# Numerical Methods for Data Science

David Bindel

# Table of contents

**Preface** 5

**1 Introduction** 7

1.1 Overview and philosophy . . . 7
1.2 Readings . . . 8
1.2.1 General Numerics . . . 8
1.2.2 Numerical Linear Algebra . . . 9
1.2.3 Numerical Optimization . . . 9
1.2.4 Machine Learning and Statistics . . . 9
1.2.5 Math Background . . . 10

**I Background Plus a Bit** 11

**2 Linear Algebra Background** 13

2.1 Vector spaces . . . 13
2.2 Examples . . . 14
Dual spaces . . . 14
Concrete spaces . . . 15
Matrix spaces . . . 15
Polynomial spaces . . . 16
Linear maps . . . 18
$k$-times differentiable functions . . . 18
2.3 Quasimatrices . . . 18
2.4 Bases . . . 19
Example: Polynomial bases . . . 20
2.5 Norms . . . 22
Norms on concrete spaces . . . 22
Norms on polynomials . . . 23
Consistency and induced norms . . . 23
2.6 Inner products . . . 25
Inner products on concrete spaces . . . 26
Inner products on polynomials . . . 26
Gram-Schmidt . . . 27
2.7 Maps and matrices . . . 28

2.8 Block matrices . . . 29
2.9 Schur complements . . . 30
2.10 The canonical forms . . . 30
Linear maps . . . 31
Linear operators . . . 32
Quadratic forms . . . 33
Other functions . . . 34
2.11 Important invariants . . . 34
Invariant norms . . . 34
Stable rank . . . 35
Spectral invariants . . . 35

**3 Calculus, Optimization, Analysis 38**
3.1 Multivariate differentiation . . . 38
Chain rule . . . 39
Implicit functions . . . 40
3.2 Taylor approximation . . . 41
Single variable . . . 41
Multivariable case . . . 42
Finite differencing . . . 42
Matrix series . . . 43
Neumann series . . . 43
3.3 Optimization . . . 44
Derivative tests . . . 44
Equality constraints . . . 45
KKT conditions . . . 46
Physical interpretation . . . 46
Convexity . . . 47
3.4 Metric spaces . . . 48
3.5 Lipschitz constants . . . 49
3.6 Contraction mappings . . . 49
Banach fixed point theorem . . . 49
Local convergence . . . 50
Preventing escape . . . 51

**4 Probability Background 52**
4.1 Probability basics . . . 52
4.2 The Monte Carlo idea . . . 53
4.3 Examples . . . 54
4.4 Random number generation . . . 55
4.4.1 Bernoulli random variables . . . 55
4.4.2 Exponential random variables . . . 55
4.4.3 Sampling from an empirical distribution . . . 56

4.4.4 Sampling from the unit disk . . . 56
4.4.5 Distribution with an exponential tail . . . 57
4.5 Variance reduction . . . 58
4.5.1 Importance sampling . . . 58
4.5.2 Control variates . . . 59
4.5.3 Antithetic variates . . . 60

**5 CS Background** **61**
5.1 Order notation and performance . . . 61
5.2 Graph theory and sparse matrices . . . 61

**6 Error Analysis Basics** **62**
6.1 Floating point . . . 62
6.2 Sensitivity, conditioning, and types of error . . . 62

**7 Julia Fundamentals** **64**
7.1 Building matrices and vectors . . . 64
7.2 Concatenating matrices and vectors . . . 65
7.3 Transpose and rearrangemenent . . . 66
7.4 Submatrices, diagonals, and triangles . . . 66
7.5 Matrix and vector operations . . . 67
7.6 Things best avoided . . . 67

**References** **68**

# Preface

This is an incomplete draft of a text to accompany Cornell CS 6241: Numerical Methods for Data Science. It is being written as we proceed through the Fall 2023 semester, using Quarto as a typesetting system and Julia as the programming language for most of the computational examples. Both the book draft and the course materials are available via GitHub, and I welcome comments via email or pull request.

The course is designed for a target audience of beginning graduate students with a firm foundation in linear algebra, probability and statistics, and multivariable calculus, along with some background in numerical analysis. The focus is on numerical methods, with an eye to how thoughtful design of numerical methods can help us solve problems of data science. I am deliberately vague about what I mean by "data science," but my hope is that students will find this material useful whether they are interested in computational statistics, in data-driven models from science and engineering, or in machine learning. Topically, the course is organized into roughly six units, each of about two weeks:

1. *Least squares and regression*: direct and iterative linear and nonlinear least squares solvers direct randomized approximations and preconditioning; Newton, Gauss-Newton, and IRLS methods for nonlinear problems; regularization; robust regression.
2. *Matrix and tensor data decompositions*: direct methods, iterations, and randomized approximations for SVD and related decomposition methods; nonlinear dimensionality reduction; non-negative matrix factorization; tensor decompositions.
3. *Low-dimensional structure in function approximation*: active subspace / sloppy model approaches to identifying the most relevant parameters in high-dimensional input spaces and model reduction approaches to identifying low-dimensional structure in high-dimensional output spaces.
4. *Kernel interpolation and Gaussian processes*: statistical and deterministic interpretations and error analysis for kernel interpolation; methods for dealing with ill-conditioned kernel systems; and methods for scalable inference and kernel hyper-parameter learning.
5. *Numerical methods for graph data*: implication of different graph structures for linear solvers; graph-based coordinate embedding methods; analysis methods based on matrix functions; computation of centrality measures; and spectral methods for graph partitioning and clustering.
6. *Learning models of dynamics*: system identification and auto-regressive model fitting; Koopman theory; dynamic mode decomposition.

The reader may have noticed that this list of topics is ambitious for a one-semester course even for students with a strong numerical analysis background. In practice, students come to this course from a variety of backgrounds and while they often have some grounding in computational statistics, machine learning, etc, the majority have not had even a semester introductory survey in numerical analysis, let alone a deeper dive. Hence, the long term goal of this work is a textbook with two parts, which I tend to think of as "Numerical Methods *applied to* Data Science" and "Numerical Methods *for* Data Science." The plan is that the first part should correspond to an undergraduate numerical methods survey covering the standard topics, with example applications drawn from data science; and the second part should correspond to more specialized methods. If things go according to plan, the result might be a book with about three semesters worth of material taught, with some identified paths through it that might correspond to reasonable one-semester course plans.

# 1 Introduction

## 1.1 Overview and philosophy

The title of this course is “Numerical Methods for Data Science.” What does that mean? Before we dive into the course technical material, let’s put things into context. I will not attempt to completely define either “numerical methods” or “data science,” but will at least give some thoughts on each.

*Numerical methods* are algorithms that solve problems of continuous mathematics: finding solutions to systems of linear or nonlinear equations, minimizing or maximizing functions, computing approximations to functions, simulating how systems of differential equations evolve in time, and so forth. Numerical methods are used everywhere, and many mathematicians and scientists focus on designing these methods, analyzing their properties, adapting them to work well for specific types of problems, and implementing them to run fast on modern computers. Scientific computing, also called Computational Science and Engineering (CSE), is about applying numerical methods — as well as the algorithms and approaches of discrete mathematics — to solve “real world” problems from some application field. Though different researchers in scientific computing focus on different aspects, they share the interplay between the domain expertise and modeling, mathematical analysis, and efficient computation.

I have read many descriptions of *data science*, and have not been satisfied by any of them. The fashion now is to call oneself a data scientist and (if in a university) perhaps to start a master’s program to train students to call themselves data scientists. There are books and web sites and conferences devoted to data science; SIAM even has a journal on the Mathematics of Data Science. But what is data science, really? Statisticians may claim that data science is a modern rebranding of statistics. Computer scientists may reply that it is all about machine learning[1] and scalable algorithms for large data sets. Experts from various scientific fields might claim the name of data science for work that combines statistics, novel algorithms, and new sources of large scale data like modern telescopes or DNA sequencers. And from my biased perspective, data science sounds a lot like scientific computing!

Though I am uncertain how data science should be defined, I am certain that a foundation of numerical methods should be involved. Moreover, I am certain that advances in data science, broadly construed, will drive research in numerical method design in new and interesting

[1]The statisticians could retort that machine learning is itself a modern rebranding of statistics, with some justification.

directions. In this course, we will explore some of the fundamental numerical methods for optimization, numerical linear algebra, and function approximation, and see the role they play in different styles of data analysis problems that are currently in fashion. In particular, we will spend roughly two weeks each talking about

- Linear algebra and optimization concepts for ML.
- Latent factor models, factorizations, and analysis of matrix data.
- Low-dimensional structure in function approximation.
- Function approximation and kernel methods.
- Numerical methods for graph data analysis.
- Methods for learning models of dynamics.

You will not strictly need to have a prior numerical analysis course for this course, though it will help (the same is true of prior ML coursework). But you should have a good grounding in calculus, linear algebra, and probability, as well as some "mathematical maturity."

## 1.2 Readings

In the next chapter, we give a lightning review of some background material, largely to remind you of things you have forgotten, but also perhaps to fill in some things you may not have seen. Nonetheless, I have never believed it is possible to have too many books, and there are many references that you might find helpful along the way. All the texts mentioned here are either openly available or can be accessed as electronic resources via many university libraries. I note abbreviations for the books where there are actually assigned readings.

### 1.2.1 General Numerics

If you want to refresh your general numerical analysis chops and have fun doing it, I recommend the *Afternotes* books by Pete Stewart. If you would like a more standard text that covers most of the background relevant to this class, you may like Heath's book (expanded for the "SIAM Classics" edition). I was involved in a book on many of the same topics, together with Jonathan Goodman at NYU. O'Leary's book on *Scientific Computing with Case Studies* is probably the closest of the lot to the topics of this course, with particularly relevant case studies. And Higham's *Accuracy and Stability of Numerical Methods* is a magesterial treatment of all manner of error analysis (highly recommended, but perhaps not as a starting point).

- Afternotes on Numerical Analysis and Afternotes Goes to Graduate School, Stewart
- Scientific Computing: An Introductory Survey, Heath
- Principles of Scientific Computing, Bindel and Goodman
- Scientific Computing with Case Studies, O'Leary
- Accuracy and Stability of Numerical Algorithms, Higham

### 1.2.2 Numerical Linear Algebra

I learned numerical linear algebra from Demmel's book, and still tend to go to it as a reference when I think about how to teach. Trefethen and Bau is another popular take, created from when Trefethen taught at Cornell CS. Golub and Van Loan's book on Matrix Computations ought to be on your shelf if you decide to do this stuff professionally, but I also like the depth of coverage in Stewart's *Matrix Algorithms* (in two volumes). And Elden's *Matrix Methods in Data Mining and Pattern Recognition* is one of the closest books I've found to the spirit of this course (or at least part of it).

- ALA: Applied Numerical Linear Algebra, Demmel
- Numerical Linear Algebra, Trefethen and Bau
- Matrix Algorithms, Vol 1 and Matrix Algorithms, Vol 2, Stewart
- Matrix Methods in Data Mining and Pattern Recognition, Elden

### 1.2.3 Numerical Optimization

My go-to book on numerical optimization is Nocedal and Wright, with the book by Gill, Murray, and Wright as a close second (the two Wrights are unrelated). For the particular case of convex optimization, the standard reference is Boyd and Vandeberghe. And given how much of data fitting revolves around linear and nonlinear least squares problems, we also mention an old favorite by Bjorck.

- NO: Numerical Optimization, Nocedal and Wright
- Practical Optimization, Gill, Murray, and Wright
- Convex Optimization, Boyd and Vandenberghe
- Numerical Methods for Least Squares Problems, Bjorck

### 1.2.4 Machine Learning and Statistics

This class is primarily about numerical methods, but the application (to tasks in statistics, data science, and machine learning) is important to the shape of the methods. My favorite book for background in this direction is Hastie, Tribshirani, and Friedman, but the first book I picked up (and one I still think is good) was Bishop. And while you may decide not to read the entirety of Wasserman's book, I highly recommend at least reading the preface, and specifically the "statistics/data mining dictionary".

- ESL: Elements of Statistical Learning, Hastie, Tribshirani, and Friedman
- Pattern Recognition and Machine Learning, Bishop
- All of Statistics, Wasserman

### 1.2.5 Math Background

If you want a quick refresher of "math I thought you knew" and prefer something beyond my own notes, Garrett Thomas's notes on "Mathematics for Machine Learning" are a good start. If you want much, much more math for ML (and CS beyond ML), the book(?) by Gallier and Quaintance will keep you busy for some time.

- Mathematics for Machine Learning, Thomas
- Much more math for CS and ML, Gallier and Quaintance

# Part I

# Background Plus a Bit

For this class, I assume you know the fundamentals of linear algebra, multivariable calculus, and probability. You should also know how to write and debug simple programs in Julia[2], or know enough programming to pick it up. But there are some things you may never have forgotten that you will need for this class, and there are other things that you might not have learned. This section will describe some of these things.

[2]The examples in this book will be in Julia. If you are unfamiliar with Julia but familiar with MATLAB or Octave, you should be able to read most of the code. The syntax may be slightly more mysterious if you primarily program in some other language, but I will generally assume that you have the computational maturity to figure things out.

(People often refer to "mathematical maturity" and mean enough facility with mathematical thinking that a student can follow somewhat advanced material and learn any missing background with minimal assistance. I am using "computational maturity" here to mean the analogous ability to fill in missing background when it comes to coding.)

# 2 Linear Algebra Background

In this class we will mostly consider vector spaces over the reals, though it is sometimes necessary to also consider complex vector spaces. This overview of linear algebra will be incomprehensibly fast for those not already basically familiar with the concepts. Nonetheless, it is probably worth reading even if you have a strong linear algebra background, as some of the concepts we employ (such as the "quasi-matrix" perspective or our take on canonical forms) are not universally taught.

## 2.1 Vector spaces

Vectors are things that we can add together and scale in a way that is consistent with how we usually think about those words. More formally, a vector space $\mathcal{V}$ over a field $\mathbb{F}$ (which we will always take to be the real numbers $\mathbb{R}$ or the complex numbers $\mathbb{C}$) is a set together with a binary operation $+ : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$ (addition) and $\cdot : \mathbb{F} \times \mathcal{V} \to \mathcal{V}$ (scalar multiplication) satisfying the following axioms:

- *Commutativity*: $\forall u, v \in \mathcal{V}, u + v = v + u$
- *Associativity*: $\forall u, v \in \mathcal{V}, (u + v) + w = u + (v + w)$
- *Existence of additivity identity*: $\exists 0 \in \mathcal{V}$ s.t. $\forall u \in \mathcal{V}, u + 0 = u$
- *Existence of additive inverses*: $\forall u \in \mathcal{V}, \exists - u \in \mathcal{V}$ s.t. $u + -u = 0$
- *Scalar multiply compatibility*: $\forall \alpha, \beta \in \mathbb{F}, v \in \mathcal{V}$, $(\alpha\beta)v = \alpha(\beta v)$
- *Scaling by 1*: If $1 \in \mathbb{F}$ is the multiplicative identity, then $\forall v \in \mathcal{V}, 1v = v$
- *Distributivity for scalar addition*: $\forall \alpha, \beta \in \mathbb{F}, u \in \mathcal{V}, (\alpha + \beta)u = \alpha u + \beta u$
- *Distributivity for vector addition*: $\forall \alpha \in \mathbb{F}, u, v \in \mathcal{V}, \alpha(u + v) = \alpha u + \alpha v$

A *subspace* of a vector space is a set $\mathcal{U} \subset \mathcal{V}$ that is closed under addition and scaling, making $\mathcal{U}$ itself a vector space. The *span* of a set of vectors $\mathcal{G} \subset \mathcal{V}$ is the smallest subspace containing $\mathcal{G}$, i.e. everything that can be written as a linear combination of elements of $\mathcal{G}$, i.e.

$$\operatorname{span}(\mathcal{G}) = \{u \in \mathcal{V} : u = \sum_{g \in \mathcal{G}} g c_g, c_g \in \mathbb{F}\}$$

We say $\mathcal{G}$ is a *spanning set* for a subspace $\mathcal{U}$ if the span of $\mathcal{G}$ is $\mathcal{U}$.

