MATH 458, NUMERICAL METHODS, FALL 2022

STEVEN HEILMAN

Contents

Date: December 10, 2022

1. Matlab and the NCM Toolbox

Matlab is freely available as a download for USC students. You should download and install this software on your personal computer. Instructions for downloading and installing this software can be found here: https://software.usc.edu/matlab/ . If you have not done so already, you should create a Mathworks account, associated to your USC email address (https://www.mathworks.com/login). Once you have installed Matlab, you should then install the NCM package, available at the bottom of this page:
https://www.mathworks.com/moler/chapters.html.

Once the NCM package is installed, you can access some of its features by just typing `ncmgui` in the Matlab command line, and then pressing Enter.

1.1. **Introduction.** As an introduction to Matlab, let's begin with some basic syntax. Arithmetic operations such as `1+4` or `5-2.5` produce their expected outputs. Multiplication uses the asterisk symbol, so that `3*6` evaluates to 18. Also `6/3` evaluates to 2. Exponents use the `^` symbol, so `2^3` evaluates to 8. The irrational number π is built into Matlab, so that typing `pi` and pressing enter results in

3.1416.

To see more decimal places of π, type `format long`, press enter, then type `pi` again to get

3.141592653589793.

To revert back to a display of only four decimal places, type `format short`.

To see fifty decimal places of π, the command `vpa(pi,50)` returns

3.14159265358979323846264338327950288419716939937510.

Here `vpa` refers to variable precision arithmetic, to be discussed later. For more information on a command such as `pi`, type `help pi` and press enter.

Variables are defined using the equals sign[1]. For example, `x=2` assigns the value 2 to the variable x. With this assignment, `3*x` produces the output 6. Vectors can also be assigned to variable names: `x=[2,3]` or `x=[2 3]` assigns the row vector $(2, 3)$ to the variable name x. Similarly, `x=[2;3]` assigns the column vector $\binom{2}{3}$ to the variable name x. The first entry[2] of a vector x can be accessed with the command `x(1)`. The apostrophe symbol performs a transpose operation. For example, if x is a row vector, then `x'` will be a column vector. A 2×3 matrix can be created with the command `[3 1 2 ; 5 3 6]`.

Matlab syntax streamlines vector and matrix operations. For example, `2*[3,4]` evaluates to `[6,8]`. Component-wise operations can also be done by adding a period symbol as a

[1]Matlab allows you to assign a number value to a variable x, and then assign a vector value to x, and then assign a number value to x. Other programming languages do not allow variables to change types.

[2]Other programming languages might denote the first entry of a vector as its zeroth entry.

prefix to a multiplication or division operation (no such prefix should be used for addition or subtraction). For example:

- `[2,3]+[4,5]` evaluates to `[6,8]`.
- `[6,8]./[2,4]` evaluates to `[3,2]`.
- `[6,8].*[2,4]` evaluates to `[12,32]`.

The last command should not be confused with the dot product of two vectors, such as `[6,8]*[2;4]`, which evaluates to $6 \cdot 2 + 8 \cdot 4 = 12 + 32 = 44$.

One of the first topics in this course will be finding the zeros of single variable functions. For example, the polynomial

$$f(x) = x^2 - x - 1, \qquad \forall\, x \in \mathbb{R}$$

is quadratic with two real zeros. (From the quadratic formula, f as zeros at $\frac{1 \pm \sqrt{1+4}}{2} = \frac{1 \pm \sqrt{5}}{2}$.) Functions are dealt with in Matlab in a few different ways. We can treat f as an anonymous function with the command

```
f = @(x) x.^2 -x-1
```

Then `f(1)` returns -1 and `f(2)` returns 1. Here we used the vectorized multiplication syntax `.^` in defining f since the plotting function

```
ezplot(f)
```

requires that f accept vector inputs. Grid lines can be added to the plot with the `grid on` command. The axes can be defined by e.g. `axis([-2 2 -1.5 0])` The zeros of f can be approximated numerically with the command `fzero(f,1)`. This command searches for a zero of f near 1.

