\ \alpha x_2 + \beta y_2 \\ \vdots \\ \alpha x_n + \beta y_n \end{bmatrix}.$$

Therefore, (9.1) and (9.2) together imply (9.3).

9.2 Subspaces of $\mathbb{R}^n$

Recall that a set V is a **subset** of some other set, say W, if $x \in V \implies x \in W$. One writes $V \subset W$ to denote V is a subset of W. We allow $V = W$ and hence any set is a subset of itself.

Subspace of $\mathbb{R}^n$

Suppose that $V \subset \mathbb{R}^n$ is nonempty, that is, V is a subset of $\mathbb{R}^n$ and it contains at least one element.

Def. V is a **subspace** of $\mathbb{R}^n$ if any linear combination constructed from elements of V and scalars in $\mathbb{R}$ is once again an element of V. One says that V is **closed under linear combinations.** In symbols, $V \subset \mathbb{R}^n$ is a subspace of $\mathbb{R}^n$ if for all real numbers α and β, and all vectors v_1 and v_2 in V

$$\boxed{\alpha v_1 + \beta v_2 \in V.} \tag{9.4}$$

From the equivalence of (9.3) with the separate conditions given in (9.1) and (9.2), one is free to check that a subset is a subspace by checking individually that it is **closed under vector addition** and **closed under scalar times vector multiplication**.

Being “closed under something” simply means that if you perform the operation “something” on an element of a set, you get a new element that is once again an element of the set. For us, “something” is the operation of "forming linear combinations", “doing vector addition”, or “doing scalar times vector multiplication”. If you do one of these operations and you end up with something new that is NOT in the subset V, then V is NOT a subspace.

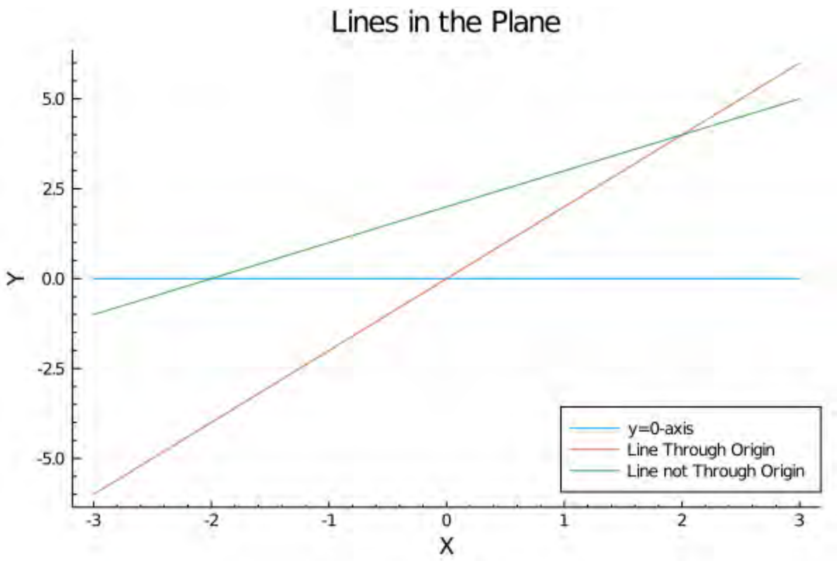


Figure 9.1: If a line does not pass through the origin, then it does not contain the origin, and hence it cannot be a subspace.

Easy first test for subspaces: Every subspace must contain the zero vector. Why? Suppose that $V \subset \mathbb{R}^n$ is a subspace and that $v \in V$. Then $0 \cdot v \in V$ because V is closed under scalar times vector multiplication. But $0 \cdot v = 0$, the zero vector. Figure 9.1 drives home the point that, even in $\mathbb{R}^2$, not all lines are subspaces. In fact, almost no lines are subspaces! It's a very special case when the y-intercept equals zero.

Example 9.1 *Let $V \subset \mathbb{R}^2$ be the set of all points that lie on a line $y = mx + b$, that is*

$$V := \left\{ \begin{bmatrix} x \\ mx + b \end{bmatrix} \;\middle|\; x \in \mathbb{R} \right\}.$$

Then V is a subspace of $\mathbb{R}^2$ if, and only if, $b = 0$, that is, the line must pass through the origin.

Solution: V contains the zero vector if, and only, if the y-intercept is zero. But this means that $b = 0$. Now, $0 \in V$ is a *necessary condition*, but not a *sufficient condition* for V to be a subspace. So, let's check if V, with $b = 0$ is closed under vector addition and scalar times vector multiplication. V is then

$$V := \left\{ \begin{bmatrix} x \\ mx \end{bmatrix} \;\middle|\; x \in \mathbb{R} \right\}.$$

We take

$$v_1 = \begin{bmatrix} x_1 \\ mx_1 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} x_2 \\ mx_2 \end{bmatrix}$$

for x_1 and x_2 arbitrary real numbers. Then

$$v_1 + v_2 = \begin{bmatrix} x_1 + x_2 \\ mx_1 + mx_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ m(x_1 + x_2) \end{bmatrix} \in V,$$

and hence V is closed under vector addition. To be extra clear, we note that

$$v_1 + v_2 = \begin{bmatrix} x \\ mx \end{bmatrix}, \text{ for } x = x_1 + x_2,$$

and that is why $v_1 + v_2 \in V$.

We now let $\alpha \in \mathbb{R}$ be arbitrary and check scalar times vector multiplication.

$$\alpha v_1 = \alpha \begin{bmatrix} x_1 \\ mx_1 \end{bmatrix} = \begin{bmatrix} \alpha x_1 \\ \alpha m x_1 \end{bmatrix} = \begin{bmatrix} \alpha x_1 \\ m(\alpha x_1) \end{bmatrix} \in V,$$

and hence V is closed under scalar times vector multiplication. To be extra clear, we note that

$$\alpha v_1 = \begin{bmatrix} x \\ mx \end{bmatrix}, \text{ for } x = \alpha x_1,$$

and that is why $\alpha v_1 \in V$.

Suppose that $b \neq 0$. Can we show that V is not a subspace without taking the shortcut of first checking that V contains the zero vector? The answer is yes. Let's do vector addition with $b \neq 0$. Then

$$v_1 = \begin{bmatrix} x_1 \\ mx_1 + b \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} x_2 \\ mx_2 + b \end{bmatrix}$$

and

$$v_1 + v_2 = \begin{bmatrix} x_1 + x_2 \\ mx_1 + mx_2 + 2b \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ (m(x_1 + x_2) + b) + \boxed{b} \end{bmatrix} \not\in V.$$

We see that we cannot write

$$v_1 + v_2 = \begin{bmatrix} x \\ mx + b \end{bmatrix}$$

for some x in $\mathbb{R}$, when $b \neq 0$, and that is why V is not closed under vector addition. You can also check that it is not closed under scalar times vector multiplication, but for the purpose of showing a subset of $\mathbb{R}^n$ is not a subspace, you only need to violate ONE of the conditions. ■

Example 9.2 *$V := \mathbb{R}^n$ is a subspace of $\mathbb{R}^n$ and $W := \{0\} \subset \mathbb{R}^n$, the zero vector, is a subspace of $\mathbb{R}^n$*

Solution: We'll let you apply the definition to V and check that it is a subspace. V is the largest subspace of $\mathbb{R}^n$, because it is all of $\mathbb{R}^n$, while W is the smallest subspace of $\mathbb{R}^n$, since it consists only of the zero vector. Is W really a subspace? Well, if we add any two vectors in W, we are adding $n \times 1$ zero vectors, and we'll get the $n \times 1$ zero vector. If we take any real number $\alpha \in \mathbb{R}$ and vector $w \in W$, then we are multiplying the $n \times 1$ zero vector by α, and we get once again the $n \times 1$ zero vector. Hence, W is closed under the operations of vector addition and scalar times vector multiplication, showing that it is a subspace. ■

Example 9.3 *$V := \left\{ \begin{bmatrix} x_1 \\ 0 \\ x_1 + 4x_2 \end{bmatrix} \,\middle|\, x_1, x_2 \in \mathbb{R} \right\}$ is a subspace of $\mathbb{R}^3$, while $W := \left\{ \begin{bmatrix} x_1 \\ 5 \\ x_1 + 4x_2 \end{bmatrix} \,\middle|\, x_1, x_2 \in \mathbb{R} \right\}$ is a NOT subspace of $\mathbb{R}^3$.*

Solution: W does not contain the zero vector and hence it cannot be a subspace. What about V? We write

$$v_1 = \begin{bmatrix} \overline{x}_1 \\ 0 \\ \overline{x}_1 + 4\overline{x}_2 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} \overline{\overline{x}}_1 \\ 0 \\ \overline{\overline{x}}_1 + 4\overline{\overline{x}}_2 \end{bmatrix}.$$

Then,

$$v_1 + v_2 = \begin{bmatrix} \overline{x}_1 + \overline{\overline{x}}_1 \\ 0 \\ \overline{x}_1 + 4\overline{x}_2 + \overline{\overline{x}}_1 + 4\overline{\overline{x}}_2 \end{bmatrix} = \begin{bmatrix} (\overline{x}_1 + \overline{\overline{x}}_1) \\ 0 \\ (\overline{x}_1 + \overline{\overline{x}}_1) + 4(\overline{x}_2 + \overline{\overline{x}}_2) \end{bmatrix} = \begin{bmatrix} x_1 \\ 0 \\ x_1 + 4x_2 \end{bmatrix}$$

for $x_1 = (\overline{x}_1 + \overline{\overline{x}}_1)$ and $x_2 = (\overline{x}_2 + \overline{\overline{x}}_2)$. Hence, $v_1 + v_2 \in V$.

Working out $\alpha v_1 \in V$ for all $\alpha \in \mathbb{R}$ follows the same pattern. We conclude that V is a subspace. ■

Null Space and Range of a Matrix

Let A be an $n \times m$ matrix. We can then define a function $f : \mathbb{R}^m \to \mathbb{R}^n$ by, for each $x \in \mathbb{R}^m$

$$f(x) := Ax \in \mathbb{R}^n. \tag{9.5}$$

The following **subsets** are naturally motivated by the function view of a matrix.

Definition:

(a) $\text{null}(A) := \{x \in \mathbb{R}^m \mid Ax = 0_{n \times 1}\}$ is the **null space** of A.

(b) $\text{range}(A) := \{y \in \mathbb{R}^n \mid y = Ax \text{ for some } x \in \mathbb{R}^m\}$ is the **range** of A.

In Example 9.6, we show that the null space and range of a matrix are in fact **subspaces**.

Example 9.4 *Find the null spaces of*

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix} \text{ and } A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}.$$

Solution:

$$A_1 x = 0 \iff \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0 \iff \begin{bmatrix} x_1 \\ -x_2 + x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \iff x = \begin{bmatrix} 0 \\ \alpha \\ \alpha \end{bmatrix}, \text{ for } \alpha \in \mathbb{R}.$$

Hence,

$$\text{null}(A_1) = \left\{ \alpha \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \,\middle|\, \alpha \in \mathbb{R} \right\}.$$

For the second matrix,

$$A_2x = 0 \iff \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \iff \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Hence,

$$\text{null}(A_2) = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\}.$$

■

Example 9.5 *Find the ranges of*

$$A_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix} \textit{ and } A_4 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}.$$

Solution: We note that A_3 is 2×3 and A_4 is 3×3. Hence,

$$\begin{aligned} \text{range}(A_3) &= \left\{ A_3x \mid x \in \mathbb{R}^3 \right\} \\ &= \left\{ \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \;\middle|\; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^3 \right\} \\ &= \left\{ \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \;\middle|\; \alpha_1, \alpha_2, \alpha_3 \in \mathbb{R} \right\} \\ &= \left\{ \alpha_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ -1 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \;\middle|\; \alpha_1, \alpha_2, \alpha_3 \in \mathbb{R} \right\} \\ &= \left\{ \alpha_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ -1 \end{bmatrix} \;\middle|\; \alpha_1, \alpha_2 \in \mathbb{R} \right\}, \end{aligned}$$

where the third column of A_3 was eliminated because it is linearly dependent on the first two columns; in fact, it is the negative of the second column.

$$\begin{aligned} \text{range}(A_4) &= \left\{ A_4x \mid x \in \mathbb{R}^3 \right\} \\ &= \left\{ \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \;\middle|\; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^3 \right\} \\ &= \left\{ \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \;\middle|\; \alpha_1, \alpha_2, \alpha_3 \in \mathbb{R} \right\} \\ &= \left\{ \alpha_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \;\middle|\; \alpha_1, \alpha_2, \alpha_3 \in \mathbb{R} \right\} \\ &= \left\{ \alpha_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \;\middle|\; \alpha_1, \alpha_3 \in \mathbb{R} \right\}, \end{aligned}$$

where the second columns was eliminated because it is dependent on the first column (in fact, it is twice the first column). ■

Example 9.6 *Show that both the null space and range of an $n \times m$ matrix A are subspaces.*

Solution: (a) We suppose that v_1 and v_2 are in null(A). Hence, $Av_1 = 0$ and $Av_2 = 0$. We form a linear combination $\alpha_1 v_1 + \alpha_2 v_2 \in \mathbb{R}^n$ and check if it is also in null(A). For the linear combination to be in null(A), we must have that A multiplying $\alpha_1 v_1 + \alpha_2 v_2$ yields zero. So we check

$$A(\alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 A v_1 + \alpha_2 A v_2 = 0 + 0 = 0.$$

Hence, null(A) is closed under linear combinations and it is therefore a subspace.

(b) We suppose that v_1 and v_2 are in range(A). Hence, there exists u_1 and u_2 such that $Au_1 = v_1$ and $Au_2 = v_2$. We form a linear combination $\alpha_1 v_1 + \alpha_2 v_2 \in \mathbb{R}^n$ and check if it is also in range(A). For the linear combination to be in range(A), we must produce a $u \in \mathbb{R}^m$ such that $Au = \alpha_1 v_1 + \alpha_2 v_2$. We propose $u = \alpha_1 u_1 + \alpha_2 u_2$ and check that

$$Au = A(\alpha_1 u_1 + \alpha_2 u_2) = \alpha_1 A u_1 + \alpha_2 A v_2 = \alpha_1 v_1 + \alpha_2 v_2,$$

and hence $\alpha_1 v_1 + \alpha_2 v_2 \in$ range(A). Because it is closed under linear combinations, range(A) is therefore a subspace. ■

Relation of Null Space and Range to Solutions of Linear Equations

Let's look now at some relations between solutions of $Ax = b$ and the null space and range of A.

(a) Suppose that $\overline{x}$ is a solution of $Ax = b$, that is, $Ax = b$, and let $\tilde{x} \in$ null(A). Is $\overline{\overline{x}} = x + \tilde{x}$ also a solution?

$$A\overline{\overline{x}} = A(\overline{x} + \tilde{x}) = \underbrace{A\overline{x}}_{b} + \underbrace{A\tilde{x}}_{0_{n\times 1}} = b.$$

Does this tell us how to generate all solutions to $Ax = b$? In fact, yes! If $\overline{x}$ and $\overline{\overline{x}}$ are any two solutions of the system of equations, then $A(\overline{x} - \overline{\overline{x}}) = A\overline{x} - A\overline{\overline{x}} = b - b = 0_{n\times 1}$, and thus $\overline{x} - \overline{\overline{x}} \in$ null(A) Hence, once we know any one solution to $Ax = b$ and we know null(A), we can generate all solutions of $Ax = b$. Moreover, if $Ax = b$ has a solution, then it is unique if, and only if, null$(A) = \{0_{m\times 1}\}$, the zero vector in $\mathbb{R}^m$.

(b) Because we can denote vectors in $\mathbb{R}^n$ by any symbol we wish, we have that

$$\begin{aligned} \text{range}(A) &:= \{y \in \mathbb{R}^n \mid y = Ax \text{ for some } x \in \mathbb{R}^m\} \\ &= \{b \in \mathbb{R}^n \mid b = Ax \text{ for some } x \in \mathbb{R}^m\} \\ &= \{b \in \mathbb{R}^n \mid Ax = b \text{ for some } x \in \mathbb{R}^m\} \\ &= \{b \in \mathbb{R}^n \mid Ax = b \text{ has a solution}\}. \end{aligned}$$

Hence, $Ax = b$ has a solution if, and only if, $b \in$ range(A).

Remark: Wait a minute! We already know that $Ax = b$ has a solution if, and only if, b is a linear combination of the columns of A. Hence, could it be true that

$$\text{range}(A) = \{\text{set of all linear combinations of the columns of } A\}?$$

Hold that thought! The answer is yes. Before we dig into this, we need to make concrete the idea of "the set of all linear combinations of a given set of vectrs".

Span of a set of Vectors

Suppose that $S \subset \mathbb{R}^n$, then S is a set of vectors. The set of all possible linear combinations of elements of S is called the span of S,

$$\text{span}\{S\} := \{\text{all possible linear combinations of elements of } S\}.$$

It follows that span$\{S\}$ is a subspace of $\mathbb{R}^n$ because, by definition, it is closed under linear combinations. This is true for any subset $S \subset \mathbb{R}^n$.

If a set S is already known to be a subspace of $\mathbb{R}^n$, then taking its span does not add any new vectors because a subspace is closed under linear combinations. Hence, $S \subset \mathbb{R}^n$ and S a subspace $\implies$ span$\{S\} = S$. When S is not a subspace, then there is at least one linear combination of elements of S that is not in S itself. In this case, the span is a bigger set. Indeed, $S \subset$ span$\{S\}$ and $S \neq$ span$\{S\}$.

Example 9.7 *Consider the vector space $\mathbb{R}^3$ and two vectors e_1 and e_2, where for $k = 1, 2$, e_k has a one in the k-the entry and zeros elsewhere. Compute span$\{e_1, e_2\}$.*

Solution: We have $S = \{e_1, e_2\}$.

$$\begin{aligned}
\text{span}\{S\} &:= \{\text{all possible linear combinations of elements of } S\} \\
&= \left\{\alpha_1 e_1 + \alpha_2 e_2 \,\middle|\, \alpha_1 \in \mathbb{R}, \alpha_2 \in \mathbb{R}\right\} \\
&= \left\{\alpha_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \,\middle|\, \alpha_1 \in \mathbb{R}, \alpha_2 \in \mathbb{R}\right\} \\
&= \left\{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ 0 \end{bmatrix} \,\middle|\, \alpha_1 \in \mathbb{R}, \alpha_2 \in \mathbb{R}\right\} \\
&= \left\{\begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \,\middle|\, x \in \mathbb{R}, y \in \mathbb{R}\right\}.
\end{aligned}$$

Hence, span$\{e_1, e_2\}$ is the x-y-plane in $\mathbb{R}^3$. ■

For this very simple and very special case, we could "recognize" what the span of a the set of vectors turned out to be and name it. More generally, we cannot "recognize" what the span looks like. We simply use it as convenient notation when we want to work with the linear combinations associated with a set of vectors and not just the vectors themselves. A span seems very abstract at the moment, but it will become more concrete in the next section when we discuss basis vectors, and then even more meaningful when we build orthogonal[1] basis vectors!

Column Span of a Matrix:

Let A be an $n \times m$ matrix. Then its columns are vectors in $\mathbb{R}^n$. Their span is called the **column span of A**.

$$\text{col span}\{A\} := \text{span}\{a_1^{\text{col}}, \ldots, a_m^{\text{col}}\}.$$

We saw in Chapter 7.3 that $Ax = b$ has a solution if, and only if, b is a linear combination of the columns of A. A more elegant way to write this is

$$\mathbf{Ax = b \text{ has a solution if, and only if, } b \in \text{col span}\{A\}} \qquad .$$

Example 9.8 *Suppose* $A = \begin{bmatrix} 3 & 2 \\ 1 & -2 \\ -1 & 1 \end{bmatrix}$ *and* $b = \begin{bmatrix} 0 \\ -8 \\ 5 \end{bmatrix}$. *Does $Ax = b$ have a solution?*

[1] You might think of them as vectors that are "perpendicular" to one another.

Solution: This is identical to the problems we solved in Chapter 7.6. We check that

$$b = -2a_1^{\text{col}} + 3a_2^{\text{col}} \in \text{span}\{a_1^{\text{col}}, a_2^{\text{col}}\},$$

and hence b is in the column span of A, and the system of linear equations has a solution. ■

Range of A Equals Column Span of A

Let A be an $n \times m$ matrix so that its columns are vectors in $\mathbb{R}^n$,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} =: \begin{bmatrix} a_1^{\text{col}} & a_2^{\text{col}} & \ldots & a_m^{\text{col}} \end{bmatrix}$$

Then

$$\boxed{\text{range}(A) := \{Ax \mid x \in \mathbb{R}^m\} = \text{span}\{a_1^{\text{col}}, a_2^{\text{col}}, \ldots, a_m^{\text{col}}\} =: \text{col span}\{A\}.}$$

Remark: Working to understand why range(A) = col span$\{A\}$ can really bring a lot of concepts together. We delay the "proof" until we have one more tool, the notion of a **basis** for a vector space or subspace.

9.3 Basis Vectors and Dimension

We consider $\mathbb{R}^n$ again, and define some special vectors.

Let I_n be the $n \times n$ identity matrix. Then $e_i :=$ i-th column of I_n. For example, when $n = 4$,

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, e_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

Sticking with $\mathbb{R}^4$ just to make things definite, we note that $\{e_1, e_2, e_3, e_4\}$ is a linearly independent set, because

$$\left(\alpha_1 e_1 + \cdots + \alpha_4 e_4 = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) \iff \Big(\alpha_1 = 0, \ldots, \alpha_4 = 0 \Big). \tag{9.6}$$

What is even more remarkable is that any vector in $\mathbb{R}^4$ can be written as a linear combination of $\{e_1, e_2, e_3, e_4\}$. Indeed,

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + x_4 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = x_1 e_1 + \cdots + x_4 e_4. \tag{9.7}$$

This is the essence of a **basis**: a set of vectors that is (a) "big enough" to generate all vectors in a vector space or a subspace by forming linear combinations and (b) "small enough" to be linearly independent.

Yes, it's kind of a "Goldilocks" notion: a set of vectors that is not too big (linearly independent) and not too small (spans the subspace). Just the right size!

Basis vectors are important because they provide a simple means to generate all vectors in a vector space or a subspace by forming linear combinations from a finite list of vectors. The basis and the subspace can be essentially treated as one and the same objects when it comes to computations: we can manipulate a subspace in a computer by computing with its basis vectors!

Basis Vectors and Dimension

Suppose that V is a subspace of $\mathbb{R}^n$. Then $\{v_1, v_2, \ldots, v_k\}$ is a **basis for V** if

1. the set $\{v_1, v_2, \ldots, v_k\}$ is linearly independent, and
2. span$\{v_1, v_2, \ldots, v_k\} = V$.

The **dimension of V is k**, the number of basis vectors[a].

We note that the above definition applies to $\mathbb{R}^n$ **because** $\mathbb{R}^n$ is a subset of itself and it is closed under linear combinations. In particular, $\mathbb{R}^n$ has dimension n, or we say that $\mathbb{R}^n$ is an n-dimensional vector space.

[a] A more correct definition is the maximum number of vectors in any linearly independent set contained in V. For ROB 101, the definition we gave is good enough

We will work some examples, though you may not find them totally convincing until we develop the notion of "orthonormal" basis vectors, that is, basis vectors where each vector forms a 90^o angle to the other vectors in the basis, and each vector has length one. First some less exciting basis vectors!

Example 9.9 *We consider the subspace of* $\mathbb{R}^3$ *defined by*

$$V := \left\{ x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^3 \mid x_1 + x_2 + x_3 = 0 \right\}.$$

Show that

$$\left\{ v_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} \right\}$$

is a basis for V *and hence* V *is a two dimensional subspace of* $\mathbb{R}^3$.

Solution: The Example stated that V is a subspace. Let's quickly show it is a subspace, just to be on the safe side. We note that

$$V := \left\{ x \in \mathbb{R}^3 \;\middle|\; \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = 0 \right\},$$

and hence $V = \text{null}(A)$. Because the null space of a matrix is a subspace, we have shown that V is a subspace.

To show that $\{v_1, v_2\}$ is a basis for V, we need that to check that

- $\{v_1, v_2\} \subset V$,
- the set $\{v_1, v_2\}$ is linearly independent, and
- span$\{v_1, v_2\} = V$.

We leave the reader to show the first two properties: that v_1 and v_2 are in V and they are linearly independent. The hard part is showing the span property, namely, that all vectors in V can be written as a linear combination of v_1 and v_2. To do this, we note that

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in V \iff x_1 + x_2 + x_3 = 0 \iff x_3 = -(x_1 + x_2) \iff x = \begin{bmatrix} x_1 \\ x_2 \\ -(x_1 + x_2) \end{bmatrix}.$$

Taking $x_1 = 1$ and $x_2 = 0$ gives v_1, while taking $x_1 = 0$ and $x_2 = 1$ gives v_2.

We have

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in V \iff x = \begin{bmatrix} x_1 \\ x_2 \\ -(x_1 + x_2) \end{bmatrix} \iff x = x_1 v_1 + x_2 v_2 \iff x \in \text{span}\{v_1, v_2\}.$$

The dimension follows from the number of elements in the basis.

Now, we could just as easily have written

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in V \iff x_1 + x_2 + x_3 = 0 \iff x_2 = -(x_1 + x_3) \iff x = \begin{bmatrix} x_1 \\ -(x_1 + x_3) \\ x_3 \end{bmatrix}.$$

Then, taking $x_1 = 1$ and $x_3 = -1$ gives v_1, while taking $x_1 = 0$ and $x_3 = -1$ gives v_2.

■

The point is that V is now rather simple to understand and manipulate as the set of linear combinations of v_1 and v_2.

Canonical or Natural Basis Vectors

Let $n \geq 1$ and, as before, $e_i :=$ i-th column of the $n \times n$ identity matrix, I_n. Then

$$\{e_1, e_2, \ldots, e_n\}$$

is a basis for the vector space $\mathbb{R}^n$. Its elements e_i are called both **natural basis vectors** and **canonical basis vectors**. The frequency of usage of one name vs the other is about fifty-fifty!

Remark: Showing linear independence is identical to (9.6) and showing that the span is all of $\mathbb{R}^n$ is the same as in (9.7).

Why Range of A Equals Column Span of A?

Let $\{e_1, e_2, \ldots, e_m\}$ be the canonical basis vectors for $\mathbb{R}^m$. Then

$$Ae_i = a_i^{\text{col}}. \tag{9.8}$$

Indeed, from our "sum over columns times rows" definition of multiplication, we have that

$$Ae_i = \sum_{k=1}^{m} a_k^{\text{col}} (e_i)_k.$$

And then because all of the entries of e_i are zero except for its i-th entry, that is,

$$(e_i)_k = \begin{cases} 1 & i = k \\ 0 & i \neq k, \end{cases}$$

we obtain (9.8). Because $\{e_1, e_2, \ldots, e_m\}$ is a basis, we can write $x = x_1 e_1 + x_2 e_2 + \cdots + x_m e_m$, and thus obtain

$$Ax = A \sum_{i=1}^{m} x_i e_i = \sum_{i=1}^{m} x_i A e_i = \sum_{i=1}^{m} x_i a_i^{\text{col}}.$$

Hence,

$$\text{range}(A) := \{Ax \mid x \in \mathbb{R}^m\} = \left\{ \sum_{i=1}^{m} x_i a_i^{\text{col}} \mid x_i \in \mathbb{R} \right\} = \text{span}\{a_1^{\text{col}}, a_2^{\text{col}}, \ldots, a_m^{\text{col}}\} =: \text{col span}\{A\}.$$

Example 9.10 *Let A be an $n \times n$ matrix. Then the columns of A form a basis for $\mathbb{R}^n$ if, and only if,* $\det(A) \neq 0$.

As a special case, we can take $A = I_n$, the columns of which give the canonical basis vectors.

Solution: **[Trigger Warning: This is a proof. You may want to skip it.]**
Let's first assume that $\det(A) \neq 0$. Then we must show that the columns of A are linearly independent and they span $\mathbb{R}^n$. We take these one at a time. We denote the columns of A by $\{v_1 = a_1^{\text{col}}, v_2 = a_2^{\text{col}}, \dots, v_n = a_n^{\text{col}}\}$.

To show linear independence, we need to establish that the only solution to

$$\alpha_1 a_1^{\text{col}} + \alpha_2 a_2^{\text{col}} + \cdots + \alpha_n a_n^{\text{col}} = 0_{n\times 1} \tag{9.9}$$

is the trivial solution, $\alpha_1 = 0, \alpha_2 = 0, \dots, \alpha_n = 0$. But this is equivalent to

$$\underbrace{\begin{bmatrix} a_1^{\text{col}} & a_2^{\text{col}} & a_n^{\text{col}} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}}_{\alpha} = 0_{n\times 1}$$

$$Updownarrow$$

$$A\alpha = 0_{n\times 1}$$

From $\det(A) \neq= 0$, we know that the solution to $A\alpha = 0$ is unique, and hence the only solution is the trivial solution $\alpha = 0_{n\times 1}$. ■

Example 9.11 *Let A be an $n \times n$ matrix. Then the columns of A form a basis for $\mathbb{R}^n$ if, and only if, the columns of A are linearly independent.*

As a special case, we can take $A = I_n$, the columns of which give the canonical basis vectors.

Solution: **[Trigger Warning: This is a proof. You may want to skip it.]** Let's suppose that $A = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$, so that $\{v_1, v_2, \dots, v_n\}$ are its columns. If the set $\{v_1, v_2, \dots, v_n\}$ forms a basis of $\mathbb{R}^n$, then by the definition of a basis, the set must be linearly independent. So that direction is trivially true.

The other direction is much less obvious: suppose that the column vectors $\{v_1, v_2, \dots, v_n\}$ are linearly independent, then do they form a basis of $\mathbb{R}^n$? Well, they then meet the first part of the definition! But is it true that they satisfy the second part of the definition, namely,

$$\text{span}\{v_1, v_2, \dots, v_n\} = \mathbb{R}^n?$$

Well, spans are simply linear combinations, so the question becomes, can every vector in $\mathbb{R}^n$ be written as a linear combination of the columns of A?

Let's transform this to a question about the existence of solutions to systems of linear equations. We let $b \in \mathbb{R}^n$ be arbitrary and ask if there exists $x \in \mathbb{R}^n$ that solves the equation

$$Ax = b, \tag{9.10}$$

meaning that $x_1 v_1 + x_2 v_2 + \cdots + x_n v_n = b$. If we could show that $\det(A) \neq 0$, then we know that we could write $x = A^{-1}b$, and then we would have shown that $b \in \text{span}\{v_1, v_2, \dots, v_n\}$ and, because b is arbitrary, that $\text{span}\{v_1, v_2, \dots, v_n\} = \mathbb{R}^n$.

How can we attack the problem of showing that $\det(A) \neq 0$? Well, let's use the LU Factorization. So, we write $A = L \cdot U$, and for simplicity, we forget about the permutation matrix. We know that L and U are $n \times n$ and triangular, $\det(L) = 1$, and that A and U have the same number of linearly independent columns, which must be n. It follows that there are no zeros on the diagonal of U and hence $\det(U) \neq 0$. Then, from the product rule of determinants,

$$\det(A) = \det(L) \cdot \det(U) = \det(U) \neq 0,$$

and we're done. If we had included the permutation matrix, then the result would have been $\det(A) = \pm\det(U) \neq 0$, which does not change the result. ■

The following brings together–into one place–everything we know for **square** systems of equations.

A Potpourri of Facts about Square Systems of Linear Equations

Let A be an $n \times n$ matrix and associate its column with vectors in $\mathbb{R}^n$ by $\{v_1 = a_1^{\text{col}}, v_2 = a_2^{\text{col}}, \dots, v_n = a_n^{\text{col}}\}$. Then the following statements are equivalent:

- for every vector $b \in \mathbb{R}^n$, the equation $Ax = b$ has a unique solution;
- $\det(A) \neq 0$;
- the set $\{v_1, v_2, \dots, v_n\}$ is linearly independent;
- the set $\{v_1, v_2, \dots, v_n\}$ is a basis of $\mathbb{R}^n$;
- $\text{range}(A) = \mathbb{R}^n$;
- $\text{col span}\{A\} = \mathbb{R}^n$;
- for every vector $b \in \mathbb{R}^n$, the equation $Ax = b$ has a solution;
- for some vector $b \in \mathbb{R}^n$, the equation $Ax = b$ has a solution and it is unique; and
- the equation $Ax = 0_{n \times 1}$ has a unique solution.

Can you piece together all of these facts? If not, let the leader of your recitation section know, or, work with student colleagues on Piazza!

Example 9.12 ***Optional Read (One more subspace and basis example):*** *Suppose that $n = n_1 + n_2$, where $n_1 \geq 1$ and $n_2 \geq 1$, and let M be an $n_1 \times n_2$ matrix. Let's note that we can write any vector $x \in \mathbb{R}^n$ by stacking two vectors $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n_2}$ as in*

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

Define a subset of $\mathbb{R}^n$ by

$$V := \left\{ x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^n \mid x_1 + Mx_2 = 0 \right\}.$$

Show that V is a subspace of $\mathbb{R}^n$ and that

$$\left\{ v_1 = \left[\begin{array}{c} -Me_1 \\ e_1 \end{array} \right], v_2 = \left[\begin{array}{c} -Me_2 \\ e_2 \end{array} \right], \dots, v_{n_2} = \left[\begin{array}{c} -Me_{n_2} \\ e_{n_2} \end{array} \right] \right\} \tag{9.11}$$

is a basis for V, where the e_i are the canonical basis vectors for $\mathbb{R}^{n_2}$. This will show that V is an n_2-dimensional subspace of $\mathbb{R}^n$.

Solution:[**Trigger warning: the following is essentially a proof. It shows you the steps you must follow to establish that a finite set of vectors is a basis for a subspace.**] V is the null space of the matrix $\left[\begin{array}{cc} I_{n_1 \times n_1} & M \end{array} \right]$ because

$$\left[\begin{array}{cc} I_{n_1 \times n_1} & M \end{array} \right] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \iff x_1 + Mx_2 = 0.$$

Hence, V is a subspace. Moreover, we see that $x_1 + Mx_2 = 0 \iff x_1 = -Mx_2$ and thus

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in V \iff x = \left[\begin{array}{c} -Mx_2 \\ x_2 \end{array} \right], x_2 \in \mathbb{R}^{n_2}. \tag{9.12}$$

Let $\{e_1, e_2, \dots, e_{n_2}\} \subset \mathbb{R}^{n_2}$ be the canonical basis vectors for $\mathbb{R}^{n_2}$ and define

$$\{v_1, v_2, \dots, v_{n_2}\} \subset \mathbb{R}^n$$

by

$$v_i := \left[\begin{array}{c} -Me_i \\ e_i \end{array} \right]. \tag{9.13}$$

By (9.12), v_i is indeed an element of V. To show the vectors in (9.13) form a basis, we need to investigate:

- Is the set $\{v_1, v_2, \ldots, v_{n_2}\}$ linearly independent?

- Does the set $\{v_1, v_2, \ldots, v_{n_2}\}$ span V?

We look at these one at a time. Let $\alpha_1, \alpha_2, \ldots, \alpha_{n_2}$ be real numbers and consider the linear combination $\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_{n_2} v_{n_2}$. Then,

$$\begin{aligned}
0 &= \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_{n_2} v_{n_2} \\
&\Updownarrow \\
\begin{bmatrix} 0 \\ 0 \end{bmatrix} &= \alpha_1 \begin{bmatrix} -Me_1 \\ e_1 \end{bmatrix} + \alpha_2 \begin{bmatrix} -Me_2 \\ e_2 \end{bmatrix} + \cdots + \alpha_{n_2} \begin{bmatrix} -Me_{n_2} \\ e_{n_2} \end{bmatrix} \\
&\Updownarrow \\
\begin{bmatrix} 0 \\ 0 \end{bmatrix} &= \begin{bmatrix} -\alpha_1 Me_1 \\ \alpha_1 e_1 \end{bmatrix} + \begin{bmatrix} -\alpha_2 Me_2 \\ \alpha_2 e_2 \end{bmatrix} + \cdots + \begin{bmatrix} -\alpha_{n_2} Me_{n_2} \\ \alpha_{n_2} e_{n_2} \end{bmatrix} \\
&\Downarrow \text{ (one way implication)} \\
0 &= \alpha_1 e_1 + \alpha_2 e_2 + \cdots + \alpha_{n_2} e_{n_2} \\
&\Updownarrow \\
&\alpha_1 = 0, \alpha_2 = 0, \ldots, \alpha_{n_2} = 0,
\end{aligned} \tag{9.14}$$

where the last line follows from the linear independence of the canonical basis vectors. Hence, $\{v_1, v_2, \ldots, v_{n_2}\}$ is linearly independent.

Next, we note that any vector $x_2 \in \mathbb{R}^{n_2}$ can be written as a linear combination

$$x_2 = \alpha_1 e_1 + \alpha_2 e_2 + \cdots \alpha_{n_2} e_{n_2},$$

because the canonical basis vectors are a basis. Hence, from (9.12), we have that $x \in V$ can be written as

$$\begin{aligned}
x &= \begin{bmatrix} -Mx_2 \\ x_2 \end{bmatrix} \\
&= \begin{bmatrix} -M\left(\alpha_1 e_1 + \alpha_2 e_2 + \cdots \alpha_{n_2} e_{n_2}\right) \\ \alpha_1 e_1 + \alpha_2 e_2 + \cdots \alpha_{n_2} e_{n_2} \end{bmatrix} \\
&= \alpha_1 \begin{bmatrix} -Me_1 \\ e_1 \end{bmatrix} + \alpha_2 \begin{bmatrix} -Me_2 \\ e_2 \end{bmatrix} + \cdots + \alpha_{n_2} \begin{bmatrix} -Me_{n_2} \\ e_{n_2} \end{bmatrix} \\
&= \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_{n_2} v_{n_2},
\end{aligned}$$

and therefore, $x \in V \iff x \in \text{span}\{v_1, v_2, \ldots, v_{n_2}\}$. We conclude that (9.11) is a basis for V and hence V has dimension n_2. ■

9.4 Some Handy Matrix Facts

In the context of the range and null space of a matrix, here are the key matrix properties that you will encounter frequently when applying linear algebra to your engineering problems. We'll explain why they are true at the end of the Chapter. For now, let's go have some fun with orthogonal vectors!

Definition of Rank and Nullity

For an $n \times m$ matrix A,

Def. rank$(A) := \dim$ col span$\{A\}$.

Def. nullity$(A) := \dim$ null(A).

Remark: If a system of equations $Ax = b$ has a solution, $\bar{x}$, then $A(\bar{x} + \eta) = b$ for all $\eta \in$ null$A$. Hence, nullity(A) is measuring the "dimension" of the set of solutions. Because col span$\{A\} \subset \mathbb{R}^n$, we see that rank$(A) \leq n$. From the properties below, we have rank$(A) =$ rank$(A^\top) \leq m$. Putting these together, we deduce that

$$\text{rank}(A) \leq \min\{n, m\},$$

that is, the rank of a matrix is upper bounded by the the smaller of the number of its rows and the number its columns of A.

Another very useful result is

$$\text{rank}(A) + \text{nullity}(A) = m, \tag{9.15}$$

the number of columns in A.

Useful Properties of Rank and Nullity

For an $n \times m$ matrix A, key results on its rank and nullity are given below:

Fact rank$(A) +$ nullity$(A) = m$, the number of columns in A. This is known as the **Rank-Nullity Theorem**.

Fact rank$(A^\top A) =$ rank(A).

Fact rank$(A^\top) =$ rank(A).

Fact nullity$(A^\top \cdot A) =$ nullity(A).

Fact nullity$(A^\top) + m =$ nullity$(A) + n$.

Fact For any $m \times k$ matrix B, rank$(A \cdot B) \leq$ rank(A).

Fact For any $p \times n$ matrix C, rank$(C \cdot A) \leq$ rank(A).

9.5 Dot Product, Orthonormal Vectors, Orthogonal Matrices

9.5.1 Dot Product or Inner Product

Definition: We let $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^n$ be **column vectors**,

$$u := \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

The **dot product** of u and v is defined as

$$u \bullet v := \sum_{k=1}^{n} u_k v_k.$$

We note that

$$u^\top \cdot v = \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \sum_{k=1}^{n} u_k v_k =: u \bullet v$$

For many people, this is how they remember the **dot product:** as $u^\top v$. In fact, you are welcome to use this as the definition of the dot product.

The dot product is also called the **inner product**. The terminology of inner product is very common. In ROB 101, you can choose your preferred terminology in this regard. Your instructors prefer *inner product*.

Example 9.13 *Compute the dot product for*

$$u := \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}, \quad v = \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix}.$$

Solution:

$$u \bullet v = (1)(2) + (0)(4) + (3)(1) = 5$$

$$u^\top v = (1)(2) + (0)(4) + (3)(1) = 5.$$

You can use either notation. ■

Example 9.14 *Compute the inner product for*

$$u := \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \end{bmatrix}, \quad v = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

Solution:

$$u \bullet v = (1)(0) + (0)(1) + (-1)(0) + (0)(1) = 0$$

$$u^\top v = (1)(0) + (0)(1) + (-1)(0) + (0)(1) = 0.$$

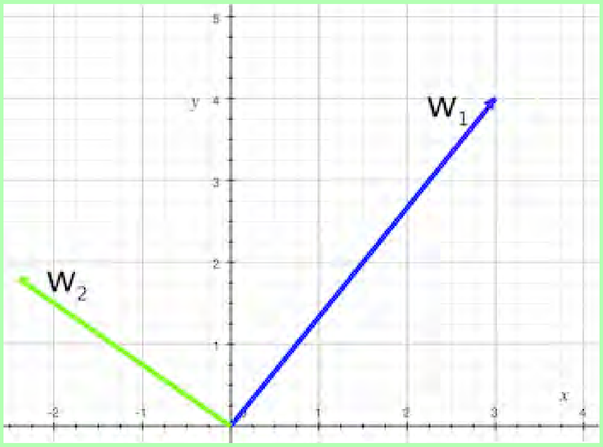
You can use either notation. ■

Key Use of the Dot (aka Inner) Product of Two Vectors

The inner product will provide us a generalization of a right angle ($90\deg$ angle) between two vectors in $\mathbb{R}^n$.

$$w_1 \perp w_2 \iff w_1 \bullet w_2 = 0 \iff w_1^\top w_2 = 0$$

(Read it as: w_1 is **orthogonal** to w_2 if, and only if, their inner product is zero. Orthogonal means "at right angle")



Source: `https://study.com/academy/lesson/the-gram-schmidt-process-for-orthonormalizing-vectors.html`

Reading the values from the graph, we have

$$w_1 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}, w_2 = \begin{bmatrix} -\frac{7}{3} \\ \frac{7}{4} \end{bmatrix} \implies w_1 \bullet w_2 = w_1^\top w_2 = -3\frac{7}{3} + 4\frac{7}{4} = 0$$

It's very useful and amazing that this works for all vector spaces $\mathbb{R}^n$, as long as $n \geq 2$. Can you picture a right angle in $\mathbb{R}^{27}$? Neither can your instructors, but later we'll see how useful the idea can be!

Pythagorean Theorem in $\mathbb{R}^n$, n $\geq$ 2.

Suppose that $w_1 \perp w_2$. Then,

$$||w_1 + w_2||^2 = ||w_1||^2 + ||w_2||^2.$$

Remark In the above plot, draw the line from w_1 to w_2 and call its length c; in addition, label the length of w_1 as a and that of w_2 as b. Then yes, we have the classic relationship: $c^2 = a^2 + b^2$.

Example 9.15 *Why is this true?*

Solution: [**Trigger Warning: This is a proof. You may want to skip it.**] Because $w_1 \perp w_2$, we know that $w_1 \bullet w_2 = 0$, which means that $w_1^\top \cdot w_2 = w_2^\top \cdot w_1 = 0$. Finally, we recall that the norm-squared of a vector v is $||v||^2 = v^\top \cdot v$. Using these facts we grind out the computation and see

$$\begin{aligned} ||w_1 + w_2||^2 &:= (w_1 + w_2)^\top \cdot (w_1 + w_2) \\ &= w_1^\top \cdot (w_1 + w_2) + w_2^\top \cdot (w_1 + w_2) \\ &= w_1^\top \cdot w_1 + w_1^\top \cdot w_2 + w_2^\top \cdot w_1 + w_2^\top \cdot w_2 \\ &= \underbrace{w_1^\top \cdot w_1}_{||w_1||^2} + \underbrace{w_1^\top \cdot w_2}_{0} + \underbrace{w_2^\top \cdot w_1}_{0} + \underbrace{w_2^\top \cdot w_2}_{||w_2||^2} \\ &= ||w_1||^2 + ||w_2||^2. \end{aligned}$$

■

Example 9.16 *Determine which pairs of vectors, if any, are orthogonal*

$$u = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}, w = \begin{bmatrix} -5 \\ 0 \\ 1 \end{bmatrix}.$$

Solution

$$u \bullet v = u^\top \cdot v = \begin{bmatrix} 2 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} = (2)(1) + (1)(3) + (-1)(5) = 0$$
$$u \bullet w = u^\top \cdot w = \begin{bmatrix} 2 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} -5 \\ 0 \\ 1 \end{bmatrix} = (2)(-5) + (1)(0) + (-1)(1) = -11$$
$$v \bullet w = v_\top \cdot w = \begin{bmatrix} 1 & 3 & 5 \end{bmatrix} \cdot \begin{bmatrix} -5 \\ 0 \\ 1 \end{bmatrix} = (1)(-5) + (3)(0) + (5)(1) = 0,$$

and hence, $u \perp v$, $u \not\perp w$, and $v \perp w$. In words, u is orthogonal to v, u is not orthogonal to w, and v is orthogonal to w. ■

9.5.2 Orthonormal Vectors

A set of vectors $\{v_1, v_2, \ldots, v_n\}$ is **orthogonal** if, for all $1 \leq i, j \leq n$, and $i \neq j$

$$v_i \bullet v_j = 0. \tag{9.16}$$

We can also write this as $v_i^\top v_j = 0$ or $v_i \perp v_j$.

A set of vectors $\{v_1, v_2, \ldots, v_n\}$ is **orthonormal** if,

- they are orthogonal, and
- for all $1 \leq i \leq n$, $||v_i|| = 1$.

Example 9.17 *Scale the vector w so that its norm becomes one,*

$$w = \begin{bmatrix} -5 \\ 0 \\ 1 \end{bmatrix}.$$

Solution: In general, if $\alpha := ||w|| \neq 0$ and we define $\tilde{w} := \frac{1}{\alpha} w$, then

$$||\tilde{w}|| = 1.$$

This is true because

$$\begin{aligned} ||\tilde{w}|| &= ||\frac{1}{\alpha} \cdot w|| \\ &= \left|\frac{1}{\alpha}\right| \cdot ||w|| \ \text{(property of norms)} \\ &= \frac{1}{\alpha} \cdot ||w|| \ \ (\frac{1}{\alpha} \text{ is positive}) \\ &= \frac{1}{\alpha} \cdot \alpha \ \ \text{(definition of } \alpha) \\ &= \frac{\alpha}{\alpha} = 1 \ \ \text{(how scalar multiplication and division work)} \end{aligned}$$

Hence, we need to form $\frac{w}{||w||}$, which gives

$$\tilde{w} := \frac{1}{||w||} \cdot w = \frac{1}{\sqrt{26}} \begin{bmatrix} -5 \\ 0 \\ 1 \end{bmatrix}.$$

■

Example 9.18 *From Example 9.16, we already know that the set $\{u, v\}$ is orthogonal, that is, $u \perp v$. Make it an orthonormal set.*

$$u = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}.$$

Solution: We need to normalize their lengths to one. We compute

$$||u|| = \sqrt{(2)^2 + (1)^2 + (-1)^2} = \sqrt{6}$$
$$||v|| = \sqrt{(1)^2 + (3)^2 + (5)^2} = \sqrt{35}$$

and thus

$$\left\{ \tilde{u} := \frac{1}{\sqrt{6}} \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}, \tilde{v} := \frac{1}{\sqrt{35}} \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \right\}$$

is an orthonormal set of vectors. ■

9.5.3 Orthogonal Matrices

This section is super cool! You will learn about a kind of matrix whose inverse is equal to its transpose!! This is only the second inverse formula that your instructors want you to know and actually use!!!

The hardest aspect of the this section is the vocabulary BECAUSE **a square matrix is orthogonal** if its columns are **orthonormal vectors**. I know, why not call them orthonormal matrices? Because that would be too easy? Because, by making it confusing, we can check who really knows what they are doing and who does not? No, it's another one of those historical accidents that we live with. Fortunately, the terminology *orthonormal matrices* is used for a rectangular version of orthogonal matrices, and in applications, those are very important too.

Let's recall something about the sizes of matrices. If a matrix Q is $n \times m$, then its transpose is $m \times n$. Therefore, we can form $Q^\top \cdot Q$ and have an $m \times m$ square matrix and we can form $Q \cdot Q^\top$ and have an $n \times n$ square matrix.

Orthonormal and Orthognal Matrices

An $n \times m$ rectangular matrix Q is **orthonormal**:

- if $n > m$ (tall matrices), its columns are orthonormal vectors, which is equivalent to $Q^\top \cdot Q = I_m$; and
- if $n < m$ (wide matrices), its rows are orthonormal vectors, which is equivalent to $Q \cdot Q^\top = I_n$.

A square $n \times n$ matrix is **orthogonal** if $Q^\top \cdot Q = I_n$ and $Q \cdot Q^\top = I_n$, and hence, $Q^{-1} = Q^\top$.

Remarks:

- For a **square matrix,** $n = m$, $(Q^\top \cdot Q = I_n) \iff (Q \cdot Q^\top = I_n) \iff (Q^{-1} = Q^\top)$.
- For a tall matrix, $n > m$, $(Q^\top \cdot Q = I_m) \not\Longrightarrow (Q \cdot Q^\top = I_n)$.
- For a wide matrix, $m > n$, $(Q \cdot Q^\top = I_n) \not\Longrightarrow (Q^\top \cdot Q = I_m)$.

Determinants of Orthogonal Matrices Suppose that Q is $n \times n$ and orthogonal so that $Q^\top Q = I_n$. Then

$$[\det(Q)]^2 = 1,$$

and hence $\det(Q) = \pm 1$.

This follows from the determinant rule for a product of matrices, once you know for a square matrix A that $\det(A^\top) = \det(A)$, a property that we have not emphasized.

9.6 Constructing Orthonormal Vectors: the Gram-Schmidt Process

We already know how to normalize a set of orthogonal vectors to create a set of orthonormal vectors. Hence, we next look at how we might go about building a set of orthogonal vectors from vectors that are not orthogonal. We'll then learn the Gram-Schmidt Process, which is simply an algorithmic form of our process for building orthogonal vectors from non-orthogonal vectors.

9.6.1 Building Orthogonal Vectors

Example 9.19 *Suppose we have two vectors in* $\mathbb{R}^3$*,*

$$u_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \text{ and } u_2 = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}.$$

It is easy to compute $u_1^\top \cdot u_2 = 2 \neq 0$*, and thus the two vectors are not orthogonal. Find, if possible, two vectors* v_1 *and* v_2 *such that*

- $v_1 \perp v_2$,
- *span*$\{v_1\} =$ *span*$\{u_1\}$*, and*
- *span*$\{v_1, v_2\} =$ *span*$\{u_1, u_2\}$.

In other words, v_1 *and* v_2 *are orthogonal and yet they "generate" the same subspace as* u_1 *and* u_2*, which are not orthogonal.*

Solution: If we set $v_1 = u_1$, then we trivially satisfy span$\{v_1\} =$ span$\{u_1\}$. Let's see if we can write v_2 as a linear combination of u_1 and u_2 in such a way that $v_2 \bullet v_1 = 0$ and span$\{v_1, v_2\} =$ span$\{u_1, u_2\}$, where we have used the fact that

$$v_1 \perp v_2 \iff v_1 \bullet v_2 = 0 \iff v_2 \bullet v_1 = 0.$$

Step 1: $v_1 := u_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$

Step 2: $v_2 := u_2 - \alpha v_1$, where we seek to choose α such that $v_2 \bullet v_1 = 0$. We compute

$$\begin{aligned} v_2 \bullet v_1 &= (u_2 - \alpha v_1) \bullet v_1 \\ &= u_2 \bullet v_1 - \alpha v_1 \bullet v_1. \end{aligned}$$

If $v_1 \bullet v_1 \neq 0$, then we can set $u_2 \bullet v_1 - \alpha v_1 \bullet v_1 = 0$ and solve for α, namely

$$\alpha = \frac{u_2 \bullet v_1}{v_1 \bullet v_1}.$$

Important Formula to Build $v_1 \perp v_2$ from u_1 and u_2 while Preserving Spans

$$\begin{aligned} v_1 &= u_1 \\ v_2 &= u_2 - \left(\frac{u_2 \bullet v_1}{v_1 \bullet v_1}\right) v_1 \end{aligned} \tag{9.17}$$

$$\begin{aligned} \text{span}\{v_1\} &= \text{span}\{u_1\} \\ \text{span}\{v_1, v_2\} &= \text{span}\{u_1, u_2\} \end{aligned}$$

In our case,

$$\begin{aligned} u_2 \bullet v_1 = u_2^\top \cdot v_1 &= \begin{bmatrix} 1 & -1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 2 \\ v_1 \bullet v_1 = u_1 \bullet u_1 = u_1^\top \cdot u_1 &= \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 3 \neq 0, \end{aligned}$$

and thus

$$\alpha = \frac{u_2 \bullet v_1}{v_1 \bullet v_1} = \frac{2}{3}.$$

and hence

$$v_2 = v_2 = u_2 - \alpha v_1 = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix} - \frac{2}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ -\frac{5}{3} \\ \frac{4}{3} \end{bmatrix}$$

Did it work? Let's check if the vectors are orthogonal, that is, $v_1 \perp v_2$,

$$v_1 \bullet v_2 = v_1^\top \cdot v_2 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} \\ -\frac{5}{3} \\ \frac{4}{3} \end{bmatrix} = 0.$$

What about the span property? Well, as we noted when we started, span$\{v_1\}$ = span$\{u_1\}$ because $v_1 = u_1$.

What about span$\{v_1, v_2\}$ = span$\{u_1, u_2\}$? This part becomes a bit technical, so feel free to stop reading here and skip to the next subsection. We first ask, what does it even mean that span$\{v_1, v_2\}$ = span$\{u_1, u_2\}$? Well, it means that if we take all linear combinations of the vectors $\{v_1, v_2\}$, we obtain the same "set of vectors" as taking all linear combinations of the vectors $\{u_1, u_2\}$ We note that $v_1 = u_1$ and hence,

$$v_2 = u_2 - \alpha v_1 \tag{9.18}$$

$$\Updownarrow$$

$$v_2 = u_2 - \alpha u_1 \tag{9.19}$$

$$\Updownarrow$$

$$u_2 = v_2 + \alpha v_1 \tag{9.20}$$

From (9.19), we have that

$$\text{span}\{v_1, v_2\} = \text{span}\{u_1, u_2 - \alpha u_1\} \subset \text{span}\{u_1, u_2\},$$

while from (9.20),

$$\text{span}\{u_1, u_2\} = \text{span}\{v_1, v_2 + \alpha v_1\} \subset \text{span}\{v_1, v_2\}.$$

But for arbitrary subsets S_1 and S_2,

$$S_1 = S_2 \iff (S_1 \subset S_2 \text{ and } S_2 \subset S_1),$$

and thus we have shown that span$\{v_1, v_2\}$ = span$\{u_1, u_2\}$.

To be clear, this kind of proof was maybe a bit over the top for ROB 101! ■

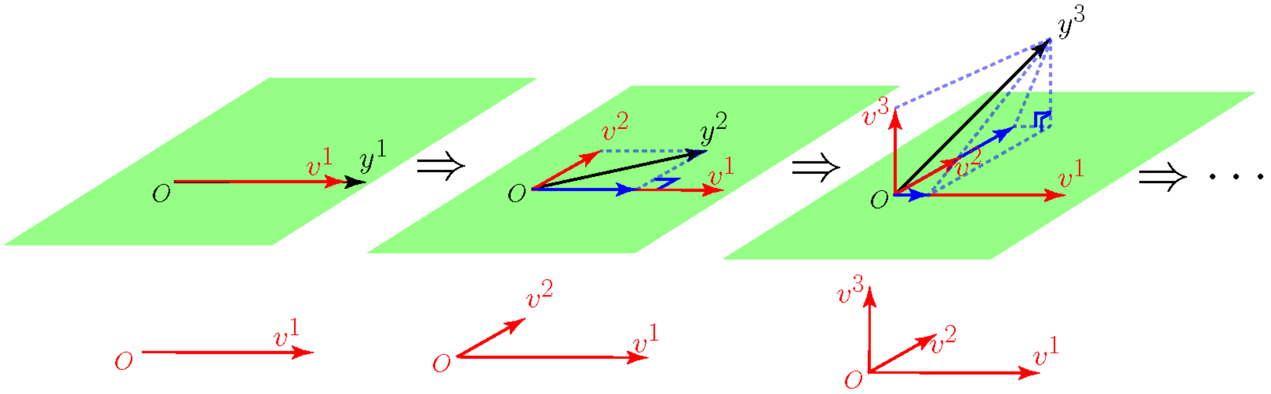


Figure 9.2: Illustration of the Gram Schmidt Process. In the above, $y^i \leftrightarrow u_i$ and $v^i \leftrightarrow v_i$. Image courtesy of Abhishek Venkataraman

9.6.2 Gram-Schmidt Process or the Gram-Schmidt Algorithm

Gram-Schmidt Process

Suppose that that the set of vectors $\{u_1, u_2, \dots, u_m\}$ is linearly independent and you generate a new set of vectors by

$$\begin{aligned} v_1 &= u_1 \\ v_2 &= u_2 - \left(\frac{u_2 \bullet v_1}{v_1 \bullet v_1}\right) v_1 \\ v_3 &= u_3 - \left(\frac{u_3 \bullet v_1}{v_1 \bullet v_1}\right) v_1 - \left(\frac{u_3 \bullet v_2}{v_2 \bullet v_2}\right) v_2 \\ &\vdots \\ v_k &= u_k - \sum_{i=1}^{k-1} \left(\frac{u_k \bullet v_i}{v_i \bullet v_i}\right) v_i \quad \text{(General Step)} \end{aligned} \tag{9.21}$$

Then the set of vectors $\{v_1, v_2, \dots, v_m\}$ is

- orthogonal, meaning, $i \neq j \implies v_i \bullet v_j = 0$
- span preserving, meaning that, for all $1 \le k \le m$,

$$\text{span}\{v_1, v_2, \dots, v_k\} = \text{span}\{u_1, u_2, \dots, u_k\}, \tag{9.22}$$

  and
- linearly independent.

The last item is redundant, but it worth stating so as to emphasize if you start with a set of **basis vectors**, then you will end with a **basis made of orthogonal vectors**. If you then normalize the **orthogonal basis**, you will produce an **orthonormal basis**!

Suggestion: If you do not see the pattern in the steps of the Gram-Schmidt Algorithm, please compare (9.21) to (9.17).

Example 9.20 *You are given that the set below is a basis for* $\mathbb{R}^3$*. Produce from it an orthonormal basis.*

$$\{u_1, u_2, u_3\} = \left\{ \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \right\}$$

Solution:

Step 1 is to apply Gram-Schmidt to produce an orthogonal basis.

$$
\begin{aligned}
v_1 &= u_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \\
v_1 \bullet v_1 &= (v_1)^\top v_1 = 2;
\end{aligned}
$$

$$
\begin{aligned}
v_2 &= u_2 - \frac{u_2 \bullet v_1}{v_1 \bullet v_1} v_1 \\
&= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \underbrace{\begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}}_{3} \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ 3 \end{bmatrix} \\
v_2 \bullet v_2 &= v_2^\top v_2 = \frac{19}{2}
\end{aligned}
$$

$$
\begin{aligned}
v_3 &= u_3 - \frac{u_3 \bullet v_1}{v_1 \bullet v_1} v_1 - \frac{u_3 \bullet v_2}{v_2 \bullet v_2} v_2 \\
&= \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} - \underbrace{\begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}}_{1} \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} - \underbrace{\begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}}_{3\frac{1}{2}} \frac{1}{\frac{19}{2}} \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ 3 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix} - \begin{bmatrix} -\frac{7}{38} \\ \frac{7}{38} \\ \frac{21}{19} \end{bmatrix} = \begin{bmatrix} -\frac{6}{19} \\ \frac{6}{19} \\ -\frac{2}{19} \end{bmatrix}.
\end{aligned}
$$

Collecting the answers, we have

$$
\{v_1, v_2, v_3\} = \left\{ \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ 3 \end{bmatrix}, \begin{bmatrix} -\frac{6}{19} \\ \frac{6}{19} \\ -\frac{2}{19} \end{bmatrix} \right\}
$$

Step 2: Normalize to obtain an orthonormal basis (often useful to do this, but not always required).

$$
\begin{aligned}
\tilde{v}_1 &= \frac{v_1}{\|v_1\|} = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \\
\tilde{v}_2 &= \frac{v_2}{\|v_2\|} = \frac{\sqrt{38}}{38} \begin{bmatrix} -1 \\ 1 \\ 6 \end{bmatrix} \\
\tilde{v}_3 &= \frac{v_3}{\|v_3\|} = \frac{\sqrt{19}}{19} \begin{bmatrix} -3 \\ 3 \\ -1 \end{bmatrix}
\end{aligned}
$$

All of this is quite tedious by hand, while being super fast and fun in Julia!