Any subspace can be generated as the span of some generating set, but we place special emphasis on the case when every element $\mathcal{U}$ is a *unique* linear combination of $\mathcal{G}$. In this case,

we say the elements of $\mathcal{G}$ are *linearly independent*. A necessary and sufficient condition for linear independence of $\mathcal{G} \subset \mathcal{V}$ is that the vector $0 \in \mathcal{V}$ has the unique representation as a linear combination

$$0 = \sum_{g \in \mathcal{G}} 0g.$$

If a spanning set is linear independent, it also has minimal cardinality.

If $\mathcal{G} = \{v_1, \dots, v_k\}$ is a spanning set of minimal cardinality, then any element $u$ in the generated set $\mathcal{U}$ has a *unique* representation

$$u = \sum_{j=1}^{k} v_j c_j$$

for coefficients $c_j \in \mathbb{F}$. In this case, we say the set $\mathcal{G}$ is *linearly independent*.

A *sum* of two subspaces $\mathcal{U} \subset \mathcal{V}$ and $\mathcal{W} \subset \mathcal{V}$ is the set $\mathcal{U} + \mathcal{W} \equiv \{u + w : u \in \mathcal{U}, w \in \mathcal{W}\}$. We say $\mathcal{U} + \mathcal{W}$ is a *direct sum* if there is a unique decomposition of each element $v \in \mathcal{U} + \mathcal{W}$ as $v = u + w$ for $u \in \mathcal{U}$ and $v \in \mathcal{V}$. The sum $\mathcal{U} + \mathcal{W}$ is a direct sum iff the unique decomposition of 0 is as $0 + 0$. Alternately, we can say that if $\mathcal{G}$ and $\mathcal{H}$ are linearly independent on their own, then $\operatorname{span}(\mathcal{G}) + \operatorname{span}(\mathcal{H})$ is a direct sum iff $\mathcal{G} \cup \mathcal{H}$ is a linearly independent subset of $\mathcal{V}$. When we know a sum of subspaces is a direct sum, we often use the symbol $\oplus$ instead of $+$.

## 2.2 Examples

Some common examples of vector spaces include:

### Dual spaces

For every vector space $\mathcal{V}$ over a field $\mathbb{F}$, there is an associated *dual space*[1] of all[2] linear maps from $\mathcal{V}$ into $\mathbb{F}$

$$\mathcal{V}^* = \{w^* : \mathcal{V} \to \mathbb{F} \text{ s.t. } w^* \text{ is linear}\},$$

i.e. a *dual vector* (also called a *linear functional*) $w^* \in \mathcal{V}^*$ represent a map such that $w^*(\alpha v) = \alpha(w^* v)$ and $w^*(u + v) = w^* u + w^* v$ for all scalars $\alpha \in \mathbb{F}$ and vectors $u, v \in \mathcal{V}$.

[1]When describing dual spaces, Prof. Kahan always used to look sternly over his glasses and pronounce: "Vector spaces are like potato chips; you can never have only one."

[2]We typically restrict our attention to continuous linear maps – or, in the case of a normed vector space, to maps that are bounded. However, this distinction only matters when we are dealing with infinite-dimensional spaces.

## Concrete spaces

Concrete vector spaces are just lists of elements of $\mathbb{F}$ (real or complex numbers for this class), which we conventionally arrange into a column[3]:

$$c = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}.$$

The vector space $\mathbb{F}^m$ is useful both in its own right and because it is the main vector space we represent in the computer. Any finite-dimensional vector space of dimension $n$ can be represented as the image of $\mathbb{F}^n$ under a *basis map*.

A vector in Julia directly represents a vector in $\mathbb{F}^n$ as an array of floating point numbers laid out sequentially in memory. Entries in a vector in Julia are separated by semicolons or commas; if white space is used instead, the vector is interpreted as a row vector (an element of $\mathbb{F}^{1 \times n}$).

```julia
# Example (column) vector
[1.0; 0.0; 2.5]
```

```
3-element Vector{Float64}:
 1.0
 0.0
 2.5
```

```julia
# Example row vector
[1.0  2.0  3.0]
```

```
1×3 Matrix{Float64}:
 1.0  2.0  3.0
```

## Matrix spaces

A matrix $A \in \mathbb{F}^{m \times n}$ is a two-dimensional array of elements of $a_{ij} \in \mathbb{F}$ (real or complex) with row index $i \in [m]$ and column index $j \in [n]$[4]. While matrices can be seen as just a special type of concrete vector space where we use two integer indices rather than one, they are most usefully interpreted as representations of linear maps between vector spaces (or other types of maps discussed later in this chapter).

[3] In statistics, concrete vectors are frequently arranged as rows by default; the column-oriented perspective we use is common in numerical computing.

[4] We use $[n]$ for natural numbers $n$ to denote the index set $\{1, 2, \ldots, n\}$.

A matrix in Julia directly represents a vector in $\mathbb{F}^{m \times n}$ as an array of floating point numbers laid out column-by-column in memory. The same memory can be interpreted as a vector of length $mn$ where the entries are laid out in column-major order.

```
# Example of a 2-by-3 matrix
[1.0 3.0 5.0;
 2.0 4.0 6.0]
```

```
2×3 Matrix{Float64}:
 1.0  3.0  5.0
 2.0  4.0  6.0
```

```
# Flatten a 2-by-3 matrix into a length 6 vector
let
  A = [1.0 3.0 5.0;
       2.0 4.0 6.0]
  A[:]
end
```

```
6-element Vector{Float64}:
 1.0
 2.0
 3.0
 4.0
 5.0
 6.0
```

### Polynomial spaces

The polynomial space $\mathcal{P}_d$ consists of all univariate polynomials of degree at most $d$:

$$\mathcal{P}_d = \left\{ \sum_{j=0}^{d} c_j x^j : c_j \in \mathbb{F} \right\}.$$

Polynomial spaces are very useful in applications, but they are also highly useful pedagogically as examples of familiar finite-dimensional vector spaces other than the concrete spaces $\mathbb{F}^n$.

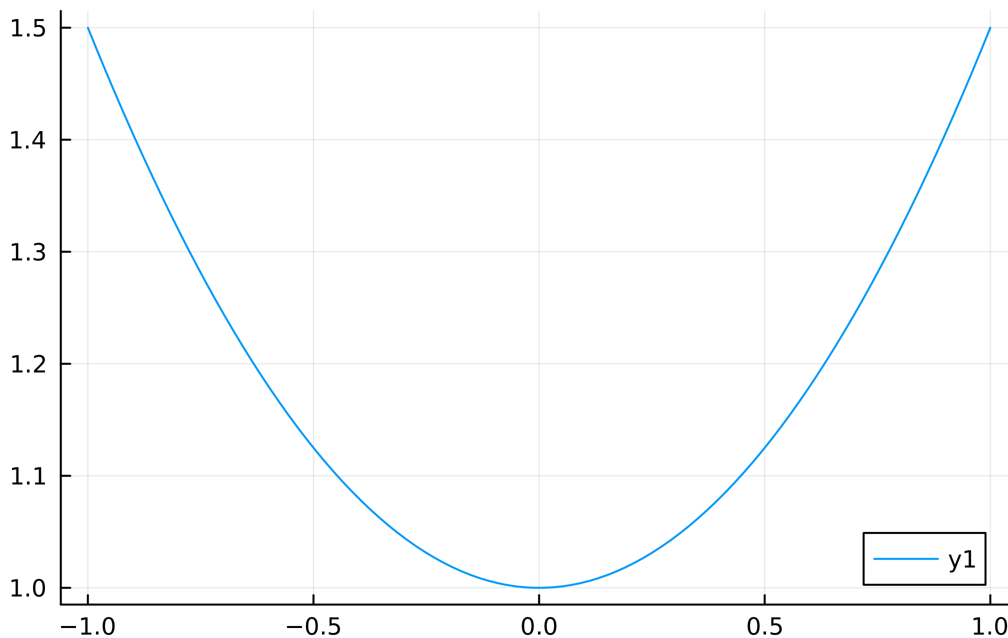
Polynomials in Julia can be represented as objects from Polynomials.jl, a library that directly supports vector algebra with polynomials along with differentiation, integration, and root-finding.

```
let
  p = Polynomial([1, 0, 2])  # Represent 1 + 2x^2
  q = Polynomial([4, 5])     # Represent 4 + 5x
  s = p + q                  # This should represent 5 + 5x + 2x^2
  s, s(2)                    # Show s and evaluation at x = 2
end
```

```
(Polynomial(5 + 5*x + 2*x^2), 23)
```

Alternately, we may choose to write polynomials in Julia directly as functions. In this case, Julia does not know about addition and scalar multiplication, which must be implemented directly. So while we cannot write `q = 2*p`, for example, we are fine defining `q(x) = 2*p(x)`.

```
let
  p(x) = 1.0 + x^2/2
  xs = range(-1.0, 1.0, length=100)
  plot(xs, p.(xs))
end
```



### Linear maps

The space $\mathcal{L}(\mathcal{U}, \mathcal{V})$ is the space of all[5] linear maps between vector spaces $\mathcal{U}$ and $\mathcal{V}$:

$$\mathcal{L}(\mathcal{U}, \mathcal{V}) = \{L : \mathcal{U} \to \mathcal{V} \text{ s.t. } L \text{ is linear}\}.$$

The dual space $\mathcal{V}^* = \mathcal{L}(\mathcal{V}, \mathbb{F})$ is an important special case.

### $k$-times differentiable functions

We will frequently care about the vector space of $k$-times continuously differentiable functions from a domain $\Omega \subset \mathbb{F}^n$ into $\mathbb{F}$. Unlike all our other examples, this is an infinite-dimensional vector space. Finite-dimensional vector spaces have a number of nice properties that infinite-dimensional vector spaces (like $\mathcal{C}^k(\Omega, \mathbb{F})$) often lack. The technical details of infinite-dimensional vector spaces are the topic of functional analysis courses. We will largely elide these details, but there are a few points later in the class where it will be necessary to deal with such annoyances.

## 2.3 Quasimatrices

A column-wise *quasi-matrix* is an ordered list of vectors $v_1, \dots, v_k \in \mathcal{V}$ written as

$$V = \begin{bmatrix} v_1 & \dots & v_k \end{bmatrix}.$$

We refer to the vectors $v_j$ as the *columns* of the quasi-matrix. The primary use of quasi-matrices is to give us a compact notation for writing linear combinations of the columns; for a coefficient vector $c \in \mathbb{F}^k$, we write

$$Vc \equiv \sum_{j=1}^{k} v_j c_j.$$

Following this notation, we will use the symbol $V$ to refer either to the quasi-matrix or to the induced linear mapping from $\mathbb{F}^n$ to $\mathcal{V}$. The range space of $V$ is simply the span of the columns; we will refer to this space as either $\mathcal{R}(V)$ or as $\operatorname{span}(V)$.

In a row-wise quasi-matrix, we write a list of dual vectors $w_1^*, \dots, w_k^* \in \mathcal{V}^*$ as

$$W^* = \begin{bmatrix} w_1^* \\ \vdots \\ w_k^* \end{bmatrix}$$

[5] If $\mathcal{U}$ and $\mathcal{V}$ are normed vector spaces, we restrict our attention to all *bounded* linear maps. This is a distinction that only matters in the infinite dimensional setting, so if you only care about finite-dimensional spaces, you may promptly forget about this footnote.

A row-wise quasi-matrix gives us a compact notation for writing a concrete vector of linear functionals applied to a vector, i.e.

$$c = W^* v \text{ means } c_j = w_j^* v.$$

As with column-wise quasi-matrices, we will refer interchangeably to a row-wise quasi-matrix $W^*$ and the induced linear map from $\mathcal{V} \to \mathbb{F}^k$.

A matrix $A \in \mathbb{F}^{m \times n}$ can be interpreted as either a column-wise quasi-matrix (where the columns of $A$ are viewed as vectors in $\mathbb{F}^m$) or as a row-wise quasi-matrix (where the rows of $A$ are viewed as row vectors in $(\mathbb{F}^n)^*$).

## 2.4 Bases

We deal with vectors from two perspectives:

- *Abstract*: A vector is an object that can be scaled or added to other vectors.
- *Concrete*: A vector is a column of numbers.

We map between the abstract and concrete pictures of (finite-dimensional) vector spaces using a basis.

A *basis* quasi-matrix[6] for a vector space $\mathcal{V}$ is a quasi-matrix $V$ with linearly independent columns that spans $\mathcal{V}$ (the number of columns is the dimension of $\mathcal{V}$). The basis quasi-matrix represents an invertible linear map from $\mathbb{F}^n$ to $\mathcal{V}$. We write $V^{-1}$ to denote the inverse map, which we think of as a row-wise quasi-matrix. The rows in $V^{-1}$ form a basis for the dual space $\mathcal{V}^*$; this basis is called the *dual basis* to $V$. We note that the composition $V^{-1}V$ is an identity map on the concrete space $\mathbb{F}^n$, while $VV^{-1}$ is the identity map on the abstract space $\mathcal{V}$.

When $W$ and $V$ are two separate bases for the same space, the *change of basis matrix* $A = W^{-1}V$ tells us how to transform the coefficients in the $V$ basis into coefficients in the $W$ basis. That is, if we write $v \in \mathcal{V}$ as $v = Vc = Wd$, we have

$$v = Vc \tag{2.1}$$
$$d = W^{-1}v \tag{2.2}$$
$$d = W^{-1}Vc = Ac. \tag{2.3}$$

Using distributivity, we can interpret $A$ column-wise: the columns of $A$ represent the vectors $v_1, \dots, v_n$ written in terms of the $W$ basis

$$d = W^{-1}v = W^{-1}\left(\sum_{j=1}^{n} v_j c_j\right) = \sum_{j=1}^{n} (W^{-1}v_j)c_j = Ac.$$

[6] In most linear algebra texts, we talk about a basis *set* of $\mathcal{V}$, i.e. a linearly independent spanning set (with no particular ordering). Using a quasi-matrix instead of a set just means we also pick an ordering, which we generally do in computation anyhow.

The $A$ matrix must also be invertible, and the matrix $A^{-1} = V^{-1}W$ represents the linear mapping from the $W$ basis coefficients back to the $V$ basis coefficients, and the columns of $A^{-1}$ represent the vectors $w_1, \dots, w_n$ written in terms of the $V$ basis.

### Example: Polynomial bases

For example, the *power basis* of the vector space $\mathcal{P}_d$ of polynomials of degree at most $d$ in one variable is the basis of $d+1$ monomials $\begin{bmatrix} x^0 & x^1 & x^2 & \dots & x^d \end{bmatrix}$. Using this basis, we might concretely represent a polynomial $p(x) = 1 + x^2/2 \in \mathcal{P}_2$ as

$$p(x) = \begin{bmatrix} 1 & x & x^2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0.5 \end{bmatrix}.$$

For numerical purposes, we often prefer the *Chebyshev bases* for $\mathcal{P}_d$, with basis vectors given by

$$T_0(x) = 1 \tag{2.4}$$
$$T_1(x) = x \tag{2.5}$$
$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \text{ for } k \geq 1 \tag{2.6}$$

With respect to this basis, we might concretely represent $p(x) = 1 + x^2/2 \in \mathcal{P}_2$ as

$$p(x) = \begin{bmatrix} T_0(x) & T_1(x) & T_2(x) \end{bmatrix} \begin{bmatrix} 1.25 \\ 0 \\ 0.25 \end{bmatrix}.$$

For a given degree $d$, let us denote the power basis and the Chebyshev bases as

$$P = \begin{bmatrix} x^0 & x^1 & \dots & x^d \end{bmatrix} \tag{2.7}$$
$$T = \begin{bmatrix} T_0(x) & T_1(X) & \dots & T_d(x) \end{bmatrix} \tag{2.8}$$

The three-term recurrence relationship between the Chebyshev polynomials lets us write a simple computation for the matrix representing $P^{-1}T$ mapping from Chebyshev coefficients to coefficients in the power basis.

```
function cheb2power(d)
  A = zeros(d+1, d+1)
  A[1,1] = 1.0               # First column represents T_0 in power basis
  if d > 1 A[2,2] = 1.0 end  # Second column represents T_1 in power basis
  for j = 2:d
    # Compute representation of T_{j+1} = 2*x*T_{j} - T_{j-1}
    A[2:d+1,j+1] = 2*A[1:d,j]  # Multiplication by x shifts power basis coefficients
```

```
    A[:,j+1] -= A[:,j-1]       # Subtract off the T_{j-1} coefficients
  end
  return UpperTriangular(A)
end
```

```
cheb2power (generic function with 1 method)
```

The 3-by-3 matrix representing the map from the Chebyshev coefficients to power coefficients can be computed by `cheb2power(2)`.

```
cheb2power(2)
```

```
3×3 UpperTriangular{Float64, Matrix{Float64}}:
 1.0  0.0  -1.0
  ⋅   1.0   0.0
  ⋅    ⋅    2.0
```

Hence, we have

$$T(x) = P(x) \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

or, reading column-by-column,

$$T_0(x) = x^0 \tag{2.9}$$

$$T_1(x) = x^1 \tag{2.10}$$

$$T_2(x) = 2x^2 - x^0 \tag{2.11}$$

This matrix is *upper triangular*; that is, all entries below the main diagonal are zero. Intuitively, we expect this because $T_j(x)$ is a degree $j$ polynomial, so the maximum monomial involved is $x^j$.