We can also manually choose x-inputs while plotting the function f. For example, if we set `x=linspace(0,1,1000)`, then x is a row vector of length 1000 consisting of 1000 equally spaced points between 0 and 1, inclusive. A nearly identical command `x=0:.001:1` results in a row vector of points starting at 0, increasing by .001 at each index, and ending at 1 (producing a row vector of length 1001.) In either case, we can then plot the values of the function f on these x-inputs with the command

```
plot(x,f(x))
```

With the vector x already defined, we can also plot the function f more directly, without assigning a variable name to the function. For example, we can plot f on the interval $[0, 1]$ with the following command:

```
plot(x,x.^2 -x-1),
```

A function can also be entered using the symbolic toolbox. (To clear any previous variable assignments, use the `clear all` command, and to close all previous plots, use the `close all` command.) To instantiate a symbolic variable x, use the command `syms x`. Then

```
f = x.^2 -x-1
```

defines f as a function of the symbol x. This function can be plotted as before using `ezplot(f)`. However, we can also apply some symbolic operations to f, such as `diff(f)`, resulting in

```
2*x-1,
```

which is a symbolic differentiation of the function f.

Adding a semicolon `;` to the end of a command suppresses its output. For example, `5+6;` will compute $5+6$ but then produce no output.

Built-in functions include: `sin, cos, tan, log, exp`. For-loops use the following syntax:

```
for i=1:10
    i
end
```

This will print the integers from 1 to 10 in increasing order. While-loops use the following syntax:

```
i=1
while i<10
    i=i+1
end
```

Single conditionals use the following syntax:

```
if x<10
    x
else
    x+1
end
```

Multiple conditionals use the following syntax:

```
if x<10
    x
elseif   10<=x<12
    x+1
elseif   12<=x<13
    x+2
else
    x+3
end
```

Exercise 1.1. In Matlab, do the following:

- Perform the following operation, and report the result:
$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + 4 \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}.$$
- Plot the function $f(x) = x^3 + e^x$ for x values in the interval $[0, 3]$.
- Describe the output of the following program.

```
x=1
while x~=0
    x=x/2
end
```

Exercise 1.2. In Matlab, the logical value 0 represents a false statement, and the logical value 1 represents a true statement. For example, `3<5` evaluates to a logical 1, and `5<3` evaluates to a logical 0.

Matlab's logical operations include: `&` for logical and, `|` for logical or, `~` for logical negation. Matlab's relational operations include: `<` for less than, `<=` for less than or equal to, `==` for equality, `~=` for not equality.

- Compute the following expression by hand, and in Matlab:

```
( (2<3) & (4<2) ) | ~(4<8).
```

- Describe the output of the following program.

```
x=1
while (x<5) & ~(x<-5)
    x=x+rand
end
```

- Logical operations also apply to vectors (where 1 denotes true, and 0 denotes false). Compute the following expression by hand, and in Matlab:

```
([ 0 1 0] & [1 1 0] ) | [0 0 1].
```

2. Floating Point Number System

Definition 2.1. The most common number system used on computers is the **double precision floating point** system. This number system includes any number of the form

$$\pm(1.a_1a_2\cdots a_{52})\cdot 2^{b_{11}\cdots b_1-1023} = \pm\Big(1+\sum_{i=1}^{52}2^{-i}a_i\Big)\cdot 2^{\sum_{j=0}^{10}2^j b_{j+1}-1023},$$

where $a_1,\ldots,a_{52},b_1,\ldots,b_{11}\in\{0,1\}$ are binary digits, **and** $b_1,\ldots,b_{11}$ are not all 0 and they are not all 1. Numbers of this form are called **normal numbers**. The 52-bit binary number $.a_1\cdots a_{52}$ is called the **mantissa**, and the 11-bit exponent $b_{11}\cdots b_1-1023$ is called the **exponent** of the floating point number. One bit is need to store the sign (+ or −) for a total of $52+11+1=64$ bits.