■

9.7 QR Factorization and Solutions of Linear Equations

QR Factorization

Suppose that A is an $n \times m$ matrix with linearly independent columns. Then there exists an $n \times m$ matrix Q with orthonormal columns and an upper triangular, $m \times m$, invertible matrix R such that $A = Q \cdot R$. Moreover, Q and R are constructed as follows:

- Let $\{u_1, \ldots, u_m\}$ be the columns of A with their order preserved so that

$$A = \begin{bmatrix} u_1 & u_2 & \cdots & u_m \end{bmatrix}$$

- Q is constructed by applying the Gram-Schmidt Process to the columns of A and normalizing their lengths to one,

$$\{u_1, u_2, \ldots, u_m\} \xrightarrow[\text{Process}]{\text{Gram-Schmidt}} \{v_1, v_2, \ldots, v_m\}$$

$$Q := \begin{bmatrix} \frac{v_1}{||v_1||} & \frac{v_2}{||v_2||} & \cdots & \frac{v_m}{||v_m||} \end{bmatrix}$$

- Because $Q^\top Q = I_m$, it follows that $A = Q \cdot R \iff R := Q^\top \cdot A$.
- Recalling that the columns of A are linearly independent, if, and only if $x = 0$ is the unique solution to $Ax = 0$, we have that

$$x = 0 \iff Ax = 0 \iff Q \cdot Rx = 0 \iff Q^\top \cdot Q \cdot Rx = Q^\top \cdot 0 \iff Rx = 0 \iff \det(R) \neq 0,$$

where the last step follows because R is square.

Remark: Because R is upper triangular, everything below its diagonal is zero. Hence, the calculation of R can be sped up by extracting its coefficients from the Gram-Schmidt Process instead of doing the indicated matrix multiplication. We explore this in HW.

Example 9.21 *Compute the QR Factorization of* $A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 3 & 1 \end{bmatrix}$.

Solution: We extract the columns of A and obtain

$$\{u_1, u_2, u_3\} = \left\{ \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \right\}$$

From Example 9.20, we have that

$$\left\{ \tilde{v}_1 = \frac{v_1}{\|v_1\|} = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \tilde{v}_2 = \frac{v_2}{\|v_2\|} = \frac{\sqrt{38}}{38} \begin{bmatrix} -1 \\ 1 \\ 6 \end{bmatrix}, \tilde{v}_3 = \frac{v_3}{\|v_3\|} = \frac{\sqrt{19}}{19} \begin{bmatrix} -3 \\ 3 \\ -1 \end{bmatrix} \right\}$$

and therefore,

$$Q \approx \begin{bmatrix} 0.707107 & -0.162221 & -0.688247 \\ 0.707107 & 0.162221 & 0.688247 \\ 0.000000 & 0.973329 & -0.229416 \end{bmatrix}$$

and

$$R = Q^\top A \approx \begin{bmatrix} 1.41421 & 2.12132 & 0.707107 \\ 0.00000 & 3.08221 & 1.13555 \\ 0.00000 & 0.00000 & 0.458831 \end{bmatrix}.$$

As a numerical check, we also compute how close $Q^\top$ is to being a matrix inverse of Q,

$$Q^\top \cdot Q - I = \begin{bmatrix} -2.22045e-16 & 9.71445e-17 & 1.11022e-16 \\ 9.71445e-17 & 0.00000e-17 & -2.49800e-16 \\ 1.11022e-16 & -2.49800e-16 & 0.00000e-17 \end{bmatrix}.$$

In addition, we check that $\det(Q) = -1.0$. ■

Optional Read: Comprehension Enhancement or a Bridge Too Far: Let's see if we can understand a bit better what we have computed. In Example 9.20, we were given that the set $\{u_1, u_2, u_3\}$, constructed from the columns of A, is linearly independent. The columns of Q, $\{\frac{v_1}{\|v_1\|}, \frac{v_2}{\|v_2\|}, \frac{v_3}{\|v_3\|}\}$, form an orthonormal basis for span$\{u_1, u_2, u_3\}$. Moreover, Gram-Schmidt naturally gives us a triangular relationship among the two sets of linearly independent vectors

$$\begin{aligned} \text{span}\{u_1\} &= \text{span}\{\frac{v_1}{\|v_1\|}\} \\ \text{span}\{u_1, u_2\} &= \text{span}\{\frac{v_1}{\|v_1\|}, \frac{v_2}{\|v_2\|}\} \\ \text{span}\{u_1, u_2, u_3\} &= \text{span}\{\frac{v_1}{\|v_1\|}, \frac{v_2}{\|v_2\|}, \frac{v_3}{\|v_3\|}\}. \end{aligned}$$

The triangular structure of R is a reflection of this triangular relationship between the columns of A and the columns of Q. In particular, we can write u_1 as a linear combination of $\frac{v_1}{\|v_1\|}$, u_2 as a linear combination of $\frac{v_1}{\|v_1\|}$ and $\frac{v_2}{\|v_2\|}$, and finally, u_3 as a linear combination of $\frac{v_1}{\|v_1\|}$, $\frac{v_2}{\|v_2\|}$, and $\frac{v_3}{\|v_3\|}$. If we use r_{ij} to denote the coefficients in the linear combinations, we end up with

$$\begin{aligned} u_1 &= r_{11}\frac{v_1}{\|v_1\|} \\ u_2 &= r_{12}\frac{v_1}{\|v_1\|} + r_{22}\frac{v_2}{\|v_2\|} \\ u_3 &= r_{13}\frac{v_1}{\|v_1\|} + r_{23}\frac{v_2}{\|v_2\|} + r_{33}\frac{v_3}{\|v_3\|}. \end{aligned}$$

Writing this out in matrix form then gives $A = Q \cdot R$,

$$\begin{aligned} \underbrace{\begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}}_{A} &= \underbrace{\begin{bmatrix} r_{11}\frac{v_1}{\|v_1\|} & r_{12}\frac{v_1}{\|v_1\|} + r_{22}\frac{v_2}{\|v_2\|} & r_{13}\frac{v_1}{\|v_1\|} + r_{23}\frac{v_2}{\|v_2\|} + r_{33}\frac{v_3}{\|v_3\|} \end{bmatrix}}_{Q \cdot R} \\ &= \underbrace{\begin{bmatrix} \frac{v_1}{\|v_1\|} & \frac{v_2}{\|v_2\|} & \frac{v_3}{\|v_3\|} \end{bmatrix}}_{Q} \cdot \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{bmatrix}}_{R} \end{aligned}$$

In case that last step was too much, too fast. We break it down into Q multiplying the various columns of R,

$$\underbrace{\begin{bmatrix} \frac{v_1}{\|v_1\|} & \frac{v_2}{\|v_2\|} & \frac{v_3}{\|v_3\|} \end{bmatrix}}_{Q} \cdot \begin{bmatrix} r_{11} \\ 0 \\ 0 \end{bmatrix} = r_{11}\frac{v_1}{\|v_1\|}$$

$$\underbrace{\begin{bmatrix} \frac{v_1}{\|v_1\|} & \frac{v_2}{\|v_2\|} & \frac{v_3}{\|v_3\|} \end{bmatrix}}_{Q} \cdot \begin{bmatrix} r_{12} \\ r_{22} \\ 0 \end{bmatrix} = r_{12}\frac{v_1}{\|v_1\|} + r_{22}\frac{v_2}{\|v_2\|}$$

$$\underbrace{\begin{bmatrix} \frac{v_1}{\|v_1\|} & \frac{v_2}{\|v_2\|} & \frac{v_3}{\|v_3\|} \end{bmatrix}}_{Q} \cdot \begin{bmatrix} r_{13} \\ r_{23} \\ r_{33} \end{bmatrix} = r_{13}\frac{v_1}{\|v_1\|} + r_{23}\frac{v_2}{\|v_2\|} + r_{33}\frac{v_3}{\|v_3\|}.$$

Solutions of Linear Equations via the QR Factorization

Suppose that A is $n \times n$ and its columns are linearly independent. Let $A = Q \cdot R$ be its QR Factorization. Then

$$(Ax = b) \iff (Q \cdot Rx = b) \iff (Rx = Q^\top b). \tag{9.23}$$

Hence, whenever $\det(A) \neq 0$, the suggested "pipeline" for solving $Ax = b$ is

- factor $A =: Q \cdot R$,
- compute $\overline{b} := Q^\top b$, and then
- solve $Rx = \overline{b}$ via back substitution.

Example 9.22 *Use the suggested pipeline to solve the system of linear equations*

$$\underbrace{\left[\begin{array}{ccc} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 3 & 1 \end{array}\right]}_{A} x = \underbrace{\left[\begin{array}{c} 1 \\ 4 \\ 7 \end{array}\right]}_{b}.$$

Solution: From Example 9.21,

$$\underbrace{\left[\begin{array}{ccc} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 3 & 1 \end{array}\right]}_{A} = \underbrace{\left[\begin{array}{ccc} 0.707107 & -0.162221 & -0.688247 \\ 0.707107 & 0.162221 & 0.688247 \\ 0.000000 & 0.973329 & -0.229416 \end{array}\right]}_{Q} \cdot \underbrace{\left[\begin{array}{ccc} 1.41421 & 2.12132 & 0.707107 \\ 0.00000 & 3.08221 & 1.13555 \\ 0.00000 & 0.00000 & 0.458831 \end{array}\right]}_{R}.$$

We form

$$\overline{b} := Q^\top b = \left[\begin{array}{c} 3.53553 \\ 7.29996 \\ 0.45883 \end{array}\right]$$

and then use back substitution to solve

$$\underbrace{\left[\begin{array}{ccc} 1.41421 & 2.12132 & 0.707107 \\ 0.00000 & 3.08221 & 1.13555 \\ 0.00000 & 0.00000 & 0.458831 \end{array}\right]}_{R} x = \underbrace{\left[\begin{array}{c} 3.535534 \\ 7.299964 \\ 0.458831 \end{array}\right]}_{\overline{b}},$$

which yields

$$x = \left[\begin{array}{c} -1 \\ 2 \\ 1 \end{array}\right].$$

■

Least Squares via the QR Factorization

Suppose that A is $n \times m$ and its columns are linearly independent (tall matrix). Let $A = Q \cdot R$ be its QR Factorization. Then $A^\top A = R^\top \cdot Q^\top \cdot Q \cdot R = R^\top \cdot R$ and thus

$$(A^\top \cdot Ax = A^\top b) \iff (R^\top \cdot Rx = R^\top \cdot Q^\top b) \iff (Rx = Q^\top b), \tag{9.24}$$

where we have used the fact that R is invertible. Hence, whenever the columns of A are linearly independent, the suggested "pipeline" for computing a least squared error solution to $Ax = b$ is

- factor $A =: Q \cdot R$,
- compute $\overline{b} := Q^\top b$, and then
- solve $Rx = \overline{b}$ via back substitution.

Yes! The two pipelines are identical!! Your surprise will be tempered when you go back to our discussion of least squared error solutions of linear equations where we noted that if A is square and its determinant is non-zero, then

$$(Ax = b) \iff (A^\top \cdot Ax = A^\top b).$$

The only difference in the two cases is that when A is square, Q is an orthogonal matrix whereas, when A is a tall matrix, then Q is an orthonormal matrix. Yes, it's a subtle difference! Is it an important difference? Not really, as long as you only form $Q^\top \cdot Q$ and you avoid $Q \cdot Q^\top$.

Example 9.23 *We rework Example 8.1 from Chapter 8.2, where we seek a least squared error solution to the system of linear equations*

$$\underbrace{\begin{bmatrix} 1.0 & 1.0 \\ 2.0 & 1.0 \\ 4.0 & 1.0 \\ 5.0 & 1.0 \\ 7.0 & 1.0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 4 \\ 8 \\ 10 \\ 12 \\ 18 \end{bmatrix}}_{b}. \tag{9.25}$$

Solution: Since the columns of A are linearly independent, we compute the QR factorization of A and obtain

$$Q = \begin{bmatrix} 0.102598 & 0.730297 \\ 0.205196 & 0.547723 \\ 0.410391 & 0.182574 \\ 0.512989 & 0.000000 \\ 0.718185 & -0.365148 \end{bmatrix}, R = \begin{bmatrix} 9.74679 & 1.94936 \\ 0.0 & 1.09545 \end{bmatrix}, \text{ and } Q^\top \cdot b = \begin{bmatrix} 25.23906 \\ 2.556038 \end{bmatrix}.$$

We then use back substitution to solve

$$\begin{bmatrix} 9.74679 & 1.94936 \\ 0.0 & 1.09545 \end{bmatrix} x = \begin{bmatrix} 25.23906 \\ 2.556038 \end{bmatrix}$$

which yields

$$x^* = \begin{bmatrix} 2.1228 \\ 2.3333 \end{bmatrix}.$$

■

For small made-up problems like this one, there is no real numerical advantage to using a sophisticated solution like our "suggested pipeline". In real engineering, it makes a huge difference. Prof. Grizzle's and Ghaffari's students use the suggested pipeline when working with Cassie Blue.

9.8 Optional Read: Modified Gram-Schmidt Algorithm

The classical Gram-Schmidt Process is straightforward to understand, which is why it is taught in courses. Unfortunately, it behaves poorly under the round-off error that occurs in digital computations! Here is a standard example:

$$u_1 = \begin{bmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{bmatrix}, u_2 = \begin{bmatrix} 1 \\ 0 \\ \varepsilon \\ 0 \end{bmatrix}, u_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \varepsilon \end{bmatrix}, \varepsilon > 0$$

Let $\{e_1, e_2, e_3, e_4\}$ be the standard basis vectors corresponding to the columns of the 4×4 identity matrix. We note that

$$u_2 = u_1 + \varepsilon(e_3 - e_2)$$
$$u_3 = u_2 + \varepsilon(e_4 - e_3)$$

and thus, for $\epsilon \neq 0$,

$$\text{span}\{u_1, u_2\} = \text{span}\{u_1, (e_3 - e_2)\}$$
$$\text{span}\{u_1, u_2, u_3\} = \text{span}\{u_1, (e_3 - e_2), (e_4 - e_3)\}$$

Hence, Gram-Schmidt applied to $\{u_1, u_2, u_3\}$ and $\{u_1, (e_3 - e_2), (e_4 - e_3)\}$ should "theoretically" produce the same orthonormal vectors. To check this, we go to Julia, and for $\varepsilon = 0.1$, we do indeed get the same results. You can verify this yourself. However, with $\varepsilon = 10^{-8}$,

$$Q_1 = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & -0.7071 & -0.7071 \\ 0.0000 & 0.7071 & 0.0000 \\ 0.0000 & 0.0000 & 0.7071 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & -0.7071 & -0.4082 \\ 0.0000 & 0.7071 & -0.4082 \\ 0.0000 & 0.0000 & 0.8165 \end{bmatrix}$$

where

$$Q_1 = \begin{bmatrix} \frac{v_1}{\|v_1\|} & \frac{v_2}{\|v_2\|} & \frac{v_3}{\|v_3\|} \end{bmatrix}$$

has been computed with Classical-Gram-Schmidt for $\{u_1, u_2, u_3\}$ while

$$Q_2 = \begin{bmatrix} \frac{v_1}{\|v_1\|} & \frac{v_2}{\|v_2\|} & \frac{v_3}{\|v_3\|} \end{bmatrix}$$

has been computed with Classical-Gram-Schmidt for $\{u_1, (e_3 - e_2), (e_4 - e_3)\}$. Hence we do NOT obtain the same result!

Modified Gram-Schmidt has better Numerical Performance

```
for $k = 1 : n$
    $v_k = u_k$   #copy over the vectors
end

for $k = 1 : n$
    $v_k = \frac{v_k}{\|v_k\|}$
    for $j = (k+1) : n$
        $v_j = v_j - (v_j \bullet v_k) v_k$   #Makes $v_j$ orthogonal to $v_k$
    end
end
```

At **Step 1**, v_1 is normalized to length one, and then $v_2, \ldots, v_n$ are redefined to be orthogonal to v_1. At **Step 2:** v_2 is normalized to length one, and then $v_3, \ldots, v_n$ are redefined to be orthogonal to v_2. We note that they were already orthogonal to v_1. At **Step** k**:** v_k is normalized to length one, and then $v_{k+1}, \ldots, v_n$ are redefined to be orthogonal to v_k. We note that they were already orthogonal to $v_1, \ldots, v_{k-1}$.

Hence, if Modified Gram-Schmidt is so great, when applied to $\{u_1, u_2, u_3\}$ and $\{u_1, (e_3 - e_2), (e_4 - e_3)\}$, it should produce the same orthonormal vectors and it does! To check this, we go to Julia for $\varepsilon = 10^{-8}$ and obtain

$$Q_1 = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & -0.7071 & -0.7071 \\ 0.0000 & 0.7071 & 0.0000 \\ 0.0000 & 0.0000 & 0.7071 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & -0.7071 & -0.7071 \\ 0.0000 & 0.7071 & 0.0000 \\ 0.0000 & 0.0000 & 0.7071 \end{bmatrix}$$

where Q_1 and Q_2 are defined above. When one is equipped with the right Algorithm, the world is truly a marvelous place.

9.9 What to do When Ax = b has an Infinite Number of Solutions?

We consider $Ax = b$ and recall what we know about its solutions:

- $b \in$ col span$\{A\} \iff$ a solution exists;
- the solution is unique if, and only if, the columns of A are linearly independent; and thus
- if there exists one solution and the columns of A are linearly dependent, then there exist an infinite number of solutions.

Underdetermined Equations

The columns of A will be linearly dependent when $Ax = b$ has fewer equations than unknowns. In other words, A is $n \times m$ and $m > n$; we've been calling these wide matrices: more columns than rows. When dealing with an equation $Ax = b$ with fewer equations than unknowns, one says that it is **underdetermined**. Why? Because, to determine x uniquely, at a minimum, we need as many equations as unknowns.

Is there a difference between being underdetermined and having an infinite number of solutions? Yes. It's possible to be underdetermined and have no solution at all when $b \not\in$ col span$\{A\}$. If the rows of A are linearly independent, then

$$Ax = b \text{ is underdetermined} \iff Ax = b \text{ has an infinite number of solutions.}$$

The rows of A being linearly independent is equivalent to the columns of $A^\top$ being linearly independent.

When $Ax = b$ has an infinite number of solutions, is there a way that we can make one of them appear to be more interesting, more special, or just flat out "better" than all the other solutions? Is there a property that we could associate with each solution and optimize our choice of solution with respect to that property? The most common approach is to choose the solution with minimum norm!

Minimum Norm Solution to Underdetermined Equations

Consider an underdetermined system of linear equations $Ax = b$. If the rows of A are linearly independent (equivalently, the columns of $A^\top$ are linearly independent), then

$$x^* = \min_{Ax=b} ||x|| \iff x^* = A^\top \cdot (A \cdot A^\top)^{-1} b \iff x^* = A^\top \alpha \text{ and } A \cdot A^\top \alpha = b. \tag{9.26}$$

We recommend that the minimum norm solution x^* be computed with the right-hand side of (9.26) so that the matrix inverse is avoided, but for small problems, the middle answer is fine.

Suppose we do the QR Factorization of $A^\top$ instead of A itself, so that

$$A^\top = Q \cdot R.$$

Because the columns of $A^\top$ are linearly independent, R is square and invertible. It follows that $A = R^\top \cdot Q^\top$ and $A \cdot A^\top = R^\top \cdot R$ because $Q^\top \cdot Q = I$. Using these facts, (9.26) can be rewritten as

$$x^* = \min_{Ax=b} ||x|| \iff x^* = Q \cdot \left(R^\top\right)^{-1} b \iff x^* = Q\beta \text{ and } R^\top \beta = b. \tag{9.27}$$

We note that $R^\top$ is lower triangular, and thus $R^\top \beta = b$ can be solved via forward substitution. Hence, our suggested "pipeline" for underdetermined problems $Ax = b$ is

- Check that the columns of $A^\top$ are linearly independent and compute $A^\top = Q \cdot R$.
- Solve $R^\top \beta = b$ by forward substitution.
- $x^* = Q\beta$.

Remark: In case you are curious, β in (9.27) is related to α in (9.26) by $\beta = R\alpha$. The two x^* are the same!

Example 9.24 *Use the suggested pipeline to determine a minimum norm solution to the system of underdetermined equations*

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix}}_{A} x = \underbrace{\begin{bmatrix} 1 \\ 4 \end{bmatrix}}_{b}.$$

Solution: The columns of $A^\top$ are linearly independent and we compute the QR Factorization to be

$$A^\top = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 0 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.707107 & -0.408248 \\ 0.707107 & 0.408248 \\ 0.000000 & 0.816497 \end{bmatrix}}_{Q} \cdot \underbrace{\begin{bmatrix} 1.41421 & 2.12132 \\ 0.00000 & 1.22474 \end{bmatrix}}_{R}.$$

We solve $R^\top \beta = b$ and obtain

$$\beta = \begin{bmatrix} 0.70711 \\ 2.04124 \end{bmatrix},$$

and then $x^* = Q\beta$ to arrive at the final answer

$$x^* = \begin{bmatrix} -0.33333 \\ 1.33333 \\ 1.66667 \end{bmatrix}.$$

To verify that x^* is indeed a solution, we substitute it into $Ax - b$ and obtain

$$Ax^* - b = \begin{bmatrix} 0.000000 \\ -4.441e\text{-}16 \end{bmatrix},$$

which looks like a pretty good solution! ■

Is x^* in Example 9.24 the solution of smallest norm? Of course! Don't you trust us? To which you respond, "of course not!"

Let's see about that. What are other solutions? They all look like $x^* + \bar{x}$, where $A\bar{x} = 0$, because then

$$A(x^* + \bar{x}) = Ax^* + A\bar{x} = b + 0 = b.$$

We compute that all solutions to $Ax = 0$ have the form

$$\bar{x} = \gamma \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix},$$

and therefore, all solutions to $Ax = b$ have the form

$$x_{\text{sol}} = x^* + \bar{x}.$$

The next thing we can check is that for all $\gamma \in \mathbb{R}$, $x^* \perp \bar{x}$, and hence, by the Pythagorean Theorem, we have that

$$||x_{\text{sol}}||^2 = ||x^* + \bar{x}||^2 = ||x^*||^2 + ||\bar{x}||^2 = ||x^*||^2 + 3\gamma^2.$$

It follows that

$$\min_{\gamma} ||x_{\text{sol}}||^2 = \min_{\gamma} \left(||x^*||^2 + 3\gamma^2\right) = ||x^*||^2 + \min_{\gamma} \left(3\gamma^2\right)$$

and thus the minimum occurs for $\gamma = 0$. Hence, x^* is indeed the minimum norm solution!

Optional Read: The Pythagorean Theorem is a powerful ally when one seeks to establish minimum norm properties. We use it to show that (9.26) has the claimed minimum norm property among all solutions of $Ax = b$. All of the ideas are actually present in our analysis of Example 9.24.

The proposed minimum norm solution to $Ax = b$ has the form $x^* = A^\top \alpha$, which is a linear combination of the columns of $A^\top$. Indeed, for A an $n \times m$ matrix we write

$$A = \begin{bmatrix} a_1^{\text{row}} \\ \vdots \\ a_n^{\text{row}} \end{bmatrix} \text{ and } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \text{ so that } A^\top \alpha = \begin{bmatrix} \left(a_1^{\text{row}}\right)^\top & \cdots & \left(a_n^{\text{row}}\right)^\top \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} = \alpha_1 \left(a_1^{\text{row}}\right)^\top + \cdots + \alpha_n \left(a_n^{\text{row}}\right)^\top.$$

We next note that $A\bar{x} = 0$ if, and only if, for all $1 \leq i \leq n$, $a_i^{\text{row}}\bar{x} = 0$, which is equivalent to $\left(a_i^{\text{row}}\right)^\top \perp \bar{x}$. Hence, we have that

$$x^* \perp \bar{x}$$

A general solution to $Ax = b$ can be written as $x_{\text{sol}} = x^* + \bar{x}$, where $\bar{x}$ is any solution to $Ax = 0$. Applying the Pythagorean Theorem we have

$$||x_{\text{sol}}||^2 = ||x^* + \bar{x}||^2 = ||x^*||^2 + ||\bar{x}||^2.$$

Because $||x^*||^2 + ||\bar{x}||^2$ is smallest for $\bar{x} = 0$, it follows that

$$\min_{\bar{x}} ||x_{\text{sol}}||^2 = \min_{\bar{x}} ||x^*||^2 + ||\bar{x}||^2 = ||x^*||^2,$$

which shows that (9.26) really is the minimum norm solution.

9.10 (Optional Read): Why the Matrix Properties on Rank and Nullity are True

We conclude the Chapter by proving the key properties of rank and nullity of a matrix. For an $n \times m$ matrix A, we recall that

Def. $\text{rank}(A) := \dim \text{col span}\{A\} \subset \mathbb{R}^n$.

Def. $\text{nullity}(A) := \dim \text{null}(A) \subset \mathbb{R}^m$.

Useful Properties of Rank and Nullity

For an $n \times m$ matrix A, the following are true:

(a) $\text{rank}(A) + \text{nullity}(A) = m$, the number of columns in A.

(b) $\text{nullity}(A^\top \cdot A) = \text{nullity}(A)$.

(c) $\text{rank}(A^\top \cdot A) = \text{rank}(A)$.

(d) $\text{rank}(A \cdot A^\top) = \text{rank}(A^\top)$.

(e) For any $m \times k$ matrix B, $\text{rank}(A \cdot B) \leq \text{rank}(A)$.

(f) $\text{rank}(A^\top) = \text{rank}(A)$.

(g) $\text{rank}(A^\top \cdot A) = \text{rank}(A \cdot A^\top)$.

(h) $\text{nullity}(A^\top) + m = \text{nullity}(A) + n$.

Proofs:

(a) If $\text{rank}(A) = m$, then the columns of A are linearly independent, which implies that $x = 0$ is the unique solution of $Ax = 0$. Hence, $\text{null}(A) = \{0_{m \times 1}\}$. It follows that the $\text{nullity}(A) = 0$ and therefore (9.15) holds. If $\text{rank}(A) = 0$, then A must be the zero matrix, and hence $\text{null}(A) = \mathbb{R}^m$, and we once again verify (9.15).

Hence, we define $\rho := \text{rank}(A)$ and suppose that $0 < \rho < m$. Our goal is to determine the dimension of the null space of A.

From the equivalence of the range and column span of a matrix, we know that A has ρ linearly independent columns and $m-\rho$ columns that are dependent on them. Because permuting the order of the columns of A does not change its rank, we assume without loss of generality that the first ρ columns of A are linearly independent and for $\rho+1 \leq j \leq m$, we have

$$a_j^{\rm col} \in \{a_1^{\rm col}, a_2^{\rm col}, \ldots, a_\rho^{\rm col}\},$$

which means there exist coefficients $\beta_{ij} \in \mathbb{R}$ such that

$$a_j^{\rm col} = \beta_{1j}\, a_1^{\rm col} + \beta_{2j}\, a_2^{\rm col} + \cdots + \beta_{\rho j}\, a_\rho^{\rm col}. \tag{9.28}$$

Based on the above, we partition the columns of A as

$$A = [A_1 \;\; A_2]$$

where A_1 is given by the first ρ independent columns of A and A_2 consists of the remaining dependent columns. From (9.28), it follows that $A_2 = A_1 B$, where

$$B = \left[\begin{array}{ccc} \beta_{1(\rho+1)} & \cdots & \beta_{1m} \\ \vdots & \vdots & \vdots \\ \beta_{\rho(\rho+1)} & \cdots & \beta_{\rho m} \end{array}\right].$$

Therefore, we have

$$A = [A_1 \;\; A_1 B] = A_1 \left[I_\rho \;\; B\right], \tag{9.29}$$

where the columns of A_1 are linearly independent. From the same reasoning in Chapter 7.4.5 that we employed in our Pro Tip, we have

$$Ax = 0 \iff A_1 \left[I_\rho \;\; B\right] x = 0 \iff A_1^\top A_1 \left[I_\rho \;\; B\right] x = 0 \iff \left[I_\rho \;\; B\right] x = 0,$$

where the last equality is because $A_1^\top A_1$ is invertible. Based on the above, we partition x as

$$x = \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right],$$

where $x_1 \in \mathbb{R}^\rho$ and $x_2 \in \mathbb{R}^{m-\rho}$, and obtain

$$x = 0 \iff x_1 + Bx_2 = 0.$$

From here, we can apply the result in Example 9.12 and deduce that the dimension of the null space of A is $m-\rho$, which completes the proof.

(b) Our Pro Tip in Chapter 7.4.5 showed that $Ax = 0 \iff A^\top A x = 0$. Hence,

$$\text{null}(A^\top A) = \text{null}(A),$$

and therefore their nullities agree.

(c) Combining (a) and (b) proves (c).

(d) True by starting with $A^\top$ in place of A and then recognizing that $\left(A^\top\right)^\top = A$.

(e) From the sum of columns times row form of matrix multiplication, we have that the columns of $A \cdot B$ are a linear combination of the columns of A. Hence, col span$\{A \cdot B\} \subset$ col span$\{A\}$, which implies that rank$(A \cdot B) \leq$ rank(A).

(f) Combining (e) with (c) we have

$$\text{rank}(A^\top) \leq \text{rank}(A^\top \cdot A) = \text{rank}(A)$$

and then combining (e) with (d) we have

$$\text{rank}(A) \leq \text{rank}(A \cdot A^\top) = \text{rank}(A^\top).$$

Hence, rank$(A^\top) \leq$ rank$(A) \leq$ rank$(A^\top)$, and therefore rank$(A^\top) =$ rank(A).

(g) Combining (f) with (c) and (d) proves (g).

(h) Combining (a) with (f) implies (h).

■

9.11 Looking Ahead

So far in ROB 101, we've only looked at linear problems. However, it turns out that techniques from Linear Algebra, namely vectors and matrices, can be very useful for some problems involving nonlinear functions. Hence, we will disengage from pure Linear Algebra and explore two very interesting problems:

1. Root finding: this is the problem of finding $x \in \mathbb{R}^n$ such that $f(x) = 0$. A special case would be $f(x) = Ax - b$, in which case, "root finding" is the same as solving $Ax = b$, a problem we know a lot about! What we will do is assume a result from Calculus[2] which says that near a root of $f(x) = 0$, we can approximate the nonlinear function $f(x)$ by an affine function, $Ax - b$. Solving $Ax - b = 0$ will give us an approximation of a solution to $f(x) = 0$. We can then re-approximate the function $f(x)$ near our current estimate of the root and attempt to improve the quality of the solution. Putting this in a for-loop gives us an algorithm.

2. Minimizing a real-valued function of x: this is the problem of finding x^* such that

$$x^* = \arg\min_{x \in \mathbb{R}^n} c(x),$$

where $c : \mathbb{R}^n \to [0, \infty)$. A special case is

$$c(x) := ||Ax - b||^2,$$

our least-squared error solution to $Ax = b$. Modern engineering is broadly based on "optimization", the process of maximizing the efficiency of some process or minimizing the energy consumed in making a product. In Robotics, we formulate "perception" problems as one of minimizing the error in the estimated position of objects that we "perceive" with a camera or LiDAR, for example.

You will find both of these tools broadly applicable throughout your engineering career, and of course, in your engineering subjects at UofM. In Project 3, we will see how to use optimization to do "balance control" of a Segway! Your project will be a simplified version of an algorithm that could be used on Cassie Blue, the amazing bipedal robot at Michigan.

[2] We will teach you the result without proof. For theory, you can see Calculus I.

Chapter 10

Changing Gears: Solutions of Nonlinear Equations

Learning Objectives

- Extend our horizons from linear equations to nonlinear equations.
- Appreciate the power of using algorithms to iteratively construct approximate solutions to a problem.
- Accomplish all of this without assuming a background in Calculus.

Outcomes

- Learn that a root is a solution of an equation of the form $f(x) = 0$.
- Learn two methods for finding roots of real-valued functions of a real variable, that is for $f : \mathbb{R} \to \mathbb{R}$, namely the Bisection Method and Newton's Method
- Become comfortable with the notion of a "local slope" of a function at a point and how to compute it numerically.
- Linear approximations of nonlinear functions.
- Extensions of these ideas to vector-valued functions of several variables, that is $f : \mathbb{R}^m \to \mathbb{R}^n$, with key notions being the gradient and Jacobian of a function and their use in the Newton-Raphson Algorithm.

10.1 Motivation and Simple Ideas

The focus of ROB 101 has been systems on linear equations. Long before you came to ROB 101, however, you had studied some Algebra and solved a few nonlinear equations. Our goal here is to develop numerical methods for finding solutions to **systems of nonlinear equations**.

We will limit our notion of a solution to the set of real numbers or real vectors. Limiting our search for solutions to the real numbers has consequences. We already know that

$$x^2 + 1 = 0,$$

for example, has no real solutions because its discriminant is $\Delta = b^2 - 4ac = -4 < 0$. Nevertheless, many interesting problems in Engineering and Science can be formulated and solved in terms of "real solutions" to systems of equations[1].

Root of an Equation

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function. Then $f(x) = 0$ defines an equation. A solution to the equation is also called a **root**[a]; that is $x^* \in \mathbb{R}^n$ is a root of $f(x) = 0$ if

$$f(x^*) = 0. \tag{10.1}$$

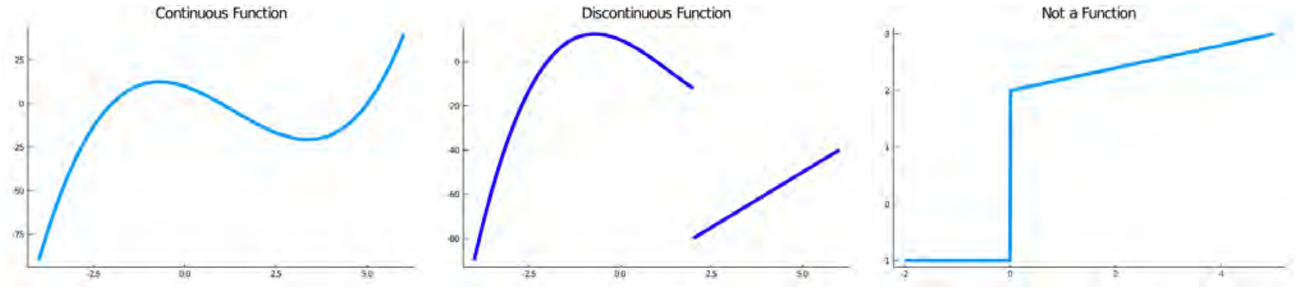
Just as with quadratic equations, it is possible that (10.1) has multiple real solutions or no real solutions.

You may wonder if we could seek solutions to $f(x) = \pi$, for example, and if we were to do that, would we still call them roots? Technically, the answer is no. The term root is reserved for solutions to $f(x) = 0$. However, if we define a new function, $\bar{f}(x) := f(x) - \pi$, then

$$\bar{f}(x^*) = 0 \iff f(x^*) - \pi = 0 \iff f(x^*) = \pi,$$

and x^* is a root of our new function $\bar{f}(x)$. If this seems like we are splitting hairs, yeah, it's hard to disagree with that sentiment!

[a] In the 9th century, Arab writers usually called one of the solutions to a polynomial equation **jadhr** (âĂIJrootâĂİ), and their medieval European translators used the Latin word **radix**; see https://www.britannica.com/science/root-mathematics.



(a) (b) (c)

Figure 10.1: Examples of a continuous function, a discontinuous function, and a graph that is not a function. Yes, in (c), the point $x = 0$ is mapped to the interval $[-1, 2]$. To be a function, each point in the domain can only map to a single point in the range.

We will say very informally that a function $f : \mathbb{R} \to \mathbb{R}$ is **continuous** if you can draw the graph of $y = f(x)$ on a sheet of paper without lifting your pencil (from the paper)! Figure 10.1-(a) clearly passes this test while Fig. 10.1-(b) does not. Figure 10.1-(c) "seems" to pass the "without lifting your pencil test," but the graph does not represent a function! Recall that a function is a rule that associates to each element of the domain, a single value in the range, meaning, that for a given $x \in \mathbb{R}$, there can be only one value of $y \in \mathbb{R}$ such that $y = f(x)$. In Fig. 10.1-(c), for $x = 0$, we have $f(x) = y$ for all $y \in [-1, 2]$, which makes it not a function. What about the part of the graph where the "non-function" is constant, does that also make it not a function? No, it is fine for the single value of $y = -1.0$ to be associated with many values of x; it's the other way around that is a problem. Functions map points to points and not points to non-trivial sets, such as the interval $[-1, 2]$.

[1] In addition, we should not overlook the fact that as a first introduction to the subject of numerical methods for solving systems of nonlinear equations, working with real numbers and vectors is a great place to start!

In Calculus, you will encounter a formal definition, which goes something like this: $f : \mathbb{R} \to \mathbb{R}$ is **continuous at a point $\mathbf{x_0} \in \mathbb{R}$** if for every $\epsilon > 0$, there exists a $\delta > 0$ such that,

$$|x - x_0| < \delta \implies |f(x) - f(x_0)| < \epsilon.$$

And then one says that f is **continuous** if it is continuous at x_0 for all x_0 in its domain of definition! For our purposes, the pencil test is good enough.

10.2 Bisection

We begin with the most straightforward and intuitive method for finding roots of scalar equations

$$f(x) = 0,$$

where $f : \mathbb{R} \to \mathbb{R}$, that is, f maps real numbers to real numbers. The method is based on the following fact, which, once again, is something you will encounter in Calculus. In ROB 101, we are giving you a reason to pay attention to the result when it is presented in Calculus!

Intermediate Value Theorem

Assume that f is a continuous real valued function and you know two real numbers $a < b$ such that $f(a) \cdot f(b) < 0$. Then there exists a real number c such that

- $a < c < b$ (c is between a and b), and
- $f(c) = 0$ (c is a root).

The values a and b are said to **bracket the root**, c.

Remarkably, this leads to a "method" for approximating roots with arbitrary accuracy! Here is the basic idea.

Bisection Algorithm Pseudo Code

```
• Initialize: define $a < b$ such that $f(a) \cdot f(b) < 0$
• Start: compute $c := \frac{a+b}{2}$, the midpoint of the interval $[a, b]$.
• Two things are possible:
    – $f(c) = 0$, in which case, we are done. $x^* = c$.
    – $f(c) \neq 0$, in which case, either $f(c) \cdot f(a) < 0$ or $f(c) \cdot f(b) < 0$. (We'll leave to you the task of "proving" that
      at least one of these statements must be true and they cannot both be true.)
• If $f(c) \cdot f(a) < 0$
    $b = c$;  # $b$ updates while $a$ stays the same
  Else
    $a = c$;  # $a$ updates while $b$ stays the same
  End If
• Loop Back to Start. ( Wash, rinse, and repeat! )
```

Now, as written, the above is not an effective algorithm because it may never terminate, meaning it could loop for ever and ever. For example, suppose you wanted to solve $x^2 - 2 = 0$. You know that answer is $x^* = \sqrt{2}$, an irrational number. You might think to start with the initial guesses being $a = 0$ and $b = 2$, because then $f(0) \cdot f(2) = (-2) \cdot (2) = -4 < 0$. However, $c = \frac{a+b}{2} = 1$, a rational number, and because $f(c) \cdot f(b) < 0$, your next step is $a = 1$ and $b = 2$. In fact, you can check that $[a \;\; c \;\; b]$ evolves like this

a	c	b
0.0	1.0	2.0
1.0	1.5	2.0
1.0	1.25	1.5
1.25	1.375	1.5
1.375	1.4375	1.5
1.375	1.40625	1.4375
1.40625	1.421875	1.4375
1.40625	1.4140625	1.421875
1.4140625	1.41796875	1.421875
1.4140625	1.416015625	1.41796875
1.4140625	1.4150390625	1.416015625

the point being that c will always be a rational number and hence it will never be true that $f(c) = 0$. Of course, we can get very close to zero and we need to define what does close enough mean!

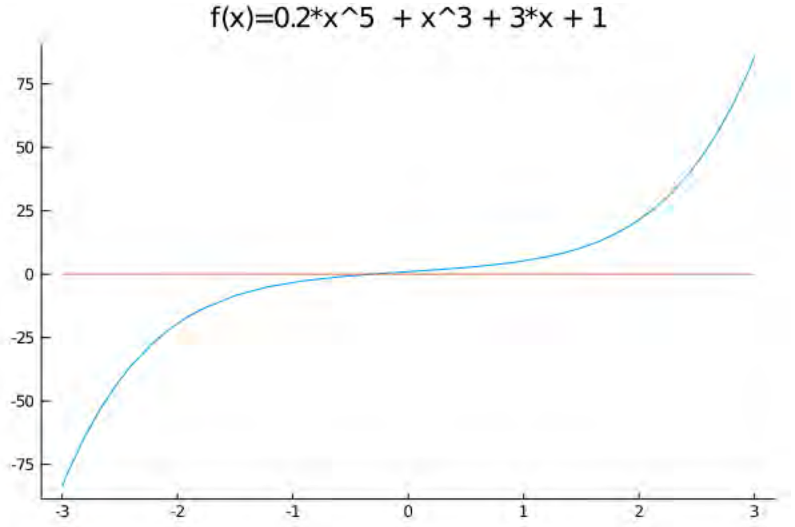


Figure 10.2: Plot of $y = 0.2x^5 + x^3 + 3x + 1$. There does not exist any formula that provides the roots of general quintic polynomials (no quintic equation)! If we want to find a root, we are forced to use numerical methods.

Example 10.1 *Figure 10.2 presents a graph of the function* $f(x) = 0.2x^5 + x^3 + 3x + 1$. *Find a root of the function, that is, find a solution of*

$$0.2x^5 + x^3 + 3x + 1 = 0.$$

Because formulas for the roots of quintic polynomials do not exist[2]*, you must use a numerical method.*

Solution: We apply the bisection method. Based on Fig. 10.2, we'll bracket the root with $a = -2$ and $b = 1$. We run the algorithm and obtain the following data

$\mathbf{a}$	$\mathbf{c = \frac{a+b}{2}}$	$\mathbf{b}$	$\mathbf{sign\ (f(a) \cdot f(c))}$	$\mathbf{f(c)}$	
-2.0	-0.5	1.0	$+1.0$	-0.6312500000000001	
-0.5	0.25	1.0	-1.0	1.7658203125	
-0.5	-0.125	0.25	-1.0	0.623040771484375	
-0.5	-0.3125	-0.125	-1.0	0.031386375427246094	(10.2)
-0.5	-0.40625	-0.3125	$+1.0$	-0.28801019787788396	
-0.40625	-0.359375	-0.3125	$+1.0$	-0.12573728393763295	
-0.359375	-0.3359375	-0.3125	$+1.0$	-0.046580093697411895	
-0.3359375	-0.32421875	-0.3125	$+1.0$	-0.007453918341161714	

[2] In fact, N. Abel proved in 1826 that formulas for roots do not exist for families of polynomials of degree higher than four. In 1835, while still in his teens, E. Galois was able to determine a necessary and sufficient condition for any given polynomial to be solvable by "roots", thereby resolving a problem that had been open for 350 years. The story goes that he wrote down his solution days before being killed in a duel.

Figure 10.3 shows the evolution of the bracketing points a and b as well as the midpoint c for the first four steps of the algorithm, while Fig. 10.4 zooms in to show more detail. From (10.2), the logic of the algorithm can be pinned down.

Logic of the Algorithm in Detail

- In Step 1, we compute $c = \frac{a+b}{2}$ and $f(a) \cdot f(c) > 0$. Recall that at each step, the Intermediate Value Theorem says we need $f(a) \cdot f(b) < 0$ to ensure that there exists a $c \in (a, b)$ such that $f(c) = 0$. Because $f(a) \cdot f(c) > 0$, we know without checking that $f(b) \cdot f(c) < 0$, and therefore, in the next step, we update $a = c$ and leave b unchanged so that $f(a) \cdot f(b) < 0$. Similar logic applies in the followign steps.
- As noted, in Step 2, we have $a_{\text{new}} = c = -0.5$, while $b = 1.0$ is unchanged. This gives $c = \frac{a+b}{2} = 0.25$ and $f(a) \cdot f(c) < 0$.
- Hence, in Step 3, we have $b_{\text{new}} = c = -0.5$, while $a = 0.25$ is unchanged. This gives $c = \frac{a+b}{2} = -0.125$ and $f(a) \cdot f(c) < 0$.
- Hence, in Step 4, we have $b_{\text{new}} = c = -0.125$, while $a = 0.25$ is once again unchanged. This gives $c = \frac{a+b}{2} = -0.3125$ and $f(a) \cdot f(c) < 0$.

■

From the zooms in Fig. 10.4, we observe that the more we zoom into our function at a point, the more it looks like a straight line! In fact, already at Step 4, we see that if we had the formula for the line that approximates the function, we'd use it to approximate the root instead of doing more iterations with the Bisection Algorithm.

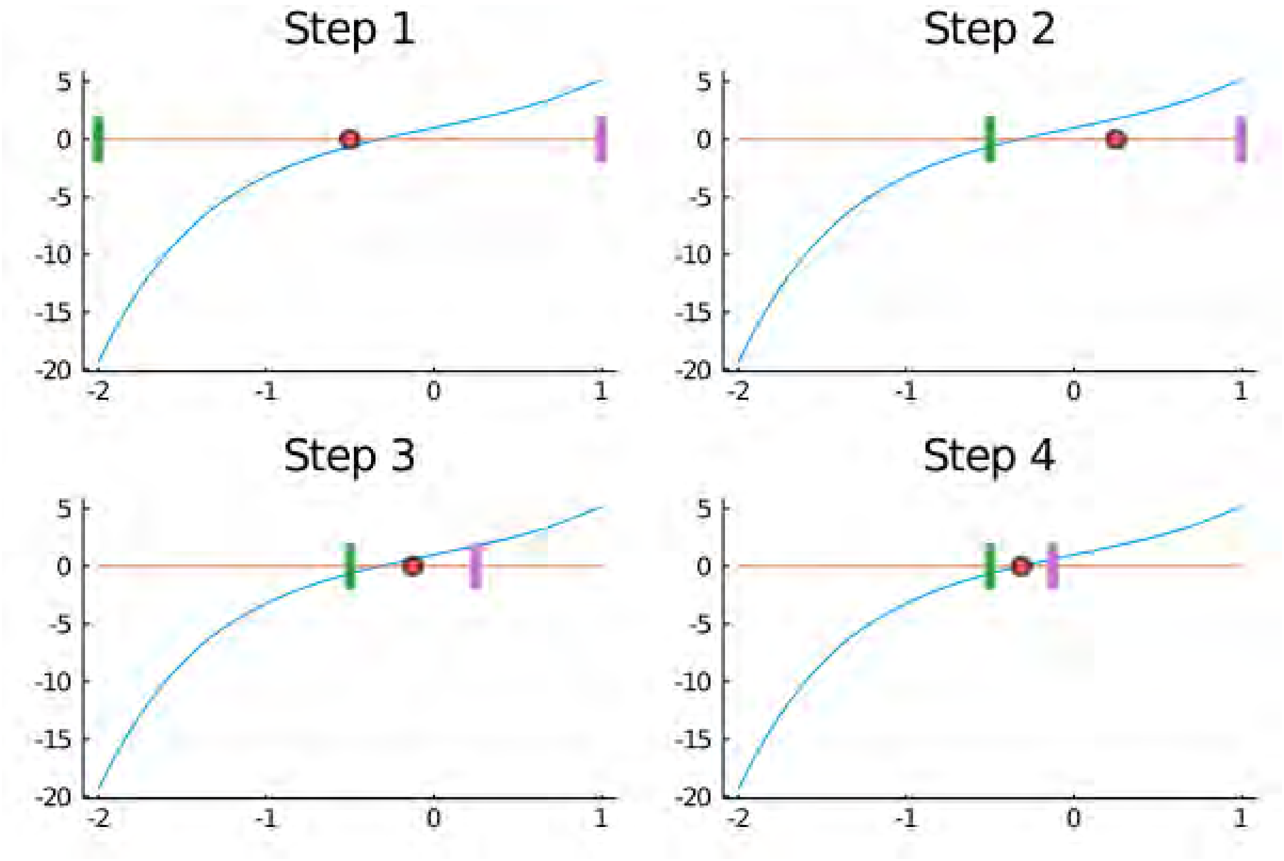


Figure 10.3: Evolution of the bracketing points a and b as well as the midpoint c in the first four steps of the Bisection Algorithm for finding a root of $0.2x^5 + x^3 + 3x + 1 = 0$. It is very clear that the algorithm hones in on a root!

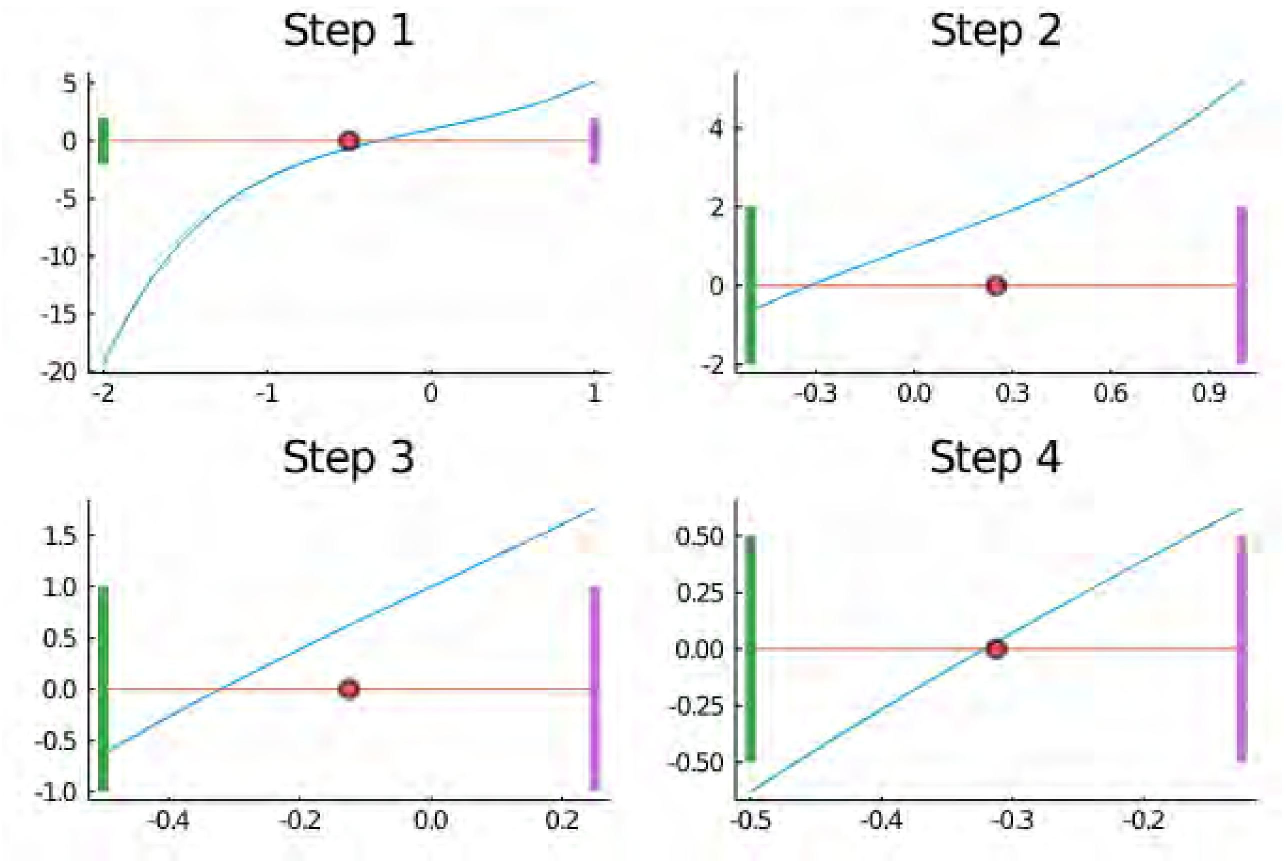


Figure 10.4: Zooms of the first four steps of the Bisection Algorithm for finding a root of $0.2x^5 + x^3 + 3x + 1 = 0$ that lies between -1 and 2. Observe that as we zoom into the function at a point, it looks more and more like a straight line!

Linear Approximations of Functions can be Very Useful

Let's write the "line" in Step 4 of Fig. 10.4 in the form

$$y = y_c + m(x - c),$$

where y_c is the value of the line at $x = c$ and m is the slope of the line. Using the data in (10.2), and the traditional notion of "rise over run" to define the slope, we obtain

$$\begin{aligned} c &= -0.3125 \\ y_c &= f(c) \approx 0.0313864 \\ m &= \frac{f(b) - f(a)}{b - a} = \frac{0.623041 - (-0.63125)}{-0.125 - (-0.5)} \approx 3.34478 \end{aligned}$$

In Fig. 10.5, we visually illustrate how good of a fit the "line approximation"

$$y = 0.0313864 + 3.34478(x + 0.3125) = 3.34478x + 1.07663$$

provides to the function. To show its utility, we set $y = 0$ and solve for x. Doing so, we obtain

$$\boxed{x^* = -0.321884 \implies f(x^*) = 0.00030674,}$$

an estimate of the root that it is much better than the value given by the Bisection Algorithm at Step 4! In fact, the Bisection Algorithm has to muddle along until its 12-th iteration to better this approximation of the root. **Issac Newton made this same observation back in 1669 and turned it into an algorithm for finding roots of equations.**

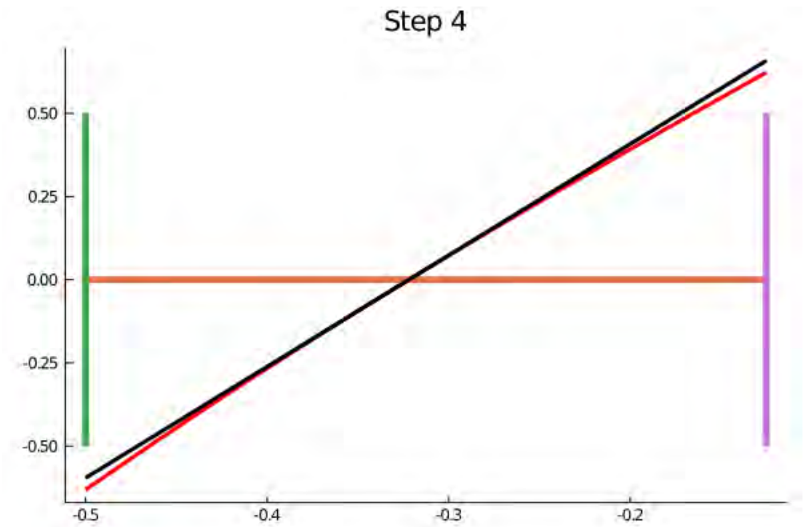


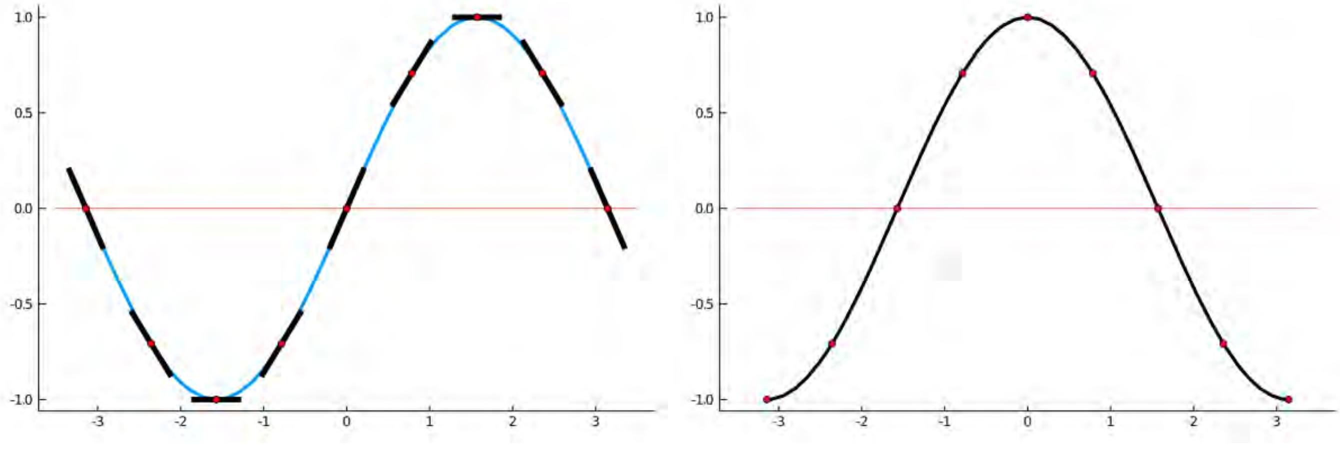
Figure 10.5: Linear Approximation (black) of $f(x) = 0.2x^5 + x^3 + 3x + 1$ compared to the function itself (red). The linear approximation is very good in a sufficiently small region.

Bisection Algorithm with Sanity Checks and Tolerance Included

The algorithm takes as input a generic function $f(x)$, bracketing points a and b, and a tolerance value, tol, for terminating the algorithm, where convergence is declared when $|f(c)| \le \mathrm{tol}$. The algorithm also terminates after 10^4 iterations. The function returns the final values for c, $f(c)$, and k, the number of iterations it took to meet the convergence criteria.

```
function Bisection(f,a,b,tol)
    Flag=1  # Positive flag means all is well in algorithm land
    if !(a < b)
        println("a is not strictly less than b")
        Flag=0
    end
    if !( f(a)*f(b) < 0)
        println("Fails test provided by the Intermediate Value Theorem")
        Flag=0
    end
    if tol < 1e-15
        println("tolerance is too tight")
        Flag=0
    end
c= (a+b)/2.0
fc=f(c)
k=0
while (abs(fc) > tol) & (k < 1e4) & (Flag > 0)
    if fc == 0
        Flag=-1
            println("Root is $c")
    elseif fc*f(a) < 0
        b=copy(c)
    else
        a=copy(c)
    end
   c = (a+b)/2
   fc=f(c)
   k=k+1
end
 return c, fc, k
end
```

10.3 The Concept of a Derivative and Its Numerical Approximation



(a)

(b)

Figure 10.6: (a) The line segments represent the local slope ("rise over run") of $f(x) = \sin(x)$ at the points $[-\pi, -\frac{3\pi}{4}, \ldots, \frac{3\pi}{4}, \pi]$. Notice that each line segment is also a local linear approximation of the function. In a practical sense, what this means is that in a small region about a given point, we can replace the function with a local linear equivalent and then use linear techniques to analyze the function! In Calculus, the "local slope of a function" is called the derivative of the function. (b) The derivative of $f(x)$ is another function, denoted $\frac{df(x)}{dx}$. In Calculus, you will learn that $\frac{d}{dx}\sin(x) = \cos(x)$. Here, were are NOT using Calculus. We have computed the derivative numerically and plotted it! The maximum error in our numerical estimation of the derivative is less than 6.58×10^{-6}.

Another concept that you will learn in Calculus is the **derivative of a function**. Geometrically, it is the slope of the function at a given point, say $x_0 \in \mathbb{R}$. Note that if $x_1 \leq x_0 < x_2$, then the "rise" of the function over the interval (x_1, x_2) would be $df(x_0) := f(x_2) - f(x_1)$, while the "run" would be $dx = x_2 - x_1$, and hence the "slope" would be

$$\text{slope} := \frac{\text{rise}}{\text{run}} = \frac{df(x_0)}{dx} = \frac{f(x_2) - f(x_1)}{x_2 - x_1}.$$

In Fig. 10.6, we have attached short line segments with slopes corresponding to the derivative of the function $\sin(x)$ computed at a number of points. The hope is that this helps you grasp the geometric meaning of a derivative of a function at point as the "local slope" of the function at that point. We see that the "local slope of the function" varies with x. To tie the idea of "slope equals rise over run" to a given point, say x_0, we let $h \neq 0$ be a small number and then we define x_1 and x_2 in terms of x_0 and h, by $x_1 = x_0$ and $x_2 = x_0 + h$. This leads to

$$\frac{df(x_0)}{dx} = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{f(x_0 + h) - f(x_0)}{(x_0 + h) - x_0} = \frac{f(x_0 + h) - f(x_0)}{h}. \tag{10.3}$$

In Calculus, one analyzes what happens in the limit when h becomes very small, and in particular, one works hard to understand when the ratio in (10.3) approaches a well defined value as h becomes smaller and smaller. While we'll explore this a bit in HW, it is beyond the scope of our effort here.

Numerical Approximations of a Derivative

We will adopt the traditional notation from Calculus for the limiting value in (10.3), namely

$$\frac{df(x_0)}{dx} := \lim_{h\to 0} \frac{f(x_0+h)-f(x_0)}{h}. \tag{10.4}$$

In practice, we will use "small values" for h and never compute the exact limit. Hence, we have an **approximation for the derivative** at a point, namely

$$\frac{df(x_0)}{dx} \approx \frac{f(x_0+h)-f(x_0)}{h}, \tag{10.5}$$

which is called a **forward difference approximation to the derivative**. Note that we have replaced the informal term "slope" with the symbol for the derivative at a point, namely $\frac{df(x_0)}{dx}$.

You can also do a **backward difference approximation to the derivative**,

$$\frac{df(x_0)}{dx} \approx \frac{f(x_0)-f(x_0-h)}{h}, \tag{10.6}$$

and a **symmetric difference approximation**, where you go both forward and backward from the point x_0,

$$\frac{df(x_0)}{dx} \approx \frac{f(x_0+h)-f(x_0-h)}{2h}. \tag{10.7}$$

The forward and backward difference approximations to the derivative are in fact exact for linear functions, while the symmetric difference approximation is *exact* for quadratic polynomials. The symmetric difference is also sometimes called a **central difference**.