To convert a vector of coefficients in the power basis into a vector of Chebyshev coefficients, we solve a linear system involving $A = P^{-1}T$. We prefer not to form the $A^{-1}$ explicitly; instead, we solve a linear system using the backslash operator.

```
let
  A = cheb2power(2)  # Compute Chebyshev -> power basis coefficient mapping
  A\[1.0; 0.0; 0.5]  # Solve for the Chebyshev coefficients for our example
end
```

```
3-element Vector{Float64}:
 1.25
 0.0
 0.25
```

## 2.5 Norms

A *norm* $\|\cdot\|$ provides a way to measure the lengths of vectors. Norms are characterized by three properties for any scalar $\alpha$ and vectors $u, v$:

**Positive definiteness** $\|v\| \geq 0$; and $\|v\| = 0$ iff $v = 0$
**Homogeneity** $\|\alpha v\| = |\alpha|\|v\|$
**Subadditivity (aka triangle inequality)** $\|u + v\| \leq \|u\| + \|v\|$

All norms are *equivalent* on finite dimensional spaces[7]; that is, if $\|\cdot\|$ and $\|\cdot\|_*$ are two different norms on a finite-dimensional space $\mathcal{V}$, then there exist positive real constants $c, C$ such that for all $v \in \mathcal{V}$

$$c\|v\| \leq \|v\|_* \leq C\|v\|.$$

However, the constants may be rather large!

In normed infinite-dimensional spaces, we often insist that the space be *complete* with respect to the norm – that is, if we have a Cauchy sequence in the space, it must converge. Such spaces are called *Banach spaces*.

### Norms on concrete spaces

The three most common vector norms we work with for the spaces $\mathbb{R}^n$ and $\mathbb{C}^n$ are

**Euclidean norm (aka 2-norm)** $\|v\|_2 = \sqrt{\sum_{j=1}^n |v_j|^2}$
**Max norm (aka sup norm, $\infty$-norm)** $\|v\|_\infty = \max_j |v_j|$
**1-norm** $\|v\|_1 = \sum_{j=1}^n |v_j|$

These norms are all implemented in the Julia LinearAlgebra package:

```
let
   x = [3; 4]
   norm(x,Inf), norm(x,2), norm(x,1)
end
```

[7] On infinite dimensional spaces, norms are *not* all equivalent.

```
(4.0, 5.0, 7.0)
```

Many other norms can be related to one of these three norms. For example, we can frequently connect these norms to other norms by *scaling*: for any norm $\|\cdot\|$ and any invertible matrix $A$, the function $\|\cdot\|_*$ given by $\|v\|_* = \|Av\|$ is also a norm.

### Norms on polynomials

We define similar norms for more general vector spaces. For example, for the polynomial spaces $\mathcal{P}_d$ on a finite interval (say $[-1, 1]$), we have three common norms analogous to the norms on $\mathbb{R}^n$ and $\mathbb{C}^n$:

**Euclidean norm (aka 2-norm)** $\|p\|_2 = \sqrt{\int_{-1}^1 |p(x)|^2 \, dx}$
**Max norm (aka sup norm, $\infty$-norm)** $\|p\|_\infty = \max_{-1 \leq x \leq 1} |p(x)|$
**1-norm** $\|p\|_1 = \int_{-1}^1 |p(x)| \, dx$

It is worth noting that these norms are *not* equivalent to applying the same norms to a vector of coefficients[8]! Unfortunately, the Polynomials.jl library does not implement these norms, though it is not so difficult to do so[9].

### Consistency and induced norms

Suppose $\mathcal{U}$ and $\mathcal{V}$ are normed vector spaces. A norm on $\mathcal{L}(\mathcal{U}, \mathcal{V})$ is *consistent* with the norms on $\mathcal{U}$ and $\mathcal{V}$ (or *submultiplicative*) if for all $u \in \mathcal{U}$ and $L \in \mathcal{L}(\mathcal{U}, \mathcal{V})$,

$$\|Lu\|_\mathcal{V} \leq \|L\| \|u\|_\mathcal{U}.$$

The *induced* norm on $\mathcal{L}(\mathcal{U}, \mathcal{V})$ is the tightest norm such that consistency holds

$$\|L\|_{\mathcal{L}(\mathcal{U},\mathcal{V})} = \sup_{0 \neq u \in \mathcal{U}} \frac{\|Lu\|_\mathcal{V}}{\|u\|_\mathcal{U}} = \sup_{u \in \mathcal{U} : \|u\|_\mathcal{U} = 1} \|Lu\|_\mathcal{V}.$$

A particularly important special case is the induced norm on the dual space $\mathcal{V}^* = \mathcal{L}(\mathcal{V}, \mathbb{F})$ where $\mathcal{V}$ is a field over $\mathbb{F}$.

[8]There is a basis of polynomials (the *normalized Legendre polynomials*) for which applying the $L^2$ norm (the Euclidean norm described here) to a polynomial $p \in \mathcal{P}_d$ is equivalent to applying the $L^2$ norm to the vector of coefficients. But this is a special case.
[9]This is a good exercise for the interested student!

For matrices in $\mathbb{R}^{m\times n}$ and $\mathbb{C}^{m\times n}$, the induced 1-norm and max-norm are simple to compute:

$$\|A\|_\infty = \max_{i\in[m]} \sum_{j=1}^{n} |a_{ij}|; \tag{2.12}$$

$$\|A\|_1 = \max_{j\in[n]} \sum_{i=1}^{m} |a_{ij}|. \tag{2.13}$$

The norm induced by the Euclidean norm on vector spaces (sometimes called the *spectral norm*) is rather more complicated to compute, and is given by the largest singular value of $A$. We describe this below. Fortunately, the *Frobenius norm* is simple to compute and is consistent with the Euclidean norm, even if it is not induced by it. The Frobenius norm is given by

$$\|A\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}.$$

In Julia, the `norm` function computes a vector norm for a vector or for a matrix flattened into a vector. For example:

```
let
  A = [1.0 3.0;
       2.0 4.0]
  # These are equivalent to computing norms of x = [1; 2; 3; 4]
  norm(A,Inf), norm(A,2), norm(A,1)
end
```

```
(4.0, 5.477225575051661, 10.0)
```

These are all legitimate vector norms, but only `norm(A,2)` (which computes the Frobenius norm) even gives a consistent norm. To compute induced norms, we need the `opnorm` function

```
let
  A = [1.0 3.0;
       2.0 4.0]
  # These are equivalent to computing norms of x = [1; 2; 3; 4]
  opnorm(A,Inf), opnorm(A,2), opnorm(A,1)
end
```

```
(6.0, 5.464985704219042, 7.0)
```

## 2.6 Inner products

An *inner product* $\langle \cdot, \cdot \rangle$ is a function from two vectors into the real numbers (or complex numbers for complex vector space). It satisfies the following properties for all vectors $u, v, w$ and scalars $\alpha$

**Positive definiteness** $\langle v, v \rangle \geq 0$ and $\langle v, v \rangle = 0$ iff $v = 0$
**Linearity in the first slot** $\langle u + v, w \rangle = \langle u, w \rangle + \langle v, w \rangle$ and $\langle \alpha u, w \rangle = \alpha \langle u, w \rangle$
**Symmetry (or Hermitian-ness)** $\langle u, v \rangle = \overline{\langle v, u \rangle}$

where the overbar in the last case corresponds to complex conjugation (for complex vector spaces). Every inner product defines a corresponding norm

$$\|v\| = \sqrt{\langle v, v \rangle}.$$

The inner product and associated norm satisfy the *Cauchy-Schwarz* inequality

$$|\langle u, v \rangle| \leq \|u\| \|v\|.$$

In the real case, we are also able to recover the inner product given the norm

$$\langle u, v \rangle = \frac{1}{2}(\|u + v\|^2 - \|u\|^2 - \|v\|^2).$$

In the complex case, this formula only gives the real part of the inner product.

A *Hilbert space* is an inner-product space that is complete under the associated norm (i.e. all Cauchy sequences converge). All finite-dimensional inner-product spaces are Hilbert spaces, as are the infinite-dimensional inner-product spaces that we find most interesting. If $\mathcal{V}$ is a Hilbert space, the *Riesz representation theorem* tells us that every (continuous[10]) linear functional $f \in \mathcal{V}^*$ can be written in terms of an inner product with a unique $w \in \mathcal{V}$:

$$f(v) = \langle v, w \rangle.$$

The map gives us a linear isomorphism (in the real case) or antilinear isomorphism (in the complex case) between $\mathcal{V}$ and $\mathcal{V}^*$.

Where norms give a notion of length, inner products also give a notion of angle. If $u$ and $v$ are nonzero vectors, the angle $\theta$ between them satisfies

$$\cos(\theta) = \frac{\langle u, v \rangle}{\|u\| \|v\|}.$$

Apart from its geometric significance, this "cosine similarity" between vectors plays an important role in many data science and machine learning applications.

[10]All linear functionals are continuous in a finite-dimensional vector space; this makes a difference only in infinite-dimensional spaces.

### Inner products on concrete spaces

The *standard inner product* on $\mathbb{C}^n$ (also called the *dot product*) is

$$\langle x, y \rangle = y^* x = \sum_{j=1}^{n} x_j \bar{y}_j.$$

In this case, the Riesz map from the column vector $y$ to the functional (row vector) $y^*$ is just given by the conjugate transpose operation.

For the standard inner product, we have not only the Cauchy-Schwarz inequality, but also the very useful inequality[11]

$$|\langle x, y \rangle| \leq \|x\|_1 \|y\|_\infty$$

But the standard inner product is not the only inner product, just as the standard Euclidean norm is not the only Euclidean norm! In general, if $M \in \mathbb{C}^{n \times n}$ is Hermitian (or symmetric in the real case) and positive definite (i.e. $x^* M x \geq 0$ with equality iff $x = 0$), then

$$\langle x, y \rangle_M = y^* M x$$

is an inner product. In fact, all possible inner products on $\mathbb{C}^m$ can be written in this way. Alternately, every symmetric positive definite matrix may be written in many ways as $M = A^* A$, giving us a representation in terms of the standard inner product composed with a linear transformation:

$$\langle x, y \rangle_M = (Ax) \cdot (Ay).$$

### Inner products on polynomials

As before, it is useful to consider inner products on $\mathcal{P}_d$ as well as on $\mathbb{R}^n$ and $\mathbb{C}^n$. The standard inner product ($L^2$ inner product) on $\mathcal{P}_d$ over an interval $[-1, 1]$, for example, is given by

$$\langle p, q \rangle = \int_{-1}^{1} p(x) \overline{q(x)} \, dx.$$

In an inner product space, two vectors are said to be orthogonal if their inner product is zero. A set of vectors are *orthonormal* if they are mutually orthogonal and each vector has unit length in the Euclidean norm. Orthonormal bases are particularly convenient. In $\mathbb{R}^n$ (or $\mathbb{C}^n$) with the standard inner product, the standard basis $\begin{bmatrix} e_1 & e_2 & \dots e_n \end{bmatrix}$ (where $e_k$ is the vector of

[11] This is a special case of the Hölder inequality, which states that $|\langle x, y \rangle| \leq \|x\|_p \|y\|_q$ for the so-called $\ell^p$ norms when $1/p + 1/q = 1$.

all zeros except for a one in the $k$th position) is orthonormal. In $\mathcal{P}_d$ with the $L^2$ inner product on $[-1, 1]$, the *Legendre polynomials* form orthogonal bases; these are given by

$$P_0(x) = 1 \tag{2.14}$$
$$P_1(x) = x \tag{2.15}$$
$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x). \tag{2.16}$$

The Legendre polynomials have Euclidean length of

$$\|P_n(x)\| = \sqrt{\frac{2}{2n+1}};$$

the *normalized Legendre polynomials* are scaled to unit length, and so form an orthonormal basis

$$Q_n(x) = \sqrt{\frac{2n+1}{2}} P_n(x).$$

## Gram-Schmidt

Let $X$ be a basis quasi-matrix for an $n$-dimensional space $\mathcal{V}$. The *Gram-Schmidt orthonormalization* process gives us a way of constructing an orthonormal basis $Q$ for $\mathcal{V}$ such that for all $k \leq n$

$$\operatorname{span}(\{x_1, \dots, x_k\}) = \operatorname{span}(\{q_1, \dots, q_k\}).$$

The classical construction is usually written as

$$\tilde{q}_k = x_k - \sum_{j=1}^{k-1} q_j \langle x_k, q_j \rangle \tag{2.17}$$
$$q_k = \tilde{q}_k / \|\tilde{q}_k\|. \tag{2.18}$$

Let $r_{jk} = \langle x_k, q_j \rangle$ for $j < k$ and $r_{kk} = \|\tilde{u}_k\|$; then the Gram-Schimdt construction can be re-interpreted as a factorization of the quasi-matrix $X$:

$$X = QR$$

where $R$ is the upper triangular matrix with coefficients computed in the Gram-Schimdt construction. This factorization is sometimes called the QR factorization of the (quasi)matrix $X$.

For numerical comptuation, the classical Gram-Schmidt process is rarely used. However the QR factorization, computed using different algorithms, is broadly useful.

## 2.7 Maps and matrices

There are a variety of types of maps that are of interest in linear algebra, all of which have matrix representations under appropriate choices of bases. These include

- *Linear maps* between two different vector spaces $\mathcal{U}$ and $\mathcal{V}$.
- *Linear operators* mapping a vector space $\mathcal{V}$ to itself.
- *Bilinear forms* mapping $\mathcal{U} \times \mathcal{V} \rightarrow \mathbb{F}$, which are linear in each argument independently. These are most frequently used when $\mathbb{F} = \mathbb{R}$. An important special case is *symmetric* bilinear forms $a : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{F}$ such that $a(u, v) = a(v, u)$.
- *Sesquilinear forms* mapping $\mathcal{U} \times \mathcal{V} \rightarrow \mathbb{C}$ that are linear in the first argument and antilinear in the second argument. An important special case is *Hermitian* forms $a : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{C}$ such that $a(u, v) = \overline{a(v, u)}$.
- *Quadratic forms* mapping $\mathcal{V} \rightarrow \mathbb{R}$. These are homogeneous of degree 2, i.e. if $\phi : \mathcal{V} \rightarrow \mathbb{R}$ is a quadratic form, then $\phi(\alpha v) = |\alpha|^2 \phi(v)$.

Bilinear forms (over a real space) and sesquilinear forms (over an complex space) can be also be thought of as linear or antilinear maps into the dual space, i.e.

$$v \in \mathcal{V} \mapsto w^* \in \mathcal{U}^* \text{ via } w^* u = a(u, v).$$

In an inner product space, we can define a linear map from $\mathcal{V}$ to $\mathcal{U}^*$ by composing this map with the map from $\mathcal{U}^*$ to $\mathcal{U}$ via the Riesz representation theorem.

Each of these types of linear algebraic objects can be represented as a matrix via a choice of basis. Suppose $U$ and $V$ are basis quasi-matrices for the vector spaces $\mathcal{U}$ and $\mathcal{V}$ with $u = Uc$ and $v = Vd$. Then we have the following matrix representations

- *Linear maps*: If $L : \mathcal{U} \rightarrow \mathcal{V}$ and $v = Lu$ then $d = Ac$ where $A = V^{-1} L U$.
- *Linear operators*: If $L : \mathcal{V} \rightarrow \mathcal{V}$, we have only a single basis, and the matrix representation is $A = V^{-1} L V$.
- *Bilinear forms*: If $a : \mathcal{U} \times \mathcal{V}$ is a bilinear form, then $a(u, v) = d^T A c$ where the matrix element $a_{ij} = a(u_j, v_i)$ is an evaluation of the form on a pair of basis vectors. For symmetric bilinear forms, the matrix representation is also symmetric (i.e. $A = A^T$).
- *Sesquilinear forms*: If $a : \mathcal{U} \times \mathcal{V}$ is a bilinear form, then $a(u, v) = d^* A c$ where the matrix element $a_{ij} = a(u_j, v_i)$ is an evaluation of the form on a pair of basis vectors. For Hermitian bilinear forms, the matrix representation is also Hermitian (i.e. $A = A^*$).
- *Quadratic forms*: If $\phi : \mathcal{V} \rightarrow \mathbb{R}$ is a quadratic form, then $\phi(v) = d^* A d$. The matrix $A$ is Hermitian. The real part of $a_{jk}$ is $(\phi(v_j + v_k) - \phi(v_j) - \phi(v_k))/2$ and the imaginary part is $(\phi(v_j - i v_k) - \phi(v_j) - \phi(v_k))/2$.

Matrix representations of the same linear operator with respect to different bases are said to be *similar*. Changing from the basis $V$ to a basis $VX$ (where $X$ is invertible) transforms the matrix representation $A$ to $X^{-1} A X$; this is called a *similarity transformation*. Similar matrices

have the same eigenvalues, because the eigenvalues are a basis-independent property of linear operators.

Matrix representations of the same quadratic form (or Hermitian sesquilinear form) with respect to different bases are said to be *congruent*. Changing from the basis $V$ to a basis $VX$ (where $X$ again is invertible) transforms the matrix representation $A$ to $X^*AX$; this is called a *congruence transformation*. Congruent matrices share a property called *Sylvester's inertia*, as this is a basis-independent property of quadratic forms. We will have more to say about this shortly in our discussion of canonical forms.

## 2.8 Block matrices

Now suppose that we are interested in a linear mapping $L : \mathcal{U} \to \mathcal{V}$ where the spaces have basis quasi-matrices $U$ and $V$, respectively. Now partition the $U$ and $V$ bases into disjoint sets of columns, written as

$$U = \begin{bmatrix} U_1 & U_2 & \dots & U_q \end{bmatrix} \tag{2.19}$$

$$V = \begin{bmatrix} V_1 & V_2 & \dots & V_q \end{bmatrix}. \tag{2.20}$$

The partitioning of the bases corresponds to a partitioning of the two spaces as a direct sum of subspaces

$$\mathcal{U} = \mathcal{U}_1 \oplus \mathcal{U}_2 \oplus \dots \oplus \mathcal{U}_q \tag{2.21}$$

$$\mathcal{V} = \mathcal{V}_1 \oplus \mathcal{V}_2 \oplus \dots \oplus \mathcal{V}_p \tag{2.22}$$

If $A = V^{-1}LU$ is the matrix representing $L$ with respect to the $U$ and $V$ bases, then together with the partitioning of the bases comes a partitioning of $A$ into blocks:

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1q} \\ A_{21} & A_{22} & \dots & A_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \dots & A_{pq} \end{bmatrix} \tag{2.23}$$

where each submatrix $A_{ij}$ corresponds to the "piece" of the mapping $L$ from the $\mathcal{U}_j$ contribution to $u$ to the $\mathcal{V}_i$ contribution to $Lu$.

We can similarly partition matrices associated with other types of maps. In the case of operators and quadratic forms (or symmetric or Hermitian forms), sensible partitionings of the associated matrices generally yield diagonal blocks.

## 2.9 Schur complements

Suppose $L : \mathcal{U} \to \mathcal{U}$ is an operator on a vector space and $\mathcal{U} = \mathcal{U}_1 \oplus \mathcal{U}_2$ is a decomposition of $\mathcal{U}$ into a direct sum of subspaces. Then we have the block representation of the operator as

$$L = \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{bmatrix}.$$

If $L$ is invertible, we can similarly write the inverse in block form as

$$L^{-1} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

Assuming $L_{11}$ is invertible, we can write the block factorization:

$$L = \begin{bmatrix} I & 0 \\ L_{21}L_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} L_{11} & L_{12} \\ 0 & L_{22} - L_{21}L_{11}^{-1}L_{12} \end{bmatrix},$$

and by forward and backward substitution we have that $M_{22} = S^{-1}$ where

$$S = L_{22} - L_{21}L_{11}^{-1}L_{12}$$

is the *Schur complement* of $L_{11}$ in $L$.

Because they arise naturally in the process of algorithms like Gaussian elimination, Schur complements play an important role in numerical methods for solving linear systems. But Schur complements are also important in various non-numerical settings, such as in Bayesian statistics, where conditioning a multivariate Gaussian prior on linear measurements yields a multivariate Gaussian posterior distribution whose covariance is a Schur complement in the prior covariance.

## 2.10 The canonical forms

We start with abstract vector spaces and functions on them, but we compute with bases and matrices. The matrix associated with a linear algebra function always depends on the choice of basis, and so we ask: what basis would make the matrix as simple as possible? This simplest possible matrix representation is known as a *canonical form*. For computational purposes, we often want to restrict ourselves to orthonormal bases; therefore, for each type of function, we list two flavors of canonical forms – those associated with a general choice of basis and those associated with orthonormal bases. If we start with a matrix representation of some function on concrete vector spaces, we can also think of the canonical forms as *matrix factorizations*.

## Linear maps

Suppose $L : \mathcal{V} \to \mathcal{U}$ is a linear map between two different vector spaces with $\dim(\mathcal{V}) = n$ and $\dim(\mathcal{U}) = m$.

### General bases

Suppose we decompose $\mathcal{U} = \mathcal{U}_1 \oplus \mathcal{U}_2$ where $\mathcal{U}_1 = \text{range}(L)$, and we decompose $\mathcal{V} = \mathcal{V}_1 \oplus \mathcal{V}_2$ where $\mathcal{V}_2 = \text{null}(L)$. Then for any associated choice of bases, we have a block matrix representation of the form

$$\begin{bmatrix} A_{11} & 0 \\ 0 & 0 \end{bmatrix}.$$

where $A_{11} \in \mathbb{C}^{r \times r}$ is an invertible matrix. For appropriate choices of bases, we have the block matrix representation

$$\begin{bmatrix} I_{r \times r} & 0 \\ 0 & 0 \end{bmatrix}.$$

That is, the canonical for a linear map between two different spaces can be described just by one number: the rank $r$ (from which we can also determine the null space dimension $n - r$). We can also write this as the (quasi)matrix factorization

$$L = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^{-1}$$

### Orthonormal bases

If we restrict ourselves to orthonormal bases, we can still get the same general block form by decomposing $U$ into the range space and its orthogonal complement and decomposing $V$ into the null space and its orthogonal complement. However, we cannot get all the way to a representation of the leading block as an identity matrix – the best we can do is to get to a diagonal matrix with positive entries. Using the fact that the inverse of a unitary matrix is given by its conjugate transpose (or the Riesz map of the columns, in the more general case), we have the factorization

$$L = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_{11} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^* = U_1 \Sigma_{11} V_1^*,$$

where the $r$-by-$r$ matrix $\Sigma_{11}$ has diagonal entries $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$. The diagonals of the matrix are known as the *singular values* of the map, with associated bases of *singular vectors*.

## Linear operators

Now suppose $L : \mathcal{V} \rightarrow \mathcal{V}$. This case is different from the linear map because we only get to choose *one* basis, rather than two.

### General bases

Over the complex numbers, we can choose a basis $X$ that usually renders the matrix for $L$ diagonal (and gives us something nearly diagonal otherwise):

$$L = XJX^{-1}, \quad J = \begin{bmatrix} J_{\lambda_1} & & & \\ & J_{\lambda_2} & & \\ & & J_{\lambda_3} & \\ & & & \ddots \end{bmatrix},$$

where the submatrices $J_\lambda$ are *Jordan blocks* of the form

$$J_\lambda = \begin{bmatrix} \lambda & 1 & & & \\ & \lambda & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda & 1 \\ & & & & \lambda \end{bmatrix}.$$

The columns of the basis $X$ are *eigenvectors* or *generalized eigenvectors*. This is known as the *Jordan canonical form*. Most matrices are *diagonalizable*, so that all the Jordan blocks are 1-by-1 and we have a complete basis of eigenvectors.

### Orthonormal bases

Over the complex numbers, we can choose an orthonormal basis $U$ that gives us an upper triangular matrix

$$L = UTU^*,$$

where the diagonal elements $t_{jj}$ are eigenvalues of $L$. This is known as the (complex) *Schur canonical form* of $L$. In this basis, the columns no longer correspond to eigenvectors. However, each prefix of columns $u_1, \ldots u_k$ spans an *invariant subspace* of $L$; that is $L$ maps this space into itself.

If we restrict ourselves to the reals, we have the *real Schur form* of $L$. This is close to the complex Schur form except that we insist on real basis vectors and $T$ is *block* upper triangular, with some 2-by-2 blocks corresponding to complex conjugate pairs of eigenvalues.

## Quadratic forms

Now suppose $\phi : \mathcal{V} \rightarrow \mathbb{R}$ is a quadratic form.

### General bases

For every quadratic form, there is a basis which we can write in partitioned form as

$$V = \begin{bmatrix} V_+ & V_- & V_0 \end{bmatrix}$$

such that we have the block matrix representation

$$\phi(Vc) = \begin{bmatrix} c_+ \\ c_- \\ c_0 \end{bmatrix}^* \begin{bmatrix} I & & \\ & -I & \\ & & 0 \end{bmatrix} \begin{bmatrix} c_+ \\ c_- \\ c_0 \end{bmatrix}$$

Let $\nu_+$, $\nu_-$, and $\nu_0$ be the number of columns in the three parts of the basis. The triple $\nu = (\nu_+, \nu_-, \nu_0)$ is called *Sylvester's inertia* (or sometimes the *metric signature*) for the quadratic form, and characterizes the form in much the same way that the rank characterizes a linear map. Geometrically, Sylvester's inertia describes the number of directions of positive curvature, negative curvature, and zero curvature for the bowl or saddle described by the quadratic form. A quadratic form is *positive definite* if $\nu_+ = n$, *positive semi-definite* if $\nu_+ + \nu_0 = n$, *negative definite* if $\nu_- = n$, and *negative semi-definite* if $\nu - + \nu_0 = n$. A quadratic form with both $\nu_+$ and $\nu -$ nonzero is *strongly indefinite* (also sometimes called a *saddle*).

If $A$ is the representation of $\phi$ under some arbitrary basis $W$ and $W = VX$, we have the factorization

$$A = X^* \begin{bmatrix} I & & \\ & -I & \\ & & 0 \end{bmatrix} X;$$

in this case, we would say $A$ is *congruent* to the diagonal matrix described by the inertia.

### Orthonormal bases

If we restrict our attention to orthonormal bases, the canonical form of the matrix representation is a simple real diagonal matrix:

$$\phi(Vc) = c^* \Lambda c, \quad \Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$$

where the eigenvalues $\lambda_j$ are typically listed in descending order. The number of positive, negative, and zero eigenvalues is given by Sylvester's inertia.

If $A$ is the representation of $\phi$ under some orthonormal basis $W$ and $W = VQ^*$ (where $Q$ is a unitary matrix, i.e. $Q^*Q = I$), then we have the factorization

$$A = Q\Lambda Q^*;$$

in this case, we would say $A$ is related to $\Lambda$ by a *unitary congruence*. A unitary congruence is also a similarity transform, hinting at a rich relation between the interpretation of the symmetric eigenvalue problem in terms of operators or in terms of quadratic forms.

### Other functions

What of bilinear and sesquilinear forms? For these functions, the appropriate canonical forms are essentially the same as those that we have already described. In the case of real symmetric bilinear or complex Hermitian sesquilinear forms, there is an associated quadratic form $\phi(v) = a(v, v)$, and the canonical form of the bilinear/sesquilinear form is the same as the canonical form for the quadratic form. In the case of bilinear or sesquilinear forms on two different spaces, the appropriate canonical form is the same as for a linear map.

## 2.11 Important invariants

The canonical forms of different maps that appear in linear algebra reveal important invariants of the maps that do not depend on the specific matrix representation. These include the rank of a linear map and the inertia of a quadratic form, but also the singular values and the eigenvalues (and their geometric and algebraic multiplicities). However, sometimes we want invariants that are simpler (and maybe easier to compute with than singular values and eigenvalues). We might also want invariants that are continuous under small changes to the map, which the rank, inertia, and eigenvalue multiplicities generally are not. In this section, we review a few additional invariants that see common use.

### Invariant norms

A matrix norm is said to be *unitarily invariant* if for any matrix $A$ and unitary matrices $U$ and $V$ of appropriate dimension, $\|A\| = \|U^*A\| = \|AV\|$. Such matrix norms characterize a linear mapping between inner product spaces, independent of the specific orthonormal basis chosen. Therefore, any unitarily invariant norm on a matrix space must depend only on the singular values of the matrix.

The *Schatten p-norm* on a matrix space is defined in terms of the $p$-norm on the vector of singular values, i.e.

$$\left( \sum_{i=1}^{n} \sigma_i^p \right)^{1/p}.$$

The three most frequently used Schatten norms are:

- $p = 2$: The *Frobenius norm* $\|A\|_F = \sqrt{\sum_{i=1}^n \sigma_i^2}$. The Frobenius norm can also be written as the two-norm of the vector of coefficients, i.e. $\|A\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$.
- $p = \infty$: The *spectral norm* $\|A\|_2 = \sigma_1$. This is the same as the operator norm induced by the vector 2-norm(s).
- $p = 1$: The *nuclear norm* $\|A\|_* = \sum_{j=1}^n \sigma_j$ (also sometimes called the *trace norm*). When $A$ is positive semi-definite, the nuclear norm is equal to $\operatorname{tr}(A) = \sum_i a_{ii}$.

We often use the Frobenius norm to measure the distance between data matrices and the spectral norm when considering the error in approximating a linear map. The nuclear norm appears somewhat less frequently, but plays an important role in methods for finding low-rank solutions to optimization problems over matrix spaces.

The *Ky Fan k-norm* on a matrix spaces is the sum of the largest $k$ singular values. The most frequent examples are $k = 1$ (the spectral norm) and $k = n$ (the nuclear norm).

## Stable rank

There are many times when we would like a low-rank approximate solution to some matrix equation or optimization problem. However, this is computationally awkward, as the rank is not a continuous function of the matrix entries! A useful continuous lower bound on the rank of a matrix is the *stable rank*:

$$\|A\|_F^2 / \|A_2\|^2 = \sum_{j=1}^n \left(\frac{\sigma_i}{\sigma_1}\right)^2.$$

If the matrix $A$ has orthonormal columns (if tall and skinny) or rows (if short and fat), then the stable rank agrees with the rank. Many of the bounds on randomized algorithms for low-rank approximation of matrices are posed in terms of the stable rank.

## Spectral invariants

Unitarily invariant norms and the stable rank can all be phrased in terms of singular values, and so can be viewed as intrinsic properties of an underlying linear map between two inner product spaces. It is also useful to consider *spectral invariants* of an operator from a vector space to itself, which we can express in terms of the eigenvalues.

One invariant that occurs frequently is the *spectral radius*. If $\Lambda(A)$ denotes the spectrum of $A$, then

$$\rho(A) = \max_{\lambda \in \Lambda(A)} |\lambda|.$$

Let $v$ be aany unit-length eigenvector associated with the largest modulus eigenvalue; then for any consistent matrix norm,

$$\|A\| \geq \|Av\| = \|\lambda v\| = |\lambda| = \rho(A).$$

Therefore, the spectral radius is bounded from above by any consistent matrix norm. Conversely, at least in a finite-dimensional space, for any $\epsilon > 0$ there exists a vector space norm such that the associated operator norm satisfies

$$\rho(A) \leq \|A\| \leq \rho(A) + \epsilon.$$

We can actually get equality if no maximal modulus eigenvalue is associated with a nontrivial Jordan block.