If the derivative of $f(x)$ at a point x_0 exists, then for h sufficiently small, the forward difference, backward difference, and symmetric difference approximations to the derivative will always agree. If they provide different answers, then the limit in (10.4) **does not exist and the function is said to be not differentiable.**

```
f(x)=sin.(x)
#
#
function SymmetricDifference(f,a,b)
    # f = generic function
    # does 100 equally spaced points from a to b
    # returns x and df/dx using Symmetric Differences
    if !(b > a)
        println("You need b > a")
    end
    N=100
    h=(b-a)/(10*N)
    x=LinRange(a,b,N)
    x=collect(x)
    dfdx=0*x;
    for k=1:N
        dfdx[k]=(f(x[k]+h) - f(x[k]-h))/(2*h)
    end
    return dfdx, x
end
#
(dfdx,x)=SymmetricDifference(f,pi,-pi)
#
p1=plot(x, dfdx, legend=false,  linewidth=3, color=:black)
plot!(yzero,-3.5,3.5)
x0=[-pi -3*pi/4 -pi/2 -pi/4 0 pi/4 pi/2 3*pi/4 pi]'
```

```
df(x)=cos.(x) #Known from Calculus
              #included to make the plot look pretty??
y0=df(x0)
scatter!(x0,y0, color=:red)
plot(p1)
plot!(fmt = :png)
```

Linear Approximation at a Point

The importance of being able to approximate a function in a region about a point by a linear function cannot be overstated. When studying the Bisection Method for finding roots, we noted that as we zoomed in on the function near the root, it looked more and more like a straight line. This property holds for all points x_0 at which a function is differentiable, that is, all points at which we can compute a derivative.

The linear function $y(x)$ that passes through the point (x_0, y_0) with slope m can be written as

$$y(x) = y_0 + m\,(x - x_0)\,.$$

We use this to define the **linear approximation of a function at a point $\mathbf{x_0}$** by taking $y_0 := f(x_0)$ and $m := \frac{df(x_0)}{dx}$. This gives us

$$\boxed{\mathbf{f(x) \approx f(x_0) + \frac{df(x_0)}{dx}\,(x - x_0).}} \qquad (10.8)$$

Figure 10.7 shows the linear approximation of a cubic about a point. For the record, in Calculus, this is called a First-order Taylor Expansion. You do not need to recall this terminology in ROB 101, but when you see it again in Calculus, you can say, yeah, I know why that is important!

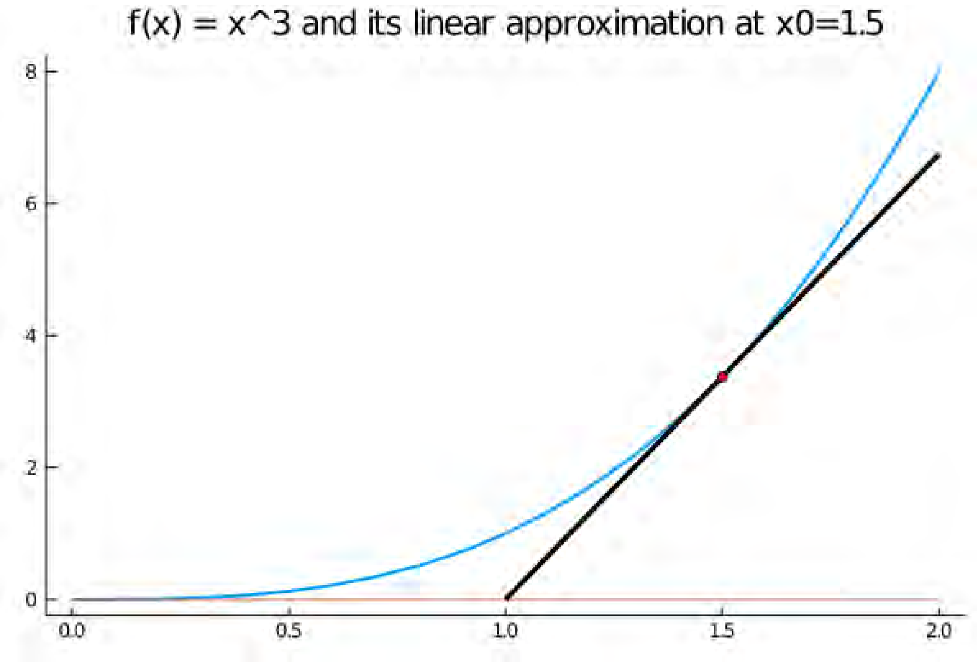


Figure 10.7: The function $f(x) = x^3$ is plotted in cyan. The value of the function at the point $x_0 = 1.5$ is indicated in red. The line in black passing through $f(x_0)$ with slope $m = \frac{df(x_0)}{dx}$ satisfies $y(x) := f(x_0) + \frac{df(x_0)}{dx}\,(x - x_0)$. The line is called the linear approximation of $f(x)$ at x_0. The linear approximation represents the function well in a sufficiently small region about x_0. This approximation can be done for any point x_0 at which the derivative exists.

Are all functions differentiable? No. A minimum requirement for a function to be differentiable at a point is that the function be continuous at that point. Are there functions that are continuous and not differentiable? Yes, the classic example is $f(x) = |x|$, which is plotted in Fig. 10.8 and discussed in Example 10.2.

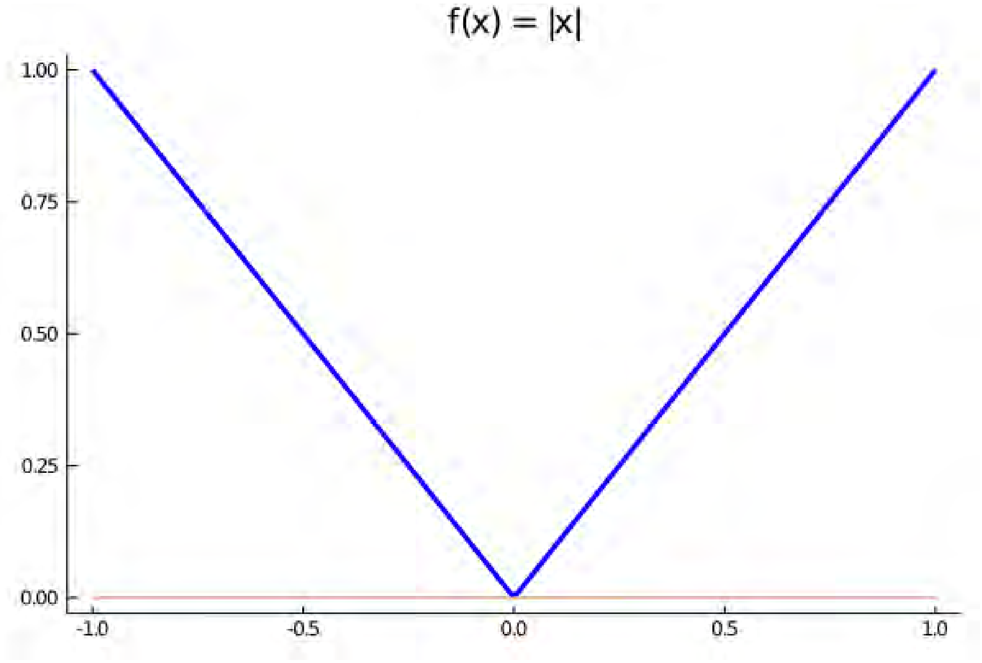


Figure 10.8: The function $f(x) = |x|$ is not differentiable at the origin ($x = 0$). The slope of the function just to the left of the origin is -1, the slope just to the right of the origin is $+1$, and the slope at the origin is undefined. Everywhere else, the function is differentiable.

Example 10.2 *Explain why the function $f(x) = |x|$ is not differentiable at $x_0 = 0$.*

Solution: We compute the forward difference, backward difference, and symmetric difference approximations to the derivative at the point $x_0 = 0$ and see if we obtain similar answers or not. For this, we let $h > 0$ be arbitrarily small. We note that then

$$|h| = h, \text{ and } |-h| = -(-h) = h.$$

Proceeding, we compute

$$\textbf{forward difference} \quad \frac{df(0)}{dx} \approx \frac{f(0+h) - f(0)}{h} = \frac{|h| - 0}{h} = \frac{h}{h} = \boxed{+1}$$

$$\textbf{backward difference} \quad \frac{df(0)}{dx} \approx \frac{f(0) - f(0-h)}{h} = \frac{0 - |-h|}{h} = \frac{-h}{h} = \boxed{-1}$$

$$\textbf{symmetric difference} \quad \frac{df(0)}{dx} \approx \frac{f(0+h) - f(0-h)}{2h} = \frac{|h| - |-h|}{2h} = \frac{h - h}{2h} = \boxed{0}$$

These three methods giving very different approximations to the "slope" at the origin is a strong hint that the function is not differentiable at the origin. What they are telling us is that by following different paths as we approach x_0, approaching x_0 from the left versus the right for example, gives different answers for the "slope" of the function at x_0. In Calculus, you'll learn that this means the function is not differentiable at x_0.

■

10.4 Newton's Method for Scalar Problems

We consider again the problem of finding roots of scalar equations $f(x) = 0$, where $f : \mathbb{R} \to \mathbb{R}$. In the Bisection Algorithm, we only required that the function be continuous. The method we develop now uses the "local slope information" of a function, and hence requires that the function be differentiable, that is, that we can define $\frac{df(x)}{dx}$.

Let x_k be our current approximation of a root of the function f. We write the linear approximation of f about the point x_k as

$$f(x) \approx f(x_k) + \frac{df(x_k)}{dx} \cdot (x - x_k). \tag{10.9}$$

We want to chose x_{k+1} so that $f(x_{k+1}) = 0$. Based on our linear approximation in (10.9), we have that

$$f(x_{k+1}) \approx 0 \iff 0 = f(x_k) + \frac{df(x_k)}{dx} \cdot (x_{k+1} - x_k).$$

If $\frac{df(x_k)}{dx} \neq 0$, we can solve for x_{k+1}, giving us

$$\frac{df(x_k)}{dx} x_{k+1} = \frac{df(x_k)}{dx} x_k - f(x_k)$$

$$\Downarrow$$

$$x_{k+1} = x_k - f(x_k) \Big/ \frac{df(x_k)}{dx}$$

For reasons that will become clear when attempt a vector version of Newton's Algorithm, let's rewrite the division operation in the above formula as

$$\boxed{x_{k+1} = x_k - \left(\frac{df(x_k)}{dx}\right)^{-1} f(x_k).}$$

The above equation is screaming for us to put it in a loop! One step of Newton's Method is shown in Fig. 10.9.

Newton's Method

The iterative process

$$x_{k+1} = x_k - \left(\frac{df(x_k)}{dx}\right)^{-1} f(x_k) \tag{10.10}$$

for finding a root of a nonlinear equation is called **Newton's Method** or **Newton's Algorithm**. Given the current approximation x_k to a root of $f(x)$, Newton's Method corrects the approximation by the term

$$-\left(\frac{df(x_k)}{dx}\right)^{-1} f(x_k) = -f(x_k) \Big/ \frac{df(x_k)}{dx}.$$

The validity of the next approximation x_{k+1} rests upon:

- the function f being differentiable;
- the derivative $\frac{df(x_k)}{dx}$ not vanishing at points generated by the algorithm in (10.10); and
- the linear equation (10.9) is a good approximation to the function.

We boxed this last item because it is easy to overlook and is often a source of failure for the algorithm. Because (10.10) has "total faith" in (10.9) being a good approximation, it sometimes takes very big "steps" (meaning $x_{k+1} - x_k$ is large) when generating x_{k+1} to zero the linear approximation in (10.9). A safer update is to go only "part way" to the linear solution. This leads to the so-called **damped** or **modified Newton Method**,

$$\boxed{x_{k+1} = x_k - \epsilon \left(\frac{df(x_k)}{dx}\right)^{-1} f(x_k)}, \tag{10.11}$$

where $0 < \epsilon < 1$. A typical value may be $\epsilon = 0.1$.

The standard way to "ensure" that Newton's Method generates points x_k such that the linear equation (10.9) is a good approximation to $f(x_k)$ is to start the algorithm "near" a root. As you can imagine, this is easier said than done! Hence, the damped version of Newton's Algorithm is very useful in practice.

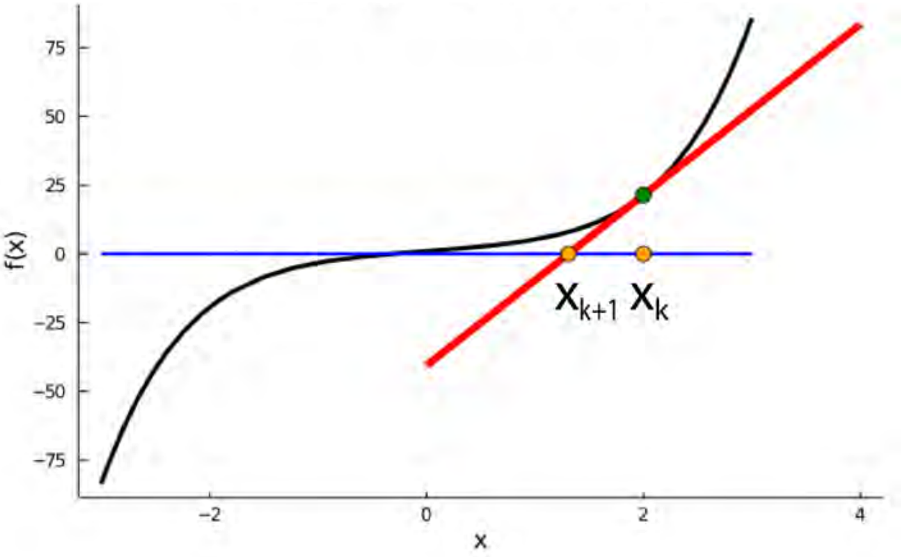


Figure 10.9: This figure demonstrates one step of Newton's Algorithm. At a point x_k, one uses the derivative to compute a linear approximation to the function. Solving for where the linear approximation (red line) crosses the x-axis gives the next value, x_{k+1}.

Visual Representation of Newton's Algorithm

Some of you are undoubtedly more visually wired than algebraically wired (did you know that was a thing?). Here are some potential visual sources:

- Wikipedia https://upload.wikimedia.org/wikipedia/commons/e/e0/NewtonIteration_Ani.gif (The words in the legend are **function** and **tangent**.) You'll find additional textual information here as well https://en.wikipedia.org/wiki/Newton%27s_method
- Kahn Academy https://www.youtube.com/watch?v=WuaI5G04Rcw
- Christine Breiner, MIT Calculus I, https://www.youtube.com/watch?v=ER5B_YBFMJo

Example 10.3 *For the same function as treated in Example 10.1, namely, $f(x) = 0.2x^5 + x^3 + 3x + 1$, find a root using Newton's Algorithm.*

Solution: We apply the basic Newton's Method in (10.9) (that is, no damping), using each of the derivative methods given in (10.5), (10.6), and (10.7). We take $x_0 = 2$ and $h = 0.01$ for the approximate derivatives. We iterate until $|f(_k)| < 10^{-4}$ or the algorithm fails by $\left|\frac{df(x_k)}{dx}\right| < 10^{-4}$.

Using the **Symmetric Difference Approximation** for the derivative, Newton's Method converges in five steps

$\mathbf{x_k}$	$\mathbf{f(x_k)}$	$\mathbf{\frac{df(x_k)}{dx}}$	$\mathbf{k}$
2.0000	21.4000	31.2209	0.0000
1.3146	8.0005	11.1709	1.0000
0.5984	3.0247	4.2025	2.0000
−0.1214	0.6341	3.0445	3.0000
−0.3296	−0.0255	3.3379	4.0000
−0.3220	−0.0001	3.3219	5.0000

(10.12)

Using the **Forward Difference Approximation** for the derivative, Newton's Method **fails** after 45 steps due to the estimated derivative vanishing

$\mathbf{x_k}$	$\mathbf{f(x_k)}$	$\frac{\mathbf{df(x_k)}}{\mathbf{dx}}$	$\mathbf{k}$
2.0000	21.4000	31.2209	0.0000
1.3146	8.0005	−1328.6981	1.0000
1.3206	8.0680	18.1162	2.0000
0.8752	4.3989	−360.9904	3.0000
⋮	⋮	⋮	⋮
$5.4e+01$	$8.9e+07$	$7.6e+03$	41.0000
$-1.2e+04$	$-4.5e+19$	$-4.5e+21$	42.0000
$-1.2e+04$	$-4.5e+19$	$-8.1e+10$	43.0000
$-5.5e+08$	$-1.0e+43$	$-1.0e+45$	44.0000
$-5.5e+08$	$-1.0e+43$	$0.0e+00$	45.0000

(10.13)

Using the **Backward Difference Approximation** for the derivative, Newton's Method converges after 23 steps

$\mathbf{x_k}$	$\mathbf{f(x_k)}$	$\frac{\mathbf{df(x_k)}}{\mathbf{dx}}$	$\mathbf{k}$
2.0000	21.4000	31.2209	0.0000
1.3146	8.0005	1351.0399	1.0000
1.3086	7.9346	17.5719	2.0000
0.8571	4.2934	369.8273	3.0000
0.8455	4.2272	12.2347	4.0000
0.5000	2.6311	163.4044	5.0000
0.4839	2.5702	9.8345	6.0000
0.2225	1.6787	92.2938	7.0000
0.2043	1.6216	8.8299	8.0000
0.0207	1.0621	58.9551	9.0000
0.0027	1.0080	8.4054	10.0000
−0.1173	0.6466	39.1841	11.0000
−0.1338	0.5963	8.0873	12.0000
−0.2075	0.3685	25.9194	13.0000
−0.2217	0.3239	7.6217	14.0000
−0.2642	0.1887	16.7398	15.0000
−0.2755	0.1524	6.8761	16.0000
−0.2976	0.0803	10.4913	17.0000
−0.3053	0.0552	5.8084	18.0000
−0.3148	0.0238	6.4494	19.0000
−0.3185	0.0116	4.5491	20.0000
−0.3210	0.0031	4.1765	21.0000
−0.3218	0.0006	3.5817	22.0000
−0.3220	0.0000	3.3919	23.0000

(10.14)

■

Symmetric Difference Makes a Difference

In general, the symmetric difference is a better approximation to the true analytical derivative than are the forward and backward difference approximations. When used in Newton's Method, the big difference in performance of the three approximate derivative methods surprised us as well!

Why do people not use it all the time? Depending on the situation, you may have the value of $f(x_k)$ already at hand, in which case, to determine a forward or backward difference, you only need one additional function evaluation, namely, either $f(x_k + h)$ or $f(x_k - h)$, whereas with the symmetric difference, you must do both additional function evaluations. If f is complicated to evaluate, that may bias you toward the computationally "lighter" methods. On the other hand, as we saw in our example with Newton's Algorithm, if you converge faster, you may still come out way ahead!

The fact that the decision of which numerical differentiation method to use is not obvious and depends on the problem being solved is actually A GREAT THING: it keeps Engineers and Applied Mathematicians employed!

10.5 Vector Valued Functions: Linear Approximations, Partial Derivatives, Jacobians, and the Gradient

When developing Newtons' Method of root finding for functions $f : \mathbb{R} \to \mathbb{R}$, we started with the notion of a derivative being the local slope of a function at a point, and from there, we were led to the idea of locally approximating a nonlinear function by a line! Once we had the idea of a linear approximation of the function about a given point, Newton's Algorithm basically fell into our lap by solving for a root of the linear approximation.

For the vector case of functions $f : \mathbb{R}^m \to \mathbb{R}^n$, we'll turn things around a bit and start with the idea of a linear approximation of the function about a point and see how that leads us to the notion of a **partial derivative**. Once we have that concept down, the rest is book keeping, in other words, the rest is developing a nice matrix-vector formulation of a derivative of a function. It sounds harder than it is. Let's do it!

Remark: A more traditional approach that starts by introducing the notion of a partial derivative and, from there, builds the gradient, the Jacobian, and only then, introduces the idea of a linear approximation, maybe better for some readers. That path is followed in Chap. 10.7.

10.5.1 Linear Approximation about a Point: Take 1

Our goal is to generalize the idea of a linear approximation of a (nonlinear) function $f : \mathbb{R}^m \to \mathbb{R}^n$ at a point x_0. What we'll do is posit that the appropriate generalization should be

$$f(x) \approx f(x_0) + A(x - x_0), \tag{10.15}$$

where A is an $n \times m$ matrix. We note that the dimensions make sense because $f(x_0)$ is $n \times 1$, $(x - x_0)$ is $m \times 1$, and therefore, $A(x - x_0)$ is $n \times 1$. So far, so good.

Let's now figure out what the columns of A need to be for (10.15) to hold of x "near" x_0. We write $A =: \begin{bmatrix} a_1^{\rm col} & a_2^{\rm col} & \cdots & a_m^{\rm col} \end{bmatrix}$, where $a_j^{\rm col}$ is the j-th column of A. Further, let $\{e_1, e_2, \dots, e_m\}$ the canonical basis vectors for $\mathbb{R}^m$ (which we recall are the columns of the $m \times m$ identity matrix). We next recall that our "sum over columns times rows" method of matrix multiplication gives us that

$$Ae_j = a_j^{\rm col},$$

which is true because, using "Julia notation",

$$(e_j)[i] = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

implies that

$$Ae_j = \sum_{i=1}^{m} a_i^{\rm col}(e_j)[i] = a_j^{\rm col}.$$

We let $x = x_0 + he_j$ be a small perturbation about the nominal vlaue x_0. We note that $x = x_0 + he_j$ **holds all components of x constant and equal to x_0, except for the j-th component, which is perturbed by an amount h**. When $h > 0$ is sufficiently small, (10.15) gives us

$$\begin{aligned} f(x_0 + he_j) &= f(x_0) + A(x_0 + he_j - x_0) \\ &\Downarrow \\ f(x_0 + he_j) &= f(x_0) + hAe_j \\ &\Downarrow \\ f(x_0 + he_j) &= f(x_0) + ha_j^{\text{col}} \\ &\Downarrow \\ f(x_0 + he_j) - f(x_0) &= ha_j^{\text{col}} \\ &\Downarrow \\ \frac{f(x_0 + he_j) - f(x_0)}{h} &= a_j^{\text{col}}. \end{aligned} \tag{10.16}$$

In other words, the j-th column of the matrix A in (10.15) is given by

$$\boxed{a_j^{\text{col}} = \frac{f(x_0 + he_j) - f(x_0)}{h},} \tag{10.17}$$

which looks suspiciously like the forward difference approximation of a derivative. In fact, it looks like here we are ignoring all variables except the j-th one and computing a derivative of f with respect to x_j. And indeed, that is exactly what we are doing! Calculus has a term for it, the **partial derivative of $\mathbf{f(x)}$** with respect to $\mathbf{x_j}$, and it uses a cool symbol,

$$\boxed{\frac{\partial f(x_0)}{\partial x_j} = \lim_{h \to 0} \frac{f(x_0 + he_j) - f(x_0)}{h}.} \tag{10.18}$$

The symbol ∂ is pronounced “partial”. We’d better dig into this!

10.5.2 Partial Derivatives

Partial Derivatives as Motivated by a Linear Approximation to a Function about a Point

If we let $\{e_1, e_2, \ldots, e_m\}$ be the natural basis vectors for $\mathbb{R}^m$, then we have three ways to **numerically approximate a partial derivative**, just as we did with a “scalar” derivative

$$\frac{\partial f(x_0)}{\partial x_j} = \begin{cases} \dfrac{f(x_0 + he_j) - f(x_0)}{h} & \text{forward difference approximation} \\ \dfrac{f(x_0) - f(x_0 - he_j)}{h} & \text{backward difference approximation} \\ \dfrac{f(x_0 + he_j) - f(x_0 - he_j)}{2h} & \text{symmetric difference approximation.} \end{cases} \tag{10.19}$$

Example 10.4 *For the function*

$$f(x_1, x_2, x_3) := \begin{bmatrix} x_1 x_2 x_3 \\ \log(2 + \cos(x_1)) + x_2^{x_1} \\ \frac{x_1 x_3}{1 + x_2^2} \end{bmatrix}, \tag{10.20}$$

compute the partial derivatives $\frac{\partial f(x_0)}{\partial x_1}$, $\frac{\partial f(x_0)}{\partial x_2}$, *and* $\frac{\partial f(x_0)}{\partial x_3}$ *at the point*

$$x_0 = \begin{bmatrix} \pi \\ 1.0 \\ 2.0 \end{bmatrix}.$$

In the next example, we’ll interpret the computed partial derivatives in terms of derivatives of scalar valued functions, which we intuitively understood as slopes of a function at a point.

Solution A We'll compute the partial derivatives in Julia, two different ways. We only need one of them to click for you.

In the first solution, we write the function given in (10.20) as $f(x)$, where $x = [x_1; x_2; x_3]$. We can then apply the numerical approximations in (10.19) directly. Because $f(x) \in \mathbb{R}^3$, the partial derivatives will also be vectors in $\mathbb{R}^3$. This follows from (10.19), where each numerator is a vector in $\mathbb{R}^3$, while the denominators are scalars.

```julia
x0=[pi;1.0;2.0]
# function defined in terms of x as a vector with components [x1; x2; x3].
function f(x)
    x1=x[1]
    x2=x[2]
    x3=x[3]
    f=[x1*x2*x3; log(2 + cos(x1) + x2^x1); (x1*x3)/(1+x2^2)]
    return f
end
h=0.001
Id=zeros(3,3)+I
e1=Id[:,1];e2=Id[:,2];e3=Id[:,3]
# Partial derivatives via symmetric differences
dfdx1=( f(x0+h*e1) - f(x0-h*e1) )/(2*h)
dfdx2=( f(x0+h*e2) - f(x0-h*e2) )/(2*h)
dfdx3=( f(x0+h*e3) - f(x0-h*e3) )/(2*h)


```

Using the above code, we determine

$$\frac{\partial f(x_0)}{\partial x_1} = \begin{bmatrix} 2.0 \\ 0.0 \\ 1.0 \end{bmatrix}, \; \frac{\partial f(x_0)}{\partial x_2} = \begin{bmatrix} 6.2832 \\ 3.1416 \\ -3.1416 \end{bmatrix}, \; \frac{\partial f(x_0)}{\partial x_3} = \begin{bmatrix} 3.1416 \\ 0.0000 \\ 1.5708 \end{bmatrix}. \tag{10.21}$$

■

Solution B In the second method, we express the function exactly as it is written in (10.20). We then have to recognize that

$$x_0 + he_1 = (x_{01} + h, x_{02}, x_{03}), \;\; x_0 + he_2 = (x_{01}, x_{02} + h, x_{03}), \text{ and } \; x_0 + he_3 = (x_{01}, x_{02}, x_{03} + h).$$

The point is, in Mathematics, we write a function that depends on several variables like this

$$f(x) = f(x_1, x_2, x_3),$$

and never like this

$$f(x) = f(\begin{bmatrix} x_1 \\ x_2 \\ x_2 \end{bmatrix}). \tag{10.22}$$

However, when we program, it is often easier to work with a function as if it were written as in (10.22), with x a column vector; as an example,

$$f(x + he_2) = f(\begin{bmatrix} x_1 \\ x_2 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ h \\ 0 \end{bmatrix}) = f(x_1, x_2 + h, x_3).$$

You will learn quickly enough that it is easier to "vectorize" (that is, put operations in a loop) expressions such as $f(x + he_2) = f(x + hId[:, 2])$ than it is expressions such as$f(x_1, x_2 + h, x_3)$, but we digress.

```julia
x0=[pi;1.0;2.0]
f(x1,x2,x3)=[x1*x2*x3; log(2 + cos(x1) + x2^x1); (x1*x3)/(1+x2^2)]
h=0.001

```

```
dfdx1=( f(pi+h,1.0,2.0) - f(pi-h,1,2) )/(2*h)
dfdx2=( f(pi,1.0+h,2.0) - f(pi,1-h,2) )/(2*h)
dfdx3=( f(pi,1.0,2.0+h) - f(pi,1,2-h) )/(2*h)

```

The results match those in (10.21). The code for the second solution looks simpler, doesn't it? But imagine writing that out if you have 25 variables! On the other hand, the code segment

```
# As a loop
n=3
dfdx=Array{Float64,2}(undef,n,0)
for k =1:n
    dfdxk=( f(x0+h*Id[:,k]) - f(x0-h*Id[:,k]) )/(2*h)
    dfdx=[dfdx dfdxk]
end
dfdx

3ÃŮ3 Array{Float64,2}:
 2.0   6.28319  3.14159
 0.0   3.14159  0.0
 1.0  -3.14159  1.5708
```

is very easy to scale up! **Vectors and matrices are really about careful bookkeeping. It's kind of sad to say it that way, but it's also kind of true.** ■

Example 10.5 *For the function in Example 10.4, interpret the components of its partial derivatives in terms of "ordinary derivatives".*

Solution Let's quite arbitrarily focus on x_2. We define a function $g : \mathbb{R} \to \mathbb{R}^3$ by

$$g(x_2) := f(\pi, x_2, 2) = \left[\begin{array}{c} \pi x_2 2 \\ \log(2+\cos(\pi)) + (x_2)^{\pi} \\ \frac{\pi 2}{1+(x_2)^2} \end{array}\right] = \left[\begin{array}{c} 2\pi x_2 \\ (x_2)^{\pi} \\ \frac{2\pi}{1+(x_2)^2} \end{array}\right],$$

where we have set $x_1 = \pi$ and $x_3 = 2$. Because the components of g only depend on the single variable x_2, we can compute their ordinary derivatives about the point $x_{02} = 1$ using symmetric differences. We do so and determine that

$$\frac{dg(x_{02})}{dx_2} \approx \frac{g(1+h) - g(1-h)}{2h} = \left[\begin{array}{c} 6.2832 \\ 3.1416 \\ -3.1416 \end{array}\right].$$

We observe that $\frac{dg(x_{02})}{dx_2} = \frac{\partial f(x_0)}{\partial x_2}$. We also note that $g(x_2)$ being a column vector with three components does not really change anything: we are simply computing the slope of each component of g.

■

Partial Wisdom

Partial derivatives are simply ordinary derivatives that we perform one variable at time, while holding all other variables constant.

$$\begin{aligned}\frac{\partial f(x_0)}{\partial x_i j} &\approx \frac{f(x_{01}, \ldots, \mathbf{x_{0j}} + \mathbf{h}, \ldots, x_{0m}) - f(x_{01}, \ldots, \mathbf{x_{0j}}, \ldots, x_{0m})}{\mathbf{h}} \\ &= \frac{f(x_0 + h e_j) - f(x_0)}{\mathbf{h}} \\ &= \frac{df(x_0 + x_j e_j)}{dx_j},\end{aligned} \tag{10.23}$$

where the last expression is kind of awesome: it underlines that we have really fixed all of the components of x EXCEPT for x_j when we compute the partial derivative with respect to the j-th component of x; therefore, $\overline{f}(x_j) := f(x_0 + x_j e_j)$ is now a function of the scalar variable x_j and we can apply the ordinary derivative. The fact that $\overline{f}(x_j) \in \mathbb{R}^n$ means it has n-components instead of one has not really changed anything. For us, working with vectors or scalars, it's all the same.

10.5.3 The Jacobian and Linear Approximation of a Function about a Point

We now turn to functions $f : \mathbb{R}^m \to \mathbb{R}^n$. Based the compact notation introduced in (10.19), we define the **Jacobian of a function** as

$$\frac{\partial f(x)}{\partial x} := \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \frac{\partial f(x)}{\partial x_2} & \cdots & \frac{\partial f(x)}{\partial x_m} \end{bmatrix} \tag{10.24}$$

by packaging the column vectors $\frac{\partial f(x)}{\partial x_j}$ into a matrix. There is no mystery here as we discovered partial derivatives as the columns of a matrix back in (10.17). We need to keep in mind that, for each value of $x \in \mathbb{R}^m$, the compact and innocent looking object

$$\frac{\partial f(x)}{\partial x}$$

is really an $n \times m$ matrix: once again, there are m columns of the form $\frac{\partial f(x)}{\partial x_j}$, and each column is an n-vector. When computing the Jacobian numerically, we typically build it up one column at a time as we did in Solution A to Example 10.4.

Just for the record, we will write out $\frac{\partial f(x)}{\partial x}$ as an $n \times m$ matrix. We write

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_m) \\ f_2(x_1, x_2, \ldots, x_m) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_m) \end{bmatrix}.$$

Writing out all of the entries in the $n \times m$ Jacobian matrix gives

$$\frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_m} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_m} \end{bmatrix}. \tag{10.25}$$

In other symbols, the ij component of $\frac{\partial f(x)}{\partial x}$ is

$$\left[\frac{\partial f(x)}{\partial x}\right]_{ij} = \frac{\partial f_i(x)}{\partial x_j},$$

which is much more intimidating than (10.24).

Perhaps it is better not to read any further? If you want to toss out the window all of the benefits of vector-matrix notation, you can compute each entry of the Jacobian matrix one by one, as in

$$\frac{\partial f_i(x)}{\partial x_j} \approx \frac{f_i(x_1, \ldots, x_j + h, \ldots, x_m) - f_i(x_1, \ldots, x_j - h, \ldots, x_m)}{2h}. \tag{10.26}$$

In case you are wondering, your instructors almost never do this when doing real robotics! We use the "vector version" of the Jacobian where we compute each column of the matrix in a loop. However, for simple examples, such as $n = m = 2$, the above very explicit scalar (means non-vector) calculations can be informative!

Linear Approximation at a Point for Functions of Vectors

The linear approximation of a (nonlinear) function $f : \mathbb{R}^m \to \mathbb{R}^n$ at a point x_0 is defined to be

$$f(x) \approx f(x_0) + A(x - x_0) = f(x_0) + \frac{\partial f(x_0)}{\partial x}(x - x_0), \tag{10.27}$$

where the $n \times m$ matrix A is the Jacobian of f at the point x_0. As we did previously, we note that the dimensions make sense because $f(x_0)$ is $n \times 1$, $(x - x_0)$ is $m \times 1$, and therefore, $\frac{\partial f(x_0)}{\partial x}(x - x_0)$ is $n \times 1$.

Example 10.6 *For the function*

$$f(x_1, x_2, x_3) := \begin{bmatrix} x_1 x_2 x_3 \\ \log(2 + \cos(x_1)) + x_2^{x_1} \\ \frac{x_1 x_3}{1 + x_2^2} \end{bmatrix}, \tag{10.28}$$

compute its Jacobian at the point

$$x_0 = \begin{bmatrix} \pi \\ 1.0 \\ 2.0 \end{bmatrix}$$

and evaluate the "accuracy" of its linear approximation.

Solution From (10.21) in Example 10.4, we have that

$$\frac{\partial f(x_0)}{\partial x_1} = \begin{bmatrix} 2.0 \\ 0.0 \\ 1.0 \end{bmatrix}, \quad \frac{\partial f(x_0)}{\partial x_2} = \begin{bmatrix} 6.2832 \\ 3.1416 \\ -3.1416 \end{bmatrix}, \quad \frac{\partial f(x_0)}{\partial x_3} = \begin{bmatrix} 3.1416 \\ 0.0000 \\ 1.5708 \end{bmatrix}.$$

Hence, packaging up the columns correctly gives the Jacobian at x_0,

$$A := \frac{\partial f(x_0)}{\partial x} = \begin{bmatrix} 2.0000 & 6.2832 & 3.1416 \\ 0.0000 & 3.1416 & 0.0000 \\ 1.0000 & -3.1416 & 1.5708 \end{bmatrix},$$

and the linear approximation is

$$f(x) \approx f(x_0) + A(x - x_0) = \begin{bmatrix} 6.2832 \\ 1.0000 \\ 3.1416 \end{bmatrix} + \begin{bmatrix} 2.0000 & 6.2832 & 3.1416 \\ 0.0000 & 3.1416 & 0.0000 \\ 1.0000 & -3.1416 & 1.5708 \end{bmatrix} \begin{bmatrix} x_1 - \pi \\ x_2 - 1.0 \\ x_3 - 2.0 \end{bmatrix}.$$

```
Jacf=[dfdx1 dfdx2 dfdx3]
```

One way to think about the question of assessing the quality of the linear approximation is to measure the error defined as

$$e(x) := ||f(x) - f_{\rm lin}(x)||,$$

where $f_{\rm lin}(x) := f(x_0) + \frac{\partial f(x_0)}{\partial x}(x - x_0)$. We will seek to estimate the maximum value of $e(x)$ over a region containing the point x_0. Define

$$S(x_0) := \{x \in \mathbb{R}^3 \mid \ |x_i - x_{0i}| \le d, i = 1, 2, 3\}$$

and

$$\text{Max Error} := \max_{x \in S(x_0)} e(x) = \max_{x \in S(x_0)} ||f(x) - f_{\rm lin}(x)||. \tag{10.29}$$

The for $d = 0.25$, we used a "random search" routine and estimated that

$$\text{Max Error} = 0.12.$$

To put this into context,

$$\max_{x \in S(x_0)} ||f(x)|| = 8.47,$$

and thus the relative error is about 1.5%. ■

10.5.4 The Gradient and Linear Approximation of a Function about a Point

The **gradient** is simply the special name given to the Jacobian of a function $f : \mathbb{R}^m \to \mathbb{R}$, that is, for each $x \in \mathbb{R}^m$, $f(x) \in \mathbb{R}$ is a scalar. Along with its special name, it comes with a special symbol!

The Gradient and Linear Approximations

The **gradient** of $f : \mathbb{R}^m \to \mathbb{R}$ is simply the partial derivatives of f with respect to x_i arranged to form a **row vector**,

$$\nabla f(x_0) := \begin{bmatrix} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{bmatrix}, \tag{10.30}$$

which we can also call a $1 \times m$ matrix. The cool symbol ∇ is usually pronounced as "**grad**" and one says "**grad f**" when speaking of ∇f.

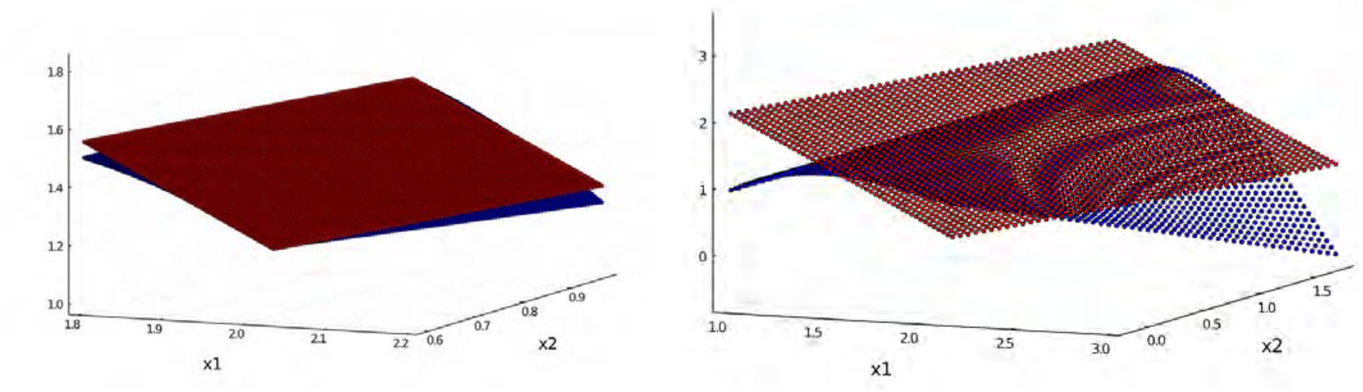
An important use of the gradient of a function of several variables is to form a **linear approximation of the function about a point**

$$f(x) \approx f(x_0) + \nabla f(x_0)(x - x_0). \tag{10.31}$$

Comparing this to (10.15), we see that $A = \nabla f(x_0)$, a $1 \times m$ matrix. Expanding (10.31) into its components gives

$$\begin{aligned} f(x) &\approx f(x_0) + \nabla f(x_0)(x - x_0) \\ &= f(x_0) + \underbrace{\begin{bmatrix} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} x_1 - x_{01} \\ x_2 - x_{02} \\ \vdots \\ x_m - x_{0m} \end{bmatrix}}_{(x - x_0)} \\ &= f(x_0) + \sum_{i=1}^{m} \frac{\partial f(x_0)}{\partial x_i}(x_i - x_{0i}). \end{aligned} \tag{10.32}$$

The linear approximation in (10.32) looks just like our linear approximation for functions of a single variable x, namely $f(x) \approx f(x_0) + \frac{df(x_0)}{dx}(x - x_0)$, where $a = \frac{df(x_0)}{dx}$ is 1×1.



(a)

(b)

Figure 10.10: (a) Close up view. (b) A more global view. The function $f(x_1, x_2) = x_1 \cos(x_2)$ is plotted in blue, while in red is shown its linear approximation about the point $x_0 = [2 \;\; \pi/4]^\top$, that is, $f_{\text{lin}}(x) := f(x_0) + \nabla f(x_0)(x - x_0)$. This approximation can be done for any point x_0 at which the partial derivatives exists. In Calculus, the red plane is also called the **tangent plane at $\mathbf{x_0}$**. These linear approximations accurately represent the nonlinear function in a small enough region about a given point, allowing us to use our Linear Algebra skills.

Example 10.7 *Compute (approximately) the gradient of* $f : \mathbb{R}^2 \to \mathbb{R}$, *for* $f(x_1, x_2) = x_1 \cos(x_2)$ *and* $x_0 = [2 \;\; \pi/4]^\top$.

Solution: Let's get the formulas for a general $h > 0$ and then we'll build a Table comparing the results for several values of h. For the partial derivative with respect to x_1, we perturb x_1 abut 2 while holding x_2 constant and equal to $\pi/4$. This gives

$$\begin{aligned}\frac{\partial f(x_0)}{\partial x_1} &\approx \frac{f(2+h,\pi/4) - f(2-h,\pi/4)}{2h} \\ &= \frac{(2+h)\cos(\pi/4) - (2-h)\cos(\pi/4)}{2h} \\ &= \frac{2h\cos(\pi/4)}{2h} \\ &= \cos(\pi/4) = \frac{\sqrt{2}}{2},\end{aligned}$$

which is independent of h and hence there is nothing more to compute. For the next partial derivative, we perturb x_2 about $\pi/4$ while holding x_1 constant and equal to 2. This gives

$$\begin{aligned}\frac{\partial f(x_0)}{\partial x_2} &\approx \frac{f(2,\pi/4+h) - f(2,\pi/4-h)}{2h} \\ &= \frac{2\cos(\pi/4+h) - 2\cos(\pi/4-h)}{2h},\end{aligned} \tag{10.33}$$

which, though we could simply it with some trigonometric identities, we'll stop here and turn to Julia. Doing so, leads to the following results

$\frac{f(2,\pi/4+h)-f(2,\pi/4-h)}{2h}$	h
-1.41421356001592	0.0001
-1.414213326670799	0.001
-1.4141899922649026	0.01
-1.4118577179998826	0.1
-1.4048043101898116	0.2
-1.3768017528243548	0.4

(10.34)

The true answer is

$$\frac{\partial f(2,\pi/4)}{\partial x_2} = -\sqrt{2} \approx -1.4142135623730951$$

■

Example 10.8 *Compute a linear approximation of $f(x_1, x_2) = x_1 \cos(x_2)$ at the point $x_0 = [2 \;\; \pi/4]^\top$.*

Solution: We know both of the partial derivatives of f at the point $x_0 = [2 \;\; \pi/4]^\top$. Hence, we have

$$\begin{aligned}f(x) &\approx f(x_0) + \nabla f(x_0)(x - x_0) \\ &= f(x_0) + \left[\begin{array}{cc} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} \end{array}\right] \left[\begin{array}{c} x_1 - x_{01} \\ x_2 - x_{02} \end{array}\right] \\ &= 2\cos(\pi/4) + \left[\begin{array}{cc} \frac{\sqrt{2}}{2} & -\sqrt{2} \end{array}\right] \left[\begin{array}{c} x_1 - 2 \\ x_2 - \pi/4 \end{array}\right]\end{aligned} \tag{10.35}$$

Figure 10.10 compares the linear approximation to the nonlinear function in a region about x_0. ■

Knowledge is Power

For root finding, an accuracy of a few percent is probably good enough. That said, **learning how to compute the partial derivatives analytically will make your code faster and will eliminate the question of how to choose the small perturbation parameter $h > 0$.**

10.5.5 Summary on Partial Derivatives

From slopes of lines→slopes of functions at points → $\frac{df(x_0)}{dx} \to \nabla f(x_0) \to \frac{\partial f(x_0)}{\partial x}$

Derivatives, gradients, and Jacobians are all generalizations of the notion of the slope of a line being its "rise over run". The derivative of a function $f : \mathbb{R} \to \mathbb{R}$ at a point x_0 is the "local" slope of the function at that point. We compute it by "rise over run", where, for example

$$\text{slope} = \frac{f(x_0+h) - f(x_0)}{h} \xrightarrow[h>0 \text{ small}]{} \frac{df(x_0)}{dx}, \tag{10.36}$$

and to make it local, we take $h > 0$ small. The gradient recognizes that a function $f : \mathbb{R}^m \to \mathbb{R}$ has a local slope in the x_1-direction, the x_2-direction, all the way up to the x_m-direction. If we let $\{e_1, e_2, \ldots, e_m\}$ be the natural basis vectors for $\mathbb{R}^m$, then we compute each component of the gradient by

$$\text{slope}_j = \frac{f(x_0+he_j) - f(x_0)}{h} \xrightarrow[h>0 \text{ small}]{} \frac{\partial f(x_0)}{\partial x_j}, \tag{10.37}$$

and we assemble the gradient by

$$\nabla f(x_0) := \left[\begin{array}{cccc} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{array} \right]. \tag{10.38}$$

Finally, for $f : \mathbb{R}^m \to \mathbb{R}^n$, each of the n components of $f(x)$ has a local slope with respect to each component of x. The bookkeeping is easiest if we leave $f(x)$ as a vector and write the Jacobian so that it **looks just like** the gradient,

$$\frac{\partial f(x_0)}{\partial x} := \left[\begin{array}{cccc} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{array} \right], \tag{10.39}$$

but now, because $f(x)$ is an n-vector, we have

$$\begin{bmatrix} \text{slope}_{1j} \\ \vdots \\ \text{slope}_{ij} \\ \vdots \\ \text{slope}_{nj} \end{bmatrix} = \frac{f(x_0+he_j) - f(x_0)}{h} \xrightarrow[h>0 \text{ small}]{} \frac{\partial f(x_0)}{\partial x_j}. \tag{10.40}$$

Compact Way to Numerically Approximate the Jacobian

If we let $\{e_1, e_2, \ldots, e_m\}$ be the natural basis vectors for $\mathbb{R}^m$, then vector notation allows us to compute each column of the Jacobian by

$$\frac{\partial f(x_0)}{\partial x_j} = \frac{f(x_0+he_j) - f(x_0)}{h}. \tag{10.41}$$

Typically, this is easier to program up than (10.26). But of course, your experience may vary! For a symmetric difference, we'd use

$$\frac{\partial f(x_0)}{\partial x_j} = \frac{f(x_0+he_j) - f(x_0-he_j)}{2h}. \tag{10.42}$$

10.6 Newton-Raphson for Vector Functions

We consider functions $f : \mathbb{R}^n \to \mathbb{R}^n$ and seek a root $f(x_0) = 0$. Note that the domain and range are both $\mathbb{R}^n$ and thus this is the nonlinear equivalent of solving a square linear equation $Ax - b = 0$. We recall that $\det(A) \neq 0$ was our magic condition for the existence and uniqueness of solutions to $Ax - b = 0$.

The **Newton-Raphson** Algorithm is precisely a vector version of Newton's Algorithm. Let x_k be our current approximation of a root of the function f. We write the linear approximation of f about the point x_k as

$$f(x) \approx f(x_k) + \frac{\partial f(x_k)}{\partial x} \cdot (x - x_k). \tag{10.43}$$

We want to chose x_{k+1} so that $f(x_{k+1}) = 0$. Based on our linear approximation in (10.43), we have that

$$f(x_{k+1}) \approx 0 \iff 0 \approx f(x_k) + \frac{\partial f(x_k)}{\partial x} \cdot (x_{k+1} - x_k). \tag{10.44}$$

If $\det\left(\frac{\partial f(x_k)}{\partial x}\right) \neq 0$ we could naively solve for x_{k+1}, giving us

$$\boxed{x_{k+1} = x_k - \left(\frac{\partial f(x_k)}{\partial x}\right)^{-1} f(x_k).}$$

By now, you know that we advise against blinding computing inverses of matrices unless you really need that matrix inverse. In our case, what we really want is x_{k+1}, and because we know to avoid computing unnecessary matrix inverses, we'll write the algorithm down in a different way. As in the scalar case, the equation is screaming for us to put it in a loop!

Newton-Raphson Algorithm

Based on (10.44), we define[a] $x_{k+1} := x_k + \Delta x_k$, where Δx_k is our update to x_k. We can then break the algorithm into two steps,

$$\left(\frac{\partial f(x_k)}{\partial x}\right) \Delta x_k = -f(x_k) \qquad (\text{solve for } \ \Delta x_k) \tag{10.45}$$

$$x_{k+1} = x_k + \Delta x_k \ \ (\text{use } \ \Delta x_k \ \text{ to update our estimate of the root}). \tag{10.46}$$

While for toy problems, we can use the matrix inverse to solve (10.45) for Δx_k, for larger problems, we recommend using an LU Factorization or a QR Factorization. Once (10.45) has been solved, x_{k+1} is updated in (10.46) and the process repeats.

A **damped Newton-Raphson Algorithm** is obtained by replacing (10.46) with

$$x_{k+1} = x_k + \epsilon \Delta x_k, \tag{10.47}$$

for some $\epsilon > 0$. The validity of the Newton-Raphson Algorithm rests upon:

- the function f being differentiable;
- the Jacobian $\frac{\partial f(x_k)}{\partial x}$ having a non-zero determinant at points generated by (10.45) and (10.46); and
- $\boxed{\text{the linear equation } f_{\text{lin}}(x) = f(x_k) + \frac{\partial f(x_k)}{\partial x}(x - x_k) \text{ being a good approximation to the function.}}$

[a] Note that $\Delta x_k = x_{k+1} - x_k$.

Example 10.9 *Find a root of* $F : \mathbb{R}^4 \to \mathbb{R}^4$ *near* $x_0 = \begin{bmatrix} -2.0 & 3.0 & \pi & -1.0 \end{bmatrix}$ *for*

$$F(x) = \begin{bmatrix} x_1 + 2x_2 - x_1(x_1 + 4x_2) - x_2(4x_1 + 10x_2) + 3 \\ 3x_1 + 4x_2 - x_1(x_1 + 4x_2) - x_2(4x_1 + 10x_2) + 4 \\ 0.5\cos(x_1) + x_3 - (\sin(x_3))^7 \\ -2(x_2)^2 \sin(x_1) + (x_4)^3 \end{bmatrix}.$$

Solution: We programmed up (10.45) and (10.46) in Julia and used a symmetric difference approximation for the derivatives, with $h = 0.1$. Below are the first five results from the algorithm:

$$x_k = \left[\begin{array}{cccccc} k=0 & k=1 & k=2 & k=3 & k=4 & k=5 \\ -2.0000 & -3.0435 & -2.4233 & -2.2702 & -2.2596 & -2.2596 \\ 3.0000 & 2.5435 & 1.9233 & 1.7702 & 1.7596 & 1.7596 \\ 3.1416 & 0.6817 & 0.4104 & 0.3251 & 0.3181 & 0.3181 \\ -1.0000 & -1.8580 & -2.0710 & -1.7652 & -1.6884 & -1.6846 \end{array}\right]$$

and

$$f(x_k) = \left[\begin{array}{rrrrrr} k=0 & k=1 & k=2 & k=3 & k=4 & k=5 \\ -39.0000 & -6.9839 & -1.1539 & -0.0703 & -0.0003 & -0.0000 \\ -36.0000 & -6.9839 & -1.1539 & -0.0703 & -0.0003 & -0.0000 \\ 2.9335 & 0.1447 & 0.0323 & 0.0028 & 0.0000 & -0.0000 \\ 15.3674 & -5.1471 & -4.0134 & -0.7044 & -0.0321 & -0.0001 \end{array}\right].$$

By iteration five, we have a good approximation of a root because $||f(x_5)|| \approx 10^{-4}$. We also provide the Jacobians at the initial and final steps,

$$\frac{\partial f(x_0)}{\partial x} = \begin{bmatrix} -19.0000 & -42.0000 & 0.0000 & 0.0000 \\ -17.0000 & -40.0000 & 0.0000 & 0.0000 \\ 0.4539 & 0.0000 & 1.0000 & 0.0000 \\ 7.4782 & 10.9116 & 0.0000 & 3.0100 \end{bmatrix} \text{ and } \frac{\partial f(x_5)}{\partial x} = \begin{bmatrix} -8.5577 & -15.1155 & 0.0000 & 0.0000 \\ -6.5577 & -13.1155 & 0.0000 & 0.0000 \\ 0.3854 & 0.0000 & 0.9910 & 0.0000 \\ 3.9296 & 5.4337 & 0.0000 & 8.5616 \end{bmatrix}$$

so that it is clear that as x_k evolves, so does the Jacobian of f at x_k. ■

10.7 (Optional Read): From the Gradient or Jacobian of a Function to its Linear Approximation

This covers the same material as in Chap. 10.5, but in reverse order. Some may find it more digestible.

Consider a function $f : \mathbb{R}^m \to \mathbb{R}^n$. We seek a means to build a linear approximation of the function near a given point $x_0 \in \mathbb{R}^m$. When $m = n = 1$, we were able to approximate a function by

$$f(x) \approx f(x_0) + \frac{df(x_0)}{dx}(x - x_0).$$

In the above, $\frac{df(x_0)}{dx}$ is a scalar. For reasons that will become clear shortly, let's denote that scalar by $a := \frac{df(x_0)}{dx}$, so that we can rewrite the linear approximation as

$$f(x) \approx f(x_0) + a(x - x_0). \tag{10.48}$$

We do this and note that a can be viewed as a 1×1 matrix, that is, an $n \times m$ matrix for $n = m = 1$. We now ask the question, for n and m not necessarily equal to one, and for $f : \mathbb{R}^m \to \mathbb{R}^n$, can we find an $n \times m$ matrix A such that

$$f(x) \approx f(x_0) + A(x - x_0). \tag{10.49}$$

The answer is (mostly) yes. To do this, we need to extend the notion of a derivative to the case of vectors. We do this first for $n = 1$ and general $m \geq 1$.

10.7.1 The Gradient

We restrict ourselves to functions $f : \mathbb{R}^m \to \mathbb{R}$. Hence, for $x \in \mathbb{R}^m$, we have $f(x) \in \mathbb{R}$ and we will make the components of x explicit in the function by writing $f(x) = f(x_1, \ldots, x_m)$. One obvious way to extend our notion of a derivative is to perturb the components of x one at time. In Calculus, these are called **partial derivatives**. We won't try to justify the terminology; it is what it is.

We continue to use $h \in \mathbb{R}$ to denote a small non-zero real number. With this notation, we define the **partial derivative of f with respect to $\mathbf{x_i}$ at a point $\mathbf{x_0}$** to be

$$\frac{\partial f(x_0)}{\partial x_i} \approx \frac{f(x_{01}, \ldots, \mathbf{x_{0i}} + \mathbf{h}, \ldots, x_{0m}) - f(x_{01}, \ldots, \mathbf{x_{0i}}, \ldots, x_{0m})}{\mathbf{h}}, \tag{10.50}$$

where we have highlighted that the increment is applied to x_i and only to x_i. What we are doing is holding constant all coordinates except the i-th one, and then applying the "usual" definition of a derivative of a function that depends on the scalar variable x_i.

Equation (10.50) is a **forward difference approximation** of the partial derivative with respect to x_i about the point x_0. Just as with our previous treatment of the derivative, we can use a backward approximation or a **symmetric difference approximation**, such as

$$\frac{\partial f(x_0)}{\partial x_i} \approx \frac{f(x_{01}, \ldots, \mathbf{x_{0i}} + \mathbf{h}, \ldots, x_{0m}) - f(x_{01}, \ldots, \mathbf{x_{0i}} - \mathbf{h}, \ldots, x_{0m})}{\mathbf{2h}}. \tag{10.51}$$

Example 10.10 *Compute (approximately) the partial derivatives of $f(x_1, x_2) = x_1 \cos(x_2)$ with respect to both x_1 and x_2 about the point $x_0 = [2 \;\; \pi/4]^\top$.*

Solution: Let's get the formulas for a general $h > 0$ and then we'll build a Table comparing the results for several values of h. For the partial derivative with respect to x_1, we perturb x_1 abut 2 while holding x_2 constant and equal to $\pi/4$. This gives

$$\begin{aligned}\frac{\partial f(x_0)}{\partial x_1} &\approx \frac{f(2+h, \pi/4) - f(2-h, \pi/4)}{2h} \\ &= \frac{(2+h)\cos(\pi/4) - (2-h)\cos(\pi/4)}{2h} \\ &= \frac{2h\cos(\pi/4)}{2h} \\ &= \cos(\pi/4) = \frac{\sqrt{2}}{2},\end{aligned}$$

which is independent of h and hence there is nothing more to compute. For the next partial derivative, we perturb x_2 about $\pi/4$ while holding x_1 constant and equal to 2. This gives

$$\begin{aligned}\frac{\partial f(x_0)}{\partial x_2} &\approx \frac{f(2, \pi/4 + h) - f(2, \pi/4 - h)}{2h} \\ &= \frac{2\cos(\pi/4 + h) - 2\cos(\pi/4 - h)}{2h},\end{aligned} \tag{10.52}$$

which, though we could simply it with some trigonometric identities, we'll stop here and turn to Julia. Doing so, leads to the following results

$\frac{f(2,\pi/4+h)-f(2,\pi/4-h)}{2h}$	h
-1.41421356001592	0.0001
-1.414213326670799	0.001
-1.4141899922649026	0.01
-1.4118577179998826	0.1
-1.4048043101898116	0.2
-1.3768017528243548	0.4

(10.53)

The true answer is

$$\frac{\partial f(2, \pi/4)}{\partial x_2} = -\sqrt{2} \approx -1.4142135623730951$$

■

Knowledge is Power

For root finding, an accuracy of a few percent is probably good enough. That said, **learning how to compute the partial derivatives analytically will make your code faster and will eliminate the question of how to choose the small perturbation parameter $h > 0$.**

The Gradient and Linear Approximations

The **gradient** of $f : \mathbb{R}^m \to \mathbb{R}$ is simply the partial derivatives of f with respect to x_i arranged to form a **row vector**,

$$\nabla f(x_0) := \left[\begin{array}{cccc} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{array} \right], \tag{10.54}$$

which we can also call a $1 \times m$ matrix. The cool symbol ∇ is usually pronounced as "**grad**" and one says "**grad f**" when speaking of ∇f.

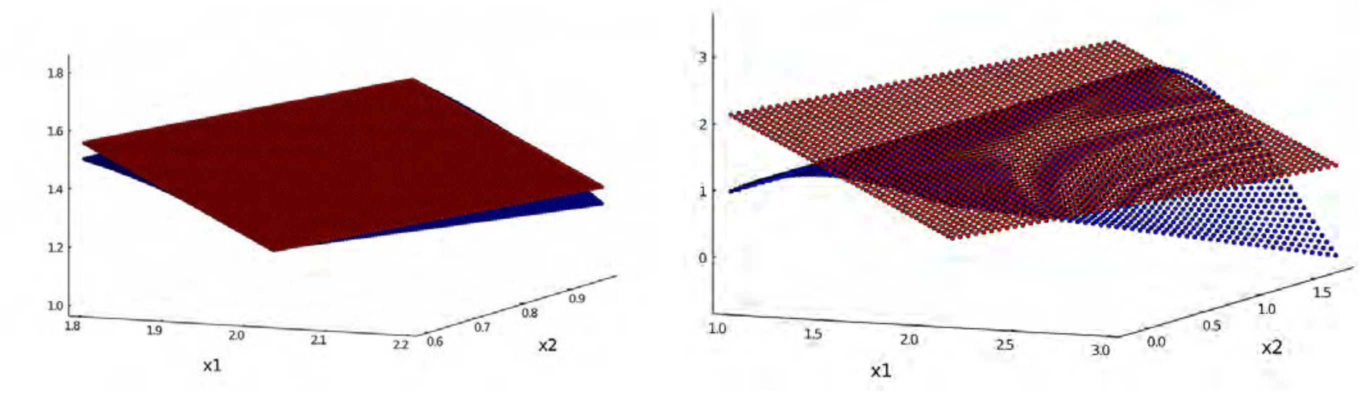
An important use of the gradient of a function of several variables is to form a **linear approximation of the function about a point**

$$f(x) \approx f(x_0) + \nabla f(x_0)(x - x_0). \tag{10.55}$$

Comparing this to (10.49), we see that $A = \nabla f(x_0)$, a $1 \times m$ matrix. Expanding (10.55) into its components gives

$$\begin{aligned} f(x) &\approx f(x_0) + \nabla f(x_0)(x - x_0) \\ &= f(x_0) + \underbrace{\left[\begin{array}{cccc} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{array} \right]}_{A} \cdot \underbrace{\left[\begin{array}{c} x_1 - x_{01} \\ x_2 - x_{02} \\ \vdots \\ x_m - x_{0m} \end{array} \right]}_{(x - x_0)} \\ &= f(x_0) + \sum_{i=1}^{m} \frac{\partial f(x_0)}{\partial x_i}(x_i - x_{0i}). \end{aligned} \tag{10.56}$$

The linear approximation in (10.56) looks just like our linear approximation for functions of a single variable x, namely $f(x) \approx f(x_0) + \frac{df(x_0)}{dx}(x - x_0)$, where $a = \frac{df(x_0)}{dx}$ is 1×1.