Many spectral invariants are expressed via the *characteristic polynomial*

$$p(z) = \prod_{i=1}^{n} (z - \lambda_i)$$

where the $\lambda_i$ are the eigenvalues of the operator (with multiplicity). Written in the monomial basis, the characteristic polynomial is

$$p(z) = z^n - \left( \sum_i \lambda_i \right) z^{n-1} + ... + (-1)^n \prod_i \lambda_i \tag{2.24}$$

$$= z^n - \operatorname{tr}(A) + ... + (-1)^n \det(A). \tag{2.25}$$

The coefficients of the characteristic polynomial are *symmetric polynomials* of the eigenvalues (i.e. polynomials of $n$ variables that are invariant under permutation of the inputs). By far the most important of these are the *trace* $\operatorname{tr}(A)$ and the *determinant* $\det(A)$. Because the eigenvalues of $zI - A$ are equal to $z - \lambda_i$, we can also write the characteristic polynomial in terms of the determinant, i.e. $p(z) = \det(zI - A)$.

The trace and the determinant have a variety of useful properties. The trace is a linear function of the entries of the matrix representation, and can be written as the diagonal sum, i.e.

$$\operatorname{tr}(A) = \sum_i a_{ii}.$$

When the dimensions make sense, the trace is invariant under cyclic permutations of matrix products; that is, if $A \in \mathbb{C}^{m \times n}, B \in \mathbb{C}^{n \times p}, C \in \mathbb{C}^{p \times m}$, we have

$$\operatorname{tr}(ABC) = \sum_{i,j,k} a_{ij} b_{jk} c_{ki} = \sum_{i,j,k} c_{ki} a_{ij} b_{jk} = \operatorname{tr}(CAB)$$

and similarly $\operatorname{tr}(ABC) = \operatorname{tr}(BCA)$. On the matrix space $\mathbb{C}^{m \times n}$, we can define *Frobenius inner product* (the analogue of the standard inner product) in terms of the trace:

$$\langle X, Y \rangle_F = \sum_{i,j} x_{ij} y_{ij}^* = \operatorname{tr}(Y^* X) = \operatorname{tr}(XY^*).$$

The determinant is not linear, but it is a *homomorphism* from operators to the complex numbers, i.e. $\det(AB) = \det(A)\det(B)$ and $\det(I) = 1$. The determinant also satisfies $\det(A^*) = \det(A)^*$. Like the trace, the determinant can be written in terms of the entries of the matrix representation without direct reference to the eigenvalues. Perhaps the most common way that determinants are taught in introductory classes is with the *Laplace formula* (or cofactor expansion)

$$\det(A) = \sum_i (-1)^{i+1} a_{1i} m_{1i}$$

where $m_{1i}$ is the determinant of the $i$th minor (the matrix with column 1 and row $i$ removed). Unfortunately, naively using the Laplace formula to compute derivatives yields an $O(n!)$ time algorithm, and alternate methods are usually used for $n$ larger than two or three.

Using the Laplace expansion, we find that the determinant of an upper triangular matrix $U$ (in which all subdiagonal elements are zero) is equal to the product of the diagonal entries, i.e. $\det(U) = \prod_{j=1}^n u_{jj}$. The determinant of a lower triangular matrix is similarly the product of the diagonal entries. Using these facts, we generally compute determinants[12] by factoring the matrix as a product of triangular or unitary matrices. Geometrically, these factorizations correspond to using volume-preserving transformations (shear transforms or rotations) to transform a parallelipiped defined by the columns of $A$ into an axis-aligned parallelipiped where we can apply the generalization of "base times width times height" formulas from high school geometry.

[12]Determinants are used to represent (signed) volumes, and it is appropriate to compute them in settings like the change of variables formula for integration, or for transformation of probability distribution functions. Most other applications of determinants from linear algebra (Cramer's rule for solving linear systems, computation of determinants to check for singularity, etc) are best avoided for numerical computation – they lead to algorithms that are inefficient, unstable, or both.

# 3 Calculus, Optimization, Analysis

I assume no refresher is needed for most single-variable calculus. But it is useful to have some facts about multi-variable calculus and a few results from mathematical analysis at hand when designing and analyzing numerical methods, and the reader may be forgiven for not remembering all of this in the notation that I find most comfortable.

## 3.1 Multivariate differentiation

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ as a directional derivative (sometimes called a Gateaux derivative) at $x$ in the direction $u$ if $g(s) = f(x + su)$ is differentiable at $s = 0$. In this case, we define the directional derivative

$$\frac{\partial f}{\partial u}(x) = \left.\frac{d}{ds}\right|_{s=0} f(x + su).$$

A function is *Frechet* differentiable if there is a linear function $f'(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$, called the *Frechet derivative* or the *Jacobian*, such that for any direction $u$

$$f(x + su) = f(x) + sf'(x)u + r(s),$$

where the remainder term $r(s)$ is $o(s)$, i.e. $r(s)/s \rightarrow 0$ as $s \rightarrow 0$. A Frechet differentiable function clearly also has Gateaux derivatives in every direction, with

$$\frac{\partial f}{\partial u}(x) = f'(x)u.$$

If we assume an inner product, the *gradient* $\nabla f(x)$ is the unique vector such that

$$\langle u, \nabla f(x) \rangle = f'(x)u.$$

For $\mathbb{C}^n$ with the standard inner product, the gradient is just the conjugate transpose of the derivative, but other inner products give other notions of gradients. The negative gradient vector gives the direction of *steepest descent* with respect to the norm associated with the inner product. In some cases, it is also interesting to consider the direction of steepest descent the respect to other norms than Euclidean norms – e.g. the 1-norm.

The Frechet derivative of a function $f$ may itself be Frechet differentiable. That is, we may have a linear function $f''(x) : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$ such that $f'(x + su) = f'(x) + sf'(x)u + o(s)$. Rather

than thinking of this as a linear map that produces linear maps, we generally think of $f''(x)$ as a *multilinear map* that takes two vectors in $\mathbb{R}^n$ as input and yields a vector in $\mathbb{R}^m$ as output. If $u$ and $v$ are the input arguments, we can write the components of the output of this map as

$$(f''(x)[u,v])_i = \sum_{j,k} f_{i,jk} u_j v_k,$$

where we use the compact "indicial notation"

$$f_{i,jk} \equiv \frac{\partial^2 f_i}{\partial x_j \partial x_k}(x).$$

When the partials are continuous near $x$, they commute, i.e. $f_{i,jk} = f_{i,kj}$. It is sometimes convenient to adopt the *Einstein summation convention* where we assume that repeated indices in a product are meant to be summed, and drop the explicit symbol $\sum_{i,j}$. For a function $\phi : \mathbb{R}^n \to \mathbb{R}$, the *Hessian* is the matrix $H_\phi$ of second partial derivatives $[H_\phi]_{ij} = \phi_{,ij}$. The Hessian matrix is symmetric (or Hermitian, in the complex case) and naturally represents a quadratic form.

A nice notational convention, sometimes called *variational* notation (as in "calculus of variations") is to write

$$\delta f = f'(x)\delta u,$$

where $\delta$ should be interpreted as "first order change in," so that a symbol like $\delta u$ is interpreted as a single object rather than a product of a scalar $\delta$ and the direction $u$. In introductory calculus classes, this sometimes is called a total derivative or total differential, though there one usually uses $d$ rather than $\delta$. There is a good reason for using $\delta$ in the calculus of variations, though, so that's typically what I do.

### Chain rule

The chain rule tells us we can interchange linearization and composition of functions. If $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^p \to \mathbb{R}^n$, then near a given $y = g(x)$ and $z = f(y)$ we have

$$\begin{aligned} f(g(x+su)) &= f(y + sg'(x)u + o(s)) && (3.1)\\ &= z + sf'(y)g'(x)u + o(s). && (3.2) \end{aligned}$$

Using variational notation,

$$\delta z = f'(y)\delta y, \quad \delta y = g'(x)\delta x.$$

or, putting things together,

$$\delta z = f'(y)g'(x)\delta x.$$

When we evaluate the composite function, the dependency between them means we generally first compute $x$, then $y$, then $z$. But associativity makes it easy to also reorder the expression as

$$\delta z = (f'(y)g'(x))\,\delta x,$$

i.e. we can compute the matrix $f'(y)g'(x)$ first, and then multiply by $\delta x$. Because this association proceeds "backwards" from the outputs to the inputs, it is sometimes called "backpropagation."

Another way of writing the same equation is to think of computing the gradient (in the standard inner product):

$$\delta z = \langle \delta x, \nabla(f \circ g) \rangle$$

where

$$\nabla_x(f \circ g) = (f'(y)g'(x))^T = g'(x)^T f'(y)^T = \nabla g(x) \nabla f(y).$$

### Implicit functions

Suppose $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is continuously differentiable, and write the Jacobian in block form as

$$F'(x, y) = \begin{bmatrix} \frac{\partial F}{\partial x} & \frac{\partial F}{\partial y} \end{bmatrix}.$$

The *implicit function theorem* tells us that if $F(x_0, y_0) = 0$ and $\partial F/\partial u$ is nonsingular at $(x_0, y_0)$, then in a neighborhood $\Omega$ containing $y_0$ we can locally define a continuously differentiable function $g : \mathbb{R}^m \to \mathbb{R}^n$ such that

$$x_0 = g(y_0) \text{ and } F(g(y), y) = 0.$$

As long as it is defined, we can differentiate such a $g$ using the chain rule. For conciseness, write $u = (g(y), y)$; then

$$\frac{\partial F}{\partial x}(u) g'(y) + \frac{\partial F}{\partial y}(u) = 0,$$

and so

$$g'(y) = -\left( \frac{\partial F}{\partial x}(u) \right)^{-1} \left( \frac{\partial F}{\partial y}(u) \right).$$

In variational notation, we would usually say that if $x = g(y)$, we have the variational relation

$$\frac{\partial F}{\partial x} \delta x + \frac{\partial F}{\partial y} \delta y = 0.$$

This variational notation often simplifies life, particularly if the arguments to $F$ are really matrices.

As an example of using variational notation to represent differentiation of an implicit function, consider the problem of differentiating $A^{-1}$ with respect to every element of $A$. I would compute this by thinking of the relation between a first-order change to $A^{-1}$ (written $\delta[A^{-1}]$) and a corresponding first-order change to $A$ (written $\delta A$). Using the product rule and differentiating the relation $I = A^{-1}A$, we have

$$0 = \delta[A^{-1}A] = \delta[A^{-1}]A + A^{-1}\delta A.$$

Rearranging a bit gives
$$\delta[A^{-1}] = -A^{-1}[\delta A]A^{-1}.$$
One *can* do this computation element by element, but it's harder to do it without the computation becoming horrible.

## 3.2 Taylor approximation

### Single variable

If $f : \mathbb{R} \to \mathbb{R}$ is differentiable, then by the fundamental theorem of calculus, we have
$$f(x+z) = f(x) + \int_0^z f'(s)\,ds.$$
With two derivatives, we can integrate again to get
$$f(x+z) = f(x) + \int_0^z \left( f'(x) + \int_0^s f''(x+t)\,dt \right) ds \tag{3.3}$$
$$= f(x) + f'(x)z + \int_0^z \int_0^{s_1} f''(x+s_2)\,ds_2\,ds_1 \tag{3.4}$$
Continuing in this manner, for a $k+1$-times differentiable function, we have
$$f(x+z) = \sum_{j=0}^{k} \frac{1}{j!} f^{(j)} z^j + r(z)$$
where
$$r(z) = \int_0^z \int_0^{s_1} \dots \int_0^{s_k} f^{(k+1)}(x+s_{k+1})\,ds_{k+1} \dots ds_2\,ds_1.$$
If $f$ has $k+1$ *continuous* derivatives (i.e. $f \in C^{k+1}$), then we can apply the mean value theorem to write the remainder as
$$r(z) = \frac{1}{(k+1)!} f^{(k+1)}(x+\xi) z^{k+1}$$
for some $\xi \in [x, x+z]$; this is the *Lagrange form* of the remainder. Often, we do not care about writing an exact formula for the remainder, and we will simply write
$$r(z) = o(z^k)$$
if we are not assuming continuity of the $k+1$ derivative, or
$$r(z) = O(z^{k+1})$$
if we do assume continuity. Here we use little $o$ and big $O$ in the order notation sense:

- $r(z) = o(g(z))$ if for any $C > 0$ there is an $\epsilon > 0$ such that for all $|z| < \epsilon$, $|r(z)| \leq Cg(z)$. Put differently, $\lim_{|z|\rightarrow 0} r(z)/g(z) = 0$.
- $r(z) = O(g(z))$ if there is an $\epsilon > 0$ and a constant $C > 0$ such that for all $|z| < \epsilon$, $|r(z)| \leq Cg(z)$.

We most frequently work with simple linear approximations, i.e.

$$f(x+z) = f(x) + f'(x)z + O(z^2),$$

though sometimes we will work with the quadratic approximation

$$f(x+z) = f(x) + f'(x)z + \frac{1}{2}f''(x)z^2 + O(z^3).$$

### Multivariable case

In more than one space dimension, the basic picture of Taylor's theorem remains the same. If $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, then

$$f(x+z) = f(x) + f'(x)z + O(\|z\|^2)$$

where $f'(x) \in \mathbb{R}^{m\times n}$ is the Jacobian matrix at $x$.

If $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$, then

$$\phi(x+z) = \phi(z) + \phi'(x)z + \frac{1}{2}z^T\phi''(z)z + O(\|z\|^3).$$

The row vector $\phi'(x) \in \mathbb{R}^{1\times n}$ is the derivative of $\phi$. A point at which the derivative is zero is a *stationary point*. The Hessian matrix $\phi''(z)$ is the matrix of second partial derivatives of $\phi$. The Hessian represents a quadratic form, and the inertia of the form (the number of positive, negative, and zero eigenvalues) can sometimes be used to tell us if a stationary point represents a local minimum or maximum (the so-called second derivative test).

Low-order Taylor expansions of multivariate functions are notationally nice, and we will rarely need to go beyond them. In the case that we do need to go further, we will use indicial notation with the summation convention, e.g.

$$f_i(x+u) = f_i(x) + f_{i,j}(x)u_j + \frac{1}{2}f_{i,jk}(x)u_ju_k + \frac{1}{6}f_{i,jkl}(x)u_ju_ku_l + \dots.$$

### Finite differencing

Suppose $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is twice continuously differentiable. Then taking the Taylor expansion with remainder

$$f'(x+hu) = f'(x) + hf'(x)u + \frac{h^2}{2}f''(x+\xi u)[u,u],$$

we have that

$$\frac{f(x+hu)-f(x)}{h} = f'(x)u + \frac{h}{2} f''(x+\xi u)[u,u] = f'(x) + O(h).$$

Therefore, we can approximate $f'(x)u$ by a *finite difference*. We can also use a *centered finite difference* for higher order accuracy (assuming continuous third derivatives):

$$\frac{f(x+hu)-f(x-hu)}{2h} = f'(x) + \frac{h^2}{6} f'''(x+\xi u)[u,u,u] = f'(x) + O(h^2).$$

Among other things, finite difference approximations are extremely useful when we want to sanity check an analytical formula for a derivative.

### Matrix series

Let $f : \mathbb{C} \to \mathbb{C}$ be represented near 0 by a power series

$$f(z) = \sum_{j=0}^{\infty} c_j z^j,$$

and suppose that the series converges absolutely for $|z| < \rho_f$. Let $A \in \mathbb{C}^{n\times n}$ be a matrix such that for some consistent norm, $\|A\| < \rho_f$. By consistency of norms, for all $j \geq 0$,

$$\|A^j\| \leq \|A\|^j,$$

and together with the triangle inequality, we have

$$\left\|\sum_{j=m}^{n} c_j A^j\right\| \leq \sum_{j=m}^{n} |c_j| \|A\|^j.$$

Therefore, the partial sums of $\sum_{j=0}^{\infty} c_j A^j$ form a Cauchy sequence in the matrix space, and must converge to some limit $f(A)$, with the bound

$$\|f(A)\| \leq \sum_{j=0}^{\infty} |c_j| \|A\|^j.$$

More broadly, if $f$ converges in an open set containing all the eigenvalues, then $f(A)$ converges.

### Neumann series

We don't need to remember a library of Taylor expansions, but it is useful to remember that for real $|\alpha| < 1$, we have the *geometric* series

$$\sum_{j=0}^{\infty} \alpha^j = (1-\alpha)^{-1}.$$

One of the most important matrix series is the *Neumann series*, which is the matrix generalization of the geometric series. If $\|A\| < 1$, then $I - A$ is invertible and has the convergent Neumann series

$$(I - A)^{-1} = \sum_{j=0}^{\infty} A^j.$$

Using the norm bounds described above, along with convergence of the Neumann series, we have

$$\|(I - A)^{-1}\| \leq \sum_{j=0}^{\infty} \|A\|^j = (1 - \|A\|)^{-1}.$$

## 3.3 Optimization

### Derivative tests

Suppose $\phi : \mathbb{R}^n \to \mathbb{R}$ is differentiable at $x$. Then

$$\phi(x + s\nabla\phi(x)) = \phi(x) + s\|\phi'(x)\|^2 + r(s), \quad r(s) = o(s).$$

If $\phi'(x)$ is nonzero, then there is some $\epsilon > 0$ so that for all $0 < s < \epsilon$,

$$\phi(x - s\nabla\phi(x)) < \phi(x) < \phi(x + s\nabla\phi(x)).$$

Hence, if $\phi'(x)$ is nonzero, then $x$ cannot be either a local minimizer or a local maximizer of $\phi$. Taking the contrapositive: if $\phi$ is Frechet differentiable on all of $\mathbb{R}^n$, any local minimizer or maximizer must occur at a *stationary point*, i.e. a point where the derivative is zero.

Suppose $\phi$ is twice differentiable near $x$, and that $x$ is a stationary point (so $\phi(x) = 0$). Then

$$\phi(x + u) = \phi(x) + \frac{1}{2}u^T\phi''(x)u + o(s^2).$$

Hence, near $x$ the dominant term in the Taylor expansion is the quadratic form associated with the Hessian $\phi''(x)$. Analyzing the Hessian gives us the *second derivative test*:

- When the Hessian is positive definite, the quadratic term is positive for $u \neq 0$, and so the stationary point is a strong local minimum.
- When the Hessian is negative definite, the quadratic term is negative for $u \neq 0$, and so the stationary point is a strong local minimum.
- When the Hessian has both positive and negative eigenvalues (directions of positive and negative curvature), the stationary point is neither a maximum nor a minimum, but a saddle point.
- When the Hessian is positive semidefinite or negative semidefinite, it could be a local minimum or a local maximum – but one needs to check higher order derivatives to be sure.

### Equality constraints

Now suppose we want to minimize $\phi : \mathbb{R}^n \to \mathbb{R}$ over $\Omega \subset \mathbb{R}^n$ defined by equality constraints:

$$\Omega = \{x \in \mathbb{R}^n : \forall i \in [m], c_i(x) = 0\}.$$

for some $c : \mathbb{R}^n \to \mathbb{R}^m$. We assume both $\phi$ and $c$ are continuously differentiable.

If $c'(x)$ is full (row) rank, then there is an $(n-m)$-dimensional space of tangent directions to $\Omega$ at $x$, given by $\{\delta x : c'(x)\delta x = 0\}$. The point $x$ cannot be a minimum or maximum if it obviously increases or decreases in one of these tangent directions; that is, a first-order necessary condition for $x$ to be an extremum is that

$$c'(x)\delta x = 0 \implies \phi'(x)\delta x = 0.$$

The row vectors of $c'(x)$ form a basis for all row vectors that satisfy this condition. That is, in order for $\phi'(x)$ to satisfy the condition, we must be able to uniquely write $\phi'(x)$ as a linear combination of the row vectors of $c'(x)$. Put differently, we require the (unique) coefficients $\lambda$ so that

$$\phi'(x) + \sum_{i=1}^m \lambda_i c_i'(x) = \phi'(x) + \lambda^T c'(x) = 0.$$

The coefficients $\lambda_i$ are known as *Lagrange multipliers*, and we can interpret this linear combination as the gradient of the *Lagrangian* function

$$L(x, \lambda) = \phi(x) + \lambda^T c(x).$$

The stationarity of the Lagrangian gives us the analogue of the first derivative test in the uncase. The analogue of the second derivative test looks at the quadratic form associated with the Hessian $\phi''$ in directions that are consistent with the constraints. That is, suppose $x_*$ is a stationary point for the Lagrangian with full rank $c'(x_*)$, and let $U$ be a basis for the null space of $c'(x_*)$. Then

- We have a strong local minimum if $U^* \phi''(x_*) U$ is positive definite.
- We have a strong local maximum if $U^* \phi''(x_*) U$ is negative definite.
- We have a saddle if $U^* \phi''(x_*) U$ is indefinite.

As before, we need to consider higher derivatives if we want to diagnose the case where $U^* \phi''(x_*) U$ is positive or negative semidefinite.

Working with an explicit null space basis is often inconvenient, particularly for high-dimensional problems with a small number of constraints. In this case, an alternate form of the second derivative test involves looking at the *bordered matrix* which has the block form

$$H = \begin{bmatrix} \phi''(x_*) & c'(x_*)^T \\ c'(x) & 0 \end{bmatrix}.$$

In this setting, the matrix $H$ is always indefinite, but we can write a version of the second derivative test in terms of the inertia:

- We have a strong local minimum if $H$ has $n - m$ positive eigenvalues.
- We have a strong local maximum if $H$ has $n - m$ negative eigenvalues.
- We have a saddle point if $H$ has at fewer than $n - m$ positive eigenvalues and fewer than $n - m$ negative eigenvalues.

## KKT conditions

Now suppose that we seek to optimize $\phi : \Omega \rightarrow \mathbb{R}$ where $\Omega$ is defined by equality and inequality constraints:

$$\Omega = \{x \in \mathbb{R}^n : c_i(x) = 0, i \in \mathcal{E} \text{ and } c_i(x) \leq 0, i \in \mathcal{I}\},$$

where $\mathcal{E}$ and $\mathcal{I}$ are index sets associated with equality and inequality constraints, respectively. Now we define the *augmented Lagrangian*

$$L(x, \lambda, \mu) = \phi(x) + \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \sum_{i \in \mathcal{I}} \mu_i c_i(x).$$

The *Karush-Kuhn-Tucker (KKT) conditions* are first-order conditions for $x_*$ to be a constrained minimizer or maximizer:

$$\begin{aligned} \nabla_x L(x_*) &= 0 & & \\ c_i(x_*) &= 0, \quad i \in \mathcal{E} & & \text{equality constraints} \\ c_i(x_*) &\leq 0, \quad i \in \mathcal{I} & & \text{inequality constraints} \\ \mu_i &\geq 0, \quad i \in \mathcal{I} & & \text{non-negativity of multipliers} \\ c_i(x_*)\mu_i &= 0, \quad i \in \mathcal{I} & & \text{complementary slackness} \end{aligned}$$

The satisfaction of the equality and inequality constraints is also called *primal feasibility*, while the satisfaction of the non-negativity of the multipliers is called *dual feasibility.* We say an inequality constraint is *active* when the associated inequality is actually zero. As with the equality-constrained case, we need a condition on the constraints to avoid degeneracy, sometimes called a *constraint qualification* condition. The most frequently used constraint qualification condition is that the gradient of the active constraint terms should be linearly independent (sometimes known as LICQ: Linearly Independent Constraint Qualification).

The second derivative test in the inequality constrained case is basically the same as the test in the equality constrained case.

## Physical interpretation

A physical picture is often a useful device for remembering the stationarity conditions for optimization problems. In the unconstrained case, we can think about solving a minimization problem by rolling a tiny ball down hill until it came to rest. If we wanted to solve a constrained minimization problem, we could build a wall between the feasible and the infeasible region.

A ball rolling into the wall can roll freely in directions tangent to the wall (or away from the wall) if those directions were downhill; at a constrained miminizer, the force pulling the ball downhill is perfectly balanced against an opposing force pushing into the feasible region in the direction of the normal to the wall. If the feasible region is $\{x : c(x) \leq 0\}$, the normal direction pointing inward at a boundary point $x_*$ s.t.~$c(x_*) = 0$ is proportional to $-\nabla c(x_*)$. Hence, if $x_*$ is a constrained minimum, we expect the sum of the "rolling downhill' ' force ($-\nabla \phi$) and something proportional to $-\nabla c(x_*)$ to be zero:

$$-\nabla \phi(x_*) - \mu \nabla c(x_*) = 0.$$

The Lagrange multiplier $\mu$ in this picture represents the magnitude of the restoring force from the wall balancing the tendency to roll downhill.

## Convexity

A function $\phi : \mathbb{R}^n \to \mathbb{R}$ is *convex* if for any distinct $x, y \in \mathbb{R}^n$ and for all $\alpha \in (0, 1)$

$$\phi\left(\alpha x + (1-\alpha)y\right) \geq \alpha\phi(x) + (1-\alpha)\phi(y).$$

We say $\phi$ is *strictly convex* if the inequality is strict.

A subset $\Omega \subset \mathbb{R}^n$ (or, more generally, a subset of a vector space) is said to be convex if for any $x, y \in \Omega$ and all $\alpha \in (0, 1)$, the points $\alpha x + (1-\alpha)y$ lie in $\Omega$. We say $\Omega$ is strictly convex if the points $\alpha x + (1-\alpha)y$ lie in the interior of $\Omega$. Convex sets are closed under intersection and direct sum.

The definition of a convex set is arguably fundamental than the notion of a convex function, as we often express arguments about the latter in terms of the former via the *epigraph* of the function.
The epigraph (or *supergraph* of a function $\phi : \mathbb{R}^n \to \mathbb{R}$ is the set on or above the graph of *phi*, i.e.

$$\operatorname{epi}(\phi) = \{(x, y) \in \mathbb{R}^n \times \mathbb{R} : y \geq f(x)\}.$$

A function $\phi$ is convex iff the epigraph is a convex set.

If $\Omega$ is a convex set and $x \in \Omega$ is a boundary point, then there is a *supporting hyperplane* at $x$ defined by a functional $w_*$ such that $y \in \Omega \implies w_*(y - x) \geq 0$. In the case of a strictly convex set, the equality can only hold when $y = x$. For functions $\phi$, this means that there is some dual vector $w$ such that for all $z$,

$$\phi(x + z) \geq \phi(x) + w^* z,$$

and for a strictly convex function, equality only holds when $z = 0$. When $\phi$ is differentiable, $w^* = \phi'(x)$. At points where $\phi$ is not differentiable, there will often be several possible choices for $w$. The collection of such choices makes up the *subgradient* at $x$. For convex functions, we

generalize the notion of a stationary point to mean "point $x$ at which the subgradient contains zero, i.e. $\phi(x+z) \geq \phi(x)$."

Convex functions are particularly nice for optimization. A convex function does not need to have a minimum or a maximum on a convex set (for example the exponential function on the real line has neither). But if $\phi$ is convex and we consider optimization over a conves set $\Omega$, then

- an interior point is a minimizer iff it is a stationary point;
- all minimizers are global minimizers;
- the set of all minimizers is convex;

If $\phi$ is strictly convex, then there is at most one global minimizer.

For functions that are twice differentiable, convexity just means that the Hessian matrix is positive semidefinite everywhere. But many important nonsmooth functions are also convex. For example, norms must be convex, but cannot be differentiable.

## 3.4 Metric spaces

A *metric space* is a set $\Omega$ together with a distance function (or metric) $d$ satisfying for all $x, y, z \in \Omega$

- *Symmetry*: $d(x, y) = d(y, x)$
- *Positive definiteness*: $d(x, y) \geq 0$ with equality iff $x = y$
- *Triangle inequality*: $d(x, y) \leq d(x, z) + d(z, y)$

Any normed vector space is also a metric space with the norm $d(x, y) = \|x - y\|$. Also, any subset of a metric space is a metric space. The topology associated with a metric space is analogous to that of the reals: a subset $\mathcal{U} \subset \Omega$ is open if for any $x \in \mathcal{U}$ there is an $\epsilon > 0$ such that the ball $B_\epsilon(u) = \{v \in \Omega : d(v, u) < \epsilon\}$ is contained within $\mathcal{U}$.

A *Cauchy sequence* in a metric space is a sequence $x_1, x_2, \ldots$ such that for any $\epsilon > 0$, points far enough out in the sequence are all within $\epsilon$ of each other (i.e. $\forall \epsilon > 0, \exists N : \forall j, k \geq N, d(x_j, x_k) < \epsilon$). A metric space is *complete* if all Cauchy sequences converge. Any closed subset of a complete metric space is itself complete.

A complete normed vector space is called a *Banach space*. Any finite-dimensional normed vector space over a complete field like $\mathbb{R}$ or $\mathbb{C}$ is a Banach space. nfinite-dimensional normed vector spaces over $\mathbb{R}$ or $\mathbb{C}$ do not always need to be complete, but most infinite-dimensional normed vector spaces we use regularly are complete.

The metric space $\Omega$ is *compact* if any open cover of $\Omega$ has a finite subcover. In the case of finite-dimensional normed vector spaces, any closed and bounded subset is compact (this is not

true in infinite-dimensional normed vector spaces). One reason we care about compactness is that any continuous function on a compact set achieves a minimum an maximum value.

## 3.5 Lipschitz constants

Suppose $f : \Omega \subset \mathcal{U} \rightarrow \mathcal{V}$ is a map between metric spaces. We say $f$ is *Lipschitz* with constant $M$ if for all $x, y \in \Omega$,

$$d(f(x), f(y)) \leq M d(x, y).$$

The concept of Lipschitz continuity is broadly useful in analysis.

If $\mathcal{U}$ and $\mathcal{V}$ are normed vector spaces and $f$ is continuously differentiable on $\Omega$, any bound on $\|f'(x)\|$ over $\Omega$ is a Lipschitz constant (and the tightest Lipschitz constant is $\sup_{x \in \Omega} \|f'(x)\|$). But a function can easily be Lipschitz even if it is not differentiable; for example, the absolute value function on $\mathbb{R}$ is Lipschitz with constant 1.

Having a Lipschitz constant is not as nice as having a derivative. However, we get some of the same nice properties For example, if $f$ and $g$ are Lipschitz functions with constants $M$ and $N$ and the composition $f \circ g$ makes sense, then $f \circ g$ is Lipschitz with constant $MN$. If $f + g$ makes sense, then it is Lipschitz with constant $M + N$. If $f$ and $g$ are Lipschits and bounded and the product $\langle f, g \rangle$ makes sense, then $\langle f, g \rangle$ is Lipschitz with constant $M \max \|g\| + N \max \|f\|$. And if $f$ is $k$-times continuously differentiable and the $k$th derivative has Lipschitz constant $M$, then we have that the residual error in Taylor approximation through the $k$th degree term is bounded by $Mr^{k+1}/(k+1)!$, where $r$ is the distance from the center of the Taylor series.

## 3.6 Contraction mappings

A *contraction mapping* $G : \Omega \rightarrow \Omega$ is a Lipschitz function on a set $\Omega$ with constant $\alpha < 1$. We sat the map $G$ is *locally contractive* near $x$ if it is Lipschitz with constant $\alpha < 1$ in some local neighborhood of $x$. Contraction mappings are a useful tool both for showing the existence and uniqueness of solutions to systems of equations (or optimization problems) and for constructing algorithms to find such solutions.