(a)

(b)

Figure 10.11: (a) Close up view. (b) A more global view. The function $f(x_1, x_2) = x_1 \cos(x_2)$ is plotted in blue, while in red is shown its linear approximation about the point $x_0 = [2 \;\; \pi/4]^\top$, that is, $f_{\text{lin}}(x) := f(x_0) + \nabla f(x_0)(x - x_0)$. This approximation can be done for any point x_0 at which the partial derivatives exists. In Calculus, the red plane is also called the **tangent plane at $\mathbf{x_0}$**. These linear approximations accurately represent the nonlinear function in a small enough region about a given point, allowing us to use our Linear Algebra skills.

Example 10.11 *Compute a linear approximation of $f(x_1, x_2) = x_1 \cos(x_2)$ at the point $x_0 = [2 \;\; \pi/4]^\top$.*

Solution: From Example 10.10, we know both of the partial derivatives of f at the point $x_0 = [2 \;\; \pi/4]^\top$. Hence, we have

$$\begin{aligned} f(x) &\approx f(x_0) + \nabla f(x_0)(x - x_0) \\ &= f(x_0) + \left[\begin{array}{cc} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} \end{array} \right] \left[\begin{array}{c} x_1 - x_{01} \\ x_2 - x_{02} \end{array} \right] \\ &= 2\cos(\pi/4) + \left[\begin{array}{cc} \frac{\sqrt{2}}{2} & -\sqrt{2} \end{array} \right] \left[\begin{array}{c} x_1 - 2 \\ x_2 - \pi/4 \end{array} \right] \end{aligned} \tag{10.57}$$

Figure 10.11 compares the linear approximation to the nonlinear function in a region about x_0. ■

10.7.2 Expanding on Vector Notation

We have carefully defined derivatives and partial derivatives of functions at given points. We used the notation x_0 for the given point to make it seem like some concrete value, a fixed scalar in $\mathbb{R}$ of vector in $\mathbb{R}^m$. However, in practice, such as in Newton's Algorithm, we keep updating the point x_0 at which we are computing derivatives and linear approximations of functions. Hence, because the point is not really fixed, we can just call it x. Then, for $f : \mathbb{R} \to \mathbb{R}$, we have

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x-h)}{2h}, \tag{10.58}$$

and for $f : \mathbb{R}^m \to \mathbb{R}$, where $x = (x_1, x_2, \dots, x_m)$, we have

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x_1, \dots, x_i + h, \dots, x_m) - f(x_1, \dots, x_i - h, \dots, x_m)}{2h}. \tag{10.59}$$

With this notation in mind, we now define **derivatives and partial derivatives for vector valued functions**. When $f : \mathbb{R} \to \mathbb{R}^n$, we have that x is a scalar and $f(x)$ is a vector with n components, as in

$$f(x) = \left[\begin{array}{c} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{array} \right]. \tag{10.60}$$

We define its derivative with respect to the scalar variable x as

$$\frac{df(x)}{dx} := \left[\begin{array}{c} \frac{df_1(x)}{dx} \\ \frac{df_2(x)}{dx} \\ \vdots \\ \frac{df_n(x)}{dx} \end{array} \right], \tag{10.61}$$

by differentiating each of the components of $f(x)$. It's the obvious thing to do. The key is that you must keep track that when $f(x)$ is vector valued, so is its derivative, $\frac{df(x)}{dx}$.

In terms of our symmetric difference approximation to the derivative, the formula,

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x-h)}{2h}, \tag{10.62}$$

is still valid and is even written in Julia the same way! Yes, you could re-write the above as

$$\frac{df(x)}{dx} \approx \left[\begin{array}{c} \frac{f_1(x+h) - f_1(x-h)}{2h} \\ \frac{f_2(x+h) - f_2(x-h)}{2h} \\ \vdots \\ \frac{f_n(x+h) - f_n(x-h)}{2h} \end{array} \right], \tag{10.63}$$

but that's a lot of extra programming. The expression (10.62) is much more compact and convenient. This is the power of good notation.

Similarly, when $f : \mathbb{R}^m \to \mathbb{R}^n$, we know that x is a vector with m components and $f(x)$ is a vector with n components, as in

$$f(x) = \begin{bmatrix} f_1(x_1, \ldots, x_m) \\ f_2(x_1, \ldots, x_m) \\ \vdots \\ f_n(x_1, \ldots, x_m) \end{bmatrix}. \tag{10.64}$$

We define its partial derivative with respect to component x_i as

$$\frac{\partial f(x)}{\partial x_i} := \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_i} \\ \frac{\partial f_2(x)}{\partial x_i} \\ \vdots \\ \frac{\partial f_n(x)}{\partial x_i} \end{bmatrix}, \tag{10.65}$$

by doing the partial differentiation of each of the components of $f(x)$. It's the obvious thing to do.

In terms of our symmetric difference approximation to the derivative, the formula,

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x_1, \ldots, x_i + h, \ldots, x_m) - f(x_1, \ldots, x_i - h, \ldots, x_m)}{2h}, \tag{10.66}$$

is still valid and is even written in Julia the same way! Once again, you could re-write the above as

$$\frac{\partial f(x)}{\partial x_i} \approx \begin{bmatrix} \frac{f_1(x_1,\ldots,x_i+h,\ldots,x_m)-f_1(x_1,\ldots,x_i-h,\ldots,x_m)}{2h} \\ \frac{f_2(x_1,\ldots,x_i+h,\ldots,x_m)-f_2(x_1,\ldots,x_i-h,\ldots,x_m)}{2h} \\ \vdots \\ \frac{f_n(x_1,\ldots,x_i+h,\ldots,x_m)-f_n(x_1,\ldots,x_i-h,\ldots,x_m)}{2h} \end{bmatrix}, \tag{10.67}$$

but that's a lot of extra programming. The expression (10.66) is much more compact and convenient. **This is again the power of good notation. But to use the notation effectively, we have to do the bookkeeping and remember that the innocent expression**

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x_i}}$$

is really a vector with n components.

10.7.3 The Jacobian

We now turn to functions $f : \mathbb{R}^m \to \mathbb{R}^n$. Based the compact notation covered in Chapter **??**, we can define the **Jacobian of a function** as

$$\frac{\partial f(x)}{\partial x} := \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \frac{\partial f(x)}{\partial x_2} & \cdots & \frac{\partial f(x)}{\partial x_m} \end{bmatrix}. \tag{10.68}$$

Comparing the above to (10.54), we see that the **Jacobian** of a function $f : \mathbb{R}^m \to \mathbb{R}$ is the **gradient** of the function. **This is the same as saying an $n \times m$ matrix reduces to a row vector when $n = 1$.**

We need to keep in mind that, for each value of $x \in \mathbb{R}^m$, the compact and innocent looking object

$$\frac{\partial f(x)}{\partial x}$$

is really an $n \times m$ matrix. When computing it numerically, we typically build it up one column at a time as in the following Julia code.

```
F(x1,x2,x3)=[x1.*x2.*x3; log.(2+cos.(x1)) .+ x2.^x1; x1.*x3/(1 .+ x2.^2)]
h=0.01
x0=[pi;1.0;2.0]
dfdx1 =(F(x0[1]+h,x0[2],x0[3])-F(x0[1]-h,x0[2],x0[3]))/(2*h)
```

```
dfdx2 =(F(x0[1],x0[2]+h,x0[3])-F(x0[1],x0[2]-h,x0[3]))/(2*h)
dfdx3 =(F(x0[1],x0[2],x0[3]+h)-F(x0[1],x0[2],x0[3]-h))/(2*h)
dfdx=[dfdx1 dfdx2 dfdx3]

3x3 Array{Float64,2}:
 2.0   6.28319  3.14159
 0.0   3.14172  0.0
 1.0  -3.14159  1.5708

```

Just for the record, we will write out $\frac{\partial f(x)}{\partial x}$ as an $n \times m$ matrix. In case you are wondering, your instructors never do this when doing real robotics! We use the "vector version" of the Jacobian where we compute each column of the matrix. For $f : \mathbb{R}^m \to \mathbb{R}^m$,

$$\frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_m} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_m} \end{bmatrix}. \tag{10.69}$$

Hence, the ij component of $\frac{\partial f(x)}{\partial x}$ is $\frac{\partial f_i(x)}{\partial x_j}$. If one wishes, the Jacobian can be computed one element at time via

$$\frac{\partial f_i(x)}{\partial x_j} \approx \frac{f_i(x_1, \ldots, x_j + h, \ldots, x_m) - f_i(x_1, \ldots, x_j - h, \ldots, x_m)}{2h}. \tag{10.70}$$

From slopes of lines→slopes of functions at points → $\mathbf{\frac{df(x_0)}{dx}}$ → $\nabla \mathbf{f(x_0)}$ → $\mathbf{\frac{\partial f(x_0)}{\partial x}}$

Derivatives, gradients, and Jacobians are all generalizations of the notion of the slope of a line being its "rise over run". The derivative of a function $f : \mathbb{R} \to \mathbb{R}$ at a point x_0 is the "local" slope of the function at that point. We compute it by "rise over run", where, for example

$$\text{slope} = \frac{f(x_0 + h) - f(x_0)}{h} \xrightarrow[h>0 \text{ small}]{} \frac{df(x_0)}{dx}, \tag{10.71}$$

and to make it local, we take $h > 0$ small. The gradient recognizes that a function $f : \mathbb{R}^m \to \mathbb{R}$ has a local slope in the x_1-direction, the x_2-direction, all the way up to the x_m-direction. If we let $\{e_1, e_2, \ldots, e_m\}$ be the natural basis vectors for $\mathbb{R}^m$, then we compute each component of the gradient by

$$\text{slope}_j = \frac{f(x_0 + he_j) - f(x_0)}{h} \xrightarrow[h>0 \text{ small}]{} \frac{\partial f(x_0)}{\partial x_j}, \tag{10.72}$$

and we assemble the gradient by

$$\nabla f(x_0) := \begin{bmatrix} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{bmatrix}. \tag{10.73}$$

Finally, for $f : \mathbb{R}^m \to \mathbb{R}^n$, each of the n components of $f(x)$ has a local slope with respect to each component of x. The bookkeeping is easiest if we leave $f(x)$ as a vector and write the Jacobian so that it **looks just like** the gradient,

$$\frac{\partial f(x_0)}{\partial x} := \begin{bmatrix} \frac{\partial f(x_0)}{\partial x_1} & \frac{\partial f(x_0)}{\partial x_2} & \cdots & \frac{\partial f(x_0)}{\partial x_m} \end{bmatrix}, \tag{10.74}$$

but now, because $f(x)$ is an n-vector, we have

$$\begin{bmatrix} \text{slope}_{1j} \\ \vdots \\ \text{slope}_{ij} \\ \vdots \\ \text{slope}_{nj} \end{bmatrix} = \frac{f(x_0 + he_j) - f(x_0)}{h} \xrightarrow[h>0 \text{ small}]{} \frac{\partial f(x_0)}{\partial x_j}. \tag{10.75}$$

Compact Way to Numerically Approximate the Jacobian

If we let $\{e_1, e_2, \dots, e_m\}$ be the natural basis vectors for $\mathbb{R}^m$, then a more compact notation allows us to compute each column of the Jacobian by

$$\frac{\partial f(x_0)}{\partial x_j} = \frac{f(x_0 + he_j) - f(x_0)}{h}. \tag{10.76}$$

Typically, this is easier to program up than (10.70). But of course, your experience may vary! For a symmetric difference, we'd use

$$\frac{\partial f(x_0)}{\partial x_j} = \frac{f(x_0 + he_j) - f(x_0 - he_j)}{2h}. \tag{10.77}$$

10.7.4 Linear Approximation of Vector Valued Functions

Our goal remains to build linear approximations of the form $f(x) \approx f(x_0) + A(x - x_0)$. Just as with our previous investigations, the matrix A is associated with derivatives of the function. In fact, we have

$$f(x) \approx f(x_0) + \underbrace{\frac{\partial f(x_0)}{\partial x}}_{A}(x - x_0), \tag{10.78}$$

in other words, $A := \left.\frac{\partial f(x)}{\partial x}\right|_{x=x_0}$.

Example 10.12 *For the function*

$$f(x_1, x_2, x_3) := \begin{bmatrix} x_1 x_2 x_3 \\ \log(2 + \cos(x_1)) + x_2^{x_1} \\ \frac{x_1 x_3}{1+x_2^2} \end{bmatrix}, \tag{10.79}$$

compute its Jacobian at the point

$$x_0 = \begin{bmatrix} \pi \\ 1.0 \\ 2.0 \end{bmatrix}$$

and evaluate the "accuracy" of its linear approximation.

Solution Using Symmetric Differences, the Jacobian at x_0 is

$$A = \frac{\partial f(x_0)}{\partial x} \approx \begin{bmatrix} 2.00 & 6.28 & 3.14 \\ 0.00 & 3.14 & 0.00 \\ 1.00 & -3.14 & 1.57 \end{bmatrix} \tag{10.80}$$

Figure 10.11 is the limit of what we can show in plots. Another way to think about the question of assessing the quality of the linear approximation is to measure the error defined as

$$e(x) := ||f(x) - f_{\text{lin}}(x)||,$$

where $f_{\text{lin}}(x) := f(x_0) + \frac{\partial f(x_0)}{\partial x}(x - x_0)$. We will seek to estimate the maximum value of $e(x)$ over a region containing the point x_0. Define

$$S(x_0) := \{x \in \mathbb{R}^3 \mid \ |x_i - x_{0i}| \le d, i = 1, 2, 3\}$$

and

$$\text{Max Error} := \max_{x \in S(x_0)} e(x) = \max_{x \in S(x_0)} ||f(x) - f_{\text{lin}}(x)||. \tag{10.81}$$

The for $d = 0.25$, we used a "random search" routine and estimate that

$$\text{Max Error} = 0.12.$$

To put this into context,

$$\max_{x \in S(x_0)} ||f(x)|| = 8.47,$$

and thus the relative error is about 1.5%. ■

If you made it to here, you should loop back to Chap. 10.6.

10.8 Looking Ahead

We've seen that an interesting idea from Calculus, called the derivative, allows nonlinear functions to be approximated by linear functions. The matrices resulting from a derivative, gradient, or Jacobian were instrumental in building algorithms to find roots of nonlinear equations.

In this next Chapter, we'll use the derivative and the gradient to understand algorithms for finding a minimum or a maximum of a function. We'll be able to greatly extend the ideas we developed in Chapters 8.2 for least squared error solutions to $Ax = b$ when it had no solutions, Chapter 8.3 for regression of functions to data, and Chapter 9.9, where we were able to identify a unique solution of minimum norm when $Ax = b$ had an infinite number of solutions.

Chapter 11

Changing Gears Again: Basic Ideas of Optimization

Learning Objectives

- Mathematics is used to describe physical phenomena, pose engineering problems, and solve engineering problems. We close our Y1-introduction to Computational Linear Algebra by showing how linear algebra and computation allow you to use a notion of "optimality" as a criterion for selecting among a set of solutions to an engineering problem.

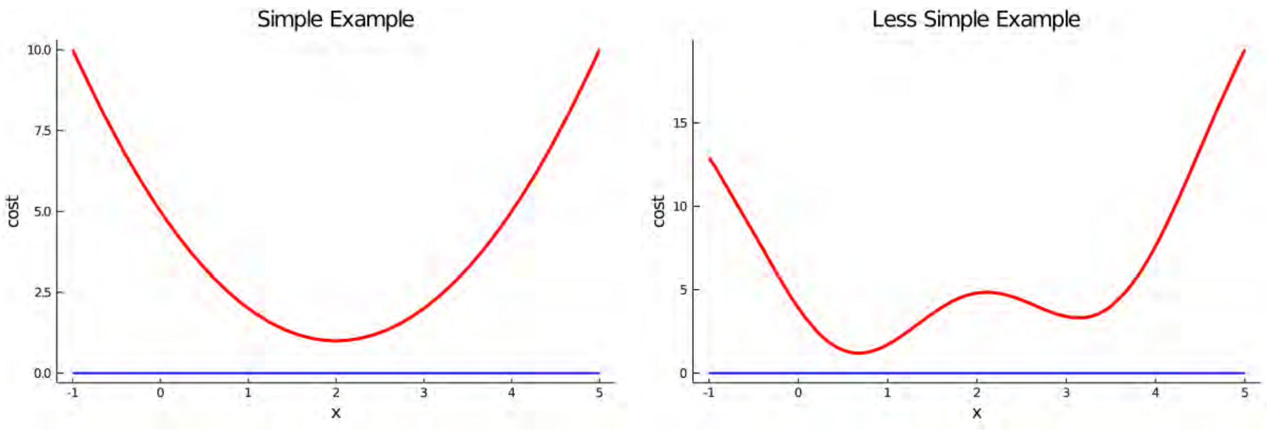
Outcomes

- Arg min should be thought of as another function in your toolbox,

$$x^* = \underset{x \in \mathbb{R}^m}{\arg\min}\, f(x).$$

- Extrema of a function occur at places where the function's first derivative vanishes.
- The gradient of a function points in the direction of maximum rate of growth.
- We will add to our knowledge of derivatives, specifically, second derivative.
- Second-order optimization methods are based on root finding
- Convex functions have global minima
- Quadratic programs are special kinds of least squares problems
- Optimization packages in Julia provide amazing tools for optimizing functions

11.1 Motivation and Basic Ideas



(a)

(b)

Figure 11.1: (a) A "simple" cost function with a global minimum at $x^* = 2$ and (b) a "less simple" cost function where there are two local minima, one at $x^* \approx 0.68$ and one at $x^* \approx 3.14$, as well as a local maximum at ≈ 2.1.

Optimization is the process of finding one or more values $x \in \mathbb{R}^m$ that minimize a function $f : \mathbb{R}^m \to \mathbb{R}$, where the scalar-valued function f is called the **cost function**. The cost should be minimum at a point x^* that is of interest to you, it should be "small" for values of x that are "near" x^* and it should be "large" for values of x that are "far" from your preferred value, x^*. In Machine Learning, a cost function for minimization is also called a **regret function** in the sense that you regret being far from your preferred value, x^*, and your regret is minimum at x^*. We will abuse notation[1] and write

$$x^* = \arg\min_{x \in \mathbb{R}^m} f(x) \tag{11.1}$$

to denote the value of x achieving the minimum, even when there may be more than one such x.

One can also do the opposite of minimization, which is to **maximize a cost function**. In this case, the Machine Learning community calls the cost function a **reward**, because hey, who does not like to maximize reward and minimize regret! We will stick to minimization throughout the Chapter until the very end, when we'll discuss briefly how maximization and minimization are very closely related.

When $m = 1$, that is, the cost function depends on a scalar variable, it is easy to graphically understand what minimization is all about. Figure 11.1a shows a "bowl-shaped" function that has a minimum at $x^* = 2$. Moreover, if the function outside of the interval $[-1, 5]$ continues growing as shown, then $x^* = 2$ is a **global minimum**, meaning that for all $x \neq x^*$,

$$f(x^*) < f(x).$$

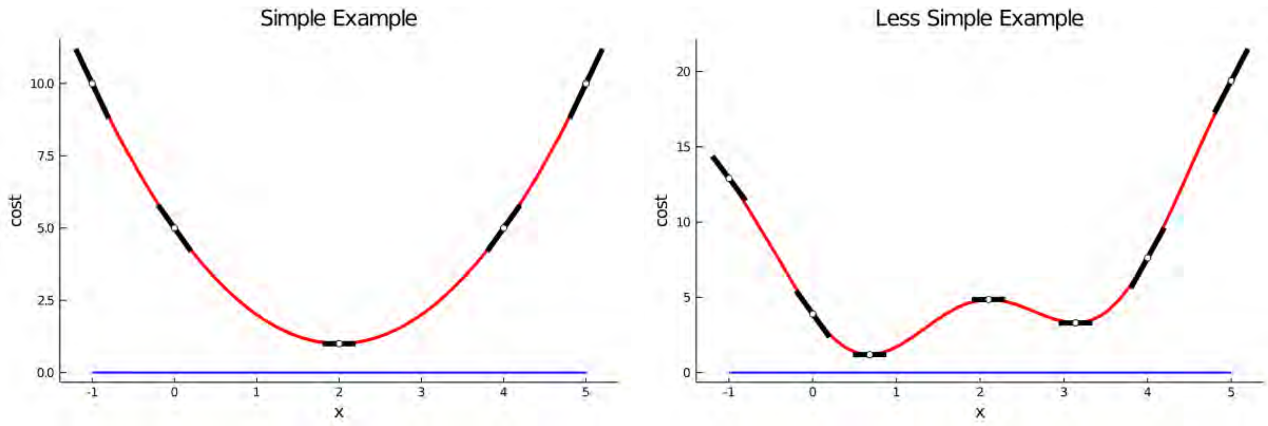
Even more special, if you imagine setting a marble on the curve at any point other than $x^* = 2$, you can easily imagine the marble rolling and settling at the global minimum (assuming some friction).

On the other hand, Fig. 11.1b presents a more complicated situation, where if the function outside of the interval $[-1, 5]$ continues growing as shown, then there is a global minimum at $x_a^* \approx 0.68$, but if you set a marble near $x = 4$, you can image it getting stuck at the local bowl at $x_b^* \approx 3.14$, while if you start the marble at $x = 1.5$, you can imagine it settling in at the local bowl $x_a^* \approx 0.68$. The local bowls in the cost function are called **local minima** in that for all x sufficiently near one of the x_i^*,

$$f(x_i^*) \leq f(x),$$

for $i \in \{a, b\}$.

[1] Abusing notation means that one is being a bit sloppy with one's use of symbols, which is what notation is! One often abuses notation when doing the right thing is a bit painful and would cause more of a distraction to explain the good notation than to caution about using poor notation!



(a) (b)

Figure 11.2: The derivatives of the cost functions have been added at strategic points. (a) A “simple” cost function with a global minimum at $x^* = 2$ and (b) a “less simple” cost function where there are two local minima, one at $x^* \approx 0.68$ and one at $x^* \approx 3.14$, and a local maximum at 2.1.

Derivative of the Cost Tells You How to Move Toward a Local Minimum

In Fig. 11.2, we have plotted the derivatives of the cost functions at several points. Recall that the slope of the line segment at each point is equal to the derivative at that point. The plots tell us some really important things:

- the derivative of the cost is zero at each local minimum;
- to the left of a local minimum, the derivative is negative (slope angles downward), while to the right of a local minimum, the derivative is positive (slope angles upward);
- hence, if you are at a point x_k, defining the next value as

$$x_{k+1} = x_k - s\frac{df(x_k)}{dx}, \tag{11.2}$$

  where $s > 0$ is called the **step size**, moves you in the direction of a local minimum (of course, if $s > 0$ is too large, you can overshoot the local minimum);
- because the derivative is zero at a local minimum, we can use the value of the derivative as a stopping criterion for the algorithm in (11.2); and
- the derivative also vanishes at local maxima, and hence if you start the algorithm in (11.2) at a local maxima, you will be stuck there!

Example 11.1 *Implement* (11.2) *in Julia to find local minima of* $f : \mathbb{R} \to \mathbb{R}$ *by*

$$f(x) = (x-2)^2 + 5(\sin(x-2))^2 + 0.03(x+1)^3 + 4.$$

Solution: We wrote the Julia code given below and selected a set of initial conditions

$$x_0 \in \left\{ \begin{array}{cccccccc} -1.00 & 0.00 & 0.68 & 2.10 & 2.15 & 3.14 & 4.00 & 5.00 \end{array} \right\} \tag{11.3}$$

for the Algorithm (11.2) such that it would be started to the left and right of local minima and very close to the local maxima. The results are reported in (11.4) for a step size of $h = 0.1$ and in (11.5) for a step size of $h = 0.2$. The Algorithm converged quickly in all cases for a step size of $h = 0.1$ and mostly failed for a step size of $h = 0.2$ (we set a limit of 10^3 iterations before terminating). **You have to chose your step sizes carefully!** for $h = 0.1$, when started just to the left of the local maximum, the algorithm converged to the local minimum at 0.68 and when started just to the right of the local maximum, it coverged to the local minimum at 3.14.

```
#Optimization
```

```
cost2(x)=(x.-2).^2 .+ 1 .- 5*(sin.(x.-2)).^2 .+ 3 .+ 0.03*(x.+1).^3
titre="Less Simple Example"
p2=plot(cost2,xmin,xmax,legend=false, title=titre, linewidth=3, color=:red )
s=0.2
delta=0.01
x0=[-1;0;0.68;2.1;2.15;3.14;4;5]
y0=cost2(x0)
IntermediateValues=zeros(1,6)
for k =1:length(x0) #try various initial values of x0
    xk=x0[k]
   dcostdxk =( cost2(xk+delta)-cost2(xk-delta) )/(2*delta)
   fk=cost2(xk)
    j=0
    while (abs(dcostdxk)>1e-5)&(j < 1e3)
        j=j+1
        xk=xk-s*dcostdxk
        dcostdxk =( cost2(xk+delta)-cost2(xk-delta) )/(2*delta)
        fk=cost2(xk)
    end
    IntermediateValues=[IntermediateValues; j s x0[k] xk fk dcostdxk]
end
display(IntermediateValues)
# Show how to get latex code for printing matrices
using Latexify
set_default(fmt = "%.4e", convert_unicode = false)
latexify(IntermediateValues) |> print
```

N0. Iterations	$\mathbf{s}$	$\mathbf{x_0}$	$\mathbf{x^*}$	$\mathbf{f(x^*)}$	$\mathbf{\frac{df(x^*)}{dx}}$
$6.0000e+00$	$1.0000e-01$	$-1.0000e+00$	$6.7838e-01$	$1.1926e+00$	$-1.1470e-06$
$6.0000e+00$	$1.0000e-01$	$0.0000e+00$	$6.7838e-01$	$1.1926e+00$	$-8.7372e-06$
$4.0000e+00$	$1.0000e-01$	$6.8000e-01$	$6.7838e-01$	$1.1926e+00$	$2.4789e-06$
$\mathbf{1.5000e+01}$	$\mathbf{1.0000e-01}$	$\mathbf{2.1000e+00}$	$\mathbf{6.7838e-01}$	$\mathbf{1.1926e+00}$	$\mathbf{-1.8652e-06}$
$\mathbf{1.2000e+01}$	$\mathbf{1.0000e-01}$	$\mathbf{2.1500e+00}$	$\mathbf{3.1369e+00}$	$\mathbf{3.3002e+00}$	$\mathbf{-5.8775e-06}$
$4.0000e+00$	$1.0000e-01$	$3.1400e+00$	$3.1369e+00$	$3.3002e+00$	$1.0699e-06$
$7.0000e+00$	$1.0000e-01$	$4.0000e+00$	$3.1369e+00$	$3.3002e+00$	$-1.0170e-06$
$8.0000e+00$	$1.0000e-01$	$5.0000e+00$	$3.1369e+00$	$3.3002e+00$	$-1.0270e-06$

(11.4)

N0. Iterations	$\mathbf{s}$	$\mathbf{x_0}$	$\mathbf{x^*}$	$\mathbf{f(x^*)}$	$\mathbf{\frac{df(x^*)}{dx}}$
FAILED	$2.0000e-01$	$-1.0000e+00$	$1.0516e+00$	$1.8579e+00$	$3.2184e+00$
FAILED	$2.0000e-01$	$0.0000e+00$	$1.0516e+00$	$1.8579e+00$	$3.2184e+00$
FAILED	$2.0000e-01$	$6.8000e-01$	$1.0516e+00$	$1.8579e+00$	$3.2184e+00$
FAILED	$2.0000e-01$	$2.1000e+00$	$4.0793e-01$	$1.6207e+00$	$-3.2184e+00$
$6.5000e+01$	$2.0000e-01$	$2.1500e+00$	$3.1369e+00$	$3.3002e+00$	$9.5055e-06$
$4.7000e+01$	$2.0000e-01$	$3.1400e+00$	$3.1369e+00$	$3.3002e+00$	$-8.7464e-06$
FAILED	$2.0000e-01$	$4.0000e+00$	$4.0793e-01$	$1.6207e+00$	$-3.2184e+00$
$6.9000e+01$	$2.0000e-01$	$5.0000e+00$	$3.1369e+00$	$3.3002e+00$	$-8.6899e-06$

(11.5)

■.

We give next a more more analytical take on our update law in (11.2). Which derivation is better? The intuition from plots or an analysis via linear approximations? The answer depends on how your brain is wired. I would say the best derivation is the one that gives you that “ah ha” moment!

Linear Approximation to the Rescue!

Let's recall our linear approximation to a function $f : \mathbb{R} \to \mathbb{R}$ near a point x_k by

$$f(x) \approx f(x_k) + \frac{df(x_k)}{dx}(x - x_k). \tag{11.6}$$

We seek to define x_{k+1} so that $f(x_{k+1}) < f(x_k)$, that is, $f(x_{k+1}) - f(x_k) < 0$. We define $\Delta x_k := x_{k+1} - x_k$ and note that (11.6) gives

$$f(x_{k+1}) - f(x_k) \approx \frac{df(x_k)}{dx}\Delta x_k. \tag{11.7}$$

If we believe in the approximation (which is fine as long as x_{k+1} is "near" x_k), then we can replace the approximation sign with an equals sign so that

$$f(x_{k+1}) - f(x_k) < 0 \iff \frac{df(x_k)}{dx}\Delta x_k < 0. \tag{11.8}$$

We see that if $\frac{df(x_k)}{dx} = 0$, there is no choice of Δx_k that yields $\frac{df(x_k)}{dx}\Delta x_k < 0$, which is why the derivative vanishing is our stopping criterion for a local extremum. We assume therefore $\frac{df(x_k)}{dx} \neq 0$, in which case

$$\Delta x_k = -s\frac{df(x_k)}{dx} \implies \frac{df(x_k)}{dx}\Delta x_k = -s\left[\frac{df(x_k)}{dx}\right]^2 < 0 \text{ for all } s > 0,$$

and our update law becomes

$$x_{k+1} = x_k + \Delta x_k = x_k - s\frac{df(x_k)}{dx},$$

which agrees with (11.2).

11.2 Contour Plots and the Gradient of the Cost

We acquired some good intuition by looking at plots of cost functions depending on a scalar variable $x \in \mathbb{R}$. We'll now augment our intuition by exploring the case $x \in \mathbb{R}^2$. We'll look at **contour plots of cost functions.** We'll also **encounter an important property of the gradient of a cost function that will generalize to arbitrary dimensions**.

We introduced the norm of a vector as a means to measure the length of vectors. We used the square of the norm in Chapter 8.2 to find a *least squared error solution* to a system of linear equations[2]. In engineering, the square of the norm is a very common cost function. In Fig. 11.3a we plot

$$f(x) := ||x||^2 = (x_1)^2 + (x_2)^2$$

on the z-axis versus x_1 and x_2. It's hard to imagine anything simpler to optimize! From the plot, we see a "generalized bowl shape", though it's hard to clearly identify where the minimum value occurs (yes, we know it occurs at $x = 0$). To the right of the plot, in Fig. 11.3b, we show a **contour plot** of $f(x_1, x_2)$. The circles about the origin show values of $x = (x_1, x_2)$ where the cost function is constant; in the inner circle, $f(x) = ||x||^2 = 1$, then $f(x) = ||x||^2 = 4$, ..., until $f(x) = ||x||^2 = 25$. In the contour plot, it is easy to see where the minimum value occurs.

Similarly, in Fig. 11.3c, we show a plot of

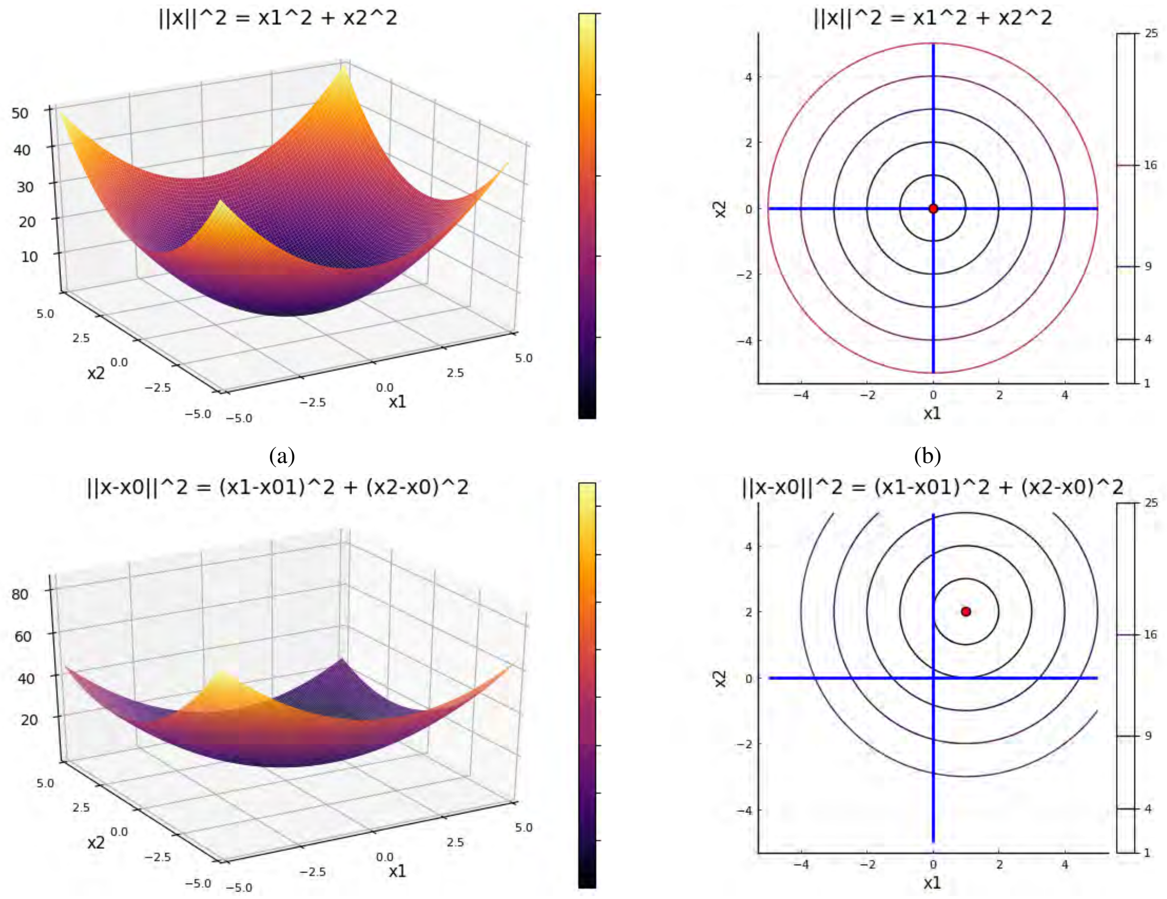
$$f(x) = ||x - x_0||^2 = (x_1 - x_{01})^2 + (x_2 - x_{02})^2 \tag{11.9}$$

for $x_0 = [1; 2]$. In the corresponding contour plot, Fig. 11.3d, it is equally easy to see where the minimum occurs. We will use the contour plots to show the correct direction to follow from a given point x_k so that one moves efficiently toward a value of x^* that is a local minimum.

Recall that the gradient of (11.9) is a row [3] vector of the form $\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \frac{\partial f(x)}{\partial x_2} \end{bmatrix}$. Hence, the transpose of the gradient is a

[2] Recall that used the square of the norm when assessing the error $e := Ax - b$ in equations that do not have exact solutions.

[3] From Calculus, in this case, the exact formula is $\nabla f(x) = [\, 2(x_1 - x_{01}) \;\; 2(x_2 - x_{02}) \,]$



(a)

(b)

(c)

(d)

Figure 11.3: The graphs of two functions are shown in (a) and (c), with their corresponding contour plots shown in (b) and (d). Contour plots are lines where a function is constant. If you have ever used a topographical map when hiking, the lines on those maps indicate constant terrain height. In our case, the lines are constant $z = f(x_1, x_2)$ values as (x_1, x_2) vary. Because the cost function we use here is very simple, its lines of constant contour are circles. More “interesting” examples will follow.

column vector in $\mathbb{R}^2$, namely

$$[\nabla f(x)]^\top = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \end{bmatrix}. \tag{11.10}$$

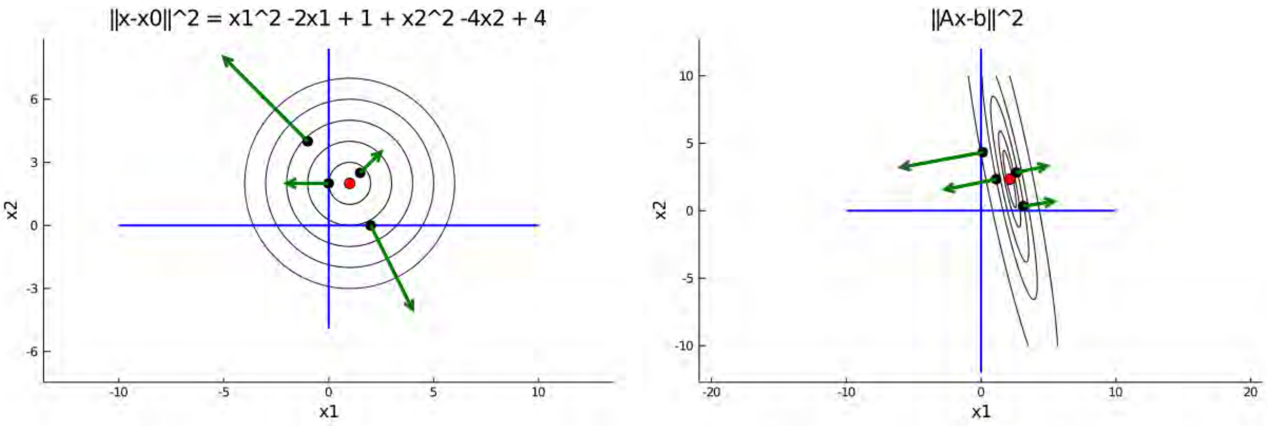
Figure 11.4a shows two contour plots. In addition, we have plotted in green arrows the gradients of the cost functions at several points and overlaid them on the contour plots. In both cases, the gradients are pointing in the direction of maximum increase of the function. It follows that the negative of the gradient is the direction of maximum decrease.

In Fig. 11.4, we could not plot the gradients at the minima because the length of the gradient vector is zero at a (local) minimum! In case you want to see why this is true, from Calculus, one obtains that

$$\nabla\left((x_1 - x_{01})^2 + (x_2 - x_{02})^2\right) = 2(x_1 - x_{01}) + 2(x_2 - x_{02}) \tag{11.11}$$

$$\nabla\left((Ax - b)^\top \cdot (Ax - b)\right) = 2(Ax - b)^\top \cdot A. \tag{11.12}$$

The minimum of $(x_1 - x_{01})^2 + (x_2 - x_{02})^2$ occurs at $x_1^* = x_{01}$ and $x_2^* = x_{02}$, and indeed, the gradient vanishes at x^*. The minimum of $||Ax - b||^2 = (Ax - b)^\top \cdot (Ax - b)$ satisfies $A^\top \cdot Ax^* = A^\top b$, and direct substitution into the corresponding gradient shows that it vanishes as well. While this is not a proof, it gives you a hint that the gradient vanishing at a local minimum may be true.



(a)

(b)

Figure 11.4: Contour Plots plus Gradients. (a) shows a very simple cost function, $||x - [1; 2]||^2$, where the contours of constant cost are circles. (b) shows a more typical situation, where the contours are squeezed in one direction. In both cases, the gradients, plotted in green arrows, are pointing in the direction of maximum increase of the function. Hence, the negative of the gradient is the direction of maximum decrease.

11.3 Gradient Descent

Gradient Descent

Equations (11.11) and (11.12) along with Fig. 11.4 tell us some really important things:

- the gradient vanishes at local minima;
- the gradient points in the direction of fastest growth of the cost function, and the negative of the gradient points in the direction of fastest decrease;
- hence, if you are at a point x_k, defining the next value as

  $$x_{k+1} = x_k - s\left[\nabla f(x_k)\right]^\top, \tag{11.13}$$

  where $s > 0$ is called the **step size**, moves you in the direction of a local minimum (of course, if $s > 0$ is too large, you can overshoot the local minimum);
- because the gradient is zero at a local minimum, we can use the value of the norm of the gradient as a stopping criterion for the algorithm in (11.13); and
- the gradient also vanishes at local maxima, and hence if you start the algorithm in (11.13) at a local maxima, you will be stuck there.

Before doing examples, we'll also show we can derive (11.13) from the linear approximation of $f : \mathbb{R}^m \to \mathbb{R}$ near a point x_k, namely,

$$f(x) \approx f(x_k) + \nabla f(x_k)\,(x - x_k)\,, \tag{11.14}$$

where $\nabla f(x_k)$ is the gradient of f at x_k and is a $1 \times m$ row vector. We seek to define x_{k+1} so that $f(x_{k+1}) < f(x_k)$, that is, $f(x_{k+1}) - f(x_k) < 0$. We define $\Delta x_k := x_{k+1} - x_k$ and note that (11.14) gives

$$f(x_{k+1}) - f(x_k) \approx \nabla f(x_k)\Delta x_k. \tag{11.15}$$

If we believe in the approximation (which is fine as long as x_{k+1} is "near" x_k), then we can replace the approximation sign with an equals sign so that

$$f(x_{k+1}) - f(x_k) < 0 \iff \nabla f(x_k)\Delta x_k < 0. \tag{11.16}$$

We see that if $\nabla f(x_k) = 0$, there is no choice of Δx_k that yields $\nabla f(x_k)\Delta x_k 0 < 0$, which is why the gradient vanishing is our stopping criterion for a local extremum. We assume therefore $\nabla f(x_k) \neq 0$, in which case selecting

$$\Delta x_k = -s\left[\nabla f(x_k)\right]^\top \implies \nabla f(x_k)\Delta x_k = -s||\left[\nabla f(x_k)\right]^\top||^2 < 0 \text{ for all } s > 0.$$

Our update law is then

$$x_{k+1} = x_k + \Delta x_k = x_k - s\left[\nabla f(x_k)\right]^\top,$$

which agrees with (11.13).

Remark: There are easy variations of the Gradient Descent Algorithm, Plestan-2020. Let's note that our key condition is

$$\nabla f(x_k)\Delta x_k < 0.$$

If we define the i-th component of Δx_k by

$$(\Delta x_k)_i := \begin{cases} 0 & \frac{\partial f(x_k)}{\partial x_i} = 0 \\ -s \ \text{sign}\left(\frac{\partial f(x_k)}{\partial x_i}\right) & \text{otherwise} \end{cases},$$

then it is still true that $\nabla f(x_k)\Delta x_k < 0 \iff \nabla f(x_k) \neq 0_{1\times m}$, and thus we are moving in a descent direction.

Example 11.2 *We'll re-visit the least squares problem from Chapter 8.3, and use gradient descent as given in* (11.13) *to solve*

$$x^* = \underset{x\in\mathbb{R}^m}{\arg\min}\, ||Ax - b||^2$$

and compare it to the closed-form solution $\left(A^\top A\right)x^* = A^\top b$, *for*

$$A = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 1.0000 & 0.2500 & 0.0625 \\ 1.0000 & 0.5000 & 0.2500 \\ 1.0000 & 0.7500 & 0.5625 \\ 1.0000 & 1.0000 & 1.0000 \\ 1.0000 & 1.2500 & 1.5625 \\ 1.0000 & 1.5000 & 2.2500 \\ 1.0000 & 1.7500 & 3.0625 \\ 1.0000 & 2.0000 & 4.0000 \end{bmatrix} \quad \textit{and} \quad b = \begin{bmatrix} 1.0000 \\ 1.0000 \\ 1.5000 \\ 2.0000 \\ 3.0000 \\ 4.2500 \\ 5.5000 \\ 7.0000 \\ 10.0000 \end{bmatrix}.$$

Solution: We apply gradient descent as given in (11.13) to the optimization problem,

$$x^* = \underset{x\in\mathbb{R}^3}{\arg\min}\, (Ax - b)^\top (Ax - b).$$

We use symmetric differences to compute the gradient of

$$f(x) := (Ax - b)^\top (Ax - b),$$

set the step size to $s = 0.01$, and the tolerance to $||\nabla f(x_k)|| < 10^{-5}$. After 2,673 iterations, we obtain

$$x^{\text{approx}} = \begin{bmatrix} 1.065144e+00 \\ -6.257368e-01 \\ 2.454536e+00 \end{bmatrix}. \tag{11.17}$$

For comparison purposes, we recall that the true answer is

$$x^* = \begin{bmatrix} 1.065152e+00 \\ -6.257576e-01 \\ 2.454545e+00 \end{bmatrix}. \tag{11.18}$$

■

The true power of optimization becomes clear when we use it to solve problems that do NOT have closed-form solutions. Check out the next section!

11.4 Extrinsic Calibration Using Gradient Descent

Extrinsic calibration is the problem of finding a rotation matrix and a translation vector that allows one to transform the 3D-(x, y, z) points measured by a LiDAR so as to align them with the data captured by a camera. The problem is interesting because it is easy to show visually what one wants to accomplish, but yet, how to formulate the objective mathematically is not clear at first glance, and for sure, there is no obvious "equation" to solve in order to compute a solution.

Here, we'll go into enough detail that we can formulate an optimization problem for "aligning" the scenes observed by the LiDAR and the camera. We'll first address the optimization problem using gradient descent and later, we'll use a more efficient technique that uses "higher-order" information about the cost function. The difference in the rates of convergence is mind blowing.

11.4.1 Introduction

Currently, basic cameras provide many more data points (pixels) for a given surface size at a given distance than even high-end LiDARs. However, cameras rely on the consistency of the ambient lighting and provide less accurate depth estimation. On the other hand, LiDARs give accurate distance measurement, and rapidly changing ambient lighting will not affect measurements of a LiDAR. Due to these different inherent properties of LiDARs and cameras, it is necessary to "fuse" the LiDAR and the camera on Cassie's torso, as shown in Fig. 11.7. By fusing, we mean overlaying a point cloud from a LiDAR to an image from a camera. The process of fusing two or more different types of data from different sensors is called an "extrinsic calibration". The problem requires high accuracy and precision; a little error in the process leads to an unusable result, as shown in Fig. 11.5b.

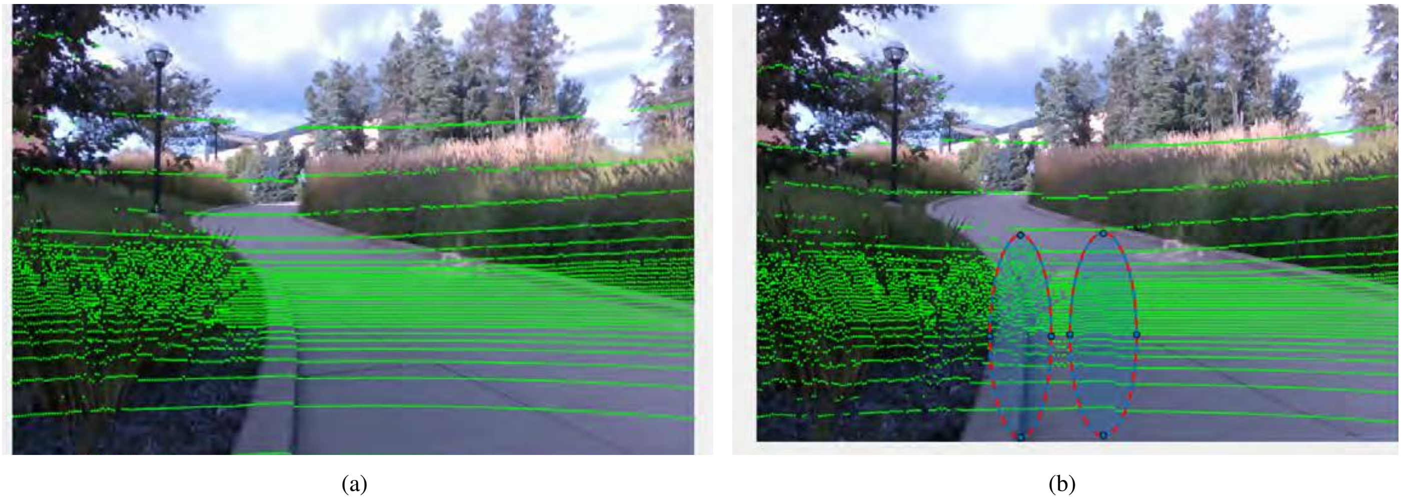


Figure 11.5: a shows good alignment of a LiDAR point cloud projected onto a camera image. b shows that a calibration result is not usable if it has a few degrees of rotation error and a few percent of translation error.

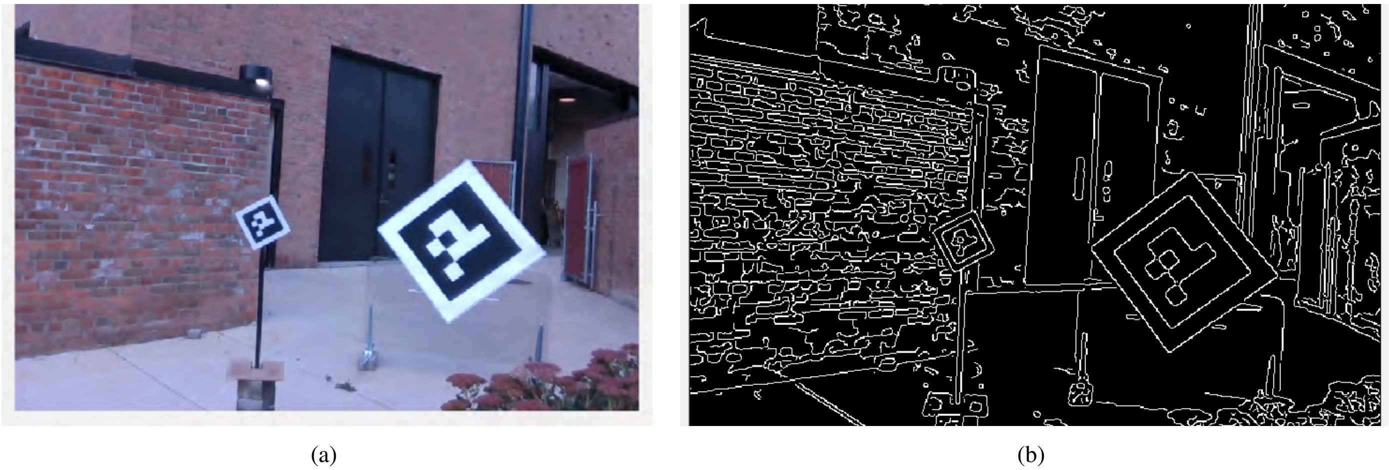


Figure 11.6: a shows a normal image and b illustrates the result of edge detection applied on a.

When we calibrate sensors, we need to find corresponding "features" captured from different sensors. Features are some specific structures in an environment, such as points, edges or corners, etc. Figure 11.6 considers edges as features and extracts them out from an image. Once we find "enough" corresponding features, we want to estimate a "rigid-body transformation" between each sensor that we want to calibrate. A rigid-body transformation H consists of a rotation matrix R and a translation vector t and is defined as

$$H := \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}. \tag{11.19}$$

When calibrating two sensors, it is good practice to indicate how the transformation is defined, by specifying that is a transformation from which sensor to which other sensor! We will use the notation $R^{\text{to sensor2}}_{\text{from sensor1}}$ and $t^{\text{to sensor2}}_{\text{from sensor1}}$ to represent the rotation and translation from the sensor 1 to sensor 2.

In the following, we will illustrate how to perform an extrinsic calibration between a LiDAR and a camera. We will walk you through the calibration process and provide some insight on how we try to take advantage of their relative strengths and avoid their weaknesses. All the images and data are collected from the torso of Cassie Blue, as shown in Fig. 11.5a.

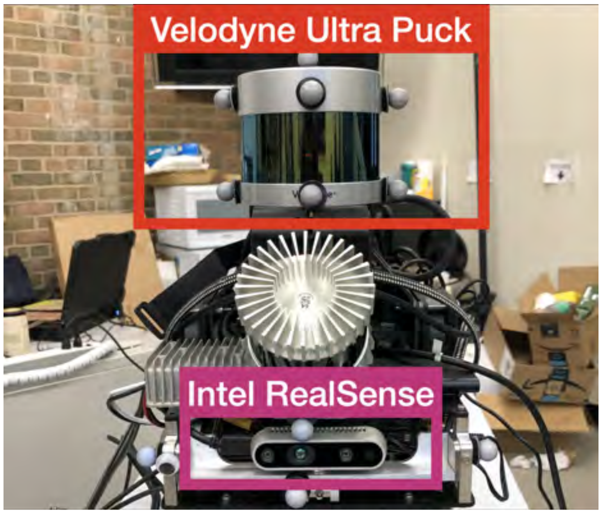


Figure 11.7: This figure shows the torso of Cassie Blue. We use it in practice to do our autonomy adventures!

11.4.2 Problem Setup and Initial Solution

In this problem, we take corners as our features, as shown in Fig. 11.8. In particular, we want to align the LiDAR vertices (green dots) with the camera corners (red dots) in Fig. 11.8b. Additionally, we assume all corresponding corners from the two sensors are already given[4], and the process of feature extraction is uploaded to our YouTube Channel! Let X and Y be the features from a LiDAR and a camera, respectively. When overlaying the two sets of features, we want the their respective coordinates in the camera frame to be as close as possible. Therefore, the problem can be formulated as

$$\begin{aligned} \left(R_L^{C^*}, t_L^{C^*}\right) &:= \underset{R,t}{\arg\min}\, f(R,t,X,Y) \\ &:= \underset{R,t}{\arg\min} \sum_{i=1}^{4n} \|\Pi\left(X_i; R, t\right) - {}_C Y_i\|_2^2, \end{aligned} \tag{11.20}$$

where R_L^C and t_L^C are the rotation and translation from the LiDAR frame to the camera frame, f is the cost function, and Π is a projection map, which we will not dive into in here, but is provided in Appendix D.8. For now, you can consider it as a black box that takes points in 3D space and maps them to their corresponding points on the 2D image-plane of the camera. We apply the gradient descent algorithm to solve (11.20) and the update function is:

$$H_{k+1} = H_k - s[\nabla f(H_k, X, Y)]^\mathsf{T}. \tag{11.21}$$

[4] To understand how to obtain corresponding corners, we encourage the readers to read "Improvements to Target-Based 3D LiDAR to Camera Calibration," by Jiunn-Kai Huang and Jessy W. Grizzle.

After 5745 iterations with a step size of 1e-8, the cost drops from 49750 to 12.12, and the result is shown in Fig. 11.9. Later, we will introduce a second-order method, the "Hessian," to solve this problem and you will be amazed when the Hessian-based algorithm converges in 14 iterations! Knowledge is power!

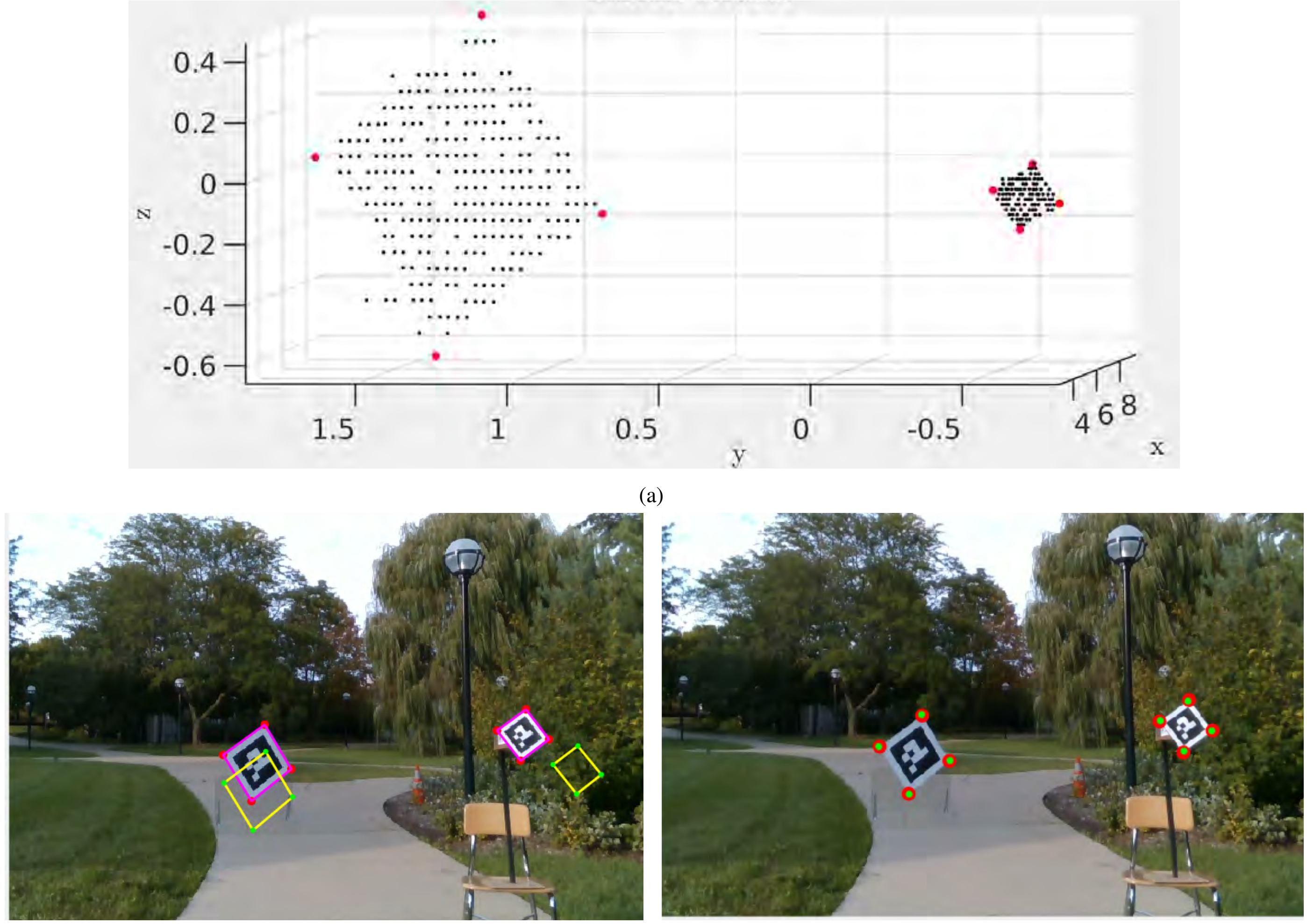
Remark: If you look into the implementation, you will find out that we do not represent the rotation matrix with nine elements even through it is a 3×3 matrix,

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \tag{11.22}$$

Instead, we use a fact you would learn in an upper level math course or a mechanics course, which says that any rotation matrix can be represented as the "matrix exponential" of a 3×3 skew symmetric matrix of the form

$$\Omega := \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix},$$

Hence, a rotation matrix depends on only three parameters, $(\omega_1, \omega_2, \omega_3)$. (It's hard to learn too much Linear Algebra; there is always one more useful fact!). Knowledge is power. Keep digging, and you will keep having fun.



(a)

(b)

(c)

Figure 11.8: a shows the LiDAR returns on the targets in black and estimated vertices in red. b illustrates the initial state of the optimization in (11.20). Our goal is to move the green dots onto the red dots, or you can also imagine moving the yellow box to the magenta box. c demonstrates the result of calibration: the LiDAR vertices (green dots) are successfully projected onto a camera corners (red dots).

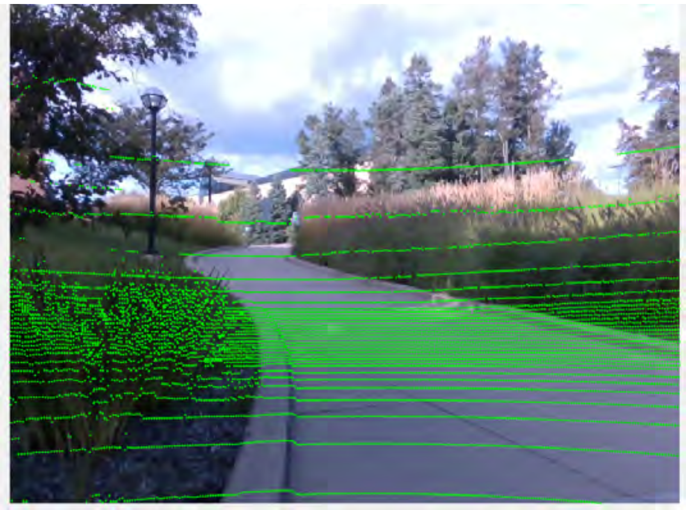


Figure 11.9: This figure demonstrates the calibration result of the gradient descent algorithm. It looks pretty good; there is still some misalignment. The main drawback of the algorithm would that a very small step size if often required to make it work.

11.5 Optimization as a Root Finding Problem: the Hessian

Gradient descent is a powerful method to solve optimization problems and we've seen it in two contexts so far: finding a (local) minimum of a cost function that depends on a scalar variable x in (11.2), and finding a (local) minimum of a cost function that depends on a vector variable $x = (x_1, x_2, \dots, x_m)$ in (11.13). Our stopping criterion was $|\frac{df(x_k)}{dx}|$ "sufficiently small" for scalar problems and $||\left[\nabla f(x_k)\right]^\top||$ "sufficiently small" for vector problems. In other words, a (locally) optimal solution x^* satisfies ${}^{df(x^*)}/_{dx} = 0$ for scalar problems and $\nabla f(x^*) = 0$ for vector problems. Said yet another way, our locally minimizing solutions are **roots of the derivative of the cost function**.