### Banach fixed point theorem

Assuming $\Omega$ is a closed subset of a Banach space[1], then $G$ has a unique fixed point $x_* \in \Omega$, i.e. a unique point such that $G(x_*) = x_*$. This fact is variously called the *contraction mapping theorem* and the *Banach fixed point theorem*. The proof is interesting because it is a construction

[1]The Banach fixed point theorem applies to any complete metric space. But all the examples in this class will be closed subsets of Banach spaces, so we will stick to that setting.

that can be carried out numerically. Let $x_0 \in \Omega$ be an arbitrary starting point, and consider the *fixed point iteration* $x_{k+1} = G(x_k)$. By contractivity,

$$\|x_{k+1} - x_k\| = \|G(x_k) - G(x_{k-1})\| \leq \alpha \|x_k - x_{k-1}\|,$$

and by induction on this fact,

$$\|x_{k+1} - x_k\| \leq \alpha^k \|x_1 - x_0\|.$$

For any $l > k$, we have

$$\begin{aligned}
\|x_l - x_k\| &= \left\| \sum_{j=k}^{l-1} (x_{j+1} - x_j) \right\| && (3.5)\\
&\leq \sum_{j=k}^{l-1} \|x_{j+1} - x_j\| && (3.6)\\
&\leq \sum_{j=k}^{l-1} \alpha^j \|x_1 - x_0\| && (3.7)\\
&\leq \alpha^k \frac{\|x_1 - x_0\|}{1 - \alpha}. && (3.8)
\end{aligned}$$

Therefore, we have a Cauchy sequence that converges to a limit point $x_*$, which is the fixed point. Uniqueness comes from the fact that if $x_*$ and $x_*'$ are both fixed points in $\Omega$, then

$$\|x_* - x_*'\| = \|G(x_*) - G(x_*')\| \leq \alpha \|x_* - x_*'\|,$$

which implies that $\|x_* - x_*'\| = 0$, so $x_* = x_*'$. Moreover, at any given step $k$, we have the error bound

$$\|x_k - x_*\| \leq \frac{\|x_{k+1} - x_k\|}{1 - \alpha}.$$

## Local convergence

Now suppose that $G$ has a fixed point $x_*$, and $\|G'(x)\| \leq \alpha < 1$ over some closed ball $\bar{B}_\rho(x_*) = \{x : \|x - x_*\| \leq \rho\}$. Then

$$\forall x \in \bar{B}_\rho(x_*), \|G(x) - x_*\| = \|G(x) - G(x_*)\| \leq \alpha \|x - x_*\| < \|x - x_*\|$$

and so $G$ maps the ball into itself. Therefore, $G$ is a contraction mapping on $\bar{B}_\rho(x_*)$, and fixed point iteration from any starting point in that ball will converge to the unique fixed point $x_*$ within the ball.

### Preventing escape

The contraction mapping theorem is useful both for telling us that a fixed point exists and is unique, and for giving us an iteration that converges to that fixed point. But sometimes it is difficult to get *global* contractivity. If we know a fixed point exists, we have just shown that a "local" notion of contractivity around that fixed point is enough. But what if we do not have global contractivity and also are not sure that a fixed point exists? Fortunately, a condition preventing "escape" from a local region of contractivity is sometimes good enough.

For example, suppose $\|G'(x_0)\| \leq \alpha$ and $G'$ is Lipschitz with constant $M$. Consider the fixed point iteration $x_{k+1} = G(x_k)$ starting from $x_0$, and let $d_1 = \|x_1 - x_0\|$. Then if $\alpha' = \alpha + M d_1 < 1$, we can show

- $G$ is Lipschitz with constant $\alpha'$ on a ball of radius $d_1/(1-\alpha')$ about $x_0$.
- By an induction: The iterates satisfy $\|x_k - x_0\| \leq \frac{1-(\alpha')^k}{1-\alpha'} d_1 < d_1/(1-\alpha')$, i.e. the iterates stay in the ball; and therefore they continue to satisfy $\|x_{k+1} - x_k\| \leq (\alpha')^k d_1$.

Therefore in this situation as well, the iterates converge to a fixed point that is at most $d_1/(1-\alpha')$ away from the starting point. As with the contractive mapping theorem, this is enough for us to show that we can show by induction that the iteration remains within a ball of radius $d_1/(1-\alpha')$ around $x_0$, that it converges to some $x_*$ in that ball, and that the convergence is geometric with rate constant $\alpha$.

# 4 Probability Background

Monte Carlo methods involve computations done with the help of random numbers. To reason about Monte Carlo methods, we need a little background in probability theory. I assume that you have seen some probability theory before, and that this is just a reminder. If you need a more thorough refresher, the book by Ross (2014) is a popular introductory text that covers discrete and continuous problems, but not more general probability measures. Another good undergraduate text by Chung and AitSahlia (2003) includes a little bit of measure theory. Good graduate texts include the books by Billingsley (1995) and by Breiman (1992). If you want a reminder that is more thorough than the one we give here, but less than a full textbook, the treatment in (Deisenroth, Faisal, and Ong 2020) is a good starting point.

## 4.1 Probability basics

When we do an experiment, there are a variety of possible outcomes that could result. These outcomes are described in terms of a *sample space* $S$. An *event* is a set $A \subset S$.[1] A probability measure is a function $P$ mapping events to non-negative real numbers such that $P(S) = 1$ and $P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$ for any countable collection of pairwise disjoint events $A_i$.

Rather than work directly with the sample space, we usually consider *random variables*. A random variable $X$ is a function on $S$.[2] In the name of concise notation, we often suppress the argument to $X$, writing expressions like $\{X \in A\}$ to denote $\{s \in S : X(s) \in A\}$.

For a discrete random variable, we write

$$P\{X \in A\} = \sum_{x \in A} p_X(x)$$

where $p_X$ is a *probability mass function* (pmf) which is everywhere between zero and one and which sums to one when the sum is taken over all possible outcomes. Similarly, for a continuous

[1] When the sample space is uncountable, we cannot generally define probabilities for arbitrary subsets of $S$. We therefore require events belong to a *sigma algebra* (also called a *Borel field*) $\mathcal{B}$; this class of sets must contain the empty set and be closed under complement and countable union. A set in the sigma algebra is called a *measurable* set.

[2] A random variable $X : S \to T$ must be *measurable*; that is, if $A$ is a measurable set in $T$, then $\{X \in A\} = \{s \in S : X(s) \in A\}$ must also be measurable.

random variable[3], we write

$$P\{X \in A\} = \int_A f_X(x)\,dx$$

where $f_X(x)$ is a *probability density function* (pdf). When the outcomes are integers or real numbers, we sometimes also care about the *cumulative distribution function* (cdf)

$$F_X(x) = P\{X \leq x\}$$

which we can get by summing the mass function or integrating the density function. The cdf is a monotonically increasing functions with limiting values $\lim_{x \to -\infty} F_X(x) = 0$ and $\lim_{x \to \infty} F_X(x) = 1$.

The *expected value* of a function $g$ of a random variable $X$ is

$$E[X] = \int_\Omega g(x) f_X(x)\,dx;$$

in the discrete case, the integral is replaced by a sum. The variance of $X$ is

$$\mathrm{Var}[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2.$$

The standard deviation is the square root of the variance, and we can think of it as a measure of how far, on average, $X$ is from its expected value.

Random variables $X$ and $Y$ are *independent* if for general choices of events $A$ and $B$ we have $P(\{X \in A\} \cap \{Y \in B\}) = P\{X \in A\} \cdot P\{Y \in B\}$. In simple Monte Carlo calculations, we typically run repeated experiments that are *independent and identically distributed* (i.i.d.). If $X_1, X_2, \ldots, X_N$ are independently drawn from the same distribution (with finite mean and variance), then by the central limit theorem, the sample mean

$$\bar{X} = \frac{1}{N} \sum_{j=1}^{N} X_j$$

is a random variable that is approximately normal (Gaussian) with mean $E[X]$ and variance $\mathrm{Var}[X]/\sqrt{N}$.

## 4.2 The Monte Carlo idea

Monte Carlo methods use random numbers to compute something that is not random. In the abstract, we write some quantity of interest $A$ as

$$A = E_f[V(X)],$$

[3] In order to have a probability density function, we technically want an *absolutely continuous* random variable.

where $X$ is a collection of random variables whose joint distribution is $f$ (sometimes written $X \sim f$) and $V(x)$ is some quantity determined by $X$. A Monte Carlo code generates many samples $X_k$, $k = 1, \dots, N$, from the distribution $f$, and then computes the approximate answer

$$A \approx \hat{A}_N = \frac{1}{N} \sum_{k=1}^{N} V(X_k).$$

If the samples $X_k$ are independent, the error is roughly $\sigma/\sqrt{N}$, where $\sigma^2 = \mathrm{var}_f(V(X))$ is the variance of the random variable $V(X)$. If we don't know the variance of $V(X)$ analytically (which is typically the case), we can use the estimate

$$\hat{\sigma}_N^2 = \frac{1}{N-1} \sum_{k=1}^{N} (V(X_k) - A)^2.$$

Sometimes we're sloppy and divide by $N$; if $N$ is small enough that this makes a significant data, we ideally should run more experiments! When we approximate $A$ by $\hat{A}_N$, we call $\hat{\sigma}_N$ an "error bar", since it describes a measure of the statistical error in our problem (the radius of a symmetric 67% confidence interval). The error bars are not the same as error *bounds*, of course, but they are useful for reasoning about the order of magnitude of the errors we expect to see.

Because statistical error is $O(1/\sqrt{N})$, it tends to be very expensive to get high accuracy with Monte Carlo methods. For some problems, though, particularly those in high dimensions, Monte Carlo methods are the most practical choice. The basic idea of Monte Carlo is simple, if expensive; much of the cleverness in Monte Carlo methods goes into *variance reduction*, which at least reduces the constant in the $O(1/\sqrt{N})$ expression. The good side of statistical error is that it is usually at least possible to estimate its order of magnitude (via error bars).

## 4.3 Examples

Monte Carlo methods have relatively low accuracy compared to deterministic methods, but they are particularly useful in a few cases:

1. Some problems are naturally probabilistic, and a Monte Carlo method may be an almost-direct translation of the problem statement. If we don't mind low accuracy, this can be a very effective way to get a feel for the answer before diving into a more exact calculation (which we might have to spend more time debugging). The standard advice is to only use Monte Carlo for things that cannot be well managed by deterministic methods; this sort of exploratory computation might be an exception.
2. The cost of deterministic methods often grows *exponentially* with the dimension of the ambient space. This causes a problem when we're interested in even moderately high dimensions. For computing integrals in high-dimensional spaces (including the sort of position-and-direction coordinates we need to describe particles in scattering problems like the one in HW 3), a Monte Carlo method is often appropriate.

3. Sometimes we are driven by data, and the data that we have is too huge to process all at once. Sampling the data by Monte Carlo methods can be a very effective approach in this case for the same reason it is effective for high-dimensional problems: the cost depends on the number of samples we draw, and not on the size or dimension of the underlying thing from which our samples are drawn.

## 4.4 Random number generation

In order to run Monte Carlo simulations, we need a source of pseudo-random numbers. One could teach an entire class on how to produce pseudo-random number generators, but we will simply state that it is a tricky business and you should use a well-designed library routine for your day-to-day draws of random bits or of numbers that are uniformly distributed in the interval $[0, 1]$. In Julia, you can use `rand` to get such uniformly distributed random samples (and `randn` to get samples from a standard normal distribution). For our purposes, we simply need to know how to turn such uniform sampling procedures into methods to sample from other distributions. I know a handful of tricks for deriving new samplers; let's investigate them by example.

### 4.4.1 Bernoulli random variables

A Bernoulli random variable generates 1 (success) with probability $p$ and 0 (failure) with probability $1 - p$. In the problem du jour, we implicitly assumed that each question was a Bernoulli trial with $p = 0.8$. Generating a Bernoulli trial from a uniform sampler is relatively simple; in Julia, we might write

```
bernoulli(p) = if rand() < p 1 else 0 end
```

```
bernoulli (generic function with 1 method)
```

Note that $P\{U < p\} = \int_0^p 1 \, du = p$, so this sampler certainly has the right properties. Also notice that we can compute a vector or matrix of Bernoulli trials simultaneously with one call to `rand`, where the arguments give the output size.

### 4.4.2 Exponential random variables

An exponential random variable with rate parameter $\lambda$ has the density function

$$f(x; \lambda) = \lambda e^{-\lambda x}, \quad x \geq 0$$

and the cumulative distribution function

$$F(x; \lambda) = 1 - e^{-\lambda x}.$$

Now, suppose that for a uniform sample $U$ we generate $X$ to satisfy $F(X; \lambda) = U$, i.e.

$$X = -\frac{1}{\lambda} \log(1 - U).$$

Then

$$P\{X \leq x\} = P\{F(X; \lambda) \leq F(x; \lambda)\} = P\{U \leq F(x; \lambda)\} = F(x; \lambda).$$

This inverse transformation trick works whenever we have a simple way to compute a cumulative distribution function. In this particular case, we might also note that $U$ and $1 - U$ have the same distribution, so we could also use

$$X' = -\frac{1}{\lambda} \log(U).$$

```
rand_exp(λ) = -log(rand())/λ
```

```
rand_exp (generic function with 1 method)
```

### 4.4.3 Sampling from an empirical distribution

Suppose we have a histogram of results from some large number of real-world experiments. If the outcomes of the experiments are integers in the range from 1 to $m$, we can define a probability mass function where $p(j)$ is the fraction of the experiments that had outcome $j$. There is a corresponding cumulative distribution function $F(j) = \sum_{i=1}^{j} p(j)$ that goes from $F(0) = 0$ to $F(m) = 1$. To draw a sample from this distribution, we would again use the inverse transformation trick: draw $U$ uniform between 0 and 1, then choose the smallest $j$ such that $F(j) > U$.

### 4.4.4 Sampling from the unit disk

Suppose we want to draw $(X, Y)$ uniformly at random from the interior of the unit circle. One way to do this is with polar coordinates: if $U_1$ and $U_2$ are uniform on $(0, 1)$, we can generate $\Theta = 2\pi U_1$ and $R = \sqrt{U_2}$ (the cdf for $R$ should be $F_R(r) = r^2$ on $[0, 1)$, so we can use the inverse transformation trick from above). Then we could compute $(X, Y) = R(\sin \Theta, \cos \Theta)$. But suppose we didn't know this, or suppose that we're thinking of the disk as a proxy for some more complicated set sitting inside the unit square. What other tactics could we use?

One simple idea is *rejection sampling*. The basic idea is

1. Draw a sample from an easy distribution $g$. In this case, we might use the uniform distribution on $[-1,1]^2$ (i.e. $g(x,y) = 1/4$ on $[-1,1]^2$ and zero elsewhere).
2. Accept the sample with probability that is a function of the sample values. In this case, we have

   $$p(x,y) = \begin{cases} 1, & x^2 + y^2 < 1 \\ 0, & \text{otherwise.} \end{cases}$$

   In this case, we accept with probability one if $X^2 + Y^2 < 1$ and with probability zero otherwise.

We then keep repeating until acceptance. The probability density associated with the accepted values is then

$$f(x,y) = \frac{1}{Z} g(x,y) p(x,y)$$

where $Z$ is some normalization constant chosen so that the acceptance probability is one. In our case, this gives us a density that is a nonzero constant on the circle and zero elsewhere, which is what we wanted.

A more geometric way of seeing rejection sampling is that we fit some shape that completely surrounds the graph of our density function (in this case, that shape is a three-dimensional box). We then draw uniformly at random from within that shape, and discard the samples that do not fall under the graph of the density function. The probability that we succeed in any given trial is equal to the fraction of the area inside the shape that lies underneath the graph of the density function.

### 4.4.5 Distribution with an exponential tail

Let's look at another example of rejection sampling. Suppose I wanted to sample from $f(x) = C^{-1} g(x) e^{-x}$ on $[0,\infty)$, where $C$ is some (possibly unknown) normalization constant and $0 < g(x) < G$. Then I could compute samples from $f$ using the following procedure:

```
function sample_exp_tail(g, gmax)
    p = 0.0
    X = 0.0
    while rand() > p
        X = -log(rand())
        p = g(x)/G
    end
    X
end
```

```
sample_exp_tail (generic function with 1 method)
```

The probability of success in this problem is the ratio of the area under the histogram for $f$ to the area under $Ge^{-x}$, or $1/G$. The expected number of rounds until success is therefore $G$.

## 4.5 Variance reduction

The problem above is a simple example of Monte Carlo integration. We now want to see how to make this simple example more efficient by reducing the variance of the estimator. We will approach this in a few different ways.

### 4.5.1 Importance sampling

Let us consider the computation

$$\sqrt{2\pi} = 2\int_0^\infty \exp(-x^2/2)\,dx.$$

Using the idea of the problem du jour, we could estimate $\sqrt{2\pi}$ by drawing uniform samples on $[0, L]$ for $L$ large enough. But this estimator has rather high variance, and the variance gets larger the larger $L$ is. This is intuitive in that most of the sample points don't really matter to the computation, since $\exp(-x^2/2)$ decays very quickly away from zero.

The integrand $\exp(-x^2/2)$ is largest near the origin, so we get the most contribution to our integral when we have samples near zero. Therefore, it makes sense to use a method that samples more frequently near the origin, rather than sampling uniformly over some large range of $x$ values

$$\sqrt{2\pi} = 2\int_0^\infty \frac{\exp(-x^2/2)}{\exp(-x)} \exp(-x)\,dx = 2E[\exp(-X^2/2)/\exp(-X)]$$

where $X$ is an exponential random variable.

```
function zmean2(N=1000)
    Y = -log.(rand(N))
    fY = exp.(-Y.^2/2)
    gY = exp.(-Y)/2
    lY = fY./gY
    mean(lY), std(lY)/sqrt(N)
end

μ, σ = zmean2()
μ-sqrt(2π), σ
```

```
(-0.0013493446051775493, 0.026315426386953576)
```

### 4.5.2 Control variates

I want to compute the expectation of $l(x) = \exp(x - x^2/2)$, but perhaps I've decided its too hard. But I know that most of the interesting behavior is near the origin, so perhaps I can approximate $l(x)$ by a polynomial over some interval close to zero. Let's try just interpolating by a quadratic at $x = 0$, $x = 1$, and $x = 2$, and discarding everything past $x = 2$:

$$h(x) = \begin{cases} \sqrt{e} - (\sqrt{e} - 1)(x-1)^2, & x \in [0,2] \\ 0, & \text{otherwise} \end{cases}.$$

While $h(X)$ is not identical to $l(X)$, the two random variables surely are correlated. Furthermore, we can compute $E[h(X)]$ analytically; a somewhat tedious calculus exercise yields

$$E[h(X)] = \sqrt{e}(1 - e^{-2}) - (\sqrt{e} - 1)(1 - 5e^{-2}).$$

The fact that $h(X)$ and $l(X)$ should be correlated, together with the fact that we can compute $E[h(X)]$ in closed form, makes $h(X)$ an ideal candidate to serve as a *control variate* with which we can construct a better estimator, as we shall now see.

First, note that

$$E[l(X)] = E[l(X) - ch(X)] + cE[h(X)].$$

So $\hat{l}_c(X) = l(X) - ch(X) + cE[h(X)]$ has the same expected value that $l(X)$ does; but

$$\operatorname{Var}[\hat{l}_c(X)] = \operatorname{Var}[l(X)] - 2c\operatorname{Cov}[l(X), h(X)] + c^2 \operatorname{Var}[h(X)].$$

If we choose $c_* = \operatorname{Cov}[l(X), h(X)]/\operatorname{Var}[h(X)]$, we have

$$\operatorname{Var}[\hat{l}_{c_*}(X)] = \operatorname{Var}[l(X)]\left(1 - \operatorname{corr}[l(X), h(X)]^2\right).$$

If $l(X)$ and $h(X)$ are highly correlated, then $\hat{l}_{c_*}(X)$ may have a much lower variance than $l(X)$. Of course, computing the covariance analytically is hard, but we can always do it numerically.

```
function zmean3(N=1000)
    e   = exp(1.0)
    Y   = -log.(rand(N))
    fY  = exp.(-Y.^2/2)
    gY  = exp.(-Y)/2
    lY  = fY./gY
    hY  = (sqrt(e).-(sqrt(e)-1)*(Y.-1.0).^2) .* (Y.<2)
    EhY = (sqrt(e)*(1-e^-2) - (sqrt(e)-1)*(1-5*e^-2))
    cs  = -sum((lY.-mean(lY)) .* (hY.-EhY))/sum((hY.-EhY).^2)
    W   = lY + cs*(hY.-EhY)
    mean(W), std(W)/sqrt(N)
end
```

```
μ, σ = zmean3()
μ-sqrt(2π), σ
```

```
(-0.0043801682373945106, 0.008297795910048079)
```

### 4.5.3 Antithetic variates

Now let's turn to the problem of computing $\pi/4$ by throwing darts at $[0,1]^2$ and seeing what fraction lie inside the unit circle. Note that if $(X_i, Y_i)$ is a uniform random sample from the square, then $(1-X_i, 1-Y_i)$ is a correlated sample. It turns out that if $\phi$ is the indicator for the unit circle, then $\phi(X_i, Y_i)$ and $\phi(1-X_i, 1-Y_i)$ have negative covariance; this makes sense, since only one of them can be outside the unit circle (though both could be the same). Therefore, the estimator $\phi(X,Y)/2 + \phi(1-X, 1-Y)/2$ actually has lower variance than $\phi(X,Y)$. This is the method of *antithetic variables.*

```
function pi_mc(N=1000)
    XY  = rand(2,N)
    XY2 = 1.0 .- XY
    trials1 = [xyj[1]^2+xyj[2]^2 < 1 for xyj in eachcol(XY) ]
    trials2 = [xyj[1]^2+xyj[2]^2 < 1 for xyj in eachcol(XY2)]
    trials = (trials1+trials2)/2
    mean(trials1), std(trials1)/sqrt(N), mean(trials), std(trials)/sqrt(N)
end

pi_mc()
```

```
(0.752, 0.013663187134877504, 0.762, 0.007900532793325948)
```

# 5 CS Background

## 5.1 Order notation and performance

Just as we use big-O notation in calculus to denote a function (usually an error term) that goes to zero at a controlled rate as the argument goes to zero, we use big-O notation in algorithm analysis to denote a function (usually run time or memory usage) that grows at a controlled rate as the argument goes to infinity. For instance, if we say that computing the dot product of two length $n$ vectors is an $O(n)$ operation, we mean that the time to compute the dot products of length greater than some fixed constant $n_0$ is bounded by $Cn$ for some constant $C$. The point of this sort of analysis is to understand how various algorithms scale with problem size without worrying about all the details of implementation and architecture (which essentially affect the constant $C$).

Most of the major factorizations of *dense* numerical linear algebra take $O(n^3)$ time when applied to square $n \times n$ matrices, though some building blocks (like multiplying a matrix by a vector or scaling a vector) take $O(n^2)$ or $O(n)$ time. We often write the algorithms for factorizations that take $O(n^3)$ time using block matrix notation so that we can build these factorizations from a few well-tuned $O(n^3)$ building blocks, the most important of which is matrix-matrix multiplication.

## 5.2 Graph theory and sparse matrices

In *sparse* linear algebra, we consider matrices that can be represented by fewer than $O(n^2)$ parameters. That might mean most of the elements are zero (e.g.~as in a diagonal matrix), or it might mean that there is some other low-complexity way of representing the matrix (e.g.~the matrix might be a rank-1 matrix that can be represented as an outer product of two length $n$ vectors). We usually reserve the word "sparse" to mean matrices with few nonzeros, but it is important to recognize that there are other *data-sparse* matrices in the world.

The *graph* of a sparse matrix $A \in \mathbb{R}^{N \times N}$ consists of a set of $N$ vertices $\mathcal{V} = \{1, 2, \ldots, N\}$ and a set of edges $\mathcal{E} = \{(i, j) : a_{ij} \neq 0\}$. While the cost of general dense matrix operations usually depends only on the sizes of the matrix involved, the cost of sparse matrix operations can be highly dependent on the structure of the associated graph.

# 6 Error Analysis Basics

## 6.1 Floating point

Most floating point numbers are essentially *normalized scientific notation*, but in binary. A typical normalized number in double precision looks like

$$(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^e$$

where $b_1 \dots b_{52}$ are 52 bits of the *significand* that appear after the binary point. In addition to the normalize representations, IEEE floating point includes subnormal numbers (the most important of which is zero) that cannot be represented in normalized form; $\pm\infty$; and Not-a-Number (NaN), used to represent the result of operations like 0/0.

The rule for floating point is that "basic" operations (addition, subtraction, multiplication, division, and square root) should return the true result, correctly rounded. So a Julia statement

```
# Compute the sum of x and y (assuming they are exact)
z = x + y
```

actually computes $\hat{z} = \text{fl}(x+y)$ where $\text{fl}(\cdot)$ is the operator that maps real numbers to the closest floating point representation. For numbers that are in the normalized range (i.e. for which $\text{fl}(z)$ is a normalized floating point number), the relative error in approximating $z$ by $\text{fl}(z)$ is smaller in magnitude than machine epsilon; for double precision, $\epsilon_{\text{mach}} = 2^{-53} \approx 1.1 \times 10^{-16}$; that is,

$$\hat{z} = z(1+\delta), \quad |\delta| \leq \epsilon_{\text{mach}}.$$

We can *model* the effects of roundoff on a computation by writing a separate $\delta$ term for each arithmetic operation in Julia; this is both incomplete (because it doesn't handle non-normalized numbers properly) and imprecise (because there is more structure to the errors than just the bound of machine epsilon). Nonetheless, this is a useful way to reason about roundoff when such reasoning is needed.

## 6.2 Sensitivity, conditioning, and types of error

In almost every sort of numerical computation, we need to think about errors. Errors in numerical computations can come from many different sources, including:

- *Roundoff error* from inexact computer arithmetic.
- *Truncation error* from approximate formulas.
- *Termination of iterations*.
- *Statistical error*.

There are also *model errors* that are related not to how accurately we solve a problem on the computer, but to how accurately the problem we solve models the state of the world.

There are also several different ways we can think about errors. The most obvious is the *forward error*: how close is our approximate answer to the correct answer? One can also look at *backward error*: what is the smallest perturbation to the problem such that our approximation is the true answer? Or there is *residual error*: how much do we fail to satisfy the defining equations?

For each type of error, we have to decide whether we want to look at the *absolute* error or the *relative* error. For vector quantities, we generally want the *normwise* absolute or relative error, but often it's critical to choose norms wisely. The *condition number* for a problem is the relation between relative errors in the input (e.g. the right hand side in a linear system of equations) and relative errors in the output (e.g. the solution to a linear system of equations). Typically, we analyze the effect of roundoff on numerical methods by showing that the method in floating point is *backward stable* (i.e. the effect of roundoffs lead to an error that is bounded by some polynomial in the problem size times $\epsilon_{\text{mach}}$) and separately trying to show that the problem is *well-conditioned* (i.e. small backward error in the problem inputs translates to small forward error in the problem outputs).

We are often concerned with *first-order* error analysis, i.e. error analysis based on a linearized approximation to the true problem. First-order analysis is often adequate to understand the effect of roundoff error or truncation of certain approximations. It may not always be enough to understand the effect of large statistical fluctuations.

# 7 Julia Fundamentals

Julia is a relatively young language initially released in 2012; the first releases of MATLAB and Python were 1984 and 1991, respectively. It has become increasingly popular for scientific computing and data science types of problems for its speed, simple MATLAB-like array syntax, and support for a variety of programming paradigms. We will provide pointers to some resources for getting started with Julia (or going further with Julia), but here we summarize some useful things to remember for writing concise codes for this class.

## 7.1 Building matrices and vectors

Julia supports general multi-dimensional arrays. Though the behavior can be changed, by default, these use one-based indexing (like MATLAB or Fortran, unlike Python or C/C++). Indexing uses square brackets (unlike MATLAB), e.g.

```
x = v[1]
y = A[1,1]
```

By default, we think of a one-dimensional array as a column vector, and a two-dimensional array as a matrix. We can do standard linear algebra operations like scaling (`2*A`), summing like types of objects (`v1+v2`), and matrix multiplication {`A*v`}.

The expression

```
w = v'
```

represents the adjoint of the vector `v` with respect to the standard inner product (i.e. the conjugate transpose). The tick operator also gives the (conjugate) transpose of a matrix. We note that the tick operator in Julia does not actually copy any storage; it just gives us a re-interpretation of the argument. This shows up, for example, if we write

```
let
  v = [1, 2]  # v is a 2-element Vector{Int64} containing [1, 2]
  w = v'      # w is a 1-2 adjoint(::Vector{Int64}) with eltype Int64
  v[2] = 3    # Now v contains [1, 3] and w is the adjoint [1, 3]'
```

```
end
```

```
3
```

Julia gives us several standard matrix and vector construction functions.

```
Z = zeros(n)   # Length n vector of zeros
Z = zeros(n,n) # n-by-n matrix of zeros
b = rand(n)    # Length n random vector of U[0,1] entries
e = ones(n)    # Length n vector of ones
D = diagm(e)   # Construct a diagonal matrix
e2 = diag(D)   # Extract a matrix diagonal
```

The identity matrix in Julia is simply `I`. This is an abstract matrix with a size that can usually be inferred from context. In the rare cases when you need a *concrete* instantiation of an identity matrix, you can use `Matrix(I, n, n)`.

## 7.2 Concatenating matrices and vectors

In addition to functions for constructing specific types of matrices and vectors, Julia lets us put together matrices and vectors by horizontal and vertical concatenation. This works with matrices just as well as with vectors! Spaces are used for horizontal concatenation and semicolons for vertical concatenation.

```
y = [1; 2]        # Length-2 vector
y = [1 2]         # 1-by-2 matrix
M = [1 2; 3 4]    # 2-by-2 matrix
M = [I A]         # Horizontal matrix concatenation
M = [I; A]        # Vertical matrix concatenation
```

Julia uses commas to separate elements of a list-like data type or an array. So `[1, 2]` and `[1; 2]` give us the same thing (a length 2 vector), but `[I, A]` gives us a list consisting of a uniform scaling object and a matrix — not quite the same as horizontal matrix concatenation.

## 7.3 Transpose and rearrangemenent

Julia lets us rearrange the data inside a matrix or vector in a variety of ways. In addition to the usual transposition operation, we can also do "reshape" operations that let us interpret the same data layout in computer memory in different ways.

```
# Reshape A to a vector, then back to a matrix
# Note: Julia is column-major
avec = reshape(A, prod(size(A)));
A = reshape(avec, n, n)

idx = randperm(n)   # Random permutation of indices (need to use Random)
Ac = A[:,idx]       # Permute columns of A
Ar = A[idx,:]       # Permute rows of A
Ap = A[idx,idx]     # Permute rows and columns
```

## 7.4 Submatrices, diagonals, and triangles

Julia lets us extract specific parts of a matrix, like the diagonal entries or the upper or lower triangle. Some operations make separate copies of the data referenced:

```
A = randn(6,6)    # 6-by-6 random matrix
A[1:3,1:3]        # Leading 3-by-3 submatrix
A[1:2:end,:]      # Rows 1, 3, 5
A[:,3:end]        # Columns 3-6

Ad = diag(A)      # Diagonal of A (as vector)
A1 = diag(A,1)    # First superdiagonal
Au = triu(A)      # Upper triangle
Al = tril(A)      # Lower triangle
```

Other operations give a *view* of the matrix without making a copy of the contents, which can be much faster:

```
A = randn(6,6)          # 6-by-6 random matrix
view(A,1:3,1:3)         # View of leading 3-by-3 submatrix
view(A,:,3:end)         # View of columns 3-6
Au = UpperTriangular(A) # View of upper triangle
Al = LowerTriangular(A) # View of lower triangle
```

## 7.5 Matrix and vector operations

Julia provides a variety of *elementwise* operations as well as linear algebraic operations. To distinguish elementwise multiplication or division from matrix multiplication and linear solves or least squares, we put a dot in front of the elementwise operations.

```
y = d.*x   # Elementwise multiplication of vectors/matrices
y = x./d   # Elementwise division
z = x + y  # Add vectors/matrices
z = x .+ 1 # Add scalar to every element of a vector/matrix

y = A*x    # Matrix times vector
y = x'*A   # Vector times matrix
C = A*B    # Matrix times matrix

# Don't use inv!
x = A\b    # Solve Ax = b *or* least squares
y = b/A    # Solve yA = b or least squares
```

## 7.6 Things best avoided

There are few good reasons to compute explicit matrix inverses or determinants in numerical computations. Julia does provide these operations. But if you find yourself typing `inv` or `det` in Julia, think long and hard. Is there an alternate formulation? Could you use the forward slash or backslash operations for solving a linear system?

# References

Billingsley, Patrick. 1995. *Probability and Measure*. Third. Wiley Series in Probability and Mathematical Statistics. Wiley.

Breiman, Leo. 1992. *Probability*. SIAM. https://doi.org/10.1137/1.9781611971286.

Chung, Kai Lai, and Farid AitSahlia. 2003. *Elementary Probability Theory*. Undergraduate Texts in Mathematics. Springer. https://doi.org/10.1007/978-0-387-21548-8.

Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong. 2020. *Mathematics for Machine Learning*. Cambridge University Press. https://mml-book.com.

Ross, Sheldon. 2014. *A First Course in Probability*. Ninth. Pearson.