In Chapter 10, we learned a lot about finding roots of nonlinear equations. We will now apply that knowledge to optimization and thereby arrive at a more powerful optimization algorithm that uses information about the second derivative of the cost function! That sounds pretty wild and scary, but you'll see that it's very do-able.

11.5.1 Second Derivatives

Calculus has developed some good notation over the past 300 plus years and we will use it. The second derivative of a function $f : \mathbb{R} \to \mathbb{R}$ is the first derivative of the function's first derivative, and is written like this,

$$\frac{d^2 f(x)}{dx^2} := \frac{d}{dx}\frac{df(x)}{dx}. \tag{11.23}$$

Let's write down the derivative of the derivative using the symmetric difference approximation to the derivative. We first recall that

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x-h)}{2h}, \tag{11.24}$$

where $h > 0$ is a small number. Let's use $\delta > 0$ instead of h for a small positive change in x when writing down the second derivative as a symmetric difference

$$\frac{d^2 f(x)}{dx^2} \approx \frac{\frac{df(x+\delta)}{dx} - \frac{df(x-\delta)}{dx}}{2\delta}. \tag{11.25}$$

We now substitute (11.24) into (11.25) to obtain

$$\begin{aligned}\frac{d^2 f(x)}{dx^2} &\approx \frac{\left[\frac{f(x+\delta+h)-f(x+\delta-h)}{2h}\right] - \left[\frac{f(x-\delta+h)-f(x-\delta-h)}{2h}\right]}{2\delta} \\ &= \frac{[f(x+\delta+h) - f(x+\delta-h)] - [f(x-\delta+h) - f(x-\delta-h)]}{4h\delta} \\ &= \frac{f(x+\delta+h) - f(x+\delta-h) - f(x-\delta+h) + f(x-\delta-h)}{4h\delta}.\end{aligned} \tag{11.26}$$

Equation (11.26) is a perfectly fine expression for the second derivative. At least for your author, keeping δ and h separate made it easier to evaluate the individual terms and see how they appeared in the equation. Your experience may vary! It's more traditional to take $\delta = h$, which we now do so as to arrive at our final expression,

$$\boxed{\frac{d^2 f(x)}{dx^2} \approx \frac{f(x+2h) - 2f(x) + f(x-2h)}{4h^2}}. \tag{11.27}$$

In terms of coding, computing a numerical approximation to the second derivative is not much different than approximating the first derivative. Comparing (11.27) to (10.7), we see there is one more term to compute and instead of dividing by $2h$, we divide by $4h^2$.

Note that if you took $h > 0$ to be something relatively "small", such as 10^{-3}, then h^2 is now really small, such as 10^{-6}. Be careful that you do not approach "machine epsilon" when doing your approximate derivatives!

11.5.2 The Hessian is the Jacobian of the Transpose of the Gradient

As the title of this section suggests, we define the **Hessian** of a function $f : \mathbb{R}^m \to \mathbb{R}$ as the Jacobian of the transpose of the gradient of the function. Using notation that comes to us from Calculus, we have that

$$\nabla^2 f(x) := \frac{\partial}{\partial x} \left[\nabla f(x)\right]^\top, \tag{11.28}$$

where $\nabla^2 f(x)$ is the notation for the Hessian of f at the point x. We underline that here, the function f depends on a vector $x \in \mathbb{R}^m$ and that $f(x) \in \mathbb{R}$ is a scalar. If $f(x) \in \mathbb{R}^m$, for $n > 1$, then the above formula makes no sense...it is just a bunch of meaningless symbols.

Equation (11.28) is a lot of notation packed into one tiny formula! Let's unpack it so as to understand it bit by bit. For a function $f : \mathbb{R}^m \to \mathbb{R}$, the transpose of its gradient is

$$\left[\nabla f(x)\right]^\top := \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_k} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}. \tag{11.29}$$

Moreover, we can approximate the indicated partial derivatives using symmetric differences,

$$\frac{\partial f(x)}{\partial x_k} = \frac{f(x + he_k) - f(x - he_k)}{2h}, \tag{11.30}$$

where $\{e_1, \ldots, e_k, \ldots, e_m\}$ is the natural basis vectors for $\mathbb{R}^m$. For a function $g : \mathbb{R}^m \to \mathbb{R}^m$, we recall that its Jacobian is

$$\frac{\partial g(x)}{\partial x} := \left[\begin{array}{ccccc} \frac{\partial g(x)}{\partial x_1} & \cdots & \frac{\partial g(x)}{\partial x_j} & \cdots & \frac{\partial g(x)}{\partial x_m} \end{array}\right]. \tag{11.31}$$

Moreover, we know how to numerically approximate the partial derivatives in (11.31) using symmetric differences and selecting $\delta > 0$ by

$$\frac{\partial g(x)}{\partial x_j} = \frac{g(x + \delta e_j) - g(x - \delta e_j)}{2\delta}. \tag{11.32}$$

To put all of this together, we take

$$g(x) := \left[\nabla f(x)\right]^\top$$

and apply (11.32) to obtain a first numerical approximation for the Hessian, namely

$$\nabla^2 f(x) \approx \frac{1}{2\delta} \left[\begin{array}{ccccc} \big(\nabla f(x + \delta e_1) - \nabla f(x - \delta e_1)\big) & \cdots & \big(\nabla f(x + \delta e_k) - \nabla f(x - \delta e_k)\big) & \cdots & \big(\nabla f(x + \delta e_m) - \nabla f(x - \delta e_m)\big) \end{array}\right]. \tag{11.33}$$

Going one step further, if we let $\left[\nabla^2 f(x)\right]_{ij}$ be the ij-entry of the Hessian matrix (that is, the entry for its i-th row and j-th column), using (11.30), we have that

$$\left[\nabla^2 f(x)\right]_{ij} \approx \frac{1}{4h\delta}\big(f(x + he_i + \delta e_j) - f(x + he_i - \delta e_j) - f(x - he_i + \delta e_j) + f(x - he_i - \delta e_j)\big). \tag{11.34}$$

In this case, setting $\delta = h$ does not really simplify the expression. In Julia, we suspect that you will find (11.33) the easiest to implement.

11.5.3 Use of the Second Derivative and Hessian in Optimization

Optimization as a Form of Root Finding

The most accessible forms of second-order optimization methods are based on applying Newton's method to the first derivative of a cost function that depends on a scalar variable $x \in \mathbb{R}$ or the Newton-Raphson Algorithm to the (transpose of the) gradient of a cost function that depends on a vector variable $x \in \mathbb{R}^m$.

Scalar optimization variable: For $f : \mathbb{R} \to \mathbb{R}$, the iterative process

$$x_{k+1} = x_k - \left(\frac{d^2 f(x_k)}{dx^2} \right)^{-1} \frac{df(x_k)}{dx} \tag{11.35}$$

will converge to a local extremum of f if the initial value x_0 is "well chosen". Because the algorithm is looking for **roots** of the first derivative, it is *indifferent* to whether the root is a local minimum or a local maximum. **In Calculus, you will learn that if the second derivative is positive at a point where the first derivative vanishes, then you have found a local minimum.** Similarly, a negative second derivative implies a local maximum.

The damped version of the algorithm

$$x_{k+1} = x_k - s \left(\frac{d^2 f(x_k)}{dx^2} \right)^{-1} \frac{df(x_k)}{dx}, \tag{11.36}$$

where $0 < s < 1$, typically performs better in practice. We note that the second derivative, $\frac{d^2 f(x)}{dx^2}$, can be approximated as in (11.27).

Vector optimization variable: For $f : \mathbb{R}^m \to \mathbb{R}$, the iterative process

$$\nabla^2 f(x_k)\, \Delta x_k = -\left[\nabla f(x_k)\right]^\top \quad \text{(solve for } \Delta x_k\text{)} \tag{11.37}$$

$$x_{k+1} = x_k + \Delta x_k \quad \text{(use } \Delta x_k \text{ to update our estimate of the optimal value)} \tag{11.38}$$

will converge to a local extremum of f if the initial value x_0 is "well chosen". Because the algorithm is looking for **roots** of the gradient, it is *indifferent* to whether the root is a local minimum, a local maximum, or what is called a "saddle point". **In Calculus, you will learn that if the Hessian is positive definite at a point where the gradient vanishes, then you have found a local minimum.** Similarly, a negative definite Hessian implies a local maximum. A discussion of these nice properties is beyond the scope of ROB 101.

While for toy problems, we can use the matrix inverse to solve (11.37) for Δx_k, for larger problems, we recommend using an LU Factorization or a QR Factorization. Once (11.37) has been solved, x_{k+1} is updated in (11.38) and the process repeats. In practice, the **damped** version of the algorithm often works better, where one replaces (11.38) with

$$x_{k+1} = x_k + s\Delta x_k, \tag{11.39}$$

for some $0 < s < 1$. We note that the Hessian, $\nabla^2 f(x)$, can be approximated with either (11.33) or (11.34).

All of the remarks made in Chapter 10 about the validity of Newton's Method or the Newton-Raphson Algorithm apply here was well.

We re-do several of the previous examples.

Example 11.3 *Based on Example 11.1, but this time, we implement* (11.35) *and* (11.36) *in Julia to find local extrema of* $f : \mathbb{R} \to \mathbb{R}$,

$$f(x) = (x-2)^2 + 5(\sin(x-2))^2 + 0.03(x+1)^3 + 4.$$

Solution: We run the algorithm from the same initial conditions as in Example 11.1 and for two values of the step size, $s \in \{0.25, 1.0\}$. The reader should note that we got somewhat lucky, and each time the algorithm converged to a root of the first derivative! We've highlighted in blue where the algorithm actually converged to a local maximum instead of a local minimum. How could we tell? The second derivative being positive implies a local minimum while it being negative implies a local maximum.

We note that the rate of convergence is much faster than for gradient descent (here, faster means fewer iterations of the algorithm). Finally, we note that the point to which the algorithm converges does depend on the initial condition. In fact, for $s = 0.25$ and $s = 1.0$, the algorithm sometimes converges to different roots of the first derivative, even when initialized at the same point.

If you can design a cost function so that it has a single extrema and it is your desired minimum, then you can avoid many of these problems. We talk a little about this in the section on "Local vs Global".

N0. Iterations	$\mathbf{s}$	$\mathbf{x_0}$	$\mathbf{x^*}$	$\mathbf{f(x^*)}$	$\mathbf{\frac{df(x^*)}{dx}}$	$\mathbf{\frac{d^2f(x^*)}{dx^2}}$
$2.2000e+01$	$1.0000e+00$	$-1.0000e+00$	$3.1369e+00$	$3.3002e+00$	$-4.5648e-09$	$9.2092e+00$
$4.0000e+00$	$1.0000e+00$	$0.0000e+00$	$6.7838e-01$	$1.1926e+00$	$-5.9785e-07$	$1.1085e+01$
$2.0000e+00$	$1.0000e+00$	$6.8000e-01$	$6.7838e-01$	$1.1926e+00$	$6.9886e-10$	$1.1085e+01$
$\mathbf{2.0000e+00}$	$\mathbf{1.0000e+00}$	$\mathbf{2.1000e+00}$	$\mathbf{2.1099e+00}$	$\mathbf{4.8543e+00}$	$\mathbf{-1.6541e-08}$	$\mathbf{-7.1982e+00}$
$\mathbf{2.0000e+00}$	$\mathbf{1.0000e+00}$	$\mathbf{2.1500e+00}$	$\mathbf{2.1099e+00}$	$\mathbf{4.8543e+00}$	$\mathbf{5.2104e-07}$	$\mathbf{-7.1982e+00}$
$2.0000e+00$	$1.0000e+00$	$3.1400e+00$	$3.1369e+00$	$3.3002e+00$	$-2.8692e-09$	$9.2092e+00$
$5.0000e+00$	$1.0000e+00$	$4.0000e+00$	$3.1369e+00$	$3.3002e+00$	$-2.6010e-09$	$9.2092e+00$
$2.3000e+01$	$1.0000e+00$	$5.0000e+00$	$-2.4271e+01$	$3.1198e+02$	$1.2562e-09$	$4.3032e+00$

(11.40)

N0. Iterations	$\mathbf{s}$	$\mathbf{x_0}$	$\mathbf{x^*}$	$\mathbf{f(x^*)}$	$\mathbf{\frac{df(x^*)}{dx}}$	$\mathbf{\frac{d^2f(x^*)}{dx^2}}$
$1.2500e+02$	$2.5000e-01$	$-1.0000e+00$	$-2.4271e+01$	$3.1198e+02$	$-9.2124e-06$	$4.3033e+00$
$4.7000e+01$	$2.5000e-01$	$0.0000e+00$	$6.7838e-01$	$1.1926e+00$	$-9.7541e-06$	$1.1085e+01$
$2.7000e+01$	$2.5000e-01$	$6.8000e-01$	$6.7838e-01$	$1.1926e+00$	$7.6022e-06$	$1.1085e+01$
$\mathbf{3.1000e+01}$	$\mathbf{2.5000e-01}$	$\mathbf{2.1000e+00}$	$\mathbf{2.1099e+00}$	$\mathbf{4.8543e+00}$	$\mathbf{9.5934e-06}$	$\mathbf{-7.1982e+00}$
$\mathbf{3.6000e+01}$	$\mathbf{2.5000e-01}$	$\mathbf{2.1500e+00}$	$\mathbf{2.1099e+00}$	$\mathbf{4.8543e+00}$	$\mathbf{-8.9863e-06}$	$\mathbf{-7.1982e+00}$
$2.8000e+01$	$2.5000e-01$	$3.1400e+00$	$3.1369e+00$	$3.3002e+00$	$9.0254e-06$	$9.2093e+00$
$4.8000e+01$	$2.5000e-01$	$4.0000e+00$	$3.1369e+00$	$3.3002e+00$	$9.9328e-06$	$9.2093e+00$
$\mathbf{6.8000e+01}$	$\mathbf{2.5000e-01}$	$\mathbf{5.0000e+00}$	$\mathbf{2.1099e+00}$	$\mathbf{4.8543e+00}$	$\mathbf{9.0945e-06}$	$\mathbf{-7.1982e+00}$

(11.41)

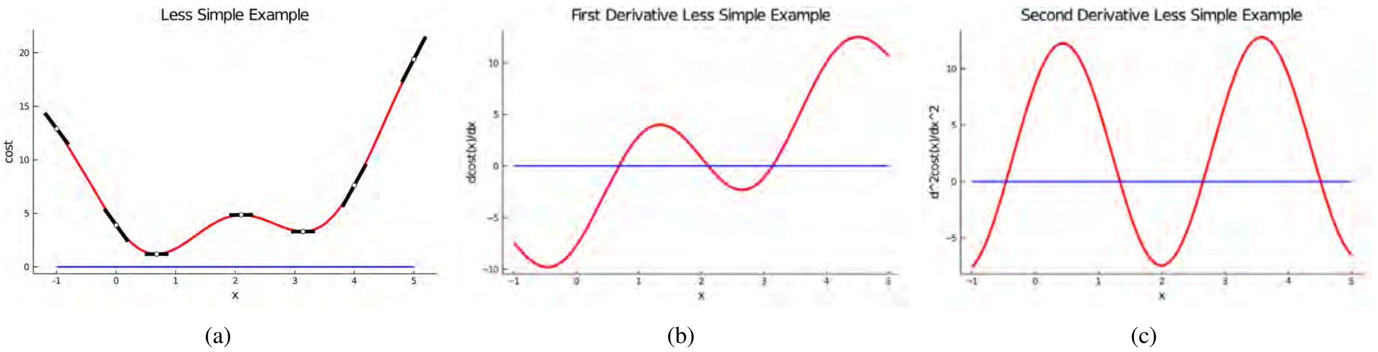


Figure 11.10: A cost function along with its first and second derivatives. We note that the extrema (both minima and maxima) in (a) correspond to the zero crossings (aka roots) of the first derivative in (b). Moreover, the sign of the second derivative at roots of the first derivative (aka, extrema) provides information on whether we have a local min, max, or neither.

An instantiation in Julia is given below.

■

```julia
#Optimization, with second derivative
#
# cost function
cost(x)=(x.-2).^2 .+ 1 .- 5*(sin.(x.-2)).^2 .+ 3 .+ 0.03*(x.+1).^3
yzero(x)=0.0*cost(x)
# x-perturbation for derivatives
```

```
delta=0.01
xmin=-1.0;xmax=5.0
# first derivative of cost
dcost(x)=(cost(x+delta)-cost(x-delta))/(2*delta)
# second derivative of cost
ddcost(x)=(dcost(x+delta)-dcost(x-delta))/(2*delta)
titre="First Derivative Less Simple Example"
p2=plot(dcost,xmin,xmax,legend=false, title=titre, linewidth=3, color=:red )
plot!(yzero,xmin,xmax, linewidth=2, color=:blue)
xlabel!("x")
ylabel!("dcost(x)/dx")
titre="Second Derivative Less Simple Example"
p3=plot(ddcost,xmin,xmax,legend=false, title=titre, linewidth=3, color=:red )
plot!(yzero,xmin,xmax, linewidth=2, color=:blue)
xlabel!("x")
ylabel!("d^2cost(x)/dx^2")
# step size
s=0.25
x0=[-1;0;0.68;2.1;2.15;3.14;4;5]
y0=cost(x0)
IntermediateValues=zeros(1,7)
for j =1:length(x0)
    xk=x0[j]
    dcostdxk = dcost(xk)
    ddcostdxk = ddcost(xk)
   fk=cost(xk)
    k=0
    while (abs(dcostdxk)>1e-5)&(k < 1e3)
        k=k+1
        xk=xk-s*dcostdxk/ddcostdxk
        fk=cost(xk)
        dcostdxk = dcost(xk)
        ddcostdxk = ddcost(xk)
    end
    IntermediateValues=[IntermediateValues; k s x0[j] xk fk dcostdxk ddcostdxk]
end
display(IntermediateValues)
display(p2)
display(p3)
png(p2, "DerivativeLessSimpleCost")
png(p3, "SecondDerivativeLessSimpleCost")
set_default(fmt = "%.4e", convert_unicode = false)
latexify(IntermediateValues) |> print
```

Example 11.4 *We re-visit the least squares problem from Example 11.2, which, in turn, came from Chapter 8.3. This time we use the Hessian and second order methods given in* (11.37) *through* (11.39) *to solve*

$$x^* = \underset{x\in\mathbb{R}^m}{\arg\min}\, ||Ax-b||^2,$$

where

$$A = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 1.0000 & 0.2500 & 0.0625 \\ 1.0000 & 0.5000 & 0.2500 \\ 1.0000 & 0.7500 & 0.5625 \\ 1.0000 & 1.0000 & 1.0000 \\ 1.0000 & 1.2500 & 1.5625 \\ 1.0000 & 1.5000 & 2.2500 \\ 1.0000 & 1.7500 & 3.0625 \\ 1.0000 & 2.0000 & 4.0000 \end{bmatrix} \quad \textit{and} \quad b = \begin{bmatrix} 1.0000 \\ 1.0000 \\ 1.5000 \\ 2.0000 \\ 3.0000 \\ 4.2500 \\ 5.5000 \\ 7.0000 \\ 10.0000 \end{bmatrix}.$$

Solution: We apply Newton-Raphson to the gradient of the cost function

$$f(x) := (Ax - b)^\top (Ax - b)\,,$$

set the step size $s \in \{0.25, 1.0\}$, and the tolerance to $||\nabla f(x_k)|| < 10^{-5}$. All derivatives are computed using symmetric differences with $h = 0.01$. For comparison purposes, we recall that the true answer is

$$x^* = \begin{bmatrix} 1.065152e+00 \\ -6.257576e-01 \\ 2.454545e+00 \end{bmatrix}. \tag{11.42}$$

For a step size of $s = 1.0$, and starting from the randomly generated initial condition

$$x_0 := \begin{bmatrix} 1.923764e+00 \\ 5.579350e+00 \\ 3.273492e-01 \end{bmatrix}, \tag{11.43}$$

the algorithm converges in one step to

$$x^* \approx \begin{bmatrix} 1.065152e+00 \\ -6.257576e-01 \\ 2.454545e+00 \end{bmatrix}. \tag{11.44}$$

Starting from the same initial condition and using a step size of $s = 0.25$, the algorithm converges in 58 iterations to

$$x^* \approx \begin{bmatrix} 1.065152e+00 \\ -6.25757e-01 \\ 2.454545e+00 \end{bmatrix}. \tag{11.45}$$

Julia code associated with the above is given below. ■

```
# Hessian on least squares
dataSet2=[
1  0 1.0
2  0.25   1.0
3  0.5    1.5
4  0.75   2.0
5  1.0    3.0
6  1.25    4.25
7  1.5    5.5
8  1.75    7.0
9  2.0    10.0]
X=dataSet2[:,2]
Y=dataSet2[:,3]
Phi=[ones(9,1) X  X.^2]
# Cheating on the matrix inverse because it is a toy problem
```

```
alphaStar=inv(Phi'*Phi)*Phi'*Y
#
#
set_default(fmt = "%.6e", convert_unicode = false)
latexify(alphaStar) |> print
#
#
F(x)= ( (Phi*[x[1];x[2];x[3]]-Y)'*(Phi*[x[1];x[2];x[3]]-Y) )
e1=[1;0;0];e2=[0;1;0];e3=[0;0;1]
h=0.01
# Symmetric differences
DirectionalDerivativeF(x,e)=(F(x+h*e)-F(x-h*e))/(2*h)
GradF(x)=[DirectionalDerivativeF(x,e1) DirectionalDerivativeF(x,e2) DirectionalDerivativeF(x,e3)]
GradFtranspose(x)=[DirectionalDerivativeF(x,e1); DirectionalDerivativeF(x,e2); DirectionalDerivativeF(x,e3)]
HessianF(x)=[(GradFtranspose(x+h*e1)-GradFtranspose(x-h*e1))/(2*h)  (GradFtranspose(x+h*e2)-GradFtranspose(x-h*e2))/(2*h)  (GradFtranspose(x+h*e3)-GradFtranspose(x-h*e3))/(2*h)]
# Initial condition
#xk=10*rand(3,1)
xk = [1.9237640987871218; 5.579349855694035; 0.32734915035269596]
@show xk
set_default(fmt = "%.6e", convert_unicode = false)
latexify(xk) |> print
GradFxk=GradF(xk)
normGradk=norm(GradFxk)
s=0.25; #step size for Hessian search
k=0
while (k <1e4)&(normGradk>1e-5)
    HessianFxk=HessianF(xk)
    xk=xk-s*inv(HessianFxk)*(GradFxk')
    k=k+1
    GradFxk=GradF(xk)
    normGradk=norm(GradFxk)
end
@show k
@show xk
latexify(xk) |> print
```

Example 11.5 *We resist the extrinsic calibration problem of Chapter 11.4.1, but this time we use the Hessian. The update equation in* (11.21) *is replaced with*

$$H_{k+1} = H_k - s\big[\nabla^2 f(H_k, X, Y)\big]^{-1}[\nabla f(H_k, X, Y)]^\mathsf{T}. \tag{11.46}$$

After 14 iterations (compared to 5745 iterations when using the gradient descent method) with a step size of 1.5, the problem converges, and the cost drops to 12.113. The result is shown in Fig. 11.11. Compared to the first-order (gradient descent) method, the second-order method (using the Hessian) is much faster!

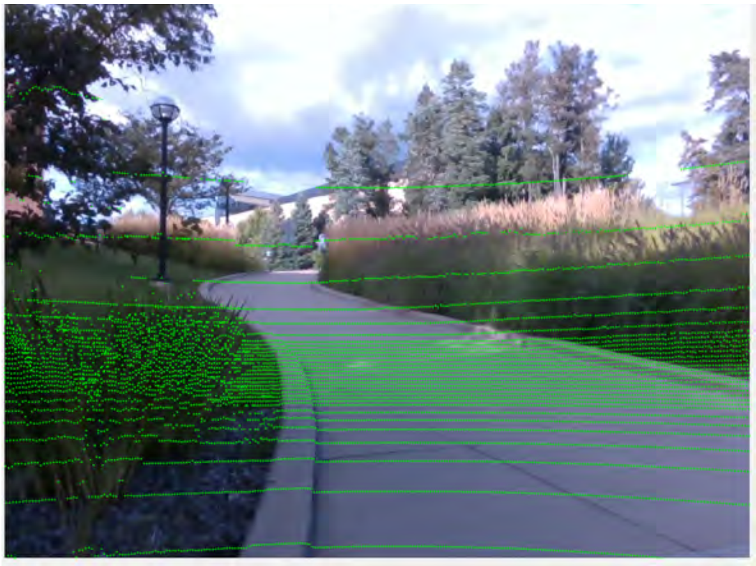


Figure 11.11: This figure demonstrates the calibration result of the Hessian method. The results look similar to the gradient descent algorithm in Fig 11.9 but the convergence speed when using the Hessian is 400 times faster than when using gradient descent.

The code to generate the results and figures for this problem is available on GitHub[5] in MATLAB; how to do it in Julia is left as a homework problem! Given the hints already provided in this Chapter, we expect you will have no trouble with it.

11.6 Local vs Global

Consider a function $f : \mathbb{R} \to \mathbb{R}$. Let's recall that the **graph** of the function is the collection of points

$$\textbf{graph of f} := \{(x, f(x)) \mid x \in \mathbb{R}\}.$$

Alternatively, you may be more comfortable thinking of it as

$$\textbf{graph of f} := \{(x, y) \mid x \in \mathbb{R},\ y = f(x)\}.$$

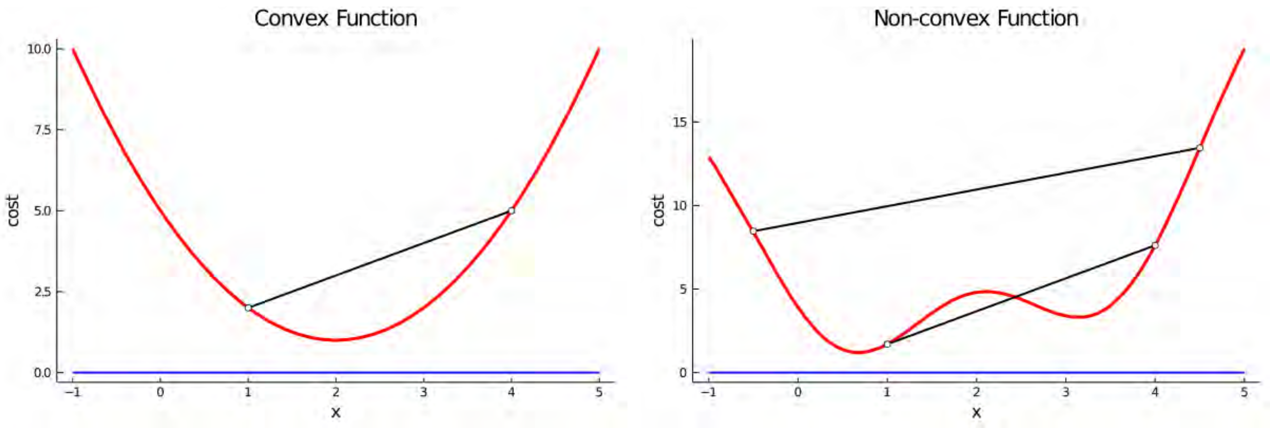
We revisit the functions used in Fig. 11.1, where this time, in Fig. 11.12, we have also indicated line segments (in black) that connect various points on the graphs of the two functions (shown in red). The lines are given by

$$y = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1), \tag{11.47}$$

for $x_1 \neq x_2$ in the domain of definition of the function.

If we can choose x_1 and x_2 such that the graph of the function is ever strictly above the corresponding line, as in Fig. 11.12b, then the **function is not convex and you can have local minima.** If, on the other hand, the graph of the function always lies below or just touches the line, for all $x_1 \neq x_2$, as in Fig. 11.12a, then the **function is convex and there are no local minima, only global minima!** Now, the value of x achieving the global minimum may not be unique. This happens when the "bottom bowl" of the function is a line with zero slope, as shown in Fig. 11.13.

[5] A platform we use to share our code. Just like you share your photos on Instagram!



(a)

(b)

Figure 11.12: (a) This is a graph of our "simple" cost function with a global minimum at $x^\ast = 2.$, while (b) a graph of our "less simple" cost function, where there are two local minima, one at $x^\ast \approx 0.68$ and one at $x^\ast \approx 3.14$, as well as a local maximum at ≈ 2.1. In each case, we have overlaid line segments that are used to check for the mathematical property called **convexity**. A convex function only always has global minima. It does not have any local minima.

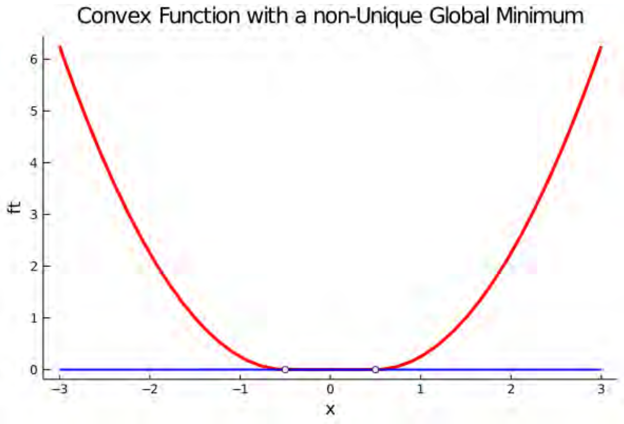


Figure 11.13: This is a plot of a convex function where there are many points achieving the global minimum of zero. In fact, the function is identically zero for $x \in [-0.5, 0.5]$.

Convex Functions

A function $f : \mathbb{R}^m \to \mathbb{R}$ is convex if for all $0 \le \alpha \le 1$, and for all $x, y \in \mathbb{R}^m$, the function satisfies

$$f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha) f(y). \tag{11.48}$$

In practice, checking this property is relatively hard at the present time, even for the case that $n = 1$, and is beyond the scope of our introduction. However, as you advance in your mathematical education, if ever a lecture is offered on **convex sets** or **convex functions**, you should pay attention! You might learn some cool and very useful stuff!

Convex optimization is not a typical subject for undergraduates in the present day and age, but maybe you guys will change that! It is becoming super important in engineering practice. Our purpose here is to let you know that such a subject as convex optimization exits so that you can be on the lookout for a course on the topic!

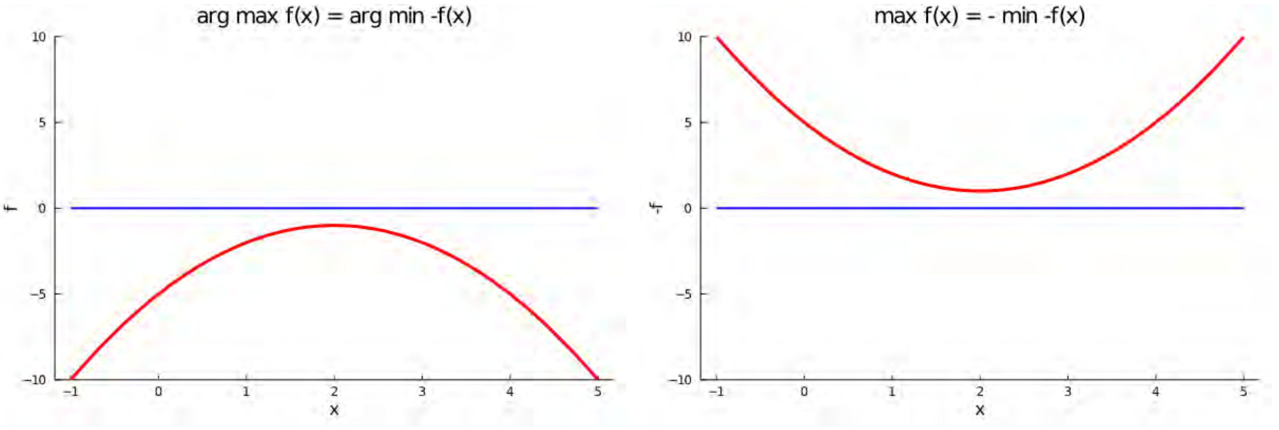
11.7 Maximization as Minimizing the Negative of a Function

The two main facts are summarized in the following equations.

$$\arg\max_{x\in R^m} f(x) = \arg\min_{x\in\mathbb{R}^m} -f(x) \tag{11.49}$$

$$\max_{x\in R^m} f(x) = -\min_{x\in\mathbb{R}^m} -f(x) \tag{11.50}$$

You should be able to convince yourself they are true by studying Fig. 11.14.



(a)

(b)

Figure 11.14: Maximization and minimization are almost the same thing! (a) Shows a function that we wish to maximize while (b) shows the negative of the function. From these plots, you can convince yourself that (11.49) and (11.50) hold!

11.8 (Optional Read): Quadratic Programs: Our first Encounter with Constraints

A **Quadratic Program** is a special kind of optimization problem with **constraints**. The cost to be minimized is supposed to be quadratic, meaning that $f : \mathbb{R}^m \to \mathbb{R}$ has the form

$$f(x) = \frac{1}{2} x^\top Q x + qx, \tag{11.51}$$

where Q is an $m \times m$ **symmetric matrix**, meaning that $Q^\top = Q$, and where q is a $1 \times m$ row vector. Moreover, instead of optimizing over all of $\mathbb{R}^m$ as in our previous problems, we are allowed to seek solutions that lie in a subset of $\mathbb{R}^m$ defined by **linear inequality** and **linear equality** constraints that are typically written in the form

$$A_{in} x \preceq b_{in} \tag{11.52}$$

$$A_{eq} x = b_{eq}. \tag{11.53}$$

The symbol $\preceq$ is a way to define "less than or equal to" for vectors; it means that each component of the vector on the left hand side is less than or equal to the corresponding component of the vector on the right hand side. As an example

$$\begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \preceq \begin{bmatrix} 4 \\ 3 \\ 4 \end{bmatrix},$$

though

$$\begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \npreceq \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix};$$

and

$$\begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \preceq \begin{bmatrix} 0 \\ 9 \end{bmatrix},$$

means that x_1 and x_2 must satisfy

$$\begin{aligned} 3x_1 + x_2 &\leq 0 \\ 2x_1 + 4x_2 &\leq 9. \end{aligned}$$

What if you really wanted $2x_1 + 4x_2 \geq 9$? Then you need to remember that when you multiply both sides by a minus sign, the inequality sign flips. Hence,

$$\begin{aligned} 3x_1 + x_2 &\leq 0 \\ 2x_1 + 4x_2 &\geq 9 \end{aligned} \iff \begin{aligned} 3x_1 + x_2 &\leq 0 \\ -2x_1 - 4x_2 &\leq -9 \end{aligned} \iff \begin{bmatrix} 3 & 1 \\ -2 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \preceq \begin{bmatrix} 0 \\ -9 \end{bmatrix}.$$

In addition, most QP solvers allow one to specify lower and upper bounds on x of the form

$$lb \preceq x \preceq ub. \tag{11.54}$$

While such constraints could always be rewritten in the form of (11.52), using (11.54) is more convenient, intuitive, and less error prone. The inclusion of constraints allows for very interesting and practical optimization problems to be posed.

Useful Fact about QPs

We consider the QP

$$\begin{aligned} x^* = \quad & \underset{x \in \mathbb{R}^m}{\arg\min} \quad \frac{1}{2} x^\top Q x + q x \\ & A_{in} x \preceq b_{in} \\ & A_{eq} x = b_{eq} \\ & lb \preceq \ x \preceq ub \end{aligned} \tag{11.55}$$

and assume that Q is symmetric ($Q^\top = Q$) and **positive definite**[a] ($x \neq 0 \implies x^\top Q x > 0$), and that the subset of $\mathbb{R}^m$ defined by the constraints is non empty, that is

$$C := \{x \in \mathbb{R}^m \mid A_{in} x \preceq b_{in}, \ A_{eq} x = b_{eq}, \ lb \preceq x \preceq ub\} \neq \emptyset. \tag{11.56}$$

Then x^* exists and is unique.

[a] Positive definite matrices are treated in Chapter B.3.

Example 11.6 *The very first optimization problem we came across in ROB 101 was least-squared-error solutions to systems of linear equations, $Ax = b$, back in Chapter 8.2, namely*

$$x^* = \underset{x \in \mathbb{R}^m}{\arg\min} \, ||Ax - b||^2. \tag{11.57}$$

We used this formulation to solve regression problems in Chapter 8.3. Show that (11.57) *is equivalent to the QP*

$$x^* = \underset{x \in \mathbb{R}^m}{\arg\min} \, \frac{1}{2} x^\top Q x + q x, \tag{11.58}$$

where

$$\begin{aligned} Q &:= A^\top \cdot A \\ q &:= -b^\top \cdot Ax. \end{aligned} \tag{11.59}$$

In particular, it is a very simple QP, with no inequality constraints and no equality constraints.

Solution: We first note that $||Ax - b||^2 = (Ax - b)^\top \cdot (Ax - b)$, where we have used the fact that the norm squared of a vector $v \in \mathbb{R}^n$ is equal[6] to $v^\top \cdot v$. Hence, multiplying out the terms, we have that

$$\begin{aligned} ||Ax - b||^2 &= (Ax - b)^\top \cdot (Ax - b) \\ &= \left(x^\top A^\top - b^\top\right) \cdot (Ax - b) \\ &= x^\top A^\top \cdot Ax - b^\top \cdot Ax - x^\top A^\top \cdot b + b^\top \cdot b \\ &= x^\top A^\top \cdot Ax - 2b^\top \cdot Ax + b^\top \cdot b, \end{aligned} \tag{11.60}$$

[6] In our case, $v = (Ax - b)$.

where we have used the fact that

$$x^\top A^\top \cdot b = b^\top \cdot Ax$$

because each side is a scalar and the transpose of a scalar is itself. Next, we note that

$$\begin{aligned} x^* &= \underset{x \in \mathbb{R}^m}{\arg\min} \, ||Ax - b||^2 \\ &= \underset{x \in \mathbb{R}^m}{\arg\min} \, \frac{1}{2}||Ax - b||^2 \\ &= \underset{x \in \mathbb{R}^m}{\arg\min} \left(\frac{1}{2} x^\top A^\top \cdot Ax - b^\top \cdot Ax + \frac{1}{2} b^\top \cdot b \right) \\ &= \underset{x \in \mathbb{R}^m}{\arg\min} \left(\frac{1}{2} x^\top A^\top \cdot Ax - b^\top \cdot Ax \right) \end{aligned}$$

because

- scaling the function to be minimized by a positive constant does not change **where the minimum occurs**, and hence does not change the value of $\arg\min$,
- and shifting the function to be minimized up or down by a constant does not change **where the minimum occurs**, and hence does not change the value of $\arg\min$!

■

Example 11.7 *The second optimization problem we came across in ROB 101 was for underdetermined systems of linear equations, $Ax = b$, back in Chapter 9.9, namely*

$$x^* = \underset{Ax=b}{\arg\min} \, ||x||^2 = \underset{Ax=b}{\arg\min} \, x^\top x. \tag{11.61}$$

This too is a rather simple QP, with $Q = I_m$, the $m \times m$ identity matrix, $q = 0_{1\times m}$, and no inequality constraints.

■

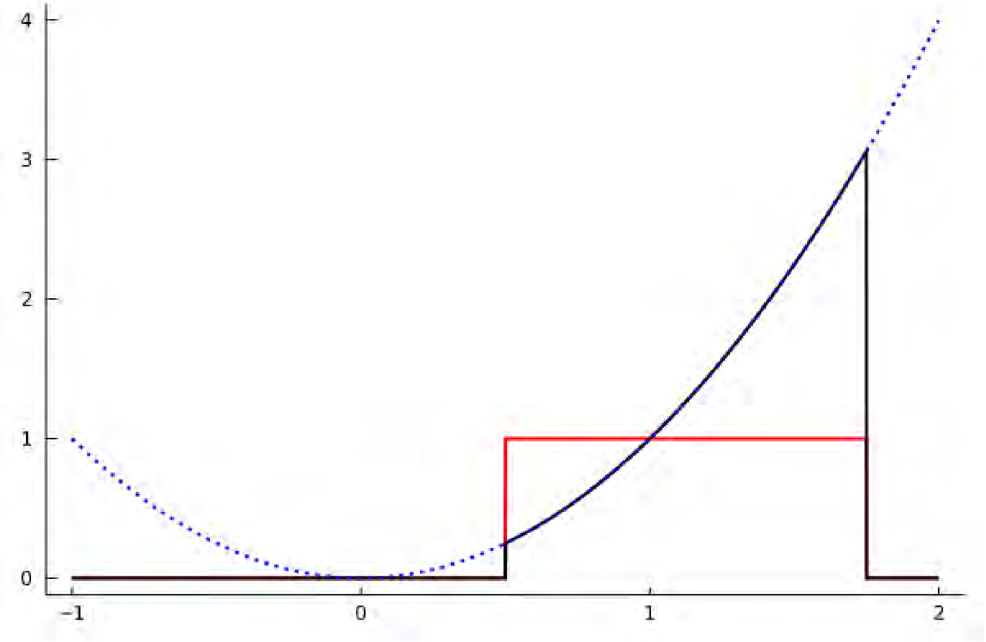


Figure 11.15: The plot shows in red the rectangle function $\text{rect}(x, a, b)$, for the values $a = 0.5$ and $b = 1.75$. The function takes on the value 1.0 for $a \leq x < b$ and equals 0.0 elsewhere. The dotted blue line is the monomial x^2, which is being applied over the interval $[-1, 2]$. In black is the monomial x^2 **multiplied** by $\text{rect}(x, a, b)$, for the same values of a and b. The function $x^2 \cdot \text{rect}(x, a, b)$ takes on the value x^2 for all $a \leq x < b$ and 0.0 elsewhere. In other words, the action of the function is now localized to the set $[a, b) \subset \mathbb{R}$. This is the basic notion of a spline, being able to localize the action of a function to a subset of x values instead of having the function extend over the entire set of x values, as with the standard monomial x^2. Splines typically give you more control in the fitting process than using high-order monomials.

Example 11.8 *We'll now put the first two examples together, while introducing you to a new way to choose "basis functions" for regression, called splines; see Fig 11.15.*

You are given the noisy data shown in Fig. 11.16. The objective is to fit a function to the data, much as we did in Chapter 8.3. We'll add a new twist by introducing a type of function called a "spline", where even though we are using our familiar monomials, they will be localized to disjoint regions of the data. The solution will provide the details!

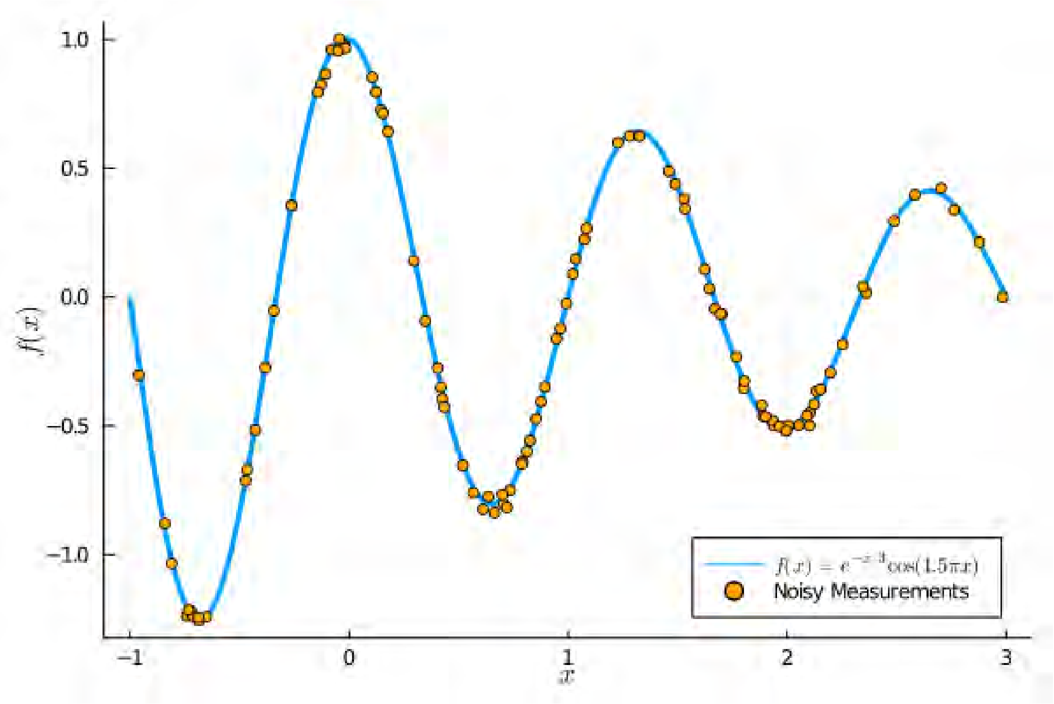


Figure 11.16: The true function (which you should pretend not to know) and a set of imperfect measurements from which you are to estimate the function.

Solution: We introduce a new function, called rectangle,

$$\text{rect}(x, a, b) = \begin{cases} 1 & a \leq x < b \\ 0 & \text{otherwise,} \end{cases}$$

that can be used to limit the action of a function to an interval of the form $[a, b)$. In Fig. 11.15, we plot the rectangle function itself and the monomial x^2 multiplied by the rectangle function.

We will use the rectangle function to divide the set $[-1, 3]$ into three subsets and fit low-degree polynomials on each subset. To do this, we define $x_{\min} := 1.0$, $x_{\max} := 2.0$, and further define

$$\begin{aligned} \Delta x &:= \frac{x_{\min} + x_{\max}}{3} \\ a_k &:= x_{\min} + (k-1)\delta x, \quad k \in 1, 2, 3, 4 \end{aligned}$$

so that

$$a = [-1.0,\ \frac{1}{3},\ \frac{5}{3},\ 3.0].$$

The components of a are called **knot points**; see https://en.wikipedia.org/wiki/Spline_(mathematics) for more information. We note that $[-1, \frac{1}{3}) \cup [\frac{1}{3}, \frac{5}{3}) \cup [\frac{5}{3}, 3) = [-1, 3)$, so technically, we have left the point $x = 3$ out of consideration. If you wish to include it, just replace the last knot point with something like 3.001.

Inspired by Fig. 11.15, we define a regressor matrix which uses the standard monomials up to degree three on each set $[a_i, a_{i+1})$,

$$\Phi(x,a) := \big[1 \;\; x\cdot \mathrm{rect}(x,a_1,a_2) \;\; \dots \;\; x^3\cdot \mathrm{rect}(x,a_1,a_2) \;\; \dots \;\; x\cdot \mathrm{rect}(x,a_3,a_4) \;\; \dots \;\; x^3\cdot \mathrm{rect}(x,a_3,a_4)\big], \tag{11.62}$$

where, x is a (column) vector of (measurement) points and a is the set of knot points.

Figure 11.17 shows the resulting fit, which is a standard least squares problem

$$\alpha^* = \underset{\alpha\in\mathbb{R}^{10}}{\arg\min} \, ||Y - \Phi\alpha||^2,$$

where Y is the vector of measured function values. We note right away the "jumps" in the fit at the two interior knot points, $a_2 = 1/3$ and $a_2 = 5/3$. The discontinuities arise from jumps in the rectangle function at the spline boundaries, as was seen in Fig. 11.15. We next show how to impose continuity at the boundaries of the splines.

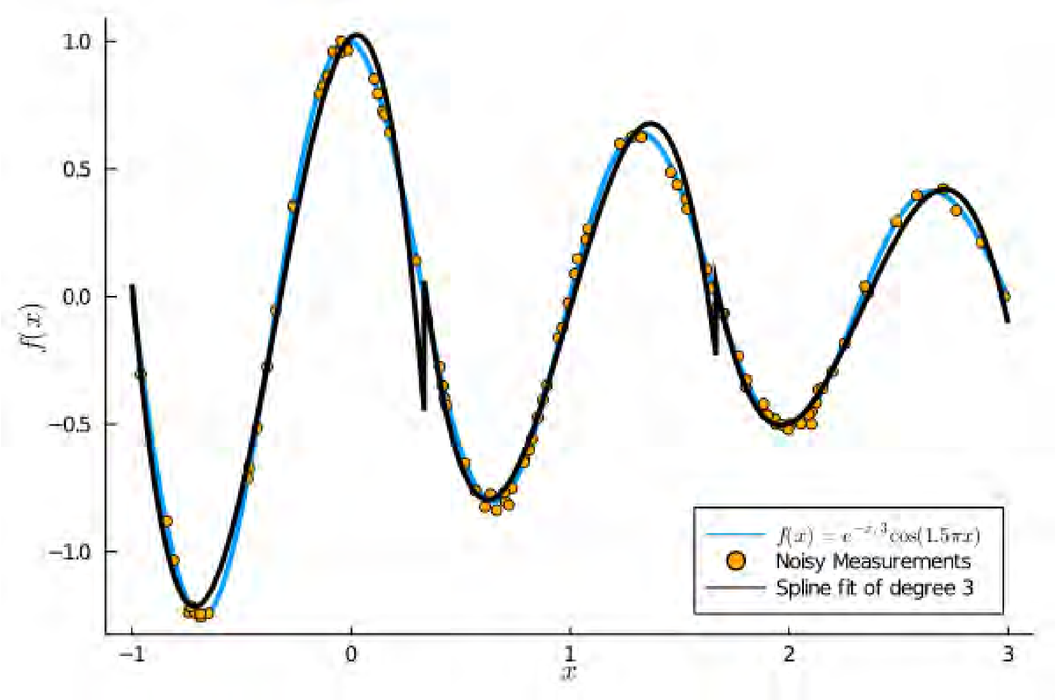


Figure 11.17: The resulting fit of a polynomial spline of degree three. A discontinuity is clear at the two interior knot points, $a_2 = 1/3$ and $a_2 = 5/3$. We next show how to achieve continuity.

To impose continuity at the interior knot points, we will use a linear constraint on the regression coefficients, α. Let $\epsilon = 10^{-3}$ and define

$$\begin{aligned} A_1 &:= \Phi(a_2-\epsilon, a) - \Phi(a_2+\epsilon, a) \\ A_2 &:= \Phi(a_3-\epsilon, a) - \Phi(a_3+\epsilon, a) \\ A_{\mathrm{eq}} &:= \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}. \end{aligned} \tag{11.63}$$

It follows that $A_{\mathrm{eq}}\alpha = 0_{2\times 1}$ forces the splines to match up at the boundary points. Indeed, if we denote $\widehat{y}(x) := \Phi(x,a)\alpha$ for an arbitrary $x \in \mathbb{R}$, then

$$A_{\mathrm{eq}}\alpha = 0_{2\times 1} \iff \widehat{y}(a_i - \epsilon) = \widehat{y}(a_i - \epsilon), i \in \{2, 3\},$$

which is what continuity is all about.

We then solve the least squares problem with a linear constraint, namely

$$\alpha^* = \underset{\substack{\alpha\,\in\,\mathbb{R}^{10} \\ A_{eq}\alpha = 0}}{\arg\min} \; ||Y - \Phi\alpha||^2. \tag{11.64}$$

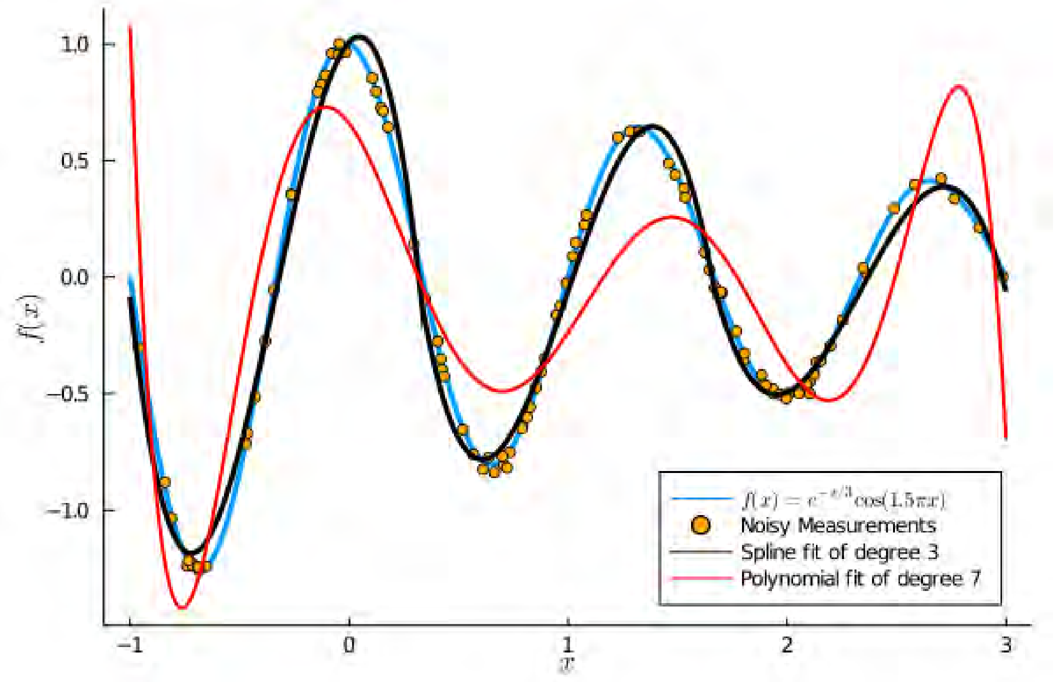


Figure 11.18: The plot shows a spline fit with continuity imposed at the interior knot points. For comparison purposes, a polynomial with the same number of free parameters is shown.

Equation 11.64 is a quadratic program; indeed, one has $Q := \frac{1}{2}\Phi^\top \Phi$, $q := -Y^\top \Phi$, $B_{eq} = 0_{2\times 1}$, and A_{eq} as above. Figure 11.18 shows the resulting fit. We note that indeed, we have removed the discontinuities in the regressed function.

■

Example 11.9 (A Graphical Example) *We provide a graphical QP example to understand the relationship between the cost function and the constraints. Consider the cost function $J(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2$ and the following constraints:*

$$\begin{aligned}
&x_1 + 2x_2 \leq 12\\
&3x_1 + 3x_2 \leq 25\\
&x_1 \leq 7\\
&x_2 \leq 5\\
&x_1 \geq 0\\
&x_2 \geq 0.
\end{aligned}$$

Write the problem in the standard form of (11.55)*, In addition, provide a contour plot of the cost function with an overlay of the constraints.*

Solution: We expand the cost function as

$$J(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2 = x_1^2 + x_2^2 - 4x_1 - 2x_2 + 5.$$

The constant term, 5, has no effect on the optimal solution of this problem, and therefore, it is common to drop it.

Remark: Some software packages for solving QPs might not include a constant term. Remember to take it into account in the end if the actual value of the cost function at the optimal solution is required. In this particular example, because we can visualize the cost function and the constraints, we will keep it.

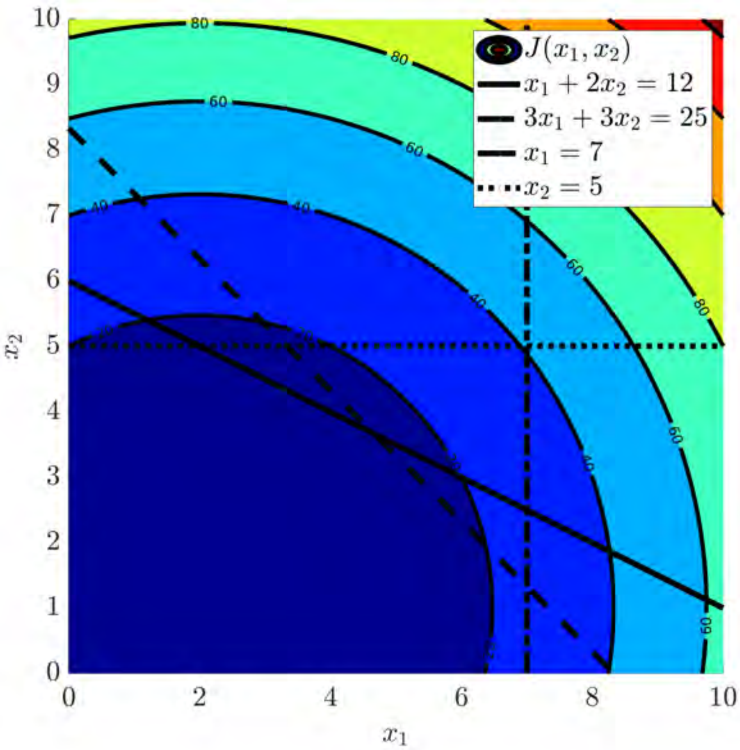


Figure 11.19: The contour plot of the cost function and constraints. The feasible region is visible in the left bottom corner of the figure.

We can now rearrange everything in the form of (11.55):

$$\begin{aligned}
\min_{x\in\mathbb{R}^m} \quad & \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\
\text{subject to} \quad & \begin{bmatrix} 1 & 2 \\ 3 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \preceq \begin{bmatrix} 12 \\ 25 \end{bmatrix} \\
& \begin{bmatrix} 0 \\ 0 \end{bmatrix} \preceq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \preceq \begin{bmatrix} 7 \\ 5 \end{bmatrix}.
\end{aligned}$$

At this point, we are almost done. Find your favorite QP solver (such as the one in Chap. 11.9), enter your problem according to the requested format and press run! Next, enjoy the results!
Because we only have two variables here, we can visualize the cost and constraints. A similar graphical approach would not be viable for large-scale problem where we might have thousands of variables. Figure 11.19 shows a contour plot of the cost function and our constraints. The region where all constraints are satisfied is called the **feasible region**. The optimal value of our problem must lie within the feasible region. Applying the QP solver in Chap. 11.9, we obtain $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$. You can check that it does not violate the constraints. ■

11.9 (Optional Read): QP Solver in Julia

We've had success with the QP solver at https://osqp.org, called OSQP. The standard form used by OSQP is a bit different than (11.55), though it is every bit as general,

$$x^* = \underset{\substack{x \in \mathbb{R}^m \\ \ell \preceq Ax \preceq \mathrm{u}}}{\arg\min} \; \frac{1}{2}x^\top Q x + q^\top x, \tag{11.65}$$

where $x \in \mathbb{R}^m$ is the optimization variable. The objective function is defined by a positive semidefinite $m \times m$ matrix Q and vector $q \in \mathbb{R}^m$. The linear inequality and equality constraints as well as upper and lower bounds are grouped together and defined by a single $n \times m$ matrix A and two $n \times 1$ vectors ℓ and u, where each component of $\ell_i \in \mathbb{R} \cup \{-\infty\}$ and $\mathrm{u}_i \in \mathbb{R} \cup \{+\infty\}$, $i \in \{1, \dots, n\}$. To impose equality constraints, one sets the corresponding entries of ℓ and u to be equal to one another. We provide a script and an illustration below to transform a problem in the form of (11.55) to that of (11.65).

```julia
] add OSQP
using Pkg
Pkg.add("Compat")
using OSQP
using Compat.SparseArrays

# Define problem data
P = sparse([4. 1.; 1. 2.])
q = [1.; 1.]
A = sparse([1. 1.; 1. 0.; 0. 1.])
l = [1.; 0.; 0.]
u = [1.; 0.7; 0.7]

# Create OSQP object
prob = OSQP.Model()

# Setup workspace and change alpha parameter
OSQP.setup!(prob; P=P, q=q, A=A, l=l, u=u)

# Solve problem
results = OSQP.solve!(prob)
```

```julia
using LinearAlgebra

function quadProg(Q,q,Aineq,Bineq,Aeq,Beq,lb,ub,tol=1e-5)
    # Begin wrapper to make QP solover in OSQP similar to quadprog in Matlab
    dimX = length(q)
    myI=sparse(zeros(dimX,dimX)+I)
    tolIneq=10.0*tol
    tolEq=0.1*tol
    # Define problem data
    P = Q
    q = q
    A = Aineq
    u = Bineq
    l = Bineq .-Inf
    if (length(lb)>0)||(length(ub)>0)
        A = [A;myI]
    end
    if (length(ub)>0)&(length(lb)>0)
        u=[u;ub]
        l=[l;lb]
    elseif (length(ub)>0)&(length(lb)==0)
        u=[u;ub]
        l=[l;ub.-Inf]
    elseif (length(ub)==0)&(length(lb)>0)
        l=[l;lb]
        u=[u;lb.+Inf]
    end
    (nreq,nceq)=size(Aeq)
    if nreq > 0
        A=[A;Aeq]
        l=[l;Beq.-tolEq]
        u=[u;Beq.+tolEq]
    end
    # End wrapper

# Create OSQP object
prob = OSQP.Model()

# Setup workspace and change alpha parameter
OSQP.setup!(prob; P=P, q=q, A=A, l=l, u=u)

# Solve problem
results = OSQP.solve!(prob)
    return results.x
end

# Example problem data (same as above)
P = sparse([2. 0.; 0. 2.])
q = [-4.; -2.]
A = sparse([1. 2.; 3. 3.; 1. 0.; 0. 1.])
l = [0.; 0.; 0.; 0.]
u = [12.; 25.; 7.; 5.]

dimX=length(q)
Aeq = Array{SparseMatrixCSC,2}(undef,0,dimX)
Beq = Vector{Float64}(undef,0)
lb = Vector{Float64}(undef,dimX).-Inf
ub = Vector{Float64}(undef,dimX).+Inf

xStar = quadProg(P,q,[A;-A],[u;-l],Aeq,Beq,lb,ub)
```

11.10 (Optional Read): Optimization Packages: The Sky is the Limit

Once you’ve coded up a few optimization algorithms, it’s time to move over and let the pros handle the programming while you focus on problem formulation. Currently, the best source for optimization packages in Julia is https://jump.dev/. From the **JuMP** homepage we learn that “JuMP is a modeling language and supporting packages for mathematical optimization in Julia. JuMP makes it easy to formulate and solve linear programming, semidefinite programming, integer programming, convex optimization, constrained nonlinear optimization, and related classes of optimization problems. You can use it to route school buses, schedule trains, plan power grid expansion, or even optimize milk output.”

What are you waiting for? There is a lot of knowledge out there. Slowly but surely, you can master it!

Appendix A

Background for Machine Learning

Learning Objectives

- Introduce material that is assumed in UofM Computer Science courses that have Math 214 as a prerequisite.
- Provide a resource for use after you leave ROB 101.

Outcomes

- Learn how to separate $\mathbb{R}^n$ into two halves via hyperplanes
- Learn the Orthogonal Projection Theorem, which is the geometric tool that underlies most least squares problems
- The notion of signed distance to a hyperplane
- An example of a max-margin classifier, a common tool in Machine Learning

A.1 Separating Hyperplanes

We continue with a geometric development that is a natural accompaniment to Chapter 9: linear structures than can be used to divide $\mathbb{R}^n$ into two pieces. The notes are based on lectures by Prof. Maani Ghaffari. This material is used in EECS 445, the undergraduate Machine Learning course, where one seeks to separate observations of a process into two categories, such as spam versus regular email, images of cats versus dogs, or a smooth walking surface versus one that undulates. The observations are typically given in the form of n-vectors and are called *object features*. Once you can handle the task of separating two categories, you are on the road to handling multiple categories, as in Fig. A.1.

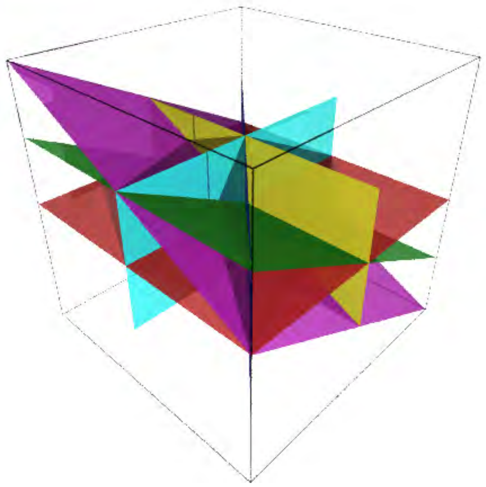


Figure A.1: This awesome figure shows multiple (hyper)planes dividing $\mathbb{R}^3$ into disjoint regions, where each region could contain features describing a different object. In this book, we will content ourselves with a single hyperplane and do it in general $\mathbb{R}^n$. Image courtesy of Kilom691 https://commons.wikimedia.org/w/index.php?curid=37508909.

We will develop the notion of a "hyperplane" as a linear object that is big enough to divide $\mathbb{R}^n$ into two halves, easy to manipulate, and can take on "any orientation or position." In $\mathbb{R}^2$, any line can divide the space into two half spaces. In $\mathbb{R}^3$, a line is not "big enough" to divide the space into two parts, though the the classical notion of a plane does the job perfectly well. In $\mathbb{R}^n$, the appropriate generalization is called a **hyperplane**!

Before we dig into the details, we firm up concepts in $\mathbb{R}^2$. While we skip the case of the real line, $\mathbb{R}$, it does provide some insight because a single point $x_c \in \mathbb{R}$ can be used to divide the real line into two halves, $H^- := \{x \in \mathbb{R} \,| x < x_c\}$ and $H^+ := \{x \in \mathbb{R} \,| x > x_c\}$. Moreover, the object being used to divide the vector space $\mathbb{R}$ into two halves has dimension zero, which is one less than the dimension of $\mathbb{R}$! Hold this thought.

A.1.1 Lines in $\mathbb{R}^2$ as Separating Hyperplanes

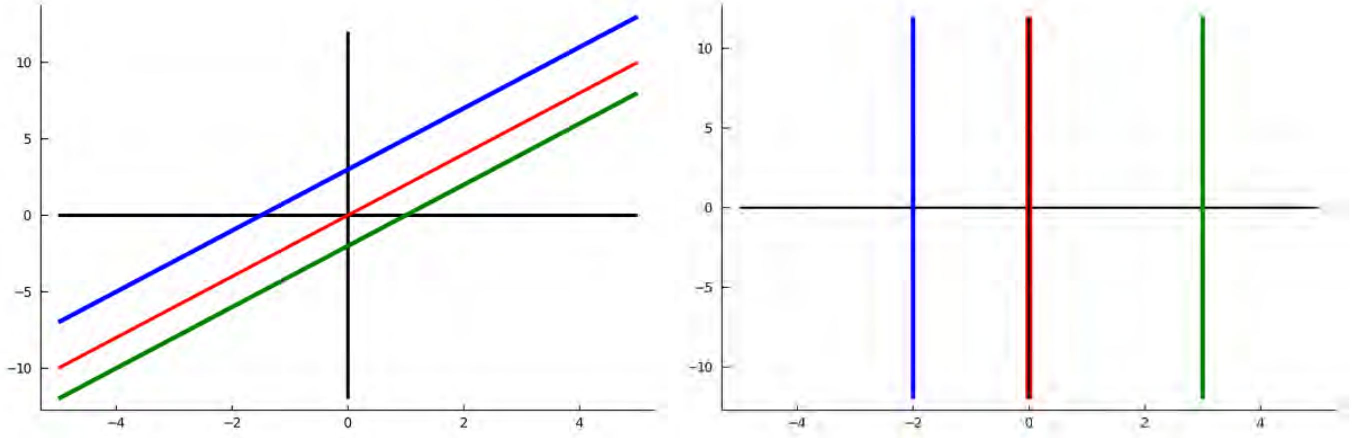
While we are very used to describing lines as things that satisfy formulas of the form $x_2 = mx_1 + b$, let's note that this description leaves out the x_2-axis, which is a perfectly fine line in $\mathbb{R}^2$. It also leaves out all lines parallel to the x_2-axis. Why is the x_2-axis not covered by this description? Because it's slope would be infinity, which is not allowed! A better way to describe a line is actually as a special kind of subset of $\mathbb{R}^2$, such as

$$\text{Line} := \{(x_1, x_2) \in \mathbb{R}^2 \mid a_0 + a_1x_1 + a_2x_2 = 0\},$$

where at least one of a_1 and a_2 is non-zero. Indeed, with this formulation,

$$x_2\text{-axis} = \{(x_1, x_2) \in \mathbb{R}^2 \mid 0 + x_1 + 0x_2 = 0\}.$$

Claim 1: Every line in $\mathbb{R}^2$ can be written as the zero set of $y(x_1, x_2) = a_0 + a_1x_1 + a_2x_2$, where at least one of a_1 and a_2 is non-zero.



(a)

(b)

Figure A.2: We know that subspaces must contain the origin. Lines in $\mathbb{R}^2$ can be viewed as translations of subspaces (loosely speaking, this means you slide them up, down, or sideways, without rotating them). In both figures, the lines corresponding to subspaces are in red while their translations are in green and blue. The blue and green lies are parallel to the red line, but are offset or translated. In (a), the lines correspond to $0 = a_0 + 1.0x_1 - 2.0x_2$, where $a_0 = 0.0$ (red), $a_0 = 3.0$ (blue), and $a_0 = -2.0$ (green) (b) The lines correspond to $0 = a_0 + 1.0x_1 + 0.0x_2$, where $a_0 = 0.0$ (red), $a_0 = 2.0$ (blue) , and $a_0 = -3.0$ (green)

Proof: If the line is given by $x_2 = mx_1 + b$, then it is the zero set of $y(x_1, x_2) = b + mx_1 - x_2$, that is, $a_0 = b$, $a_1 = m$ and $a_2 = -1$. If the line is parallel to the x_2-axis, as in $\{(x_1, x_2) \mid x_1 = a_0 = \text{a constant}, x_2 \in \mathbb{R}\}$, then we can take $y = a_0 - x_1 + 0x_2$, that is, $a_1 = -1$ and $a_2 = 0$. ■

While it may not be apparent that writing a line as the zero set of a function has any value, we next note that the function

$$y(x_1, x_2) = a_0 + a_1x_1 + a_2x_2$$

can also be used to divide $\mathbb{R}^2$ into two halves. Indeed, we define

$$\begin{aligned} H^+ &:= \{(x_1, x_2) \in \mathbb{R}^2 \, | y(x_1, x_2) = a_0 + a_1x_1 + a_2x_2 > 0\} \\ H^- &:= \{(x_1, x_2) \in \mathbb{R}^2 \, | y(x_1, x_2) = a_0 + a_1x_1 + a_2x_2 < 0\}. \end{aligned}$$

This is illustrated in Fig A.3, where the line in red is the set where $y(x_1, x_2) = a_0 + a_1x_1 + a_2x_2 = 0$, showing the utility of thinking of a line as a zero set of a function.

We want to extend these ideas to $\mathbb{R}^n$ for $n > 2$. While we could stick with formulas of the form

$$y(x_1, \ldots, x_n) = a_0 + a_1x_1 + \cdots + a_nx_n, \tag{A.1}$$

a more insightful analysis comes about from working directly with subspaces, which was hinted at in Fig. A.1 and A.2.

A.1.2 Hyper Subspaces

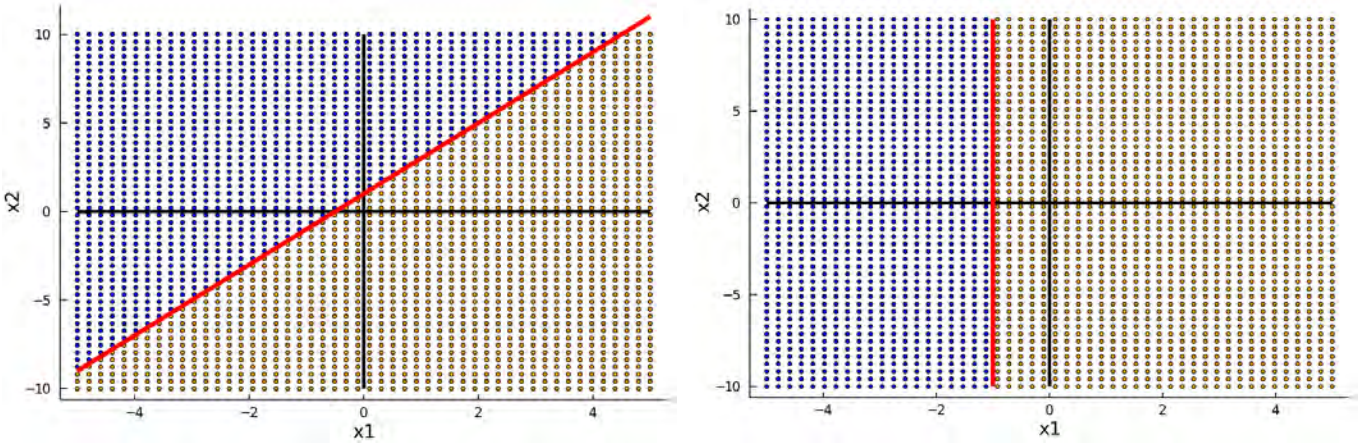
Consider the vector space $\mathbb{R}^n$ and let A be a $1 \times n$ matrix (you can also call it a row vector). We assume that A is non-zero, meaning that at least one of its components is non-zero. Viewing A as a matrix, we know that its null space

$$N := \text{null}(A) = \{x \in \mathbb{R}^n \mid Ax = 0\}$$

is a subspace of $\mathbb{R}^n$. By the **Rank-Nullity Theorem**, the dimension of N is equal to $n - 1$, one less than the dimension of $\mathbb{R}^n$. Why, because $\text{rank}(A) = 1$ due to our assumption that at least one of its columns[1] is nonzero and

$$\dim(N) = \text{nullity}(A) = \dim(\mathbb{R}^n) - \text{rank}(A) = n - 1.$$

[1] For a $1 \times n$ matrix, elements and columns are the same thing!



(a)

(b)

Figure A.3: Two examples of half spaces corresponding to (a) $y = -1.0 - 2.0x_1 + x_2$ (b) $y = -1.0 - 1.0x_1 + 0.0x_2$. The line $y = a_0 + a_1x_1 + a_2x_2 = 0$ is shown in red, while in blue is plotted the set where $y > 0$ and in brown and the set where $y < 0$. The x_1-axis and x_2-axis are in black.

A subspace with dimension one less than the ambient space in which it lives, which in our case is $\mathbb{R}^n$, is called a **co-dimension one** subspace. Though less common, you can also call it a **hyper-subspace**!

We've just seen that the null space of a rank one matrix gives rise to a co-dimension one subspace. Are all co-dimension one subspaces the null space of some matrix? The answer is yes and the easiest way to show it is by using the dot product and the Gram-Schmidt process! What? You did not see that coming?

We write $A =: a^\top$ where

$$a := \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^n.$$

We do this because

$$x \in \text{null}(A) \iff Ax = 0 \iff a^\top x = 0 \iff a \bullet x = 0 \iff x \bullet a = 0 \iff x \perp a.$$

Hence, our question of whether every co-dimension one subspace can be expressed as the null space of a rank one matrix can be rephrased as "is every co-dimension one subspace equal to the set of all vectors that are orthogonal to a non-zero vector $a \in \mathbb{R}^n$?" To answer this question, we can invoke Gram-Schmidt.

We let $N \subset \mathbb{R}^n$ be a co-dimension one subspace, meaning $\dim(N) = n - 1$. We let $\{u_1, \ldots, u_{n-1}\}$ be a basis for N. Because N is not all of $\mathbb{R}^n$, there must exist a non-zero vector $u_n \in \mathbb{R}^n$ such hat $u_n \notin N$. We skip the details, but you can then show that

$$\{u_1, \ldots, u_{n-1}, u_n\}$$

is a linearly independent set. We apply Gram-Schmidt to produce an orthogonal basis $\{v_1, \ldots, v_{n-1}, v_n\}$. By (9.22), we have that

$$N = \text{span}\{u_1, \ldots, u_{n-1}\} = \text{span}\{v_1, \ldots, v_{n-1}\}.$$

Moreover,

$$x \in N \iff x = \alpha_1 v_1 + \cdots \alpha_{n-1} v_{n-1} \iff x \perp v_n \iff v_n \bullet x = 0$$

Hyper Subspaces and Dot Products

The following are equivalent for a subspace $N \subset \mathbb{R}^n$:

- $\dim(N) = n - 1$, that is, N is a co-dimension one subspace;
- there exists $a \in \mathbb{R}^n$ not equal to zero such that $x \in N \iff x \perp a$; and
- there exists a $1 \times n$ matrix A such that $A \neq 0_{1\times n}$ and $N = \text{null}(A)$.

We note that the matrix A and the vector a are related by $A = a^\top$.

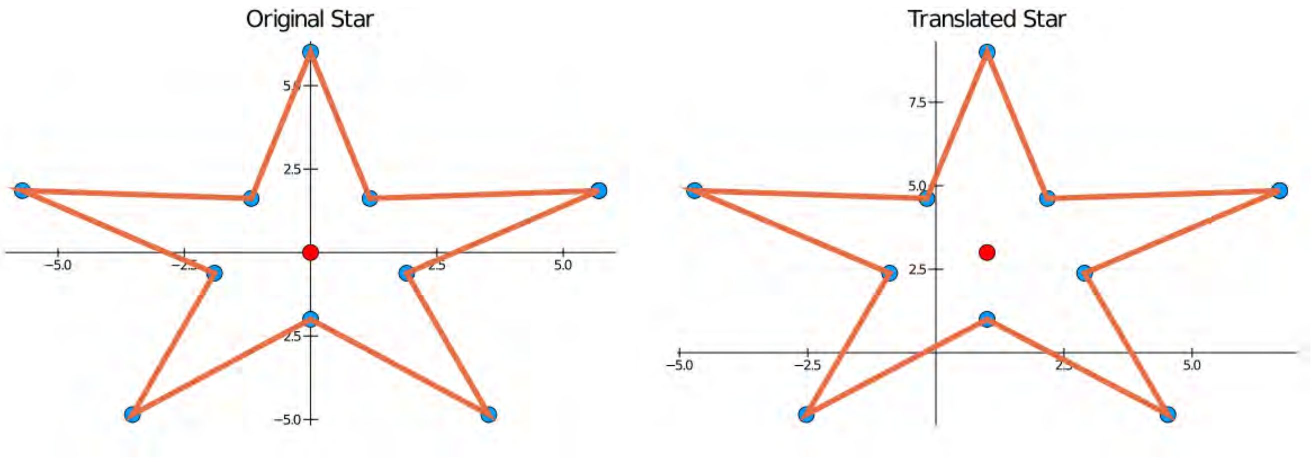
Example A.1 *Consider a matrix* $B = \begin{bmatrix} 1 & -1 \\ -1 & 2 \\ 0 & 1 \end{bmatrix}$ *and let* $N :=$ *col span*$\{B\}$*. It is clear that* $N \subset \mathbb{R}^3$ *and* $\dim(N) = 2$*. Find a vector* $a \in \mathbb{R}^3$ *such that*

$$N = \{x \in \mathbb{R}^3 \mid a \bullet x = 0\}.$$

Solution: We define $u_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$, $u_2 = \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix}$, and note that $u_3 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ is linearly independent of $\{u_1, u_2\}$. Applying Gram Schmidt with normalization to $\{u_1, u_2, u_3\}$ yields

$$\begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} 0.707107 & 0.408248 & 0.57735 \\ -0.707107 & 0.408248 & 0.57735 \\ 0.000000 & 0.816497 & -0.57735 \end{bmatrix}.$$

Hence, we can take $a = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$. It is easily checked that $a \bullet u_1 = 0$ and $a \bullet u_2 = 0$, and thus $N = \{x \in \mathbb{R}^3 \mid a \bullet x = 0\}$. ■



(a)

(b)

Figure A.4: Let $S \subset \mathbb{R}^2$ be the star-shaped object in (a) and let x_c be the vector $[2; 3]$. Then $x_c + S$ is the star-shaped object in (b), where each and every point of the object has been shifted by x_c. If you can handle this, then you should have no trouble handling the translation of a line or a plane! Image courtesy of Tribhi Kathuria.

A.1.3 Translations of Sets, Hyper Subspaces, and Hyperplanes

Definition Let $S \subset \mathbb{R}^n$ be a subset and and let $x_c \in \mathbb{R}^n$ be a vector. We define the translation of S by x_c as

$$x_c + S := \{x_c + x \mid x \in S\}.$$

Note that, because S consists of vectors in $\mathbb{R}^n$, the addition in the above formula makes sense. Figure A.4 provides an illustration.

Claim: Let $N = \{x \in \mathbb{R}^n \mid a \bullet x = 0\} \subset \mathbb{R}^n$ be a hyper subspace (means that $a \neq 0_{n\times 1}$) and let $x_c \in \mathbb{R}^n$ be a vector. Then their sum has a simple description as

$$x_c + N = \{x \in \mathbb{R}^n \mid a \bullet (x - x_c) = 0\} = \{x \in \mathbb{R}^n \mid a \perp (x - x_c)\}. \tag{A.2}$$

The proof is not important, but we give it for those who are interested. N consists of everything in $\mathbb{R}^n$ that is orthogonal to a. Hence, if $(x - x_c) \perp a$, then $(x - x_c) \in N$. Adding x_c to both sides, we have that $x \in x_c + N$. The other direction is similar. ■

Hyperplanes are Translations of Hyper Subspaces

Let $N = \{x \in \mathbb{R}^n \mid a \bullet x = 0\} \subset \mathbb{R}^n$ be a hyper subspace (means that $a \neq 0_{n\times 1}$) and let $x_c \in \mathbb{R}^n$ be a vector. Then

$$H := x_c + N \tag{A.3}$$

is called a **hyperplane**. Moreover, by (A.2), the real-valued function $y : \mathbb{R}^n \to \mathbb{R}$ defined by

$$y(x) := a \bullet (x - x_c) \tag{A.4}$$

vanishes on $H = x_c + N$ (because $H = \{x \in \mathbb{R}^n \mid y(x) = a \bullet (x - x_c) = 0\}$). It follows that $y(x)$ can be used to divide $\mathbb{R}^n$ into two halves

$$\begin{aligned} H^+ &:= \{x \in \mathbb{R}^n \mid y(x) > 0\} \\ H^- &:= \{x \in \mathbb{R}^n \mid y(x) < 0\}. \end{aligned} \tag{A.5}$$

A fanciful illustration is given in fig. A.5.

Remarks:

- Without loss of generality, it is always possible to take $x_c = \alpha a$, for $\alpha \in \mathbb{R}$.

- One can go back and forth between (A.4) and (A.1) by

$$a_0 = -a \bullet x_c \text{ and } x_c = -a_0 \frac{a}{||a||} \tag{A.6}$$

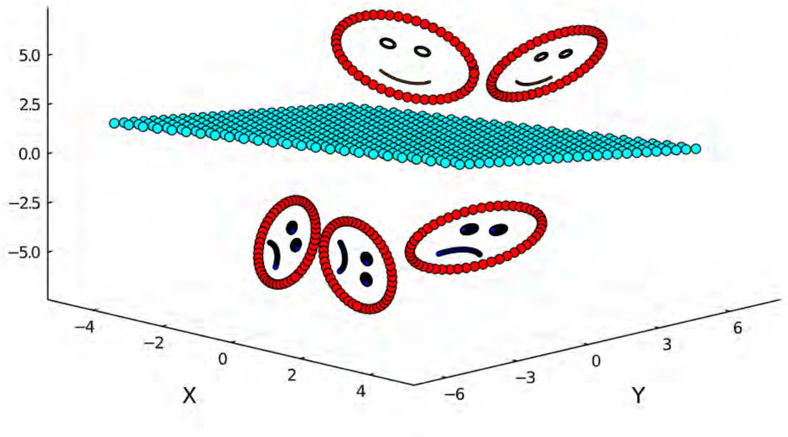


Figure A.5: A separating hyperplane where the features are smiles versus frowns. In Example A.9, we show how to design the parameters of the hyperplane, that is, $a \in \mathbb{R}^n$ and $x_c \in \mathbb{R}^n$, so as to achieve separation for given data on the features.

A.2 Orthogonal Projection

We extend the importance of the dot product (aka, inner product) by showing its fundamental role in least squares problems.

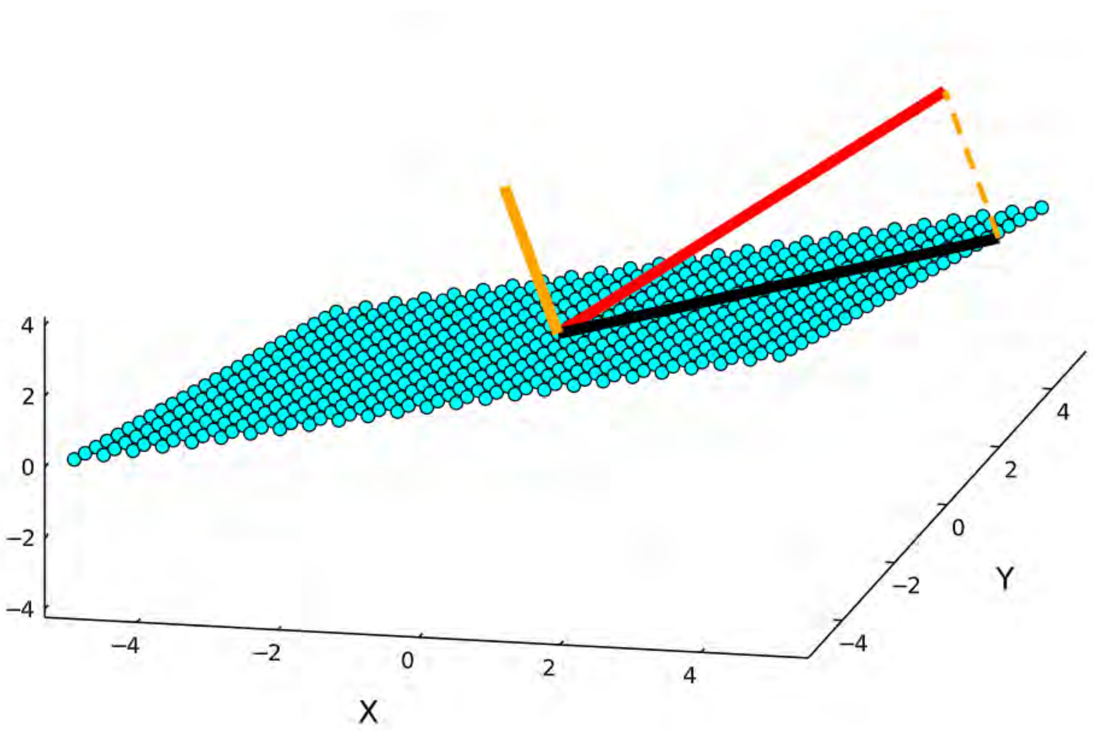


Figure A.6: A vector (in red) is orthogonally projected (in black) onto a subspace (in cyan). The error vector (in solid orange) is orthogonal to the plane. This characterizes the orthogonal projection process. The vector in dashed orange is the error vector drawn to highlight that the error forms a right angle with the projection of the vector.

Review

Consider the vector space $\mathbb{R}^n$, which we view as the set of all $n \times 1$ column vectors of real numbers. Let $v, w \in \mathbb{R}^n$ and let $V \subset \mathbb{R}^n$ be a subspace.

- $v \bullet w := v^\top w$.
- $w \bullet v = v \bullet w$.
- $v \perp w \iff v \bullet w = 0$.
- $v \perp w \implies ||v + w||^2 = ||v||^2 + ||w||^2$ (Pythagorean Theorem).
- Let $\{u_1, u_2, \dots, u_m\}$ be a basis for V. Then Gram-Schmidt produces an **orthogonal basis** that also satisfies, for all $1 \leq k \leq m$,
$$\text{span}\{u_1, u_2, \dots, u_k\} = \text{span}\{v_1, v_2, \dots, v_k\}.$$
Moreover, by the simple step of adding normalization to Gram-Schmidt, we can assume that $\{v_1, v_2, \dots, v_m\}$ is an **orthonormal basis** for V.

Projection Theorem: The Underlying Tool that Solves all Least Squares Problems

Let V be a subspace of $\mathbb{R}^n$ and let x_0 be an arbitrary point in $\mathbb{R}^n$. Then there exists a unique vector $x^* \in V$ such that

$$||x_0 - x^*|| = \min_{x \in V} ||x_0 - x||;$$

as before, we denote this vector by $x^* = \arg\min_{x \in V} ||x_0 - x||$ or by $x^* = \arg\min_{x \in V} ||x_0 - x||^2$. Moreover, the solution to the least squared error problem is uniquely characterised by

$$x^* = \arg\min_{x \in V} ||x_0 - x||^2 \iff (x_0 - x^*) \perp V \text{ and } x^* \in V. \tag{A.7}$$

The vector $x_0 - x^*$ is called the **error vector**. The vector x^* is called the **orthogonal projection of $\mathbf{x_0}$ onto V** precisely because the error vector is orthogonal to V. Recalling the Pythagorean Theorem, we have that

$$||x_0 - x^*||^2 + ||x^*||^2 = ||x_0||^2;$$

once again emphasizing that x^*, $x_0 - x^*$, and x_0 form a “generalized right triangle”.

You already know one way to compute x^* from the Projection Theorem! Really? Yes, Gram Schmidt. If $x_0 \in V$, the solution to the problem is trivial, namely $x^* = x_0$, because then the error is zero, which is as small as it gets. Hence, suppose $x_0 \notin V$ and let $\{u_1, u_2, \dots, u_m\}$ be a basis for V. Then we leave it to you to show that

$$x_0 \notin V \iff x_0 \notin \text{span}\{u_1, u_2, \dots, u_m\} \iff \{u_1, u_2, \dots, u_m, x_0\} \text{ is linearly independent.}$$

We apply Gram-Schmidt[2] to the set $\{u_1, u_2, \dots, u_m, x_0\}$. The last step gives that

$$v_{m+1} = x_0 - \sum_{k=1}^{m} \frac{x_0 \bullet v_k}{v_k \bullet v_k} v_k,$$

and moreover, we know that

$$v_{m+1} \perp \text{span}\{v_1, v_2, \dots, v_m\} = V.$$

Hence, by the Projection Theorem,

$$\boxed{x^* = \sum_{k=1}^{m} \frac{x_0 \bullet v_k}{v_k \bullet v_k} v_k,} \tag{A.8}$$

because $x_0 - x^* = v_{m+1}$ and $v_{m+1} \perp V$.

Remark: Once we know that (A.8) is true, we can simply apply Gram-Schmidt to any basis of V to produce an orthogonal basis and then apply (A.8). If we produce an **orthonormal basis**, then we know that $v_k \bullet v_k = 1$ and the formula simplifies to

$$\boxed{x^* = \sum_{k=1}^{m} (x_0 \bullet v_k)\, v_k = \sum_{k=1}^{m} \langle x_0, v_k \rangle\, v_k,} \tag{A.9}$$

where we have recalled our alternative notation for an inner product.

A second way to compute the solution follows from (A.7) and leads to the **Normal Equations**. Once again, let $\{u_1, u_2, \dots, u_m\}$ be any basis for V. Because we know that $x^* \in V$, we can pose

$$x^* = \alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_m u_m \tag{A.10}$$

as a linear combination of basis vectors for V and seek the conditions on the coefficients $\alpha_1, \alpha_2, \dots, \alpha_m$ so that

$$x_0 - x^* \perp V.$$

[2] We do not assume normalization, but you can also do that.

You can quickly convince yourself that

$$x_0 - x^* \perp V \iff x_0 - x^* \perp u_k, 1 \leq k \leq m.$$

The above constitutes m-equations, one for each k, and leads to the famous Normal Equations,

$$\underbrace{\begin{bmatrix} u_1 \bullet u_1 & u_1 \bullet u_2 & \cdots & u_1 \bullet u_m \\ u_2 \bullet u_1 & u_2 \bullet u_2 & \cdots & u_2 \bullet u_m \\ \vdots & \vdots & \ddots & \vdots \\ u_m \bullet u_1 & u_m \bullet u_2 & \cdots & u_m \bullet u_m \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}}_{\alpha} = \underbrace{\begin{bmatrix} u_1 \bullet x_0 \\ u_2 \bullet x_0 \\ \vdots \\ u_m \bullet x_0 \end{bmatrix}}_{\beta}. \tag{A.11}$$

The matrix G is called the **Gram matrix** and is invertible if, and only if, the set $\{u_1, u_2, \ldots, u_m\}$ is linearly independent. We note the the ij-entry of it is

$$G_{ij} = u_i \bullet u_j = u_i^\top u_j.$$

We'll let you work out that if you take a basis for V that is orthogonal, then G is a diagonal matrix, and if you take an orthonormal basis for V, then G is the identity matrix!

We summarize the various solutions in the following:

Computing the Solution Given by the Projection Theorem

Let V a subspace of $\mathbb{R}^n$ and $x_0 \in \mathbb{R}^n$ be given. Then $x^* = \underset{x \in V}{\arg\min} \, ||x_0 - x||^2$, the orthogonal projection of x_0 onto V, can be computed by

- $x^* = \sum_{k=1}^m (x_0 \bullet v_k) \, v_k$ if $\{v_1, \cdots, v_m\}$ is an **orthonormal basis** for V;
- $x^* = \sum_{k=1}^m \frac{x_0 \bullet v_k}{v_k \bullet v_k} v_k$ if $\{v_1, \cdots, v_m\}$ is an **orthogonal basis** for V;
- $x^* = \alpha_1 u_1 + \alpha_2 u_2 + \cdots \alpha_m u_m$, where $G\alpha = \beta$ are the Normal Equations given in (A.11), if $\{u_1, \cdots, u_m\}$ is **any basis** for V.

You instructors use all of these forms of the solution at various times when solving problems.

Example A.2 *We'll warm up on a simple example. Consider a subspace given by $V = span\{u_1, u_2\}$, where*

$$u_1 = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \end{bmatrix}, u_2 = \begin{bmatrix} 2.5 \\ 0.0 \\ 1.0 \end{bmatrix}.$$

Compute the orthogonal projection of $x_0 = \begin{bmatrix} 4.0 \\ 4.0 \\ 4.0 \end{bmatrix}$ onto V. Moreover, compute

$$x^* = \underset{x \in V}{\arg\min} \, ||x_0 - x||^2$$

in at least two different ways.

Solution A: We apply the normal equations

$$G = \begin{bmatrix} u_1^\top u_1 & u_1^\top u_2 \\ u_2^\top u_1 & u_2^\top u_2 \end{bmatrix} = \begin{bmatrix} 2.00 & 2.50 \\ 2.50 & 7.25 \end{bmatrix}$$

$$\beta = \begin{bmatrix} u_1^\top x_0 \\ u_2^\top x_0 \end{bmatrix} = \begin{bmatrix} 8.00 \\ 14.00 \end{bmatrix}$$

$$G\alpha = \beta \implies \alpha = \begin{bmatrix} 2.79 \\ 0.97 \end{bmatrix}$$

$$x^* = \alpha_1 u_1 + \alpha_2 u_2 = \begin{bmatrix} 5.21 \\ 2.79 \\ 0.97 \end{bmatrix}.$$

The results are illustrated in Fig. A.6.

Solution B: We find an orthonormal basis for V and apply (A.9). We use Gram-Schmidt with normalization and find that $V = \text{span}\{v_1, v_2\}$, for

$$v_1 = \begin{bmatrix} 0.707 \\ 0.707 \\ 0.000 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} 0.615 \\ -0.615 \\ 0.492 \end{bmatrix}.$$

Hence,

$$x^* = \left(v_1^\top x_0\right) v_1 + \left(v_2^\top x_0\right) v_2 = 5.657 v_1 + 1.969 v_2 = \begin{bmatrix} 5.212 \\ 2.788 \\ 0.970 \end{bmatrix}.$$

Solution C: Finally, we use an orthogonal basis for V and apply (A.8). For our orthogonal basis, we apply Gram-Schmidt without normalization and obtain $V = \text{span}\{v_1, v_2\}$, for

$$v_1 = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} 1.25 \\ -1.25 \\ 1.00 \end{bmatrix}.$$

Hence,

$$x^* = \frac{v_1^\top x_0}{v_1^\top v_1} v_1 + \frac{v_2^\top x_0}{v_2^\top v_2} v_2 = 4.0 v_1 + 0.970 v_2 = \begin{bmatrix} 5.212 \\ 2.788 \\ 0.970 \end{bmatrix}.$$

■.

In this next example, we apply the Normal Equations to our very first least squares problem in (8.7)! The example is essentially a proof showing how to derive our original result from the Projection Theorem. Trigger Warning: This is not for the faint of heart. Your instructors did not learn this as undergraduates, much less, first-year undergraduates! If you skip the example, we highly recommend the summary that follows it.

Example A.3 *Consider a system of linear equations $Ax = b$, where A is $n \times m$ and its columns are linearly independent. Define*

$$V := \textit{col span}\{A\}.$$

Relate the following two least squares problems

- $x^* = \underset{x \in \mathbb{R}^m}{\arg\min} \, ||Ax - b||^2$
- $v^* = \underset{v \in V}{\arg\min} \, ||b - v||^2,$

where we renamed the solution of the second optimization problem as v^* to avoid confusion later on. (Yikes! It must not be easy.)

Solution The first least squares problem is well known to us from (8.7), which we repeat here for clarity

$$x^* = \underset{x \in \mathbb{R}^m}{\arg\min} \, ||Ax - b||^2 \iff A^\top A x^* = A^\top b,$$

which we've always interpreted as the least squared error solution to over determined equations. Moreover, this provided the basis for our work on regression, which was, we recall, pretty awesome.

The second least squares problem is still kind of a mystery to us. If we believe what we were were told about its solution, then v^* is the orthogonal projection of b onto the column span of the matrix A. What could that possibly mean? Well, let's find out!

We write A out in terms of its columns, $A = \left[\begin{array}{cccc} A_1 & A_2 & \ldots & A_m \end{array} \right]$, so that

$$\text{col span}\{A\} = \text{span}\{A_1, A_2, \ldots, A_m\}.$$

When we compute the Gram matrix, we recall that $G_{ij} = A_i \bullet A_j = A_i^\top A_j$, which is precisely the ij-entry of $A^\top A$, and thus

$$G = A^\top A,$$

an amazing coincidence! We'll let you work out a few examples to see that this is true or we'll let you work out a proof! Moreover, when we compute the i-th entry of β we obtain $\beta_i = A_i \bullet b = A_i^\top b$, so that

$$\beta = A^\top b,$$

another amazing coincidence! (Or, perhaps not!). Finally, we note (aka, "let you work out") that

$$\sum_{k=1}^{m} \alpha_k A_k = A\alpha,$$

which should either be setting off alarms in your head because this many coincidences should never happen.....except for a reason! **The two least squares problems are really one and the same problem.**

To see this, we summarize what we have

- $v^* = \underset{v \in \text{col span}\{A\}}{\arg\min} \; ||b - v||^2 \iff \left(A^\top A\alpha = A^\top b \text{ and } v^* = A\alpha\right).$
- $x^* = \underset{x \in \mathbb{R}^m}{\arg\min} \, ||Ax - b||^2 \iff A^\top A x^* = A^\top b.$
- Hence, $\alpha = x^*$, and $v^* = Ax^*$ is the orthogonal projection of b onto the column span of A.
- By projecting b onto the column span of A, we have that

  $$v^* \in \text{span}\{A_1, A_2, \ldots, A_m\},$$

  and hence $Ax = v^*$ has a solution. Kind of clever, isn't it!
- The fact that all of that is being accomplished by the simple equation $A^\top A x^* = A^\top b$ is one of the **Wonders of Linear Algebra**. It's a really beautiful subject and we hope you will want to learn more about it. There is so much more to it than what we have covered in the main parts of this book.

■

Remark: This is a heavy result, so it will take you some time to wrap your head around it. The main message is that the Projection Theorem and the Normal Equations are your main tools when you approach new least squares problems. They have extensions to settings that you cannot even imagine right now. But they are always there, providing theoretical and computational support for solving least squares problems of so many kinds.

It is clear why we did not broach this result in our main treatment of least squares problems. We would have sent you running and screaming for any course but ROB 101!

Summary of the Previous Example

Suppose that $\{u_1, u_2, \ldots, u_m\}$ is a basis for a subspace $V \subset \mathbb{R}^n$ and $x_0 \in \mathbb{R}^n$. Form a matrix U with the basis vectors as its columns, that is,

$$U = \left[\begin{array}{cccc} u_1 & u_2 & \ldots & u_m \end{array}\right].$$

Then the solution to $x^* = \arg\min\limits_{x \in V} ||x_0 - x||^2$ is given by

$$\boxed{U^\top U \alpha^* = U^\top x_0,\ x^* = U\alpha^*.}$$

A.3 Signed Distance and Orthogonal Projection onto Hyperplanes

This section seeks to tie together Chapters A.1 and A.2. The material is used in Machine Learning EECS 445. While we doubt it is taught in Math 214, we'll cover it here because we know from Prof. Sindhu that it will help you.

A.3.1 Signed Distance to a Hyperplane

The function $y(x) = a \bullet (x - x_0)$ has another amazing feature: it's absolute value is the distance of a point x from the hyperplane H^0 defined by $H^0 := \{x \in \mathbb{R}^n \mid y(x) = 0\}$. Hence, $y(x)$ is called the signed distance of x to H^0 because, depending in which half plane x lies, the sign of $y(x)$ will be either $+1$ or -1. To make sense of this, we must first define the distance of a point to a subspace and then we will specialize to the case that the subspace is a hyper

From Norms to Distances

Let $V \subset \mathbb{R}^n$ be a subspace and let $x_0 \in \mathbb{R}^n$, $y_0 \in \mathbb{R}^n$, $v_c \in \mathbb{R}^n$ be points. Then

- **Definition** $d(x_0, y_0) := ||x_0 - y_0||$ is called the **distance** from x_0 to y_0.
- **Definition** $d(x_0, V) := \min\limits_{v \in V} ||x_0 - v||$ is called the **distance** from x_0 to V.

The translation of a subspace by vector is called a **linear variety**. Let $W := v_c + V$. Then one can also define

- **Definition** $d(x_0, W) := \min\limits_{w \in W} ||x_0 - w||$ is called the **distance** from x_0 to W.

Fact: If $W = v_c + V$, then $d(x_0, W) = d(x_0 - v_c, V)$.

The somewhat amazing fact is that when V is a hyperplane, the minimization problem defining the distance of a point to the hyperplane has a very simple solution! This only works for hyperplanes, that is linear varieties that are translates of hyper subspaces.

Signed Distance to a Hyperplane

Suppose that $y : \mathbb{R}^n \to \mathbb{R}$ is given by (A.1) or (A.4) and that $||a|| = 1$ (recall that one can go back and forth between the two representations by (A.6)). Let $H^0 := \{x \in \mathbb{R}^n \mid y(x) = 0\}$ be the hyperplane defined by y. Then, for all $x_0 \in \mathbb{R}^n$,

$$|y(x_0)| = d(x, H^0).$$

For this reason, $y(x)$ is called the **signed distance** from x to V. If the $||a|| \neq 1$, then

$$|y(x)| = ||a|| d(x, H^0)$$

and the the signed distance is given by $\frac{1}{||a||} y(x)$.

Example A.4 *Compute the signed distance for a hyperplane defined by*

$$P := \{(x_1, x_2) \in \mathbb{R}^2 \mid 1.5x_1 - 2.0x_2 = -4.0\}.$$

Solution We have the hyperplane is defined by $y : \mathbb{R}^2 \to \mathbb{R}$, where

$$y(x_1, x_2) = 1.5x_1 - 2.0x_2 + 4.0.$$

Hence, $a = [1.5; -2.0]$ and

$$\frac{1}{||a||} y(x_1, x_2) = \frac{1.5x_1 - 2.0x_2 + 4.0}{\sqrt{6.25}}$$

is the signed distance from $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ to the hyperplane P.

■

Example A.5 *Compute the signed distance for a hyperplane defined by*

$$P := x_c + \{x \in \mathbb{R}^n \mid x \perp a\}.$$

Solution We know that the hyperplane is defined by $y : \mathbb{R}^n \to \mathbb{R}$, where

$$y(x) = a \bullet (x - x_c).$$

Hence, $y(x) = \frac{a}{||a||} \bullet (x - x_c)$ gives the signed distance. ■

Example A.6 *For a d-dimensional hyperplane that passes through the origin and is defied by the normal vector* $[a_1; a_2; \ldots, a_d]$, *compute the signed distance function.*

Solution We know that the hyperplane is defined by $y : \mathbb{R}^n \to \mathbb{R}$, where

$$y(x) = a \bullet x$$

Hence, $y(x) = \frac{a}{||a||} \bullet x$ is the signed distance function. ■

A.3.2 Orthogonal Projection onto Linear Varieties (translations of subspaces)

Let $V \subset \mathbb{R}^n$ be a subspace (of any dimension) and $v_c \in \mathbb{R}^n$ be a point. We define the linear variety, $W := v_c + V$, as the translation of the subspace V by the vector v_c. Though it is not very common, one can consider the **orthogonal projection** of a vector $x_0 \in \mathbb{R}^n$ onto the linear variety W. The key idea is to once again pose a best approximation problem and to consider the properties that define its solution, by properly interpreting the Projection Theorem.

Small Extension of the Projection Theorem

Consider a linear variety $W := v_c + V$, where $V \subset \mathbb{R}^n$ is a subspace (of any dimension) and $v_c \in \mathbb{R}^n$ is a point. For $x_0 \in \mathbb{R}^n$ arbitrary, the following are equivalent:

(a) $w^* = \underset{w \in W}{\arg\min}\, ||x_0 - w||^2$.

(b) $w^* = v^* + v_c$, where $v^* = \underset{v \in V}{\arg\min}\, ||(x_0 - v_c) - v||^2$.

(c) $w^* = v^* + v_c$, where $v^* \in V$ and $\big((x_0 - v_c) - v^*\big) \perp V$

(d) $w^* \in W$ and $\big((x_0 - v_c) - (w^* - v_c)\big) \perp V$.

(e) $w^* \in W$ and $(x_0 - w^*) \perp V$.

The last condition, (e), emphasizes that the error term, $x_0 - w^*$, is orthogonal to V. The second condition, (b), shows how to compute w^*: orthogonally project $x_0 - v_c$ onto V, and then add v_c to the answer.

We gave (a) through (e) in the order we would use them in a proof, were we to give it! The order we chose should help you to see how each fact is a small variation of the previous one, while going straight from (a) to (e) would be rather daunting.

Example A.7 *Compute the orthogonal projection of* $x_0 = \begin{bmatrix} 4.0 \\ 4.0 \\ 4.0 \end{bmatrix}$ *onto* $W := v_c + V$, *where* $V = span\{u_1, u_2\}$,

$$u_1 = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \end{bmatrix}, u_2 = \begin{bmatrix} 2.5 \\ 0.0 \\ 1.0 \end{bmatrix}$$

and $v_c = \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \end{bmatrix}$. *Moreover, compute the norm of the error term,* $x_0 - w^*$, *which we now know is the distance of* x_0 *from the linear variety* W.

Solution: Our strategy is we form $\overline{x}_0 := x_0 - v_c = \begin{bmatrix} 3.0 \\ 2.0 \\ 1.0 \end{bmatrix}$ and compute v^*, its orthogonal projection onto V. We then have $w^* = v^* + v_c$ is the orthogonal projection of x_0 onto $W = V + v_c$.

Using our work from Example A.2, Solution B, we have that $V = \text{span}\{v_1, v_2\}$, for

$$v_1 = \begin{bmatrix} 0.707 \\ 0.707 \\ 0.000 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} 0.615 \\ -0.615 \\ 0.492 \end{bmatrix}.$$

Hence,

$$v^* = \left(v_1^\top \overline{x}_0\right) v_1 + \left(v_2^\top \overline{x}_0\right) v_2 = -1.5 v_1 - 0.182 v_2 = \begin{bmatrix} -1.727 \\ -1.273 \\ -0.182 \end{bmatrix}.$$

Hence,

$$w^* = v^* + v_c = \begin{bmatrix} -0.727 \\ 0.727 \\ 2.818 \end{bmatrix}.$$

We next compute

$$d(x_0, W) := \min_{w \in W} ||x_0 - w|| = ||x_0 - w^*|| = 2.08893$$

■

Example A.8 *As a natural continuation of Example A.7, we note that* $W \subset \mathbb{R}^3$ *is a hyperplane. Compute the signed distance of* x_0 *from* W.

Solution: We need to write the hyperplane as the zero set of a function $y : \mathbb{R}^3 \to \mathbb{R}$, where

$$y(x) = a \bullet (x - v_c),$$

and $a \in \mathbb{R}^3$ has norm one. Once again, appealing to Gram-Schmidt, we have that

$$a = \begin{bmatrix} -0.348 \\ 0.348 \\ 0.870 \end{bmatrix}.$$

Doing the required computation yields that the signed distance is

$$y(x_0) = -2.08893.$$

Comparing to our result in Example A.7, we see that

$$y(x_0) = -d(x_0, W),$$

in other words, the terminology "signed distance" is justified! ■

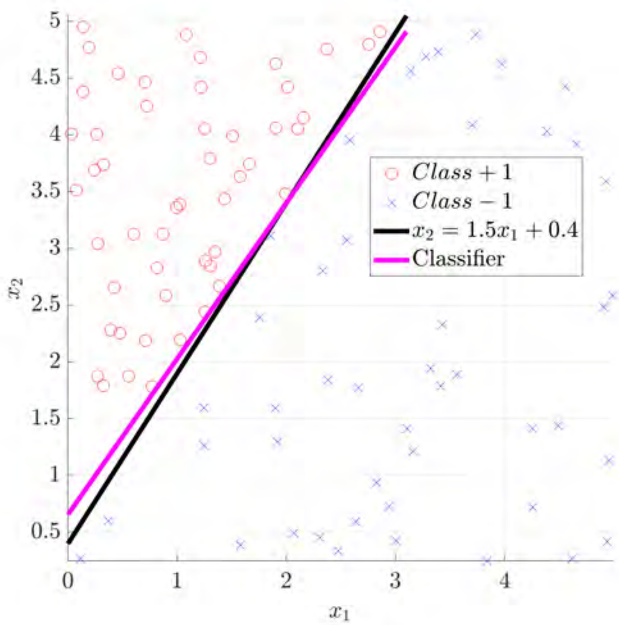


Figure A.7: Maximum margin classifier using quadratic programming.

A.4 Max-margin Classifier

In the next example, we will formulate a (linear) classifier to separate points that belong to two different categories called **class labels**. Our model is linear and separates the space into two half-spaces, as described in Chap. A.1. As shown in Chap. A.1.1, in the 2D plane, a line divides the space into two half-spaces. In general, we could be designing a **separating hyperplane**.

Example A.9 (Maximum Margin Classifier) *Use a QP to find a separating hyperplane (here, a line) for the data given in Fig. A.7. This problem appears in machine learning and is called max-margin classifier. The goal is to build a linear model that can separate two classes of data that we label as* $+1$ *and* -1*. For example, you can think of this as a model that predicts whether an email is a spam or not.*

Solution:
Figure A.7 shows shows a synthetic data set (means we generated it) with red circles, a black line, and blue crosses. We used the black line as our **ground truth** when we generated the data. All we did was randomly generate numbers in $\mathbb{R}^2$, and if they landed above the line, we labeled them with red circles, if the were below the line, we labeled them with blue crosses. Synthetic data generated in this manner is how one tests a problem formulation and solution in practice!

Problems such as this one are called toy examples. They are simple enough that we can visualize and track the solution to ensure our software works as expected. The magenta line in Fig. A.7 is the line we compute to separate the two classes of data. We let you know this ahead of time so that you will read on and see how we did it!

Our dataset consists of 2D vectors (called inputs), $x_i \in \mathbb{R}^2$, and class labels (called target or output), $\ell_i \in \{-1, +1\}$. If we have n data points, we write the dataset as

$$\mathcal{D} = \{(x_i, \ell_i)\}_{i=1}^n.$$

From Chap. A.1, the line (hyperplane) that separates the data can be written as

$$y(x) = a^\top x + a_0 = 0,$$

for $a_0 \in \mathbb{R}$ and $a \in \mathbb{R}^2$. We can also combine the normal vector, a, and the bias, a_0, into

$$w = \begin{bmatrix} a \\ a_0 \end{bmatrix}^\top \in \mathbb{R}^3,$$

and append a one to the inputs as $\big[x_i; 1\big]$ Then the side of the line (hyperplane in general) on which each data point lies can be written as

$$\begin{aligned} w^\top x_i &\geq \ \ 1 \quad \text{if} \quad \ell_i = \ \ 1, \\ w^\top x_i &\leq -1 \quad \text{if} \quad \ell_i = -1. \end{aligned}$$

These constraints state that the data points for each class must lie on the correct side of the hyperplane for it to be a separating hyperplane! We can combine two constraints into

$$\ell_i(w^\top x_i) \geq 1.$$

Finally, the problem can be formulated as the following QP:

$$\begin{aligned} \min_{w \in \mathbb{R}^3} \quad & \frac{1}{2}\|w\|^2 \\ \text{subject to} \quad & \ell_i(w^\top x_i) \geq 1, \quad i = 1, \ldots, n. \end{aligned}$$

After solving the problem, suppose $w^* = [a^*; a_0^*]$ is the optimal solution. We can predict the class label of a new input (called query point), x_{data}, by simply checking which side of the hyperplane it lies on

$$\text{class label is} \begin{cases} \text{red circles} & a^* \bullet x_{\text{data}} + a_0^* > 0 \\ \text{blue crosses} & a^* \bullet x_{\text{data}} + a_0^* < 0 \end{cases}.$$

■

Appendix B

To Learn on Your Own (if you want to): Cool and Important Things We Omitted From our Linear Algebra Introduction

Learning Objectives

- Introduce material that is commonly included in a second or third year Linear Algebra Course
- Provide a resource for use after you leave ROB 101.

Outcomes

- Complex numbers obey the same rules of arithmetic as the real numbers, if you really understand the real numbers!
- Eigenvalues and eigenvectors of square matrices
- Symmetric matrices have real eigenvalues and admit orthonormal eigenvectors
- Positive definite matrices allow one to generalize the Euclidean norm
- The Singular Value Decomposition (SVD) allows one to quantify the degree to which vectors are linearly independent. This is super useful in engineering practice.
- Matrices are good for other things than representing systems of equations: they also allow one to transform vectors in interesting ways, giving rise to the concept of *linear transformations*.
- Many more facts about basis vectors.

B.1 Complex Numbers and Complex Vectors

Here are some video resources that you may enjoy consulting:

- https://youtu.be/T647CGsuOVU
- https://youtu.be/2HrSG0fdxLY
- https://youtu.be/N9QOLrfcKNc
- https://youtu.be/DThAoT3q2V4
- https://youtu.be/65wYmy8Pf-Y

The story of complex numbers begins with the quadratic equation $x^2+1=0$, which has no real solutions! After much soul searching, the mathematics community finally embraced the notion of an **imaginary quantity** $\mathbb{i}$ defined by

$$(\mathbb{i})^2 := -1. \tag{B.1}$$

More commonly, we write this as

$$\mathbb{i} = \sqrt{-1}. \tag{B.2}$$

The set of **complex numbers** is then defined as

$$\mathbb{C} := \{x + \mathbb{i}\, y \,|x \in \mathbb{R}, y \in \mathbb{R}\}. \tag{B.3}$$

If $z = x + \mathbb{i}\, y \in \mathbb{C}$, then we define

$$\begin{aligned} x &:= \text{real}(z) \;\; \text{the } \textbf{real part} \text{ of } \; z \\ y &:= \text{imag}(z) \;\; \text{the } \textbf{imaginary part} \text{ of } \; z. \end{aligned} \tag{B.4}$$

We note that both x and y are real numbers. Complex numbers of the form $0 + \mathbb{i}\, y$ are called **imaginary numbers**. We view a **real number** $x \in \mathbb{R}$ as being a complex number of the form $x + \mathbb{i}\, 0$. In other words, we view $\mathbb{R} \subset \mathbb{C}$. In addition, we define the **complex conjugate** of $z = x + \mathbb{i}\, y$ to be

$$z^* := x - \mathbb{i}\, y, \tag{B.5}$$

that is, $\text{imag}(z^*) = -\text{imag}(z)$, while $\text{real}(z^*) = \text{real}(z)$.

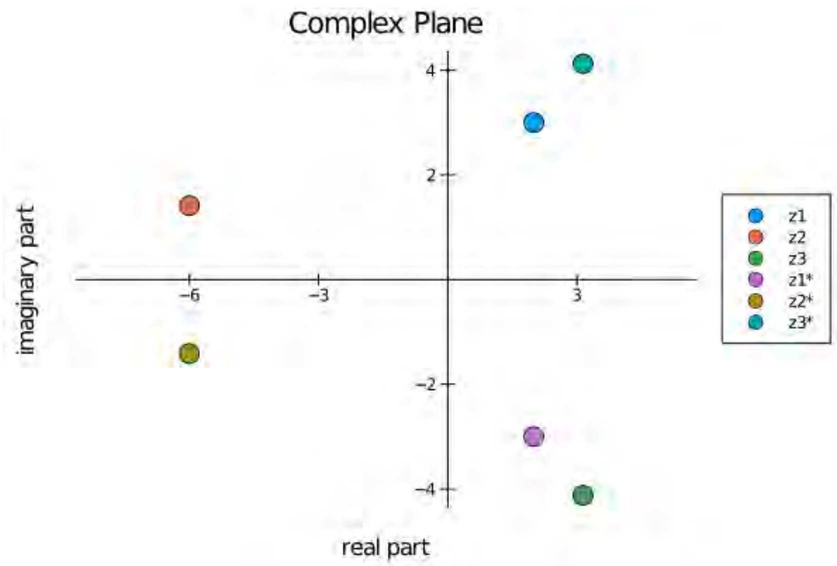


Figure B.1: The complex plane has x-axis given by the real part of a complex number and y-axis given by the imaginary part of a complex number. Here, we plot z_1, z_2, z_3 from Example B.1 and their complex conjugates.

Example B.1 *For the following are complex numbers, compute their real and imaginary parts as well as their complex conjugates. Also, plot them in the complex plane.*

$$\begin{aligned} z_1 &= 2 + \mathbb{i}\, 3 \\ z_2 &= -6 + \mathbb{i}\, \sqrt{2} \\ z_3 &= \pi - \mathbb{i}\, \sqrt{17}. \end{aligned}$$

Solution

$$\begin{array}{lll} \text{real}(z_1) = 2 & \text{imag}(z_1) = 3 & z_1^* = 2 - \mathbb{i}\, 3 \\ \text{real}(z_2) = -6 & \text{imag}(z_2) = \sqrt{2} & z_2^* = -6 - \mathbb{i}\, \sqrt{2} \\ \text{real}(z_3) = 2\pi & \text{imag}(z_3) = -\sqrt{17} & z_2^* = \pi + \mathbb{i}\, \sqrt{17}. \end{array}$$

All of these values are plotted in Fig. B.2.

■

B.1.1 Arithmetic of Complex Numbers: Enough to Get You By

We'll define all of the major arithmetic operations. Just like operations with vectors and matrices, however, it's much more fun to do the calculations in Julia than by hand!

The **addition of two complex numbers** is defined by adding their respective real and imaginary parts,

$$(x_1 + \mathbb{i}\, y_1) + (x_2 + \mathbb{i}\, y_2) := (x_1 + x_2) + \mathbb{i}\, (y_1 + y_2)\,. \tag{B.6}$$

This is very similar to how we add two vectors

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} := \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$

by adding their respective components.

The **multiplication** of two complex numbers is defined by

$$(x_1 + \mathbb{i}\, y_1) \cdot (x_2 + \mathbb{i}\, y_2) := (x_1 x_2 - y_1 y_2) + \mathbb{i}\, (x_1 y_2 + y_1 x_2)\,. \tag{B.7}$$

This formula comes from applying basic algebra to the "symbolic expression"

$$(a_1 + b_1)(a_2 + b_2) = a_1 a_2 + b_1 b_2 + a_1 b_2 + b_1 a_2$$

and then substituting in

$$\begin{aligned} a_1 &:= x_1 \\ a_2 &:= x_2 \\ b_1 &:= \mathbb{i}\, y_1 \\ b_2 &:= \mathbb{i}\, y_2. \end{aligned}$$

The term $b_1 b_2 = (\mathbb{i}\, y_1) \cdot (\mathbb{i}\, y_2) = (\mathbb{i})^2 y_1 y_2 = (-1) y_1 y_2$, which explains how the minus sign appears!

Let's note that if we multiply a complex number by its complex conjugate, then we obtain a real number. Indeed,

$$z \cdot z^* = (x + \mathbb{i}\, y) \cdot (x - \mathbb{i}\, y) = ((x)(x) - (y)(-y)) + \mathbb{i}\, ((x)(-y) + (y)(x)) = x^2 + y^2. \tag{B.8}$$

The **magnitude of a complex number** $z = x + \mathbb{i}\, y$ is denoted by $|z|$ and is defined by

$$|z| := \sqrt{x^2 + y^2}, \tag{B.9}$$

or equivalently, by

$$|z| := \sqrt{z \cdot z^*}. \tag{B.10}$$

Both definitions are common and we note that the square root makes sense[1] because the magnitude is a non-negative real number.

Using the complex conjugate, the **division of one complex number by another** can be defined, and subsequently, understood. We define

$$\frac{x_1 + \mathbb{i}\, y_1}{x_2 + \mathbb{i}\, y_2} := \frac{(x_1 x_2 + y_1 y_2) + \mathbb{i}\ (y_1 x_2 - x_1 y_2)}{(x_2)^2 + (y_2)^2}, \tag{B.11}$$

and note that the denominator is real, and thus the indicated division can be treated as multiplication by one over the denominator. It follows that when $|z_2| \neq 0$, z_1/z_2 is a well-defined complex number. The formula (B.11) is best understood from an alternative definition of complex division

$$\frac{z_1}{z_2} := \frac{z_1 \cdot z_2^*}{z_2 \cdot z_2^*} = \frac{z_1 \cdot z_2^*}{|z_2|^2}. \tag{B.12}$$

Personally, we try to avoid using either one of these formulas and do the computations in Julia! Multiplication and division of complex numbers by hand is very error prone. For probably a century, engineering faculty have been torturing students by making them do such calculations by hand; at some point, it has to stop!

Example B.2 *For the following complex numbers, compute their sum, product, division, and magnitudes,*

$$\begin{aligned} z_1 &= 2 + \mathbb{i}\, 3 \\ z_2 &= -6 + \mathbb{i}\, \sqrt{2} \end{aligned}$$

Solution

■

B.1.2 Angles of Complex Numbers and Euler's Formula: More Advanced Aspects

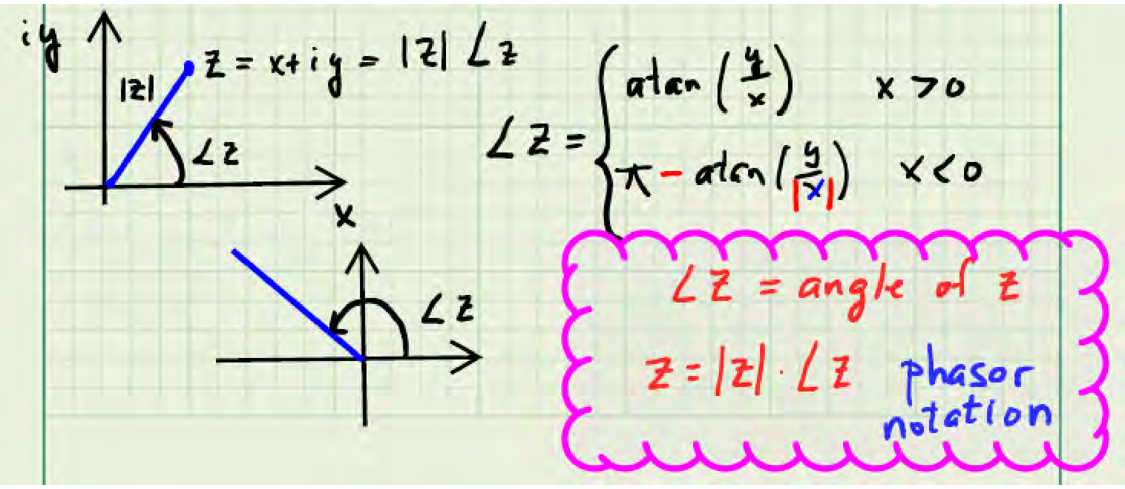


Figure B.2: It is often helpful to understand complex numbers as having a magnitude and an angle. This is similar to using polar coordinates in $\mathbb{R}^2$. Here, the angle of z is denoted by $\angle z$ instead of θ. Both conventions are common.

For a pair of real numbers (x, y), we were taught in High School how to express them in **polar coordinates** (ρ, θ), where

$$\begin{aligned} \rho &:= \sqrt{x^2 + y^2} \\ \theta &:= \begin{cases} \arctan(y/x) & x > 0 \\ \pi - \arctan(y/|x|) & x < 0 \\ \text{sign}(y)\ \frac{\pi}{2} & x = 0, y \neq 0 \\ \text{undefined} & x = 0, y = 0. \end{cases} \end{aligned} \tag{B.13}$$

[1] Julia will recognize $\text{real}(z)^2 + \text{imag}(z)^2$ as being a real number. It does not recognize $z \cdot z^*$ as a real number. In Julia, the command is $\text{abs}(z)$, just as with a real number.

From polar coordinates (ρ, θ), we computed the **Cartesian Coordinates** (x, y) as

$$\begin{aligned} x &= \rho \cos(\theta) \\ y &= \rho \sin(\theta). \end{aligned} \tag{B.14}$$

Moving beyond High School, we can also express the above in terms of the canonical basis vectors $\{e_1, e_2\}$ for $\mathbb{R}^2$ as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \rho \cos(\theta) e_1 + \rho \sin(\theta) e_2. \tag{B.15}$$

In fact, any (non-zero) vector $v \in \mathbb{R}^2$ can be expressed as

$$v = ||v|| \cos(\theta) e_1 + ||v|| \sin(\theta) e_2. \tag{B.16}$$

Equation (B.16) hints at a natural way of expressing complex numbers.

Polar Coordinates Meet Complex Numbers and their Multiplication

For a non-zero complex number $z = x + \mathbb{i}\, y$, we define its angle as in (B.13). Doing so allows us to express every (non-zero) $z \in \mathbb{C}$ as

$$z = |z| \cos(\theta) + \mathbb{i}\, |z| \sin(\theta). \tag{B.17}$$

The real and imaginary parts of z are playing the role of the basis vectors $\{e_1, e_2\}$. One can also think of $\{1.0, \mathbb{i}\}$ as being a basis for C, though this is beyond our scope. Equation (B.17) leads to a very nice way to understand the multiplication of two complex numbers.

Fact: Suppose $z_1 = |z_1| \cos(\theta_1) + \mathbb{i}\, |z_1| \sin(\theta_1)$ and $z_2 = |z_2| \cos(\theta_2) + \mathbb{i}\, |z_2| \sin(\theta_2)$ are non-zero complex numbers. Then,

$$\boxed{\begin{aligned} z_1 \cdot z_2 &= |z_1||z_2| \cos(\theta_1 + \theta_2) + \mathbb{i}\, |z_1||z_2| \sin(\theta_1 + \theta_2) \\ \frac{z_1}{z_2} &= \frac{|z_1|}{|z_2|} \cos(\theta_1 - \theta_2) + \mathbb{i}\, \frac{|z_1|}{|z_2|} \sin(\theta_1 - \theta_2). \end{aligned}} \tag{B.18}$$

When expressed in "polar form", multiplying two complex numbers is equivalent to multiplying their magnitudes and adding their angles (or phases), while dividing two complex numbers is equivalent to dividing their magnitudes and subtracting their angles (or phases). Proving (B.18) involves some trigonometric identities. You may want to give it a go. We'll provide a simpler way to understand it in a few more lines!

Remark: A positive real number has angle zero, while a negative real number has angle π (or, equivalently, $-\pi$). The angle of $\mathbb{i}$ is $\pi/2$ and the angle of $-\mathbb{i}$ is $-\pi/2$ (or, equivalently, $3\pi/2$).

The exponential function of a real number x is defined by the infinite series

$$\begin{aligned} e^x &:= \sum_{n=0}^{\infty} \frac{x^n}{k!} \\ &= 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \cdots \\ &= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \cdots, \end{aligned} \tag{B.19}$$

where $n! := 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$, the product of all integers from 1 to n.

Euler's Formula

A very famous result due to the Swiss Mathematician Leonhard Euler (https://en.wikipedia.org/wiki/Leonhard_Euler), asserts that for a given real number θ,

$$e^{\mathbb{i}\,\theta} = \cos(\theta) + \mathbb{i}\,\sin(\theta) \quad \textbf{Euler's Formula}. \tag{B.20}$$

Hence, every complex number can be written as

$$z = |z|e^{\mathbb{i}\,\theta}, \tag{B.21}$$

which leads to

Fact: Suppose $z_1 = |z_1|e^{\mathbb{i}\,\theta_1}$ and $z_2 = |z_2|e^{\mathbb{i}\,\theta_2}$ are non-zero complex numbers. Then,

$$\boxed{\begin{aligned} z_1 \cdot z_2 &= |z_1||z_2|e^{\mathbb{i}\,(\theta_1+\theta_2)} \\ \\ \frac{z_1}{z_2} &= \frac{|z_1|}{|z_2|}e^{\mathbb{i}\,(\theta_1-\theta_2)}. \end{aligned}} \tag{B.22}$$

Deriving this result for multiplication and division is much easier than (B.18), but Euler's Formula (B.20) assures us they are the same.

Remark: Deriving Euler's formula is not that hard. When you substitute $\mathbb{i}\,\theta$ into (B.19), you must first note that $(\mathbb{i})^{2n} = (-1)^n$ and $(\mathbb{i})^{2n+1} = (-1)^n\,\mathbb{i}$. If you then separate the power series into its real and imaginary parts, you will recognize the power series for $\cos(\theta)$ and $\sin(\theta)$.

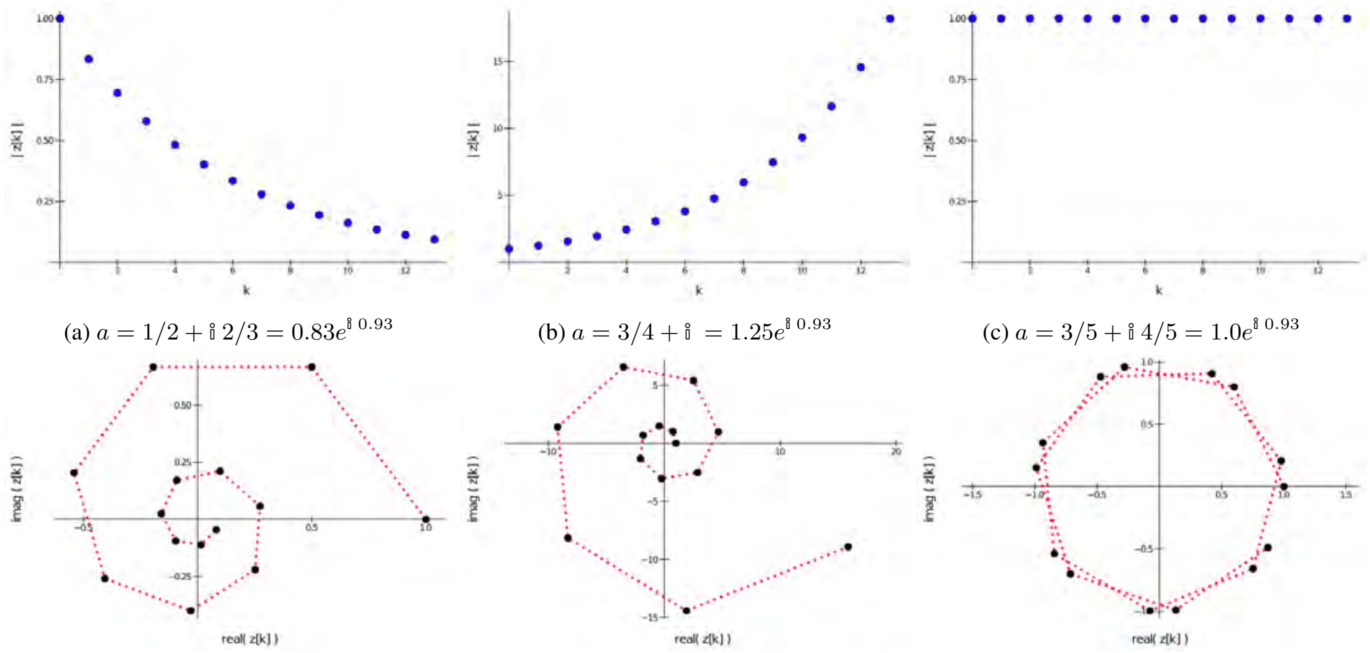
B.1.3 Iterating with Complex Numbers: Background for Eigenvalues

Consider the equation

$$z_{k+1} = az_k, \tag{B.23}$$

with $a \in \mathbb{C}$ and $z_0 \in \mathbb{C}$. Equation (B.23) is technically called a **scalar linear difference equation**, but for us, it looks not so different than iterating with the bisection method or Newton's Algorithm. We compute a few steps until the general pattern of its solution becomes clear:

$$\begin{aligned} z_1 &= az_0 \\ z_2 &= az_1 = a^2 z_0 \\ z_3 &= az_2 = a^3 z_0 \\ &\vdots \\ z_k &= a^k z_0. \end{aligned} \tag{B.24}$$



(a) $a = 1/2 + \hat{\imath}\, 2/3 = 0.83e^{\hat{\imath}\, 0.93}$ (b) $a = 3/4 + \hat{\imath} \; = 1.25e^{\hat{\imath}\, 0.93}$ (c) $a = 3/5 + \hat{\imath}\, 4/5 = 1.0e^{\hat{\imath}\, 0.93}$

(d) $a = 3/5 + \hat{\imath}\, 4/5 = 1.0\angle 53.1$ degrees (e) $a = 3/4 + \hat{\imath} \; = 1.25\angle 53.1$ degrees (f) $a = 3/5 + \hat{\imath}\, 4/5 = 1.0\angle 53.1$ degrees

Figure B.3: The solid dots illustrate the evolution of z_k in (B.23) and (B.24) when a has magnitude less than one, greater than one, and equal to one, respectively. In each case, $z_0 = 1.0 + \hat{\imath}\, 0.0$ and the angle of a was selected to be 53.1 degrees; therefore the dots rotate counterclockwise. The red dashes are present to guide the eye in connecting the dots. In (f), the points lie on a circle of radius one.

Scalar Linear Difference Equation

The general solution to $z_{k+1} = az_k$, $z_0 \in \mathbb{C}$ is $z_k = a^k z_0$. We write $a = |a|e^{\hat{\imath}\, \theta}$, where $\theta = \angle a$, the angle of a as computed in (B.13). Then from (B.22), we conclude that

$$z_k = |a|^k e^{\hat{\imath}\, k\angle a} z_0. \tag{B.25}$$

Below, we analyze three cases and show the following for $z_0 \neq 0$

- $|a| < 1 \implies |z_k| \underset{k\to\infty}{\longrightarrow} 0$
- $|a| > 1 \implies |z_k| \underset{k\to\infty}{\longrightarrow} \infty$
- $|a| = 1 \implies |z_k| = |z_0|, k \geq 0.$

See also Fig. B.3.

The following analysis supports the illustrations in Fig. B.3.

Case 1: |a|<1 We note that $\log(|a|^k) = k\log(|a|)$, and that $|a| < 1 \implies \log(|a|) < 0$. Hence,

$$\lim_{k\to\infty} |a|^k = \lim_{k\to\infty} e^{k\log(|a|)} = 0.$$

Case 2: |a|>1 We note that $\log(|a|^k) = k\log(|a|)$, and that $|a| > 1 \implies \log(|a|) > 0$. Hence,

$$\lim_{k\to\infty} |a|^k = \lim_{k\to\infty} e^{k\log(|a|)} = \infty.$$

Case 3: |a|=1 We note that when $|a| = 1$, then $|a|^k = 1$ for all $k \geq 1$, and thus this case is clear.

B.1.4 $\mathbb{C}^n$, the Space of Complex Vectors

$\mathbb{C}^n$ sounds harder than it is. It's exactly $\mathbb{R}^n$ where the scalars are complex numbers instead of real numbers. All the definitions of vector addition, linear combinations, spans, and linear independence are the same. We'll cover just a few of the basic ideas so that you get the idea.

Recall that we started by defining $\mathbb{R}^n$ as n-tuples of real numbers and then we identified it with column vectors of length n. We do that same here.

$$\boxed{\mathbb{C}^n := \{(\alpha_1, \alpha_2, \dots, \alpha_n) \mid \alpha_i \in \mathbb{C}, 1 \le i \le n\} \iff \left\{ \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} \middle| \; \alpha_i \in \mathbb{C}, 1 \le i \le n \right\} =: \mathbb{C}^n} \tag{B.26}$$

Consider two vectors $v_1 \in \mathbb{C}^n$ and $v_2 \in \mathbb{C}^n$. We define their **vector sum** by

$$v_1 + v_2 = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} := \begin{bmatrix} \alpha_1 + \beta_1 \\ \alpha_2 + \beta_2 \\ \vdots \\ \alpha_n + \beta_n \end{bmatrix},$$

that is, we sum their respective components or entries. Let γ be a complex number. Then we define

$$\gamma v := \begin{bmatrix} \gamma\alpha_1 \\ \gamma\alpha_2 \\ \vdots \\ \gamma\alpha_n \end{bmatrix},$$

that is, to **multiply a complex vector by a complex number**, we multiply each of the components of the vector by the number, just as we do for real vectors.

Let $\{v_1, v_2, \dots, v_k\}$ be a collection of vectors in $\mathbb{C}^n$. Then we define their **span** as

$$\text{span}\{v_1, v_2, \dots, v_k\} := \{\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k \mid \alpha_i \in \mathbb{C}, 1 \le i \le k\}.$$

The set of vectors $\{v_1, v_2, \dots, v_k\}$ is **linearly independent** in the vector space $\mathbb{C}^n$ if the only solution to

$$\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k = 0$$

is $\alpha_1 = 0 + \mathbb{i}\,0, \alpha_2 = 0 + \mathbb{i}\,0, \dots, \alpha_k = 0 + \mathbb{i}\,0$.

The **norm** of a complex vector

$$v = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

is

$$||v|| := \sqrt{\sum_{k=1}^{n} |\alpha_k|^2},$$

where, to be extra clear, $|\alpha_k|^2 = \alpha_k \cdot \alpha_k^*$. Moreover, if one defines the **complex conjugate of a vector** by taking the complex conjugate of each of its components, then

$$||v||^2 = (v^*)^\top \cdot v,$$

and yes, the transpose simply takes the column vector to a row vector.

B.1.5 Iterating with Matrices: The Case for Eigenvalues and Eigenvectors

We now attempt to analyze the matrix versions of (B.23) and (B.24). Recall that you saw matrix difference equations in Project 3. Our real goal is to understand

$$x_{k+1} = Ax_k, \tag{B.27}$$

with A an $n \times n$ real matrix and $x_0 \in \mathbb{R}^n$. But we'll see that allowing the entries of A to be complex and $x_0 \in \mathbb{C}^n$ does not change anything.

With this in mind, we rewrite (B.28) first as

$$x[k+1] = Ax[k],$$

with the time index denoted in square brackets, Julia style! This will allow us to use a subscript for the components of x. Next, we replace $x[k]$ with $z[k]$ to emphasize that the we allow $z[k]$ to be a complex vector. We compute a few steps of $z[k+1] = Az[k]$ until the general pattern of its solution becomes clear:

$$\begin{aligned} z[1] &= Az[0] \\ z[2] &= Az[1] = A^2 z[0] \\ z[3] &= Az[2] = A^3 z[0] \\ &\vdots \\ z[k] &= A^k z[0]. \end{aligned} \tag{B.28}$$

So far so good! Now, our challenges are:

- give conditions on A so that $||\ z[k]\ ||$ contracts, blows up, or stays bounded as k tends to infinity;
- even better, for a given initial condition $z[0]$, describe in detail the evolution of $z[k]$ for $k > 0$.

We'll start with a diagonal $n \times n$ matrix A, and for reasons that will become clear in the next section, we'll denote the entries on the diagonal by λ,

$$A = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_n \end{bmatrix}. \tag{B.29}$$

We leave it as an exercise to compute that

$$A^2 = \begin{bmatrix} (\lambda_1)^2 & 0 & 0 & 0 \\ 0 & (\lambda_2)^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & (\lambda_n)^2 \end{bmatrix}, \tag{B.30}$$

and once you have established (B.30), you will have no trouble believing that

$$A^k = \begin{bmatrix} (\lambda_1)^k & 0 & 0 & 0 \\ 0 & (\lambda_2)^k & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & (\lambda_n)^k \end{bmatrix}. \tag{B.31}$$

One thing we can do is observe that

$$\begin{bmatrix} z_1[k] \\ z_2[k] \\ \vdots \\ z_n[k] \end{bmatrix} = \begin{bmatrix} (\lambda_1)^k & 0 & 0 & 0 \\ 0 & (\lambda_2)^k & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & (\lambda_n)^k \end{bmatrix} \begin{bmatrix} z_1[0] \\ z_2[0] \\ \vdots \\ z_n[0] \end{bmatrix} \tag{B.32}$$

results in n scalar equations of the form (B.23), namely,

$$\boxed{z_j[k] = (\lambda_j)^k z_j[0],\ \ 1 \le j \le n.} \tag{B.33}$$

Linear Difference Equation with a Diagonal Matrix

The general solution to $z[k+1] = Az[k]$, $z[0] \in \mathbb{C}^n$ is $z[k] = A^k z[0]$. When A is diagonal, the solution is given in (B.32) and (B.33). Based on these results and Chapter B.1.3, we analyze three cases for $z_j[0] \neq 0$,

- $|\lambda_j| < 1 \implies |z_j[k]| \underset{k\to\infty}{\longrightarrow} 0$
- $|\lambda_j| > 1 \implies |z_j[k]| \underset{k\to\infty}{\longrightarrow} \infty$
- $|\lambda_j| = 1 \implies |z_j[k]| = |z_j[0]|, k \ge 0.$

Being a ROB 101 student, having "real" matrices replaced by diagonal matrices must be a bit disconcerting! You'll be glad to know that it is really just a step to something kind of magical: most matrices can be factored as $A = M\Lambda M^{-1}$, where $\det(M) \neq 0$ and Λ is diagonal, as in (B.29). But we get ahead of ourselves!

Key features of a Diagonal Matrix: One way that Eigenvalues and Eigenvectors come about

Let $v_j = e_j$, where e_j are the canonical basis vectors for either $\mathbb{R}^n$ or $\mathbb{C}^n$ (aka, columns of the $n \times n$ identity matrix). We have noted before that $Ae_j = a_j^{\text{col}}$. In our case, $a_j^{\text{col}} = \lambda_j e_j$. Hence, substituting in $v_j = e_j$, we arrive at the equation

$$\boxed{Av_j = \lambda_j v_j,\ \ 1 \le j \le n.} \tag{B.34}$$

Equation (B.34) is the defining relation for eigenvalues (denoted here by λ_j) and eigenvectors (denoted here by v_j). We further note that the set of vectors $\{v_1, v_2, \dots, v_n\}$ is linearly independent and spans both $\mathbb{R}^n$ and $\mathbb{C}^n$. Having a set of eigenvectors that forms a basis turns out to be a defining characteristic of matrices that are related to a diagonal matrix Λ by a transformation of the form $A = M\Lambda M^{-1}$.

Remark: If $v \in \mathbb{C}^n$ is an eigenvector, meaning $v \neq 0$ and there exists a $\lambda \in \mathbb{C}$ such that (B.34) holds, then we have that

$$\begin{aligned} Av &= \lambda v \\ A^2 v &= A(\lambda v) = \lambda A v = (\lambda)^2 v \\ &\ \vdots \\ A^k v &= (\lambda)^k v \end{aligned}$$

and hence we can analyze convergence for the difference equation $z[k+1] = Az[k]$, $z[0] = v$, even when A is not diagonal.

Remark: Suppose that A is real and that $\lambda \in \mathbb{C}$ and $v \in \mathbb{C}^n$, satisfy $Av = \lambda v$ and $v \neq 0$. Even though the eigenvalue and eigenvector are complex, their real and imaginary parts are very relevant to computations in $\mathbb{R}^n$. Decompose λ and v into their real and imaginary parts, viz

$$\begin{aligned} \lambda &=: \lambda_{\text{Re}} + \mathbb{i}\, \lambda_{\text{Im}} \\ v &=: v_{\text{Re}} + \mathbb{i}\, v_{\text{Im}}. \end{aligned} \tag{B.35}$$

Then

$$\begin{aligned} Av_{\text{Re}} &= \text{real}(Av) = \lambda_{\text{Re}} \cdot v_{\text{Re}} - \lambda_{\text{Im}} \cdot v_{\text{Im}} \\ Av_{\text{Im}} &= \text{imag}(Av) = \lambda_{\text{Im}} \cdot v_{\text{Re}} + \lambda_{\text{Re}} \cdot v_{\text{Im}}. \end{aligned} \tag{B.36}$$

Hence,

$$A \begin{bmatrix} v_{\text{Re}} \\ v_{\text{Im}} \end{bmatrix} = \begin{bmatrix} \lambda_{\text{Re}} I_n & -\lambda_{\text{Im}} I_n \\ \lambda_{\text{Im}} I_n & \lambda_{\text{Re}} I_n \end{bmatrix} \begin{bmatrix} v_{\text{Re}} \\ v_{\text{Im}} \end{bmatrix}. \tag{B.37}$$

If we write $\lambda = |\lambda| e^{\mathbb{i}\,\theta} = |\lambda|\cos(\theta) + \mathbb{i}\,|\lambda|\sin(\theta)$, then (B.37) can be rewritten as

$$A \begin{bmatrix} v_{\rm Re} \\ v_{\rm Im} \end{bmatrix} = |\lambda| \underbrace{\begin{bmatrix} \cos(\theta) I_n & -\sin(\theta) I_n \\ \sin(\theta) I_n & \cos(\theta) I_n \end{bmatrix}}_{R(\theta)} \begin{bmatrix} v_{\rm Re} \\ v_{\rm Im} \end{bmatrix}, \tag{B.38}$$

where $R(\theta)^\top \cdot R(\theta) = R(\theta) \cdot R(\theta)^\top = I_{2n}$, and hence $R(\theta)$ is an orthogonal matrix. This shows how the complex aspect of the eigenvalue and eigenvector manifests itself as a "kind of rotation" of vectors in the two dimensional subspace

$$\text{span}\{v_{\rm Re}, v_{\rm Im}\}$$

by $R(\theta)$, in addition to the scaling by $|\lambda|$. A second benefit of the latter expression is that we then have

$$A^k \begin{bmatrix} v_{\rm Re} \\ v_{\rm Im} \end{bmatrix} = |\lambda|^k \underbrace{\begin{bmatrix} \cos(k\theta) I_n & -\sin(k\theta) I_n \\ \sin(k\theta) I_n & \cos(k\theta) I_n \end{bmatrix}}_{R(k\theta)} \begin{bmatrix} v_{\rm Re} \\ v_{\rm Im} \end{bmatrix}. \tag{B.39}$$

Figure B.4 illustrates a case where $\lambda = 0.9803 \pm \mathbb{i}\,0.0965 = 0.985\angle 5.6$ degrees. The rotating and decaying nature of the solution is clearly seen in the figure. The reader should compare Figs. B.3-(a) and -(d) with Fig. B.4.

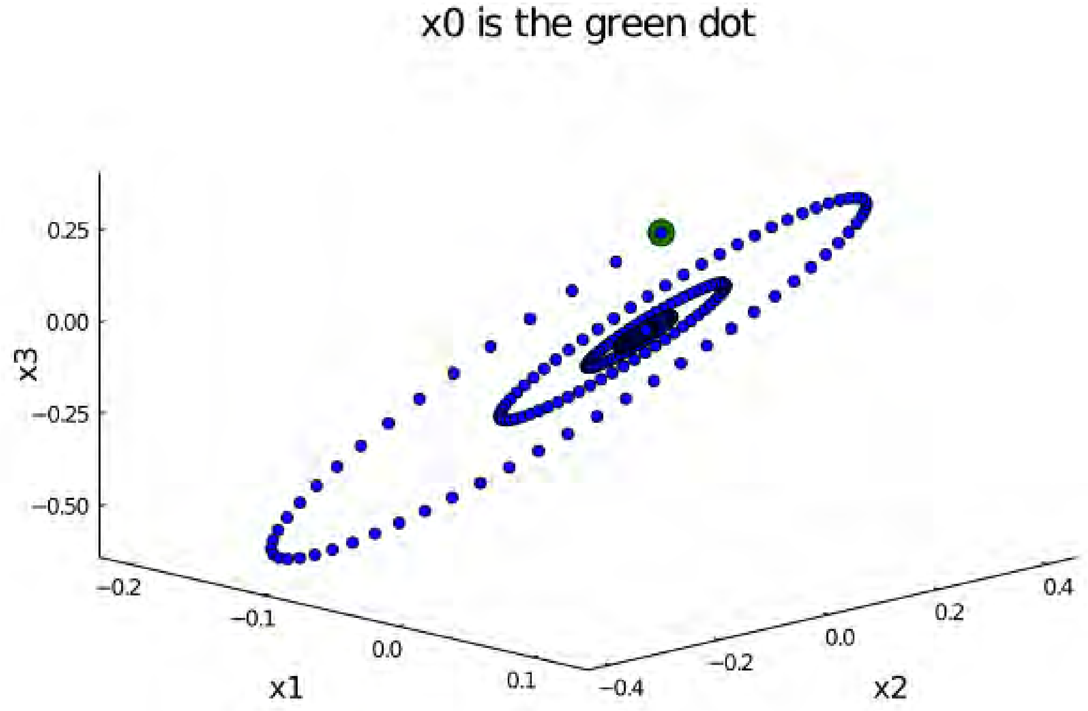


Figure B.4: The eigenvalues of a real 3×3 matrix are computed to be $0.9803 \pm \mathbb{i}\,0.0965$ and 1.100. The initial condition in green was chosen to be a linear combination of the real and imaginary parts of an eigenvector corresponding to the complex pair of eigenvalues. The resulting solution of (B.28) evolves in the plane defined by $\text{span}\{v_{\rm Re}, v_{\rm Im}\}$ as indicated by (B.37) through (B.39). This is very analogous to how complex numbers, when iterated, evolve in the Complex Plane.

B.2 Eigenvalues and Eigenvectors

The study of eigenvalues and eigenvectors is very traditional in Linear Algebra courses. We skipped them in the main portion of the book for a few reasons: (1) time is limited; (2) they get complicated really fast; and (3) their most important applications are the evolution of linear difference equations and the Singular Value Decomposition (SVD), neither of which were covered in the main portion of the text. The usual illustrative application of eigenvalues and eigenvectors is to "diagonalize" a matrix, and quite frankly, that on its own is not a compelling application. In the context of Chapter B.1.5 and your Segway Project, it does make sense.

Just in case you are starting here and skipped Chapter B.1 entirely, we start from the beginning.

B.2.1 General Square Matrices

Temporary Def. Let A be an $n \times n$ matrix with real coefficients. A scalar $\lambda \in \mathbb{R}$ is an **eigenvalue** (e-value) of A, if there exists a non-zero vector $v \in \mathbb{R}^n$ such that $A \cdot v = \lambda v$. Any such vector v is called an **eigenvector** (e-vector) associated with λ.

We note that if v is an e-vector, then so is αv for any $\alpha \neq 0$, and therefore, e-vectors are not unique. To find eigenvalues, we need to have conditions under which there exists $v \in \mathbb{R}^n$, $v \neq 0$, such that $A \cdot v = \lambda v$. Here they are,

$$A \cdot v = \lambda v \iff (\lambda I - A) \cdot v = 0 \overset{v \neq 0}{\iff} \det(\lambda I - A) = 0.$$

Example B.3 *Let A be the 2×2 real matrix $A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$. Determine, if any, its e-values and e-vectors.*

Solution: To find e-values, we need to solve

$$\det(\lambda I - A) = \begin{vmatrix} \lambda & -1 \\ 1 & \lambda \end{vmatrix} = \lambda^2 + 1 = 0.$$

We compute the discriminant of this quadratic equation and we find

$$b^2 - 4ac = -4 < 0,$$

and therefore there are no real solutions. Hence, by our *temporary definition*, this 2×2 real matrix does not have any e-values, and hence, neither does it have any e-vectors.

If we were to allow e-values to be complex numbers, then we'd have two e-values corresponding to the two complex solutions of the quadratic equation $\lambda^2 + 1 = 0$, namely, $\lambda_1 = \mathbb{i}$ and $\lambda_2 = -\mathbb{i}$.

We'll see shortly that we'll also need to allow the e-vectors to have complex entries. Hence, we need to generalize our temporary definition. ■

Permanent Definition of Eigenvalues and Eigenvectors

Let A be an $n \times n$ matrix with real or complex coefficients. A scalar $\lambda \in \mathbb{C}$ is an **eigenvalue** (e-value) of A, if there exists a non-zero vector $v \in \mathbb{C}^n$ such that $Av = \lambda v$. Any such vector v is called an **eigenvector** (e-vector) associated with λ.

Eigenvectors are not unique.

- To find e-values, we solve $\det(\lambda I - A) = 0$ because

$$A \cdot v = \lambda v \iff (\lambda I - A) \cdot v = 0 \overset{v \neq 0}{\iff} \det(\lambda I - A) = 0. \tag{B.40}$$

- To find e-vectors, we find any non-zero $v \in \mathbb{C}^n$ such that

$$(\lambda I - A) \cdot v = 0. \tag{B.41}$$

Of course, if you prefer, you can solve $(A - \lambda I)v = 0$ when seeking e-vectors.

Fundamental Theorem of Algebra (and a bit More)

Let A be an $n \times n$ matrix with real or complex coefficients. Then the following statements are true

- $\det(\lambda I - A) = \lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots \alpha_1 \lambda + \alpha_0$, and if A is real, so are the coefficients $\alpha_{n-1}, \ldots, \alpha_0$.
- The degree n polynomial $\lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots \alpha_1 \lambda + \alpha_0$ has n roots $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$ such that

  $$\det(\lambda I - A) = (\lambda - \lambda_1)(\lambda - \lambda_2) \cdots (\lambda - \lambda_n).$$

  Each of the roots λ_i, $1 \leq i \leq n$, is an e-value of A.
- The e-values $\{\lambda_1, \ldots, \lambda_n\}$ are said to be **distinct** if $\lambda_i \neq \lambda_k$ for all $i \neq k$.
- If $\lambda_i = \lambda_k$ for some $i \neq k$, then λ_i is a **repeated e-value**. The e-values can then be grouped into $1 \leq p \leq n$ sets of distinct roots $\{\lambda_1, \ldots, \lambda_p\}$ such that

  $$\det(\lambda I - A) = (\lambda - \lambda_1)^{m_1}(\lambda - \lambda_2)^{m_2} \cdots (\lambda - \lambda_p)^{m_p}.$$

  The integer m_i is called the **algebraic multiplicity** of λ_i and their sum satisfies $m_1 + m_2 + \cdots + m_p = n$.
- An e-vector associated with λ_i is computed by finding non-zero solutions to (B.41).
- If the matrix A is real, then the e-values occur in **complex conjugate pairs**, that is, if λ_i is an e-value then so is λ_i^*.
- If the matrix A is real and the e-value λ_i is real, then the e-vector v_i can always be chosen to be real, that is, $v_i \in \mathbb{R}^n$ instead of $v_i \in \mathbb{C}^n$.
- There will always be at least one non-zero solution to (B.41), and because any non-zero multiple of a solution is also a solution, there will always be an infinite number of solutions to (B.41).
- If λ_i is a repeated e-value with algebraic multiplicity m_i, then the number of linearly independent e-vectors associated with λ_i is upper bounded by m_i. Another way to say this is, $1 \leq \dim\left(\mathrm{Null}(A - \lambda_i I)\right) \leq m_i$.
- **In Julia**, after `using LinearAlgebra`, the commands are $\Lambda =$ `eigvals(A)` and $V =$ `eigvecs(A)`

Example B.4 *Let A be the 2×2 real matrix that we treated in Example B.3, namely, $A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$. Determine its e-values and e-vectors in the sense of our "permanent" definition.*

Solution: As in Example B.3, to find e-values, we solve

$$\det(\lambda I - A) = \begin{vmatrix} \lambda & -1 \\ 1 & \lambda \end{vmatrix} = \lambda^2 + 1 = 0.$$

We apply the quadratic equation and determine $\lambda_1 = \mathbb{i}$ and $\lambda_2 = -\mathbb{i}$. To find the eigenvectors, we solve

$$(A - \lambda_i I)v_i = 0.$$

The eigenvectors are

$$v_1 = \begin{bmatrix} 1 \\ \mathbb{i} \end{bmatrix}, v_2 = \begin{bmatrix} 1 \\ -\mathbb{i} \end{bmatrix}.$$

Note that the eigenvalues and eigenvectors each form complex conjugate pairs. Indeed,

$$\lambda_2 = \lambda_1^* \text{ and } v_2 = v_1^*.$$

■

Example B.5 *Let A be the $n \times n$ identity matrix. Determine its e-values and e-vectors.*

Solution: $\det(\lambda I - I) = \det((\lambda - 1)I_=0 \iff \lambda = 1$. Alternatively, you can compute that $\det(\lambda I - I) = (\lambda - 1)^n$. Hence, the e-value $\lambda = 1$ is repeated n times, that is, $m_1 = n$. What are the e-vectors? We seek to solve

$$(A - \lambda I) \cdot v = 0 \iff (I - 1 \cdot I) \cdot v = 0 \iff 0_n \cdot v = 0,$$

where 0_n is the $n \times n$ matrix of all zeros! Hence, any non-zero vector $v \in \mathbb{R}^n$ is an e-vector. Moreover, if $\{v_1, \dots, v_n\}$ is a basis for $\mathbb{R}^n$, then $\{v_1, \dots, v_n\}$ is a set of n linearly independent e-vectors associated with $\lambda_1 = 1$. ■

Example B.6 *Let $a \in \mathbb{R}$ be a constant and let A be the 4×4 matrix below. Determine its e-values and e-vectors.*

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

Solution: To find the e-values, we solve

$$\det(\lambda I - A) = \det\left(\begin{bmatrix} (\lambda - a) & -1 & 0 & 0 \\ 0 & (\lambda - a) & -1 & 0 \\ 0 & 0 & (\lambda - a) & -1 \\ 0 & 0 & 0 & (\lambda - a) \end{bmatrix}\right) = (\lambda - a)^4 = 0,$$

and hence there is one distinct e-value $\lambda_1 = a$. To solve for e-vector(s) we consider

$$0 = (A - aI) \cdot v = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot v$$

and we find that the only solutions are multiples of

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

■

We've seen the extremes! A matrix with a single distinct e-value and a complete set of e-vectors (there were n linearly independent e-vectors associated with the e-value), and another matrix with a single distinct e-value, but only one linearly independent e-vector associated with it.

When the e-values are Distinct, the e-vectors form a Basis

Let A be an $n \times n$ matrix with coefficients in $\mathbb{R}$ or $\mathbb{C}$. If the e-values $\{\lambda_1, \dots, \lambda_n\}$ are distinct, that is, $\lambda_i \neq \lambda_j$ for all $1 \leq i \neq j \leq n$, then the e-vectors $\{v_1, \dots, v_n\}$ are linearly independent in $(\mathbb{C}^n, \mathbb{C})$.

Restatement of the result: If $\{\lambda_1, \dots, \lambda_n\}$ are distinct, then $\{v_1, \dots, v_n\}$ is a basis for $(\mathbb{C}^n, \mathbb{C})$. If A is real and its e-values are real, then the e-vectors can be chosen to be real and they form a basis for $\mathbb{R}^n$.

B.2.2 Real Symmetric Matrices

We recall that a real $n \times n$ matrix A is **symmetric** if $A^\top = A$. E-values and e-vectors of symmetric matrices have nicer properties than those of general matrices.

E-values and E-vectors of Symmetric Matrices

- The e-values of a symmetric matrix are real. Because the e-values are real and the matrix is real, we can always chose the e-vectors to be real. Moreover, we can always normalize the e-vectors to have **norm one**.
- Just as with general matrices, the e-values of a symmetric matrix may be distinct or repeated. However, even when an e-value λ_i is repeated m_i times, there are always m_i linearly independent e-vectors associated with it. By applying Gram-Schmidt, we can always chose these e-vectors to be **orthonormal**.
- E-vectors associated with distinct e-values are automatically orthogonal. To be clear,

  $$\left(A^\top = A,\ Av_i = \lambda_i v_i, Av_k = \lambda_k v_k, \text{ and } \lambda_i \neq \lambda_k\right) \implies v_i \perp v_k.$$

  Since we can assume they have length one, we have that the e-vectors are **orthonormal**.
- **In summary**, when A is symmetric, there is always an orthonormal basis $\{v_1, v_2, \ldots, v_n\}$ for $\mathbb{R}^n$ consisting of e-vectors of A. In other words, for all $1 \leq i \leq n$, $Av_i = \lambda_i v_i$, $||v_i|| = 1$, and for $k \neq i$, $v_k \perp v_i$.

Factoring a Symmetric Matrix

For every real $n \times n$ symmetric matrix A, there exists an $n \times n$ diagonal matrix Λ and an $n \times n$ orthogonal matrix Q such that

$$A = Q \cdot \Lambda \cdot Q^\top. \tag{B.42}$$

Moreover,

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \text{ and } Q = \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix}$$

are constructed from the e-values of A and a corresponding set of orthonormal e-vectors.

Remark 01: From (B.42), $\det(A) = \det(Q) \cdot \det(\Lambda) \cdot \det(Q^\top) = \det(\Lambda) = \lambda_1 \cdot \lambda_2 \cdots \lambda_n$. Hence, a symmetric real matrix A is invertible if, and only if, all of its e-values are non-zero. Moreover, in this case

$$A^{-1} = Q \cdot \Lambda^{-1} \cdot Q^\top.$$

While a similar result holds for general square matrices, it requires inverting the matrix formed by stacking the e-vectors as columns, and hence is not numerically attractive. For symmetric matrices, the corresponding inverse is computed via a matrix transpose.

Remark 02: Using the fact that matrix multiplication can be realized by summing over the product of columns times rows, (B.42) can be rewritten as

$$A = \sum_{i=1}^{n} \lambda_i \left(v_i \cdot v_i^\top\right). \tag{B.43}$$

Equations (B.42) and (B.43) parallel results we will develop for the Singular Value Decomposition or (SVD). Equation (B.42) factors A into a product of three terms consisting of two orthogonal matrices and a diagonal matrix, while (B.43) is an expansion of A into "rank one" matrices.

B.3 Positive Definite Matrices

Some Definitions and Facts

Def. Let P be an $n \times n$ real matrix and $x \in \mathbb{R}^n$. Then $x^\top P x$ is called a **quadratic form**.

Def. An $n \times n$ matrix S is **skew symmetric** if $S^\top = -S$.

Fact If S is skew symmetric, then $x^\top S x = 0$ for all $x \in \mathbb{R}^n$.

Fact Let P an $n \times n$ real matrix and write

$$P = \frac{1}{2}\left(P + P^\top\right) + \frac{1}{2}\left(P - P^\top\right).$$

Then $\left(P + P^\top\right)$ is symmetric, $\left(P - P^\top\right)$ is skew symmetric, and we see that every (real) square matrix can be written as the sum of a symmetric matrix and a skew symmetric matrix.

Fact Let P an $n \times n$ real matrix. Then, for all $x \in \mathbb{R}^n$

$$x^\top P x = \frac{1}{2} x^\top \left(P + P^\top\right) x.$$

Hence, a quadratic form only depends on the symmetric part of a matrix.

Consequence: When working with a quadratic form, $x^\top P x$, one **ALWAYS** assumes that the matrix P is symmetric. Allowing the matrix to be non-symmetric does not increase the generality of the notion of a quadratic form. This is because $x^\top P x = \frac{1}{2} x^\top \left(P + P^\top\right) x$ implies that one can always replace P with its symmetric part!

Fact For an $n \times n$ symmetric real matrix P with e-values $\lambda_1, \ldots, \lambda_n$, let $\lambda_{\max} := \max_{1 \leq i \leq n} \lambda_i$ and $\lambda_{\min} := \min_{1 \leq i \leq n} \lambda_i$ be the max and min, respectively over the e-values. Then, for all $x \in \mathbb{R}^n$,

$$\lambda_{\min}\, x^\top x \leq x^\top P x \leq \lambda_{\max}\, x^\top x. \tag{B.44}$$

Because $x^\top x = ||x||^2$, the above expression is also commonly written as

$$\lambda_{\min}\, ||x||^2 \leq x^\top P x \leq \lambda_{\max}\, ||x||^2.$$

Both are useful.

Equation (B.44) is established by choosing an orthonormal set of e-vectors for P, $\{v_1, \ldots, v_n\}$, which we know forms a basis for $\mathbb{R}^n$. Hence, for all $x \in \mathbb{R}^n$, there exist coefficients $\alpha_1, \ldots, \alpha_n$ such that $x = \alpha_1 v_1 + \cdots + \alpha_n v_n$. Then, using the two facts we have at our disposal, namely (a) $\{v_1, \ldots, v_n\}$ is orthonormal and (b), $Av_i = \lambda_i v_i$, we compute

$$\begin{aligned} x^\top x =& \alpha_1^2 + \cdots + \alpha_n^2 \\ x^\top P x =& \lambda_1 \alpha_1^2 + \cdots + \lambda_n \alpha_n^2. \end{aligned}$$

It follows that

$$\lambda_{\min} x^\top x = \lambda_{\min} \alpha_1^2 + \cdots + \lambda_{\min} \alpha_n^2 \leq \lambda_1 \alpha_1^2 + \cdots + \lambda_n \alpha_n^2 \leq \lambda_{\max} \alpha_1^2 + \cdots + \lambda_{\max} \alpha_n^2 = \lambda_{\max} x^\top x,$$

showing that (B.44) holds.

Positive Definite and Semidefinite Matrices

Def. A real symmetric matrix P is **positive definite**, if for all $x \in \mathbb{R}^n$, $x \neq 0 \implies x^\top P x > 0$. The common notation for such matrices is $P > 0$.

Def. A real symmetric matrix P is positive semidefinite, if for all $x \in \mathbb{R}^n \implies x^\top P x \geq 0$. The common notation for such matrices is $P \geq 0$.

From (B.44), we arrive at the following facts.

Fact A symmetric matrix P is positive definite if, and only if, all of its eigenvalues are greater than 0.

Fact A symmetric matrix P is positive semidefinite if, and only if, all of its eigenvalues are greater than or equal to 0.

Two additional useful facts are:

Fact A symmetric $n \times n$ matrix P is positive semidefinite if, and only if, there exists a $k \times n$ matrix N such that $N^\top \cdot N = P$.

Fact A symmetric $n \times n$ matrix P is positive definite if, and only if, there exists an $n \times n$ matrix N with linearly independent columns such that $N^\top \cdot N = P$.

Example B.7 *Determine, which, if any, of the following matrices are positive definite or positive semidefinite.*

$$P_1 = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, P_2 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, P_3 = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}, P_4 = \begin{bmatrix} 4 & 1 \\ 3 & 4 \end{bmatrix}$$

Solution: Because P_4 is not symmetric, it cannot be positive definite or positive semidefinite! Using Julia, we compute the e-values of P_1, P_2, and P_3

$$\begin{aligned} P_1 &\implies \lambda_1 = 1, \lambda_2 = 3 \implies P > 0 \\ P_2 &\implies \lambda_1 = -1, \lambda_2 = 3 \implies P \ngeq 0 \text{ (neither positive semidefinite nor positive definite)} \\ P_3 &\implies \lambda_1 = 0, \lambda_2 = 5 \implies P \geq 0. \end{aligned}$$

We note that P being positive definite does NOT mean that all of its entries have to be positive! P can have entries with negative values and still be positive definite. We note that all of the entries of P being positive does NOT imply that P is even positive semidefinite. ■.

LU Factorization to Check P > 0

Computing e-values is a terrible way to determine if a matrix is positive definite or not. One of the better ways is to do an LU Factorization without permutations. The key fact is that, if an $n \times n$ matrix P is symmetric and invertible, then it can be written as

$$P = L \cdot U;$$

you can do the factorization with permuting any of the rows of P. Moreover, there is always a diagonal matrix D such that

$$U = D \cdot L^\top.$$

Determining D from U is trivial: you just normalize by the diagonal of U. Then, $P = L \cdot D \cdot L^\top$ and

$$P > 0 \iff D > 0, \text{ that is, all of the entries on the diagonal of } D \text{ are positive.}$$

Example B.8 *Determine if the randomly generated symmetric matrix P is positive definite or not.*

$$P = \begin{bmatrix} 8.241e-01 & 1.171e+00 & 1.117e+00 & 1.706e+00 & 1.021e+00 \\ 1.171e+00 & 1.574e+00 & 8.547e-01 & 1.102e+00 & 2.871e-01 \\ 1.117e+00 & 8.547e-01 & 1.238e+00 & 7.506e-01 & 1.291e+00 \\ 1.706e+00 & 1.102e+00 & 7.506e-01 & 2.943e-01 & 9.570e-01 \\ 1.021e+00 & 2.871e-01 & 1.291e+00 & 9.570e-01 & 1.448e+00 \end{bmatrix}.$$

Solution: We do the LU Factorization without permutations and obtain

$$L = \begin{bmatrix} 1.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 1.421e+00 & 1.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 1.356e+00 & 8.151e+00 & 1.000e+00 & 0.000e+00 & 0.000e+00 \\ 2.070e+00 & 1.469e+01 & 1.616e+00 & 1.000e+00 & 0.000e+00 \\ 1.239e+00 & 1.294e+01 & 1.649e+00 & 5.893e-01 & 1.000e+00 \end{bmatrix}$$

$$U = \begin{bmatrix} 8.241e-01 & 1.171e+00 & 1.117e+00 & 1.706e+00 & 1.021e+00 \\ 0.000e+00 & -8.990e-02 & -7.328e-01 & -1.321e+00 & -1.164e+00 \\ 0.000e+00 & 0.000e+00 & 5.696e+00 & 9.203e+00 & 9.393e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.296e+00 & 7.636e-01 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 & -6.909e-01 \end{bmatrix}$$

We extract the diagonal of U and we form $D \cdot L^\top$

$$D = \begin{bmatrix} 8.241e-01 & 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & -8.990e-02 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 5.696e+00 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.296e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 & -6.909e-01 \end{bmatrix}$$

$$D \cdot L^\top = \begin{bmatrix} 8.241e-01 & 1.171e+00 & 1.117e+00 & 1.706e+00 & 1.021e+00 \\ 0.000e+00 & -8.990e-02 & -7.328e-01 & -1.321e+00 & -1.164e+00 \\ 0.000e+00 & 0.000e+00 & 5.696e+00 & 9.203e+00 & 9.393e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.296e+00 & 7.636e-01 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 & -6.909e-01 \end{bmatrix}$$

and we recognize, that indeed, $U = D \cdot L^\top$.

Back to the question of determining whether P is positive definite? We see that D has non-positive entries and therefore P is not positive definite. We only formed $D \cdot L^\top$ to illustrate that $P = L \cdot D \cdot L^\top$. ■

The following results are primarily of use for "hand calculations" or proving results about positive definite matrices. We include them for completeness.

Schur Complement Theorem: A way to Decompose the Test for Being Positive Definite

Suppose that A is $n \times n$, symmetric, and invertible, B is $n \times m$, C is $m \times m$, symmetric, and invertible, and

$$M := \begin{bmatrix} A & B \\ B^\top & C \end{bmatrix},$$

which is then $(n+m) \times (n+m)$ and symmetric. Under these conditions, the following three statements are equivalent:

(a) $M > 0$.

(b) $A > 0$, and $C - B^\top \cdot A^{-1} \cdot B > 0$.

(c) $C > 0$, and $A - B \cdot C^{-1} \cdot B^\top > 0$.

Remarks:

- $C - B^\top \cdot A^{-1} \cdot B$ is called the Schur Complement of A in M.
- $A - B \cdot C^{-1} \cdot B^\top$ is called the Schur Complement of C in M.

B.4 Singular Value Decomposition or SVD

The material here is inspired by a handout prepared by Prof. James Freudenberg, EECS, University of Michigan.

B.4.1 Motivation

In abstract linear algebra, a set of vectors is either linearly independent or not. There is nothing in between. For example, the set of vectors

$$\left\{ v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} 0.999 \\ 1 \end{bmatrix} \right\}$$

is linearly independent. In this case, one looks at the set of vectors and says, yes, BUT, the vectors are "almost" dependent because when one computes the determinant

$$\det \begin{bmatrix} 1 & 0.999 \\ 1 & 1 \end{bmatrix} = 0.001,$$

the result is pretty small, so it should be fine to call them dependent.

Well, what about the set

$$\left\{ v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 10^4 \\ 1 \end{bmatrix} \right\}?$$

When you form the matrix and check the determinant, you get

$$\det \begin{bmatrix} 1 & 10^4 \\ 0 & 1 \end{bmatrix} = 1,$$

which seems pretty far from zero. So are these vectors "adequately" linearly independent?

Maybe not! Let's note that

$$\begin{bmatrix} 1 & 10^4 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 10^{-4} & 0 \end{bmatrix} = \begin{bmatrix} 1 & 10^4 \\ 10^{-4} & 1 \end{bmatrix},$$

and its determinant is zero! Hence, it's possible to add a very small perturbation to one of the vectors and make the set linearly dependent! This cannot be good.

B.4.2 Definition and Main Theorem

Rectangular Diagonal Matrix

An $n \times m$ matrix Σ is a **Rectangular Diagonal Matrix** if

$$\Sigma_{ij} = 0 \text{ for } i \neq j.$$

Alternative and equivalent way to define Rectangular Diagonal is

(a) (tall matrix) $n > m$ $\quad \Sigma = \begin{bmatrix} \Sigma_d \\ 0 \end{bmatrix}$, where Σ_d is an $m \times m$ diagonal matrix.

(b) (wide matrix) $n < m$ $\quad \Sigma = \begin{bmatrix} \Sigma_d & 0 \end{bmatrix}$, where Σ_d is an $n \times n$ diagonal matrix.

The **diagonal** of Σ is defined to be the diagonal of Σ_d.

Singular Value Decomposition (Main Theorem)

Every $n \times m$ real matrix A can be factored as

$$A = U \cdot \Sigma \cdot V^\top,$$

where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix, Σ is an $n \times m$ rectangular diagonal matrix, and the diagonal of Σ,

$$\mathrm{diag}(\Sigma) = [\sigma_1, \sigma_2, \cdots, \sigma_p],$$

satisfies $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$, for $p := \min(n, m)$.

Moreover, the columns of U are eigenvectors of $A \cdot A^\top$, the columns of V are eigenvectors of $A^\top \cdot A$, and $\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_p^2\}$ are eigenvalues of both $A^\top \cdot A$ and $A \cdot A^\top$.

The **Singular Values of A** are the elements $\{\sigma_1, \ldots, \sigma_p\}$ from the diagonal of Σ.

Another way to write the SVD of A is

$$A = \sigma_1 u_1 \cdot v_1^\top + \sigma_2 u_2 \cdot v_2^\top + \cdots + \sigma_p u_p \cdot v_p^\top,$$

where u_i and v_i are columns of U and V respectively.

$$U = \left[\begin{array}{cccc} u_1 & u_2 & \cdots & u_n \end{array}\right] \text{ and } V = \left[\begin{array}{cccc} v_1 & v_2 & \cdots & v_m \end{array}\right]. \tag{B.45}$$

This formula follows from our matrix multiplication formulation through the sum over columns times rows, where we note that the columns of V are the rows of $V^\top$.

Rank and Nullity of a Matrix

The **rank** of an $n \times m$ matrix A is the dimension of its column span and the **nullity** of A is the dimension of its null space. Let r be the number of non-zero singular values of A. Then

Fact $\mathrm{rank}(A) := \dim \mathrm{col\ span}\{A\} = r$.

Fact $\mathrm{nullity}(A) := \dim \mathrm{null}(A) = m - r$.

Example B.9 *Determine the SVD of A as well as its rank and nullity,*

$$A = \left[\begin{array}{cc} 1 & 10^4 \\ 0 & 1 \end{array}\right].$$

Solution: Using the LinearAlgebra package in Julia, we find

$$U = \left[\begin{array}{cc} 1.000e+00 & -1.000e-04 \\ 1.000e-04 & 1.000e+00 \end{array}\right]$$

$$\Sigma = \left[\begin{array}{cc} 1.000e+04 & 0.000e+00 \\ 0.000e+00 & 1.000e-04 \end{array}\right]$$

$$V = \left[\begin{array}{cc} 1.000e-04 & -1.000e+00 \\ 1.000e+00 & 1.000e-04 \end{array}\right]$$

There are two non-zero singular values, and thus $r = 2$. It follows that $\mathrm{rank}(A) = 2$ and $\mathrm{nullity}(A) = 0$.

Information about the "near" linear dependence of the columns of A is in the diagonal matrix Σ. There are two singular values, $\sigma_1 = 10^4$ and $\sigma_2 = 10^{-4}$. Their ratio is 10^8, which is an indicator that these vectors are "nearly linearly dependent". "Numerically", one would say that $r = 1$ and hence $\mathrm{rank}(A) = r = 1$ and $\mathrm{nullity}(A) = 2 - r = 1$. ■

B.4.3 Numerical Linear Independence

Illustration: 5×5 matrix. For

$$A = \begin{bmatrix} -32.57514 & -3.89996 & -6.30185 & -5.67305 & -26.21851 \\ -36.21632 & -11.13521 & -38.80726 & -16.86330 & -1.42786 \\ -5.07732 & -21.86599 & -38.27045 & -36.61390 & -33.95078 \\ -36.51955 & -38.28404 & -19.40680 & -31.67486 & -37.34390 \\ -25.28365 & -38.57919 & -31.99765 & -38.36343 & -27.13790 \end{bmatrix},$$

and the Julia commands

```
 using LinearAlgebra

 A=[-32.57514 -3.89996 -6.30185 -5.67305 -26.21851;
-36.21632 -11.13521 -38.80726 -16.86330 -1.42786;
-5.07732 -21.86599 -38.27045 -36.61390 -33.95078;
-36.51955 -38.28404 -19.40680 -31.67486 -37.34390;
-25.28365 -38.57919 -31.99765 -38.36343 -27.13790 ]

(U ,Sigma, V) = svd(A)
```

one obtains

$$U = \begin{bmatrix} -2.475e-01 & -5.600e-01 & 4.131e-01 & 5.759e-01 & 3.504e-01 \\ -3.542e-01 & -5.207e-01 & -7.577e-01 & -1.106e-02 & -1.707e-01 \\ -4.641e-01 & 6.013e-01 & -1.679e-01 & 6.063e-01 & -1.652e-01 \\ -5.475e-01 & -1.183e-01 & 4.755e-01 & -3.314e-01 & -5.919e-01 \\ -5.460e-01 & 1.992e-01 & -2.983e-02 & -4.369e-01 & 6.859e-01 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 1.325e+02 & 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 3.771e+01 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 3.342e+01 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.934e+01 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 & 7.916e-01 \end{bmatrix}$$

$$V = \begin{bmatrix} 4.307e-01 & 8.839e-01 & -5.303e-02 & 8.843e-02 & -1.503e-01 \\ 4.309e-01 & -2.207e-01 & -1.961e-01 & 7.322e-01 & 4.370e-01 \\ 4.617e-01 & -8.902e-02 & 7.467e-01 & -3.098e-01 & 3.539e-01 \\ 4.730e-01 & -3.701e-01 & 7.976e-02 & 1.023e-01 & -7.890e-01 \\ 4.380e-01 & -1.585e-01 & -6.283e-01 & -5.913e-01 & 1.968e-01 \end{bmatrix}$$

Because the **smallest singular value $\sigma_5 = 0.7916$ is less than 1% of the largest singular value $\sigma_1 = 132.5$**, in many cases, one would say that the numerical rank of A was 4 instead of 5.

This notion of numerical rank can be formalized by asking the following question: Suppose $\text{rank}(A) = r$. How far away is A from a matrix of rank strictly less than r?

The numerical rank of a matrix is based on the expansion in (B.4.2), which is repeated here for convenience,

$$A = U \cdot \Sigma \cdot V^\top = \sum_{i=1}^{p} \sigma_i u_i \cdot v_i^\top = \sigma_1 u_1 \cdot v_1^\top + \sigma_2 u_2 \cdot v_2^\top + \cdots + \sigma_p u_p \cdot v_p^\top,$$

where $p = \min\{m, n\}$, and once again, the singular values are ordered such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$. Each term $u_i \cdot v_i^\top$ is a rank-one matrix. The following will help you understand the expansion.

Exercises or Facts:

- $A \cdot A^\top = U \cdot \Sigma \cdot \Sigma^\top \cdot U^\top = \sum_{i=1}^{p} (\sigma_i)^2 \; u_i \cdot u_i^\top$
- $A^\top \cdot A = V \cdot \Sigma^\top \cdot \Sigma \cdot V^\top = \sum_{i=1}^{p} (\sigma_i)^2 \; v_i \cdot v_i^\top$
- $\left(u_i \cdot v_i^\top\right) \cdot v_j = \begin{cases} u_i & j = i \\ 0 & j \neq i \end{cases}$ and hence rank$(u_i \cdot v_i^\top) = 1$ and nullity$(u_i \cdot v_i^\top) = m - 1$
- $\left(u_i \cdot u_i^\top\right) \cdot u_j = \begin{cases} u_i & j = i \\ 0 & j \neq i \end{cases}$ and hence rank$(u_i \cdot u_i^\top) = 1$ and nullity$(u_i \cdot u_i^\top) = n - 1$
- $\left(v_i \cdot v_i^\top\right) \cdot v_j = \begin{cases} v_i & j = i \\ 0 & j \neq i \end{cases}$ and hence rank$(v_i \cdot v_i^\top) = 1$ and nullity$(v_i \cdot v_i^\top) = m - 1$
- $v_i \cdot v_i^\top$, and $u_i \cdot u_i^\top$ have e-values $\lambda_1 = 1$ distinct and $\lambda_2 = 0$ repeated $m-1$ and $n-1$ times, respectively.
- **Hint:** $\left(u_i \cdot v_i^\top\right) \cdot v_j = u_i \cdot \left(v_i^\top \cdot v_j\right) = \begin{cases} u_i & j = i \\ 0 & j \neq i \end{cases}$ because the $\{v_1, v_2, \ldots, v_m\}$ are orthonormal.

So far, we have only defined the norm of a vector. However, it is also useful to measure the "length" of matrices.

Def. (Induced Matrix Norm) Given an $n \times m$ real matrix A, the **matrix norm induced by the Euclidean vector norm** is given by:

$$||A|| := \max_{x^\top x = 1} ||Ax|| = \sqrt{\lambda_{\max}(A^\top A)}$$

where $\lambda_{\max}(A^\top A)$ denotes the largest eigenvalue of the matrix $A^\top A$. **(Recall that the matrices of the form $A^\top A$ are at least positive semidefinite and hence their e-values are real and non-negative.)** Therefore, the square root exists.

Numerical Rank

Facts: Suppose that rank$(A) = r$, so that σ_r is the smallest non-zero singular value of A.

(i) If an $n \times m$ matrix E satisfies $||E|| < \sigma_r$, then rank$(A + E) \geq r$.

(ii) There exists an $n \times m$ matrix E with $||E|| = \sigma_r$ and rank$(A + E) < r$.

(iii) In fact, for $E = -\sigma_r u_r v_r^\top$, rank$(A + E) = r - 1$.

(iv) Moreover, for $E = -\sigma_r u_r v_r^\top - \sigma_{r-1} u_{r-1} v_{r-1}^\top$, rank$(A + E) = r - 2$.

Corollary: Suppose A is square and invertible. Then σ_r measures the distance from A to the nearest singular matrix.

Illustration Continued

```
u5=U[:,5]; v5=V[:,5]; sig5=Sigma[5]
E=-sig5*u5*v5’
# Induced Norm
M=E’*E
SquareRootEigs=(abs.(eigvals(E’*E))).^0.5
#
(U ,Sigma2, V) = svd(A+E)
```

$$E = \begin{bmatrix} 4.169e-02 & -1.212e-01 & -9.818e-02 & 2.189e-01 & -5.458e-02 \\ -2.031e-02 & 5.906e-02 & 4.784e-02 & -1.066e-01 & 2.659e-02 \\ -1.966e-02 & 5.716e-02 & 4.629e-02 & -1.032e-01 & 2.574e-02 \\ -7.041e-02 & 2.048e-01 & 1.658e-01 & -3.697e-01 & 9.220e-02 \\ 8.160e-02 & -2.373e-01 & -1.922e-01 & 4.284e-01 & -1.068e-01 \end{bmatrix}$$

$$\sqrt{\lambda_i(E^\top \cdot E)} = \begin{bmatrix} 7.376e-09 \\ 2.406e-09 \\ 1.977e-09 \\ 4.163e-09 \\ 7.916e-01 \end{bmatrix}$$

$$\Sigma_2 = \begin{bmatrix} 1.325e+02 & 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 3.771e+01 & 0.000e+00 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 3.342e+01 & 0.000e+00 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.934e+01 & 0.000e+00 \\ 0.000e+00 & 0.000e+00 & 0.000e+00 & 0.000e+00 & 1.775e-15 \end{bmatrix}$$

We added a matrix with norm 0.7916 and made the (exact) rank drop from 4 to 5! How cool is that? This example shows that SVD can exactly measure how close a matrix is to being singular. We also see that $E^\top \cdot E$ has rank one: there is one non-zero e-value and the rest are (essentially) zero as the theory promised.

Other Interesting and Useful Facts

(a) **Null space:** null$(A) := \{x \in \mathbb{R}^m \mid Ax = 0\}$

(b) **Range:** range$(A) := \{y \in \mathbb{R}^n \mid \text{ such that } y = Ax \text{ for some } x \in \mathbb{R}^m\}$

(c) **Fact:** Suppose $A = U \cdot \Sigma \cdot V^\top$. Then the columns of U corresponding to non-zero singular values are a basis for range(A) and the columns of V corresponding to zero singular values are a basis for null(A), viz

$$\begin{aligned} \text{range}(A) &:= \text{span}\{u_1, ..., u_r\}, \text{and} \\ \text{null}(A) &:= \text{span}\{v_{r+1}, ..., v_m\}. \end{aligned}$$

(d) The SVD can also be used to compute an "effective" range and an "effective" null space of a matrix.

(e) **Fact:** Suppose that $\sigma_1 \geq ... \geq \sigma_r > \delta \geq \sigma_{r+1} \geq ...\sigma_p \geq 0$, so that r is the "effective" or "numerical rank" of A. (Note the δ inserted between σ_r and σ_{r+1} to denote the break point.)

(f) **Fact:** Let range$_{\text{eff}}(A)$ and null$_{\text{eff}}(A)$ denote the effective range and effective null space of A, respectively. Then we can calculate bases for these subspaces by choosing appropriate singular vectors:

$$\begin{aligned} \text{range}_{\text{eff}}(A) &:= \text{span}\{u_1, ..., u_r\}, \text{and} \\ \text{null}_{\text{eff}}(A) &:= \text{span}\{v_{r+1}, ..., v_m\}. \end{aligned}$$

B.5 Linear Transformations and Matrix Representations

Def. A function $L : \mathbb{R}^m \to \mathbb{R}^n$ is a **linear transformation** if for all $x, z \in \mathbb{R}^m$, $\alpha, \beta \in \mathbb{R}$,

$$L(\alpha x + \beta z) = \alpha L(x) + \beta L(z). \tag{B.46}$$

Just as with checking the subspace property, one can break the condition (B.46) into two separate properties,

$$\begin{aligned} L(x + z) &= L(x) + L(z) \\ L(\alpha x) &= \alpha L(x). \end{aligned}$$

You already know at least one linear transformation from $\mathbb{R}^m$ to $\mathbb{R}^n$, namely, let A be an $n \times m$ real matrix and for $x \in \mathbb{R}^m$, define $L : \mathbb{R}^m \to \mathbb{R}^n$ by

$$L(x) = Ax. \tag{B.47}$$

It is straightforward to show that (B.46) holds and thus we leave that to you. If you are trying to think of a clever linear transformation that is not built from a matrix, but you cannot come up with one, well, there is a reason for that: there are none!

Linear Transformations from $\mathbb{R}^m$ to $\mathbb{R}^n$ are Kind of Boring

Fact: Let $L : \mathbb{R}^m \to \mathbb{R}^n$ be a linear transformation. Then there exists an $n \times m$ real matrix A such that, for every $x \in \mathbb{R}^m$

$$L(x) = Ax.$$

The matrix A has a name: it is called the **matrix representation of** L. It's computation is relatively easy. Let $\{e_1, e_2, \dots, e_m\}$ be the natural basis vectors for $\mathbb{R}^m$. Define

$$a_j^{\rm col} := L(e_i) \textbf{ and } A := \left[\begin{array}{cccc} a_1^{\rm col} & a_2^{\rm col} & \cdots & a_m^{\rm col} \end{array} \right] = \left[\begin{array}{cccc} L(e_1) & L(e_2) & \cdots & L(e_m) \end{array} \right].$$

Then $a_j^{\rm col} \in \mathbb{R}^n$ because $L(x) \in \mathbb{R}^n$ for all $x \in \mathbb{R}^m$. Now, because $\{e_1, e_2, \dots, e_m\}$ is a basis, we have

$$x \in \mathbb{R}^m \iff x = x_1 e_1 + x_2 e_2 + \cdots + x_m e_m.$$

Because L is a linear transformation,

$$\begin{aligned} L(x) &= L(x_1 e_1 + x_2 e_2 + \cdots + x_m e_m) \\ &= x_1 L(e_1) + x_2 L(e_2) + \cdots + x_m L(e_m) \\ &= x_1 a_1^{\rm col} + x_2 a_2^{\rm col} + \cdots + x_m a_m^{\rm col} \\ &= Ax. \end{aligned}$$

We will give you just a hint that **there are interesting linear transformations**, but to do that, we need to build an interesting vector space, the set of polynomials of degree less than or equal to n, namely

$$P_n(t) := \{a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \; | a_i \in \mathbb{R}, 0 \le i \le n\}.$$

We note that everything in $P_n(t)$ appears to be a linear combination of the set of monomials, $\{1, t, t^2, \dots, t^n\}$. Indeed, if you take a bit more Linear Algebra, such as Math 217, you can make sense of the monomials as being vectors, and not just any vectors, but **basis vectors** for $P_n(t)$. Hence,

$$P_n(t) = \text{span}\{1, t, t^2, \dots, t^n\}.$$

It is clear that if you add any two polynomials of degree less than or equal to n, you get another polynomial of degree less than or equal to n. If you multiply a polynomial of degree less than or equal to n by a real number, you get another polynomial of degree less than or equal to n. This tells you that $P_n(t)$ satisfies all the properties of being a **vector space**: it is **closed under linear combinations**! And yes, in abstract Linear Algebra, we call the elements of $P_n(t)$ vectors.

We define $L : P_n(t) \to P_n(t)$ by $L(p(t)) := \frac{dp(t)}{dt}$, the first derivative of the polynomial $p(t)$, that is, for those of you who have taken Calculus I,

$$L(a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n) := a_1 + 2a_2 t + 3a_3 t^2 + \cdots + n a_n t^{n-1}. \tag{B.48}$$

The rules of differentiation imply that L satisfies (B.46), that is, the rules of being a linear transformation. We note that L defined in (B.48) does not look like it comes from a matrix.

Remark: Bummer! Since L does not appear to come from a matrix, does that mean that we cannot apply to it any of the computational tools that we have developed in ROB 101? The emphatic answer is: we absolutely can apply them! How? We'll show below that there is a matrix hiding inside of L! While this is quite a bit beyond the scope of ROB 101, we feel that it might provide additional motivation for you to take a more advanced Linear Algebra course!

Definition: Let $\{v\} := \{v_1, v_2, \dots, v_k\}$ be a basis for a (real) vector space V. We know that for any vector $x \in V$, there exist[2] real coefficients $\alpha_1, \alpha_2, \dots, \alpha_k$ such that

$$x = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k. \tag{B.49}$$

The column vector built by expressing x as a linear combination of the basis vectors in $\{v\}$,

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_k \end{bmatrix} \in \mathbb{R}^k, \tag{B.50}$$

is called the **representation of x with respect to the basis** $\{v_1, v_2, \dots, v_k\}$. A nice notation for it is

$$[x]_{\{v\}} := \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_k \end{bmatrix} \in \mathbb{R}^k. \tag{B.51}$$

The representation of x in V is a vector in $\mathbb{R}^k$.

Let's apply this idea to the vector space $V := P_n(t)$ and its very nice basis

$$\{v\} = \{1, t, t^2, \dots, t^n\}.$$

We noted earlier that any vector in "$x \in V$" (that is, $p(t) \in P_n(t)$) can be written as

$$x = p(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n,$$

and hence,

$$[x]_{\{v\}} := \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^{n+1}. \tag{B.52}$$

It's kind of obvious, we add two polynomials of the same degree by adding their coefficients, and (B.52) is simply enticing us to do just that. [In other words, do not overthink it!]

Hence, if polynomials of degree less than or equal to n can be represented by columns of numbers, can differentiation be represented by a matrix? The answer is yes. As in (B.48), we define $L : V \to V$ by if $x = a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \in V$,

$$L(x) := a_1 + 2a_2 t + 3a_3 t^2 + \cdots + n a_n t^{n-1}. \tag{B.53}$$

We now break this up and apply L to each of the basis elements

$$\{v\} := \{v_1 = 1, v_2 = t, v_3 = t^2, \dots, v_{n+1} = t^n\},$$

yielding

$$L(v_1) = 0, L(v_2) = v_1, L(v_3) = 2v_2, \dots, L(v_{n+1}) = n v_n, \tag{B.54}$$

and note that

$$[L(v_1)]_{\{v\}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}, [L(v_2)]_{\{v\}} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}, [L(v_3)]_{\{v\}} = \begin{bmatrix} 0 \\ 2 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}, \dots, [L(v_n)]_{\{v\}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ n-1 \\ 0 \\ 0 \end{bmatrix}, [L(v_{n+1})]_{\{v\}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ n \\ 0 \end{bmatrix}. \tag{B.55}$$

[2] The coefficients are actually unique. You should you prove that if you have two such sets of coefficients that they have to be equal. It follows form the definition of linear Independence!

We use the above vectors as the columns of a matrix A, where $a_j^{\rm col} := [L(v_j)]_{\{v\}}$,

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & n-1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & n \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}. \tag{B.56}$$

We next note that for $x = a_0 + a_1 t + \cdots + a_{n-2}t^{n-2} + a_{n-1}t^{n-1} + a_n t^n \in V$, its representation is given in (B.52) and that

$$\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & n-1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & n \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-2} \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ 2a_2 \\ \vdots \\ (n-1)a_{n-1} \\ na_n \\ 0 \end{bmatrix}; \tag{B.57}$$

moreover, the right hand side of the equation is the representation of

$$\frac{d}{dt}\left(a_0 + a_1 t + \cdots + a_{n-2}t^{n-2} + a_{n-1}t^{n-1} + a_n t^n\right) = a_1 + 2a_2 t + \cdots + (n-1)a_{n-1}t^{n-2} + na_n t^{n-1}$$

with respect to the monomials. In other symbols,

$$A\,[x]_{\{v\}} = [L(x)]_{\{v\}}. \tag{B.58}$$

A is called the **matrix representation of L with respect to the basis $\{v\}$**. In Robotics, we often need to differentiate signals in real-time on our robots. We do it using versions of (B.57) and (B.58), which transform differentiation into matrix multiplication!

B.6 Affine Transformations

All functions $f : \mathbb{R}^m \to \mathbb{R}^n$ that fail (B.46) are nonlinear! That does not seem very discerning. There are a few more classes of functions called out, and one in particular is very important in Linear Algebra is **Definition:** A function $f : \mathbb{R}^m \to \mathbb{R}^n$ is **affine** if there exists a constant vector $b \in \mathbb{R}^n$ and an $n \times m$ real matrix A such that

$$f(x) = Ax + b. \tag{B.59}$$

A bit more abstract definition is $f : \mathbb{R}^m \to \mathbb{R}^n$ is **affine** if there exists a constant vector $b \in \mathbb{R}^n$ such that the function $L : \mathbb{R}^m \to \mathbb{R}^m$ by

$$L(x) := f(x) - b$$

is linear. When dealing with $\mathbb{R}^n$ and $\mathbb{R}^m$, the two definitions are equivalent, while the more abstract definition extends to more general settings.

Appendix C

What is an Ordinary Differential Equation?

Learning Objectives

- Provide motivation for considering equations with derivatives.
- Provide a numerical means to approximate a solution to a differential equation.
- Prepare you a bit for Math 216.

Outcomes

- See $F = ma$ in a new light
- Define an ordinary differential equation, aka an ODE
- Learn how to pass from a differential equation to a difference equation
- Relate the process of iterating a difference equation to the process of solving an ODE

Equations with a derivative in them are called **differential equations**. In Robotics, we use differential equations to understand the motion of robots, such as Cassie Blue, or in the case of Project #3, a Segway. The topic of differential equations is typically delayed until a fourth-semester Calculus course. Moreover, such courses focus almost exclusively on closed-form solutions to differential equations. As you can imagine, we're not a big fan of that. Here, you will see that tools we have developed so far in ROB 101 allow us to develop elementary numerical tools for analyzing the solutions of interesting differential equations.

C.1 Preliminaries: Expanding our Concept of an Equation

Equations in a scalar variable $x \in \mathbb{R}$ probably seem pretty trivial to you by now. Examples we have seen include:

- Linear equation: $ax = b$;
- Quadratic equation: $ax^2 + bx + c = 0$;
- Cubic equation: $ax^3 + bx^2 + cx + d = 0$; or
- Trigonometric equation: $A\cos(x) + B\sin(x) + C = 0$.

One obvious way to increase the generality of our analysis tools for equations is to include several variables, $x \in \mathbb{R}^n$, $n > 1$, and we've done a bunch of that in ROB 101 as well, where we studied how to find solutions to

- System of linear equations: $Ax = b$; and
- System of nonlinear equations: $F(x) = 0$.

In the case of linear systems, we explored map building and linear regression (easy version of Machine Learning) as cool application problems. For nonlinear systems, we looked at the position of a robot gripper in $\mathbb{R}^2$ and sought values for the angles of its joints so that the gripper could be placed in a desired position.

Another way to increase the generality of our analysis tools is to expand our concept of equations to include time as a variable! This is a big conceptual step: instead of the values in our equation depending on constants or other powers of our variable x, as in the examples we listed above, we will have the value of our variable x at time t depend on x at some other time, say $t - \delta t$, for some $\delta t > 0$.

C.2 Time in a Digital Computer is Discrete

To get our heads around functions that depend on time, we'll start with "time" as it is treated in a digital computer, namely, time is a sequential variable, such as

$$k = 1, 2, 3, \ldots. \tag{C.1}$$

Digital time is similar to our *human* notion of time in that it is strictly increasing, but whereas our human notion of time is continuous (it can be divided into ever smaller increments as long as we ignore the rules of Quantum Mechanics), *digital* time is fundamentally discrete in that it increases by non-zero increments.

Suppose we denote the value of our variable x at time k by $x[k]$, and we define an *equation for x at the next time*, $k + 1$, by

$$x[k+1] = \frac{1}{2}x[k] + 1. \tag{C.2}$$

Equation (C.2) says that if we know the value of our variable x at time k, we can find the value of x at time $k + 1$ by multiplying $x[k]$ by one-half and adding one to it. That seems easy enough. OK, then, what is the value of x at time $k = 4$?

To compute $x[4]$, we need to know $x[3]$. To compute $x[3]$, we need to know $x[2]$. To compute $x[2]$, we need to know $x[1]$. If we suppose that time starts at $k = 1$, then we cannot go back any further and we realize that somehow $x[1]$ must be given to us!

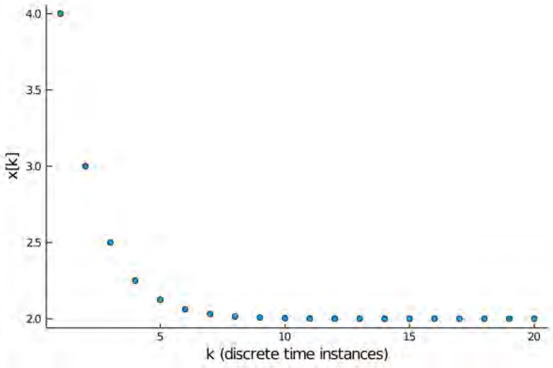


Figure C.1: Plot of the function $x : [1, 2, \ldots, 20] \to \mathbb{R}$ at discrete instances of time.

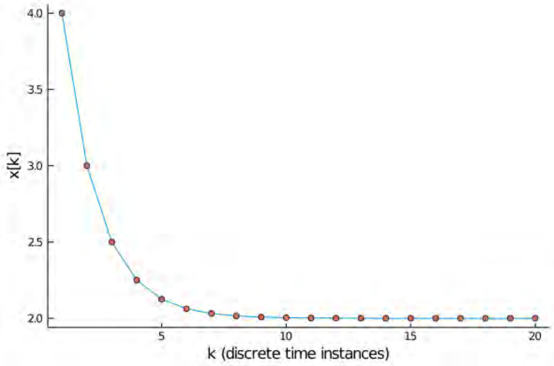


Figure C.2: Plot of the function $x : [1, 2, \ldots, 20] \to \mathbb{R}$, with the "dots" connected, giving us the impression that time is almost continuous.

Equation (C.2) is called a **difference equation** and $\mathbf{x[1]}$ is called its **initial value**. If we are given that $x[1] = 4$, we'd then compute that

$$\begin{aligned}
x[1] &= \frac{1}{2}x[0] + 1 = 3 \\
x[2] &= \frac{1}{2}x[1] + 1 = \frac{7}{2} \\
x[3] &= \frac{1}{2}x[2] + 1 = \frac{11}{4} \\
x[4] &= \frac{1}{2}x[3] + 1 = \frac{19}{8} \\
\vdots &= \qquad \vdots \\
x[k+1] &= \frac{1}{2}x[k] + 1
\end{aligned}$$

In Fig. C.1, we plot the function x for $k = 1 : 20$. We observe that it seems to converge to 2.0, which is intriguing. In Fig. C.2, we have "cheated" and made time look quasi-continuous by "connecting the dots". The Julia code for generating x and the plots is given in Sec. C.6.1.

It would seem that if we could put the "dots closer together", the effect in Fig. C.2 would be even better. Let's try!

C.3 Digital Time may be Discrete, but We Can Make the Time Increment δt Quite Small

In our previous model and plot, we implicitly took $\delta t = 1$ "unit" of time, where "unit" could have been seconds, milliseconds, months, or fortnights. We just plotted the index k and had no idea how it was directly related to a physical notion of "time". This

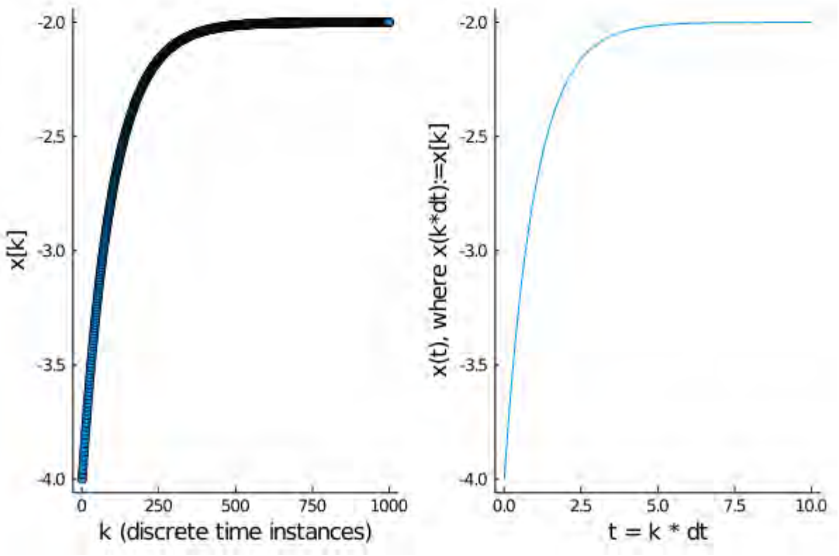


Figure C.3: Plot of the function $x : [0, 10] \to \mathbb{R}$ that solves (C.3) when its initial condition is $x[1] = 4$. The discrete interval of time $\delta t = 0.01$ is small enough that it looks continuous, and in fact, if one connects the dots, our function versus time looks like a continuous curve.

time we'll define $\delta t = 0.01$ seconds, that is, 10 milliseconds, and we'll think of $x[k]$ as representing $x(k\delta t)$. To be super explicit,

$$\begin{aligned}
x[1] &:= x(t)|_{t=\delta t} = x(0.01)\\
x[2] &:= x(t)|_{t=2\delta t} = x(0.02)\\
x[3] &:= x(t)|_{t=3\delta t} = x(0.03)\\
&\vdots\\
x[99] &:= x(t)|_{t=99\delta t} = x(0.99)\\
&\vdots\\
x[301] &:= x(t)|_{t=301\delta t} = x(3.01)\\
&\text{etc.}
\end{aligned}$$

We introduce δt into the model in a particular manner that will become clear in Sec. C.4,

$$x[k+1] = x[k] - \delta t \cdot (x[k] + 2). \tag{C.3}$$

Fig C.3 shows two plots side by side. One shows the function x plotted against the index k, while the second plot shows x plotted against time, for $t = k\delta t$. The Julia code for computing the function and making the plots is given in Sec. C.6.2.

C.4 Equations with Derivatives in them are called Differential Equations

All of us hear about "$F = ma$" in High School, one of Newton's laws for how a body of mass m accelerates under the action of applied forces F. We might also learn that "acceleration", a, is the rate of change of "velocity", v, with respect to time, t.

The notation $\frac{\mathrm{d}}{\mathrm{dt}}$

In Calculus, the rate of change of one quantity, say v, with respect to another quantity, say t, is denoted

$$\frac{dv(t)}{dt}.$$

Hence, in the language of Calculus, acceleration is related to velocity by

$$a(t) := \frac{dv(t)}{dt},$$

which, once again, is just shorthand for "acceleration is the rate of change of velocity with respect to time". This course does not assume knowledge of Calculus, so please don't sweat the introduction of this notation. For us, it is just a shorthand way of saying "rate of change with respect to time".

Using this shorthand notation, we can express Newton's Law as

$$m\frac{dv(t)}{dt} = F(t). \tag{C.4}$$

If we suppose that the body is falling in air, we might model the total force $F(t)$ acting on the body as being due to gravity and air friction,

$$F(t) = -mg - k_d v(t).$$

Doing so leads us to the equation

$$\boxed{m\frac{dv(t)}{dt} = -k_d v(t) - g}. \tag{C.5}$$

This equation says that the rate of change of the velocity as a function of time is given by a sum of two terms, one of which corresponds to gravity and the other to air resistance. Equation (C.5) is called a **differential equation**, that is, an equation that depends on "derivatives."

In (C.5), let's now replace the shorthand Calculus symbol for rate of change, $\frac{dv(t)}{dt}$, with the same kind of numerical approximation we used in our studies of root finding and optimization, namely

$$\frac{dv(t)}{dt} \approx \frac{v(t+\delta t) - v(t)}{\delta t}. \tag{C.6}$$

Substituting (C.6) into (C.5) and assuming the approximation is "good enough" (that we can change $\approx$ into $=$) give

$$\begin{aligned} \frac{dv(t)}{dt} &= \frac{1}{m}\left(-k_d v(t) - g\right) \\ &\Updownarrow \\ \frac{v(t+\delta t) - v(t)}{\delta t} &= \frac{1}{m}\left(-k_d v(t) - g\right) \\ &\Updownarrow \\ v(t+\delta t) - v(t) &= \delta t \frac{1}{m}\left(-k_d v(t) - g\right) \\ &\Updownarrow \\ v(t+\delta t) &= v(t) + \delta t \frac{1}{m}\left(-k_d v(t) - g\right). \end{aligned} \tag{C.7}$$

If we then define $t = k \cdot \delta t$ and $v[k] := v(k \cdot \delta t)$, the equation looks just like our *difference equations* in Sec. C.3. Indeed, we have $v(t+\delta t) = v(k \cdot \delta t + \delta t) = v((k+1) \cdot \delta t) = v[k+1]$, and (C.7) becomes

$$v[k+1] = v[k] - \delta t \frac{k_d}{m} v[k] - \delta t \frac{g}{m}, \tag{C.8}$$

which is very much like (C.3) and hence, we now know one way to (approximately) solve the differential equation (C.5): we set an initial condition and iterate based on (C.8).

In fact, if the mass of our falling body is $m = \frac{2}{g}$ and its drag is $k_d = m$, then (C.8) is exactly the same as (C.3). Moreover, we can physically interpret the solution of the differential equation (C.5), which is plotted in Fig. C.3, as a package is tossed out of plane. Our model tracks its evolution from the moment the parachute deploys, greatly increasing its drag, thereby slowing its velocity from an initial value of -4 to -2 units of distance per second (we never specified the units).

C.5 Discretization of ODEs of Higher Dimension

ODE

ODE is short for Ordinary Differential Equation. The word "ordinary" is used because there are other kinds of differential equations, which apparently, are less "ordinary". Everyone in the know uses the terminology ODE, hence you will too!

ODEs can be vector valued too. A linear ODE may look like this,

$$\frac{dx(t)}{dt} = Ax(t) + b.$$

Just as before, we select $\delta t > 0$ and define $x[k] := x(k \cdot \delta t)$, and re-write the differential equation as a difference equation

$$\begin{aligned}
\frac{dx(t)}{dt} &\approx \frac{x(t+\delta t) - x(t)}{\delta t} \\
&\Downarrow \\
\frac{x(t+\delta t) - x(t)}{\delta t} &\approx Ax(t) + b \\
&\Downarrow \\
x(t+\delta t) &= x(t) + \delta t \left(Ax(t) + b\right) \\
&\Downarrow \\
x[k+1] &= x[k] + \delta t A x[k] + \delta t b
\end{aligned}$$

A two-dimensional example is

$$\underbrace{\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \end{bmatrix}}_{x[k+1]} = \underbrace{\begin{bmatrix} x_1[k] \\ x_2[k] \end{bmatrix}}_{x[k]} + \delta t \cdot \underbrace{\begin{bmatrix} 0.0 & 1.0 \\ -2.0 & -1.0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1[k] \\ x_2[k] \end{bmatrix}}_{x[k]} + \delta t \cdot \underbrace{\begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}}_{b}. \tag{C.9}$$

This time, our $x[1]$ is a 2×1 vector. We'll arbitrarily specify it as $x[1] = [2 \;\; 0]^\top$. The plots of x as a function of index k and of time $t = k \cdot \delta t$ are given in Fig. C.5. The associated Julia code is given in Sec. C.6.4.

Finally, not only can the function x of time can be vector valued, it can have nonlinear terms. Here is a common example from physics: a pendulum of mass m and length ℓ swinging from a frictionless pivot satisfies the equation

$$\begin{aligned}
\frac{dx_1(t)}{dt} &= x_2(t) \\
\frac{dx_2(t)}{dt} &= -\frac{g}{\ell}\sin(x_1(t)),
\end{aligned} \tag{C.10}$$

where x_1 is the angle of the pendulum and x_2 is its angular velocity. We'll rewrite the model in vector form by defining

$$x := \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \tag{C.11}$$

so that

$$\frac{dx(t)}{dt} = \begin{bmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \end{bmatrix} := \underbrace{\begin{bmatrix} x_2(t) \\ -\frac{g}{\ell}\sin(x_1(t)) \end{bmatrix}}_{f(x(t))} =: f(x(t)) \tag{C.12}$$

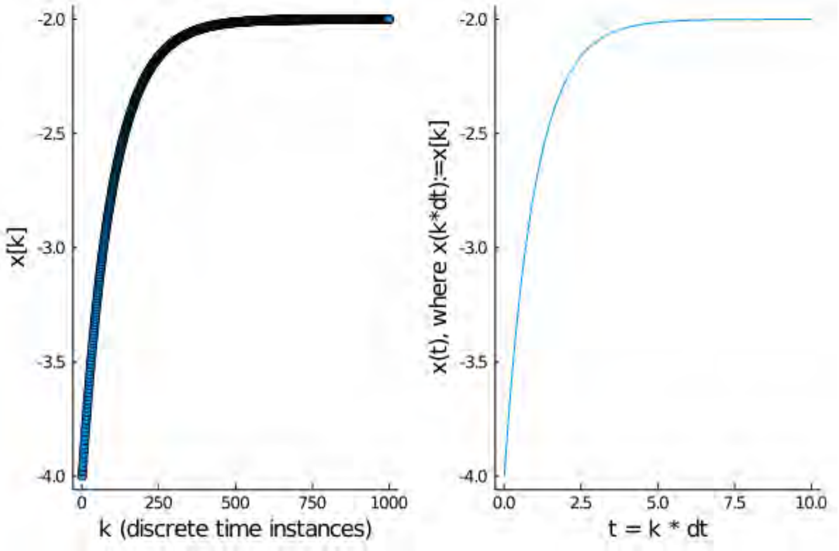


Figure C.4: Plot of the function $x : [0, 10] \to \mathbb{R}$ that solves (C.3) when its initial condition is $x[1] = 4$. The discrete interval of time $\delta t = 0.01$ is small enough that it looks continuous, and in fact, if one connects the dots, our function versus time looks like a continuous curve.

To find a solution, we discretize the model with $\delta t > 0$. To show the flexibility we have in approximating the derivative, we'll use a symmetric difference approximation this time, namely

$$\begin{gathered}\frac{dx(t)}{dt} \approx \frac{x(t+\delta t) - x(t-\delta t)}{2\delta t} \quad \text{and} \quad \frac{dx(t)}{dt} = f(x(t)) \\ \Updownarrow \\ x(t+\delta t) = x(t-\delta t) + 2\delta t f(x(t)) \quad \text{and} \quad t = k \cdot \delta t \\ \Updownarrow \\ x[k+1] = x[k-1] + \delta t f(x[k])\end{gathered} \tag{C.13}$$

We use the last line of (C.13) to iterate and compute a solution to the pendulum ODE (C.10). The solution is plotted in Fig. C.6. The associated Julia code is given in Sec. C.6.5.

Our transformation of ODEs into difference equations are examples of numerical integration methods. When we use the approximation

$$\frac{dx(t)}{dt} \approx \frac{x(t+\delta t) - x(t)}{\delta t}$$

we end up with what is called a first-order forward Euler integration method. In general, it's a pretty bad way to solve ODEs, but for a first introduction, it is great! We can do a lot with it.

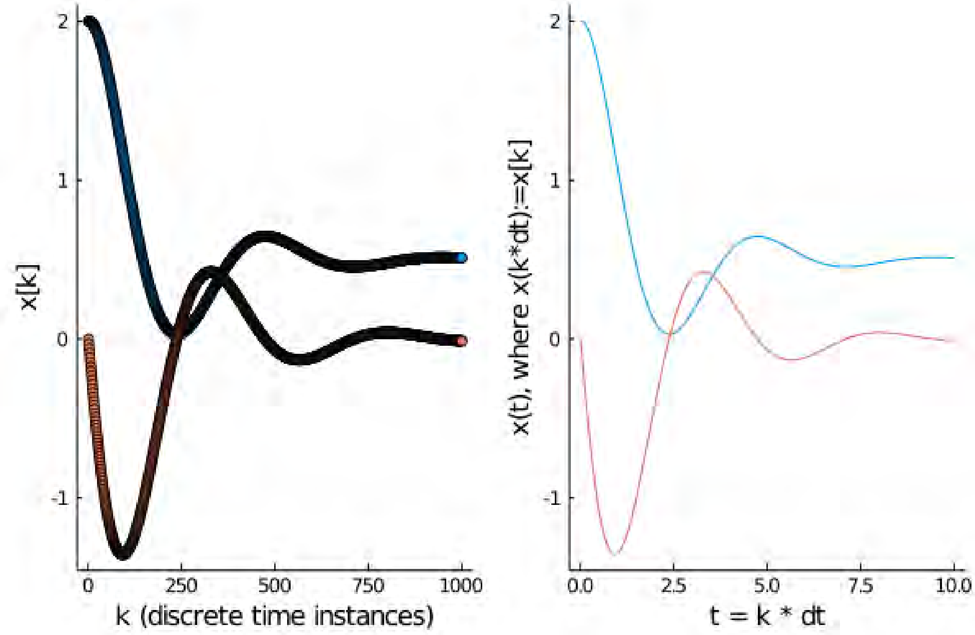


Figure C.5: Plot of the vector valued function $x : [0, 10] \to \mathbb{R}^2$ solving the ODE $\frac{dx(t)}{dt} = Ax + b$, with the solution approximated in (C.9). The discrete interval of time is small enough that it looks continuous!

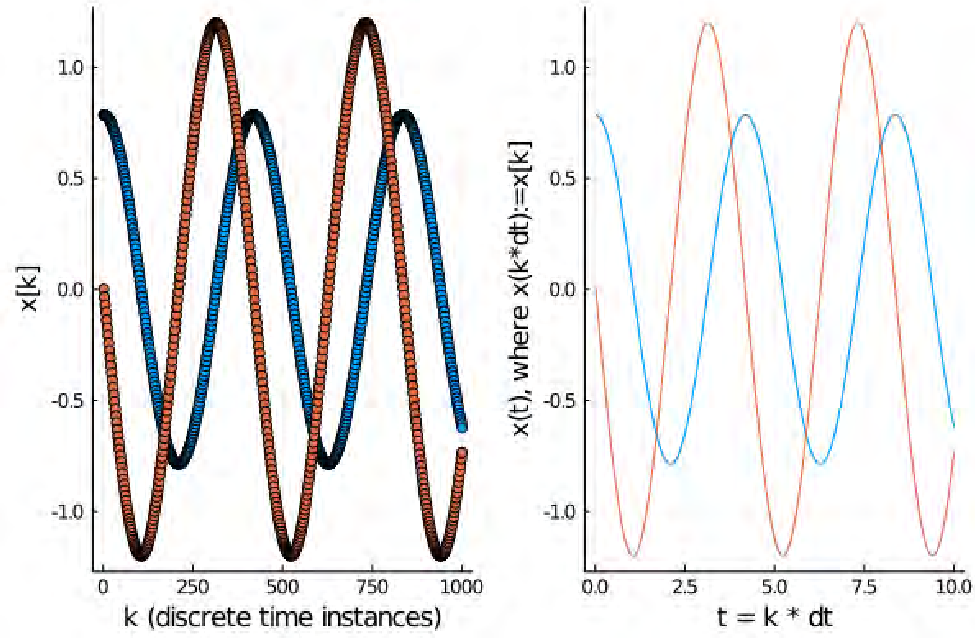


Figure C.6: Plot of a vector valued function $x : [0, 10] \to \mathbb{R}^2$ that oscillates back and forth because it corresponds to the solution the pendulum ODE given in (C.10). The blue lines are the angle of the pendulum and the brown(ish) lines are its angular velocity.

C.6 Julia Code for Generating the Figures

We provide code in Julia 1.41 for the figures appearing in this Appendix.

C.6.1 For Figures C.1 and C.2

```

# Model
dt=0.01
T=10
N=floor(Int,T/dt);
K=1:N
time = K*dt
x=zeros(N,1)
x[1]=-4
for k=1:(N-1)
    x[k+1]=(1-dt)*x[k]-2*dt
end
# Plotting
plot1=scatter(K,x,xlabel="k (discrete time instances)", ylabel="x[k]", leg=false)
plot2=plot(time,x,xlabel="t = k * dt", ylabel="x(t), where x(k*dt):=x[k]", leg=false)
plot(plot1, plot2, layout = (1, 2), legend = false)
#Turn the plot into an image that one can copy
plot!(fmt = :png)
```

C.6.2 Code for Figure C.3

```
using Plots
gr()
# Model
N=20
time=1:N
x=zeros(N,1)
x[1]=4
for k=1:(N-1)
    x[k+1]=0.5*x[k]+1
end
# Plotting
scatter(time,x)
#plot!(xlabel="k (discrete time instances)", ylabel="x[k]",
#title="Plot of as a function of time: Discrete-time Model", leg=false)
plot!(xlabel="k (discrete time instances)", ylabel="x[k]", leg=false)
#Turn the plot into an image that one can copy
plot!(fmt = :png)
#
# Second plot
#
plot(time,x)
#plot!(xlabel="k (discrete time instances)", ylabel="x[k]",
#title="Plot of as a function of time: Discrete-time Model", leg=false)
plot!(xlabel="k (discrete time instances)", ylabel="x[k]", leg=false)
scatter!(time,x)
#Turn the plot into an image that one can copy
plot!(fmt = :png)
```

C.6.3 Code for Figure C.4

```
# Model
dt=0.01
T=10
N=floor(Int,T/dt);
K=1:N
time = K*dt
```

```
x=zeros(N,1)
x[1]=4
for k=1:(N-1)
    x[k+1]=(1-dt)*x[k]+2*dt
end
# Plotting
plot1=scatter(K,x,xlabel="k (discrete time instances)", ylabel="x[k]", leg=false)
plot2=plot(time,x,xlabel="t = k * dt", ylabel="x(t), where x(k*dt):=x[k]", leg=false)
plot(plot1, plot2, layout = (1, 2), legend = false)
#Turn the plot into an image that one can copy
plot!(fmt = :png)
```

C.6.4 Code for Figure C.5

```
# Model
dt=0.01
T=10
N=floor(Int,T/dt);
K=1:N
time = K*dt
A=[0 1; -2 -1]
b=[0;1]
x=zeros(N,2)
x[1, :]=[2 0]
for k=1:(N-1)
    x[k+1,:]=x[k,:]+ dt*A*x[k,:] + dt*b
end
# Plotting
plot1=scatter(K,x,xlabel="k (discrete time instances)", ylabel="x[k]", leg=false)
plot2=plot(time,x,xlabel="t = k * dt", ylabel="x(t), where x(k*dt):=x[k]", leg=false)
plot(plot1, plot2, layout = (1, 2), legend = false)
#Turn the plot into an image that one can copy
plot!(fmt = :png)
```

C.6.5 Code for Figure C.6

```
# Model
dt=.01
T=50
N=floor(Int,T/dt);
K=1:N
time = K*dt
g=9.81 #m/s^2
l=4 # m
x=zeros(N,2)
#Pendulum equations
f(x1,x2)=[x2;-(g/l)*sin(x1)]
#Initial conditions
x[1, :]=[pi/4 0]
x[2, :]=x[1,:] + dt*f(x[1,1],x[1,2])
#Euler integration based on Symmetric Difference for dx/dt
for k=2:(N-1)
x[k+1,:]=x[k-1,:] +2*dt*f(x[k,1],x[k,2])
end
# Plotting
plot1=scatter(K,x,xlabel="k (discrete time instances)", ylabel="x[k]", leg=false)
plot2=plot(time,x,xlabel="t = k * dt", ylabel="x(t), where x(k*dt):=x[k]", leg=false)
plot(plot1, plot2, layout = (1, 2), legend = false)
#Turn the plot into an image that one can copy
plot!(fmt = :png)
```

From here on, the book is a work in progress.
Please pardon our mess during construction!

Appendix D

Camera and LiDAR Models for Students of Robotics

Learning Objectives

- [**JWG: ???**].

Outcomes

- [**JWG: ???**]

[**JWG: Jessy to write an Intro. Bruce to add background an LiDAR sensors**]

D.1 Pinhole Camera Model

Camera calibration is the process of finding the quantities internal to the camera that affect the imaging process. Today's cheap camera lenses or camera production errors may introduce a lot of distortion to images. Precise camera calibration is important when we need to deal with 3D interpretation of images, reconstruction of world models, and robot interaction with the real world. We will apply what we have learnt so far to discover the charm of camera calibration and its usage in our project.

D.2 Preliminaries: Geometrical Transformations

D.2.1 Homogeneous Coordinates

Homogeneous coordinates are widely used in computer vision and graphics. They are a nice extension of standard 3D vectors and allow us to simplify and compute various vector operations, such as translation, rotation, scaling and perspective projection. The sequence of such operations can be multiplied out into a single matrix with simple and efficient processing.Besides, homogeneous coordinates also allow to represent infinity. The Euclidean coordinate system denotes the location of an object by a triplet of numbers. However, we can't treat infinity as a regular number. Homogeneous coordinates allows by denoting infinity by $[x, y, z, 0] = \frac{[x,y,z]}{0}$ in 3D.

How to transfer between Cartesian Coordinates and Homogeneous Coordinates?

- Cartesian coordinates $\rightarrow$ Homogeneous coordinates : Multiply the coordinates by a non-zero scalar and add an extra coordinate equal to that scalar. For example, $[x, y] \rightarrow [x \cdot z, y \cdot z, z], z \neq 0$. We usually multiply the coordinates by 1.
- Homogeneous coordinates $\rightarrow$ Cartesian coordinates: Divide the Cartesian coordinates by the last coordinate and eliminate it. For example, $[x, y, z], z \neq 0 \rightarrow [x/z, y/z]$.

D.2.2 2D Translation

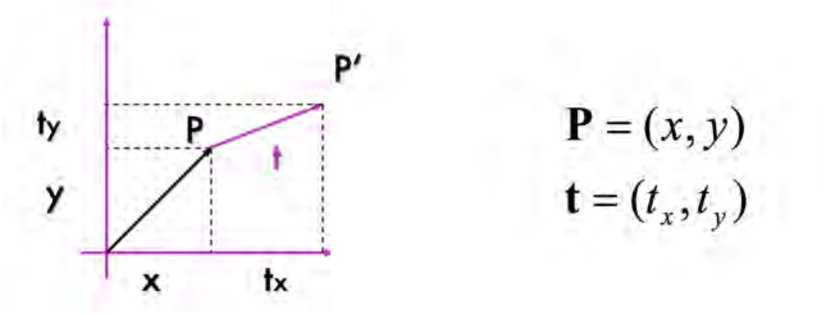


Figure D.1: Example of 2D Translation

If point **P'**(x', y') is obtained by translating **P** by $\mathbf{t}(t_x, t_y)$, then the relation between **P'** and **P** could be written as:

$$\mathbf{P'} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \mathbf{I} \cdot \mathbf{P} + \mathbf{t}$$

If we use homogeneous coordinates, $P = (x, y, 1)$, and $t = (t_x, t_y, 1)$. The the relation between P' and P becomes:

$$\mathbf{P'} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \cdot \mathbf{P} = \mathbf{T} \cdot \mathbf{P}$$

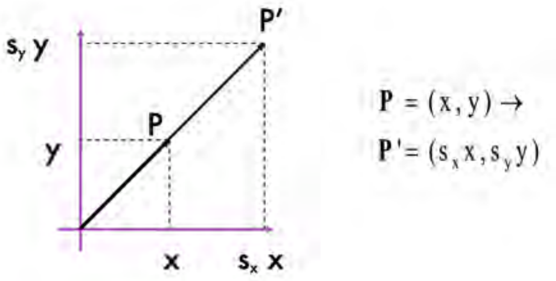


Figure D.2: Example of 2D Scaling

D.2.3 2D Scaling

If point **P'**(x', y') is obtained by scaling **P** by **s**(s_x, s_y), then the relation between **P'** and **P** could be written as:

$$\textbf{P'} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \textbf{S'} \cdot \textbf{P}$$

Similar before, the relation between **P'** and **P** could be expressed in homogeneous coordinates:

$$\textbf{P'} = \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \textbf{S'} & 0 \\ 0 & 1 \end{bmatrix} \textbf{P} = \textbf{S} \cdot \textbf{P}$$

D.2.4 2D Rotation

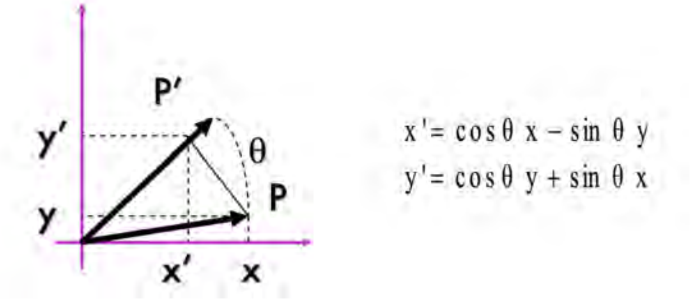


Figure D.3: Example of 2D Rotation

If point **P'**(x', y') is obtained by rotating **P** counterclockwise by θ degree, then the relation between **P'** and **P** could be written as:

$$\textbf{P'} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \textbf{R'} \cdot \textbf{P}$$

The relation between **P'** and **P** could be expressed in homogeneous coordinates:

$$\textbf{P'} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \textbf{R'} & 0 \\ 0 & 1 \end{bmatrix} \textbf{P} = \textbf{R} \cdot \textbf{P}$$

R is an orthogonal matrix and it satisfies the following properties:

- $R \cdot R^\top = R^\top \cdot R = I$
- det(R) = 1

D.2.5 Other 2D Geometrical Transformations in Homogeneous Coordinates

- Shear in x: $\begin{bmatrix} 1 & k_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- Shear in y: $\begin{bmatrix} 1 & 0 & 0 \\ k_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- Reflect about y: $\begin{bmatrix} 1 & 0 & 0 \\ k_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- General Affine Transformation: $\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$

If there are more than one type of geometric transformation, one can multiply the corresponding transformation matrix to get the final transformation matrix.
For example, if we want to find the transformation matrix after scaling P by s firstly, then rotating counterclockwise by θ degree, and finally translate by t. The position of P' could be compute by:

$$\begin{aligned} P' = \mathbf{T}\cdot\mathbf{R}\cdot\mathbf{S}\cdot P &= \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} R' & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} R'S & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \end{aligned}$$

D.3 Pinhole Model

Last section we mainly discussed the affine transformation, which preserves the parallel lines, ratio of areas, ratio of lengths on colinear lines and a few others. However, the affine transformation is not a valid assumption in general for images, nor is it valid around the image edges. The mapping from 3D object to 2D image is a perspective projection as shown in the following image:

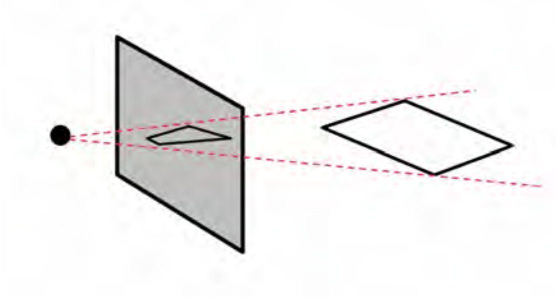


Figure D.4: Perspective schematic diagram

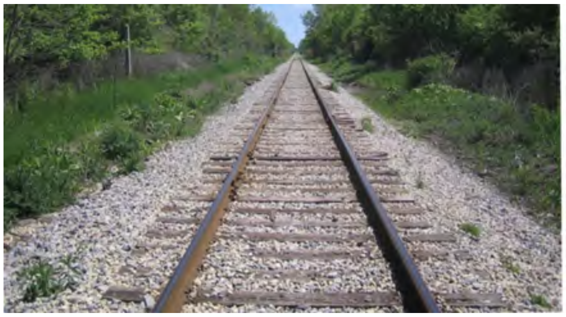


Figure D.5: Real-life perspective example: parallel rails

The **pinhole camera model** describes the mathematical relationship between the coordinates of a point in 3D space and its projection onto the image plane of an ideal pinhole camera. The following diagram illustrates the pinhole model and defines some intrinsic parameters of pinhole camera:

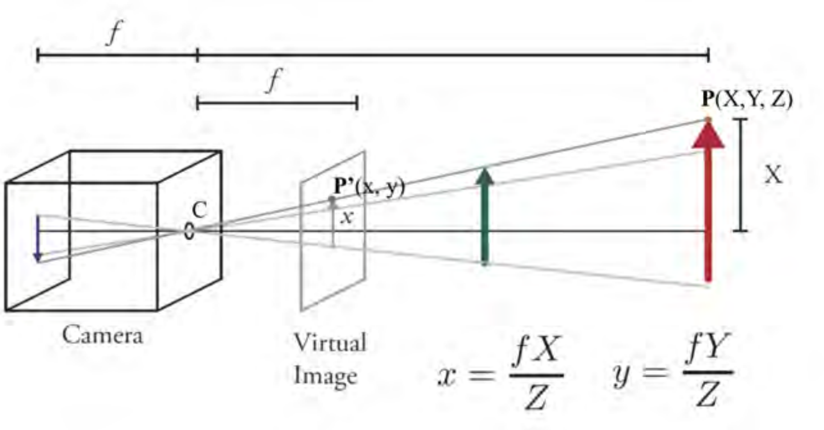


Figure D.6: Pinhole camera model

There are some key assumptions of the pinhole model:

- Aperture, i.e. the pinhole, is described as a point (point C in the image)
- No lenses are used to focus light, therefore no geometric distortions (radial distortion)
- Does not take into account that most practical cameras have only discrete image coordinates
- This model can only be used as a first order approximation of the mapping from 3D scene to 2D image. This is a perspective projection
- Ideal model that we use has identity matrix for the perspective portion of the projection matrix (**projection matrix** = [**Perspective**$|0]$ = $[\mathbf{I}_{3\times3}|0]_{3\times4}$ which assumes no perspective since the surface is close to the camera)

If **P** (X, Y, Z) is a 3D scene point, and it is projected into the 2D image by pinhole. **P'** $(x, y$ is the projective point of **P** on the 2D virtual image plane. We could find the following equations derived using similar triangles.
Remark: f is the focal length of the camera.

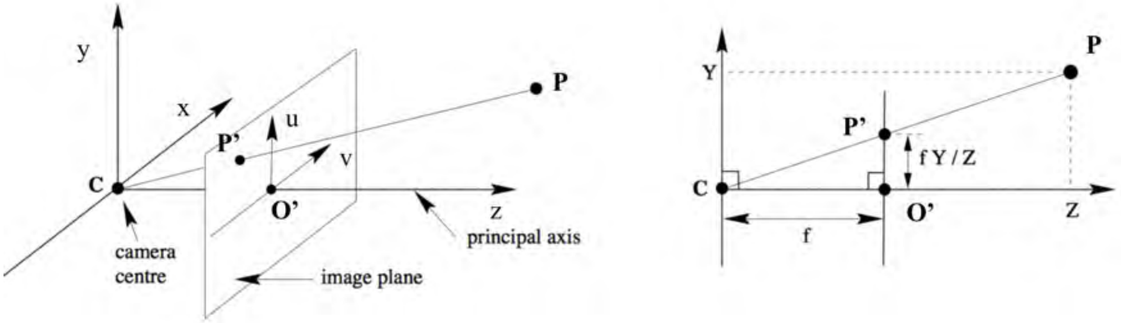


Figure D.7: Pinhole model

$$\frac{f}{Z} = \frac{y}{Y} = \frac{x}{X} \implies \begin{cases} x = \frac{fX}{Z} \\ y = \frac{fY}{Z} \end{cases} \tag{D.1}$$

The image plane is usually read as $n \times m$ pixel matrix in computer. Points in the image is then represented by pixel coordinates.

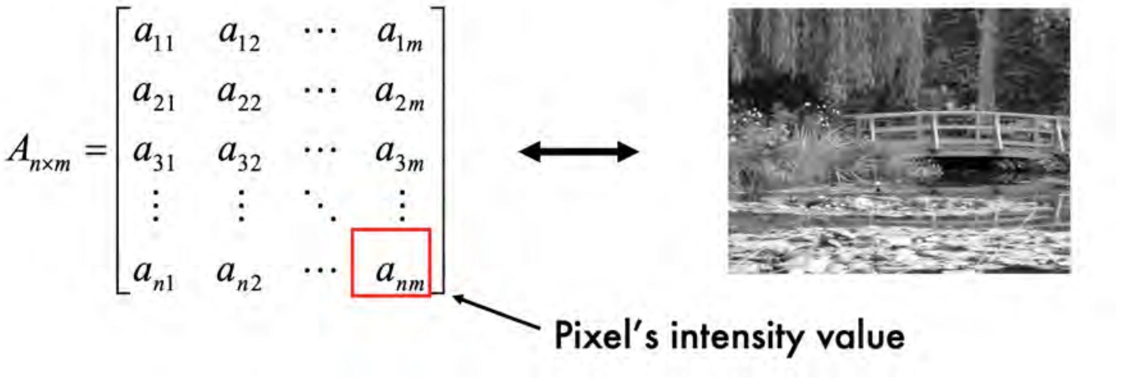

$A_{n\times m} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ a_{31} & a_{32} & \cdots & a_{3m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$
Pixel's intensity value

Figure D.8: Relation of image plane and pixel matrix

The following picture is an example of image plane, which indicates the relation between camera coordinate origin and image coordinate origin

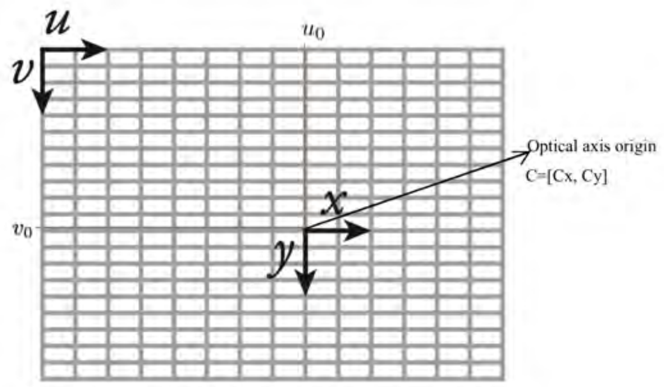


Figure D.9: Pixel coordinates plane

By expanding equation D.1, the relation between positions on the image scene (u, v) and positions in the real world (X, Y, Z) could be written as:

$$u = fs_x x^c + u_0 = \alpha\frac{X}{Z} + u_0$$
$$v = fs_y y^c + v_0 = \beta\frac{Y}{Z} + v_0$$

We assume the world frame and camera frame is aligned. s_x, s_y are the scale in x and y direction to transfer the units from metric to pixel. x^c, y^c are projective points of 3D objects in camera coordinates.

However, there are other considerations:

- Optical axis is not centered on sensor
- Pixel coordinate system orgin is in the corner of the sensor
- Pixels aren't square
- Pixel size is unknown

Therefore, we need to find a transformation matrix to consider all the possible situations to transform the camera frame to the image plane. Review 2D geometric transformations in previous section and recall the general affine transformation $\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$. We will explore more about it in the next section.

D.4 Geometric and Mathematical Relations

We would like to represent image plane coordinates in terms of real world coordinates. This is exactly the same problem as finding the conversion of 3D world coordinates to image coordinates and then inverting this transformation. In this section, we will examine the three components of this transformation.

D.4.1 Intrinsic Matrix K

Intrinsic matrix is the affine transformation matrix we mentioned before to represent the linear properties of the pinhole camera model (focal length to visual image, scale, shear, translation, and so on). The intrinsic matrix can be used to transform 2D camera coordinates (x^c, y^c) into image plane coordinates (u, v):

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x^c \\ y^c \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & ks_y & u_0 \\ 0 & fs_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^c \\ y^c \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^c \\ y^c \\ 1 \end{bmatrix}$$

Where s_x is the scale in x, s_y is the scale in y, f is the focal length, k is the ration of shear in the y direction to that in x, and (u_0, v_0) is the distance from the image center to the image plane coordinate system origin.

D.4.2 Projection Matrix

This matrix transforms the coordinates from 3D to 2D. The perspective portion of this matrix (left) is equal to the identity because we make the assumption that all captured positions are close.

$$\begin{bmatrix} x^c \\ y^c \\ 1 \end{bmatrix} = \frac{1}{Z_c} [P_{\text{perspective}} \quad 0] \begin{bmatrix} X^c \\ Y^c \\ Z^c \\ 1 \end{bmatrix} = \frac{1}{Z_c} [I_{3\times3} \quad 0]_{3\times4} \begin{bmatrix} X^c \\ Y^c \\ Z^c \\ 1 \end{bmatrix}$$

The positions are now shown in lower case and lie now on a 2D plane.

D.4.3 Extrinsic Matrix

This matrix belongs in SE(3) and it represents the transformation of position coordinates from the real world frame to the camera coordinate frame.

$$\begin{bmatrix} X^c \\ Y^c \\ Z^c \\ 1 \end{bmatrix} = [R \quad t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

D.4.4 Full Projection Matrix Workspace to Camera

Grouping everything together we get a final full projection matrix, P, such that

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{aligned} P &= K[I \quad 0][R \quad t] \\ &= \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [I \quad 0][R \quad t] \end{aligned}$$

Or with the general affine transformation as the intrinsic matrix

$$P = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} [I \quad 0][R \quad t]$$

D.5 Intrinsic Matrix Calibration

Solve the simpler problem of only finding the intrinsic matrix if the extrinsic is known. If you are already given the 2D camera coordinates (post extrinsic and projective transformation, then you can select at least 3 known points in the real world and image plane (which do not all lie within the same plane). Remember the general form

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^c \\ y^c \\ 1 \end{bmatrix}$$

For example, let's say that we know two different points then

$$\begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1^c \\ y_1^c \\ 1 \end{bmatrix}, \quad \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2^c \\ y_2^c \\ 1 \end{bmatrix}$$

The system of equations become

$$\begin{aligned} ax_1 + by_1 + c &= u_1 \\ dx_1 + ey_1 + f &= v_1 \\ ax_2 + by_2 + c &= u_2 \\ dx_2 + ey_2 + f &= v_2 \end{aligned}$$

Rearranging into the form $Ax = b$

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \end{bmatrix}$$

A quick and less rigorous derivation of the solution: We want to approximate b with the subspace of A. This means through the projection theorem that $b - Ax$ is perpendicular to Ax

$$\begin{aligned} b - Ax &\perp Ax \\ (b - Ax) \cdot (Ax) &= 0 \\ (Ax) \cdot (Ax) &= (Ax) \cdot b \\ x^T A^T Ax &= x^T A^T b \\ x &= (A^T A)^{-1} A^T b \end{aligned}$$

Then plug the corresponding values into the complete intrinsic matrix.

D.6 Nonlinear Phenomena in Calibration

In real life, because of the imperfect lens or inaccurate placement of the lens, the image would be distorted. We can use nonlinear root finding methods like Newton-Raphson to account for nonlinear phenomena. $(x_{distorted} = x_{linear} + g(x)$

- Radial Distortion
  This phenomenon occurs when light rays bend more near the edges of a lens than they do at this optical center. Smaller lens can lead to greater distortion.

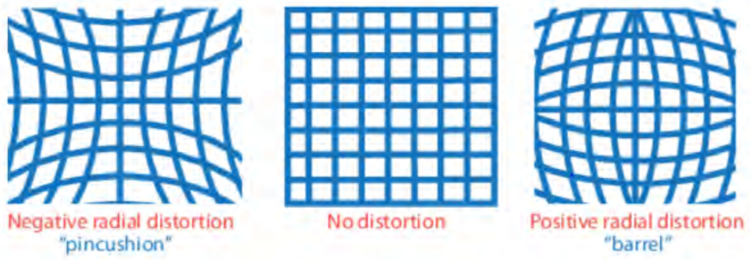


Figure D.10: Types of radial distortion

- Tangential Distortion This phenomenon occurs when the lens and the image plane are not parallel

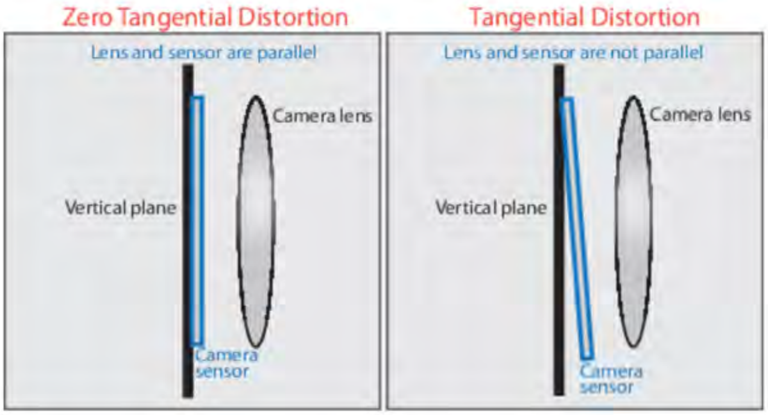


Figure D.11: Tangential distortion

D.7 Projection Map from LiDAR to Camera

D.8 Projection Map

The projection map is a mapping between a 3D and a 2D world, and is used to project a LiDAR point cloud (3D) into an image (2D). Let X be the LiDAR points and Y be the projected points on the image-plane of a camera. The standard relations are [1]

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsic parameters}} \begin{bmatrix} \mathbb{1}_{3\times3} \\ \mathbf{0}_{1\times3} \end{bmatrix}^{\mathsf{T}} \underbrace{\begin{bmatrix} R_L^C & t_L^C \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}}_{\text{extrinsic parameters}} \underbrace{\begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}}_{\text{LiDAR points}} \tag{D.2}$$

$$= K \begin{bmatrix} \mathbb{1}_{3\times3} \\ \mathbf{0}_{1\times3} \end{bmatrix}^{\mathsf{T}} H_L^C \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{D.3}$$

$$Y_i = \begin{bmatrix} u & v & 1 \end{bmatrix}^{\mathsf{T}} = \begin{bmatrix} \frac{u'}{w'} & \frac{v'}{w'} & 1 \end{bmatrix}^{\mathsf{T}}, \tag{D.4}$$

where (D.2) includes the camera's intrinsic parameters (K) and the extrinsic parameters (R_L^C, T_L^C).

[1] More information about the projection map, we refer the readers to " Multiple view geometry in computer vision second edition," by Hartley Richard and Zisserman Andrew, and "Computer vision: a modern approach," by Forsyth David A and Ponce Jean.

For later use, we combine (D.2) and (D.4) to define

$$\Pi\left(X_i; R, t\right) := Y_i, \tag{D.5}$$

the projection map from LiDAR coordinates to image coordinates. Note that it is a function of both the extrinsic variables and the LiDAR vertices.