In Matlab, the binary representation of $(-1)^c\cdot(1.a_1a_2\cdots a_{52})\cdot 2^{b_{11}\cdots b_1-1023}$ with $c\in\{0,1\}$ is ordered as

$$cb_{11}b_{10}\cdots b_1a_1a_2a_3\cdots a_{52}.$$

Below, we will discuss how the command `format hex` can show the binary representation of a number in Matlab.

The case $b_1=\cdots=b_{11}=0$ has a special meaning, corresponding to **subnormal numbers**. In this case, the corresponding number is

$$\pm(0.a_1a_2\cdots a_{52})\cdot 2^{1-1023}=\pm(0.a_1a_2\cdots a_{52})\cdot 2^{-1022}.$$

(The case $a_1=\cdots=a_{52}=0$ with a positive sign corresponds to 0, and with a negative sign it corresponds to −0. The floating point representations of 0 and −0 are technically different, despite them being formally equal.) The case $b_1\cdots b_{11}=1$ has a special meaning, denoting $\pm\infty$ if $a_i=0$ for all $1\le i\le 52$, or `NaN` (Not a Number) if $a_i\neq 0$ for some $1\le i\le 52$.

Remark 2.2. A normal number has a unique representation as a double precision floating point number.

Here the term "double" signifies that 64 is twice as large as 32. A less precise 32-bit number system, single precision floating point arithmetic, uses a 23 bit mantissa and an 8 bit exponent.

The largest exponent of a double precision floating point number is the binary digit with 11 ones (minus 1), minus 1023, i.e.

$$-1023 - 1 + \sum_{i=1}^{11} 2^{i-1} = -1024 + 2^{11} - 1 = -1024 + 2048 - 1 = 1023.$$

The smallest exponent of a double precision floating point normal number is the number 1, minus 1023, i.e. -1022.

So, the largest double precision floating point number is

$$1.1\cdots1 \cdot 2^{1023} \approx 1.8 \times 10^{308}.$$

This number in Matlab is output from the `realmax` command. We can already see some arithmetic issues with this number. For example, `realmax+1` will be equal to `realmax`. Why? (We will discuss this issue more in Section 2.1.) (To see this try the commands `realmax==realmax+1` or `realmax-(realmax+1)`.) Also, since `realmax<2^1024`, perhaps it is sensible that `2^1024` evaluates to `Inf` in Matlab (Here `Inf` denotes ∞.)

The smallest positive double precision floating point number corresponds to $a_{52} = 1$, and $a_i = 0$ for all $1 \le i \le 51$. In this case,

$$0.0\cdots01 \cdot 2^{-1022} = 2^{-52} \cdot 2^{-1022} = 2^{-1074} \approx 4.941 \times 10^{-324}.$$

Since this is the smallest positive real number, we might worry about e.g. dividing it by 2. Indeed, `2^{-1074}/2` evaluates to zero, as does `2^{-1075}`.

The smallest positive double precision floating point normal number is

$$1 \cdot 2^{1-1023} = 2^{-1022} \approx 2.225 \times 10^{-308}.$$

This number is output from the `realmin` command.

As we have seen from a few examples, arithmetic on computers results in rounding errors. Adding small integers to `realmax` results in a rounding error. And dividing 2^{-1074} by two evaluates to zero, another rounding error. The rounding error for additions close to 1 can be approximated by `eps`, known as machine epsilon. Machine epsilon is defined to be the distance from 1 to the next largest double precision floating point number, which is

$$2^{-52} \approx 2.22 \times 10^{-16}.$$

Consequently, `(1+2^(-53))-1` evaluates to 0. (We will discuss this issue more in Section 2.1.) Note also that the smallest positive subnormal number is `realmin*eps`.

By default, Matlab displays the decimal representation of a float point number, rather than its binary representation. The decimal representation can be viewed with the command `format long`. However, these decimal representations are perhaps a bit deceiving. For example, 1/10 has an exact, infinite decimal representation as

$$\frac{1}{10} = .0001100110011001\ldots = \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{0}{2^{10}} + \frac{0}{2^{11}} + \frac{1}{2^{12}} + \cdots.$$

Rewriting this in base 16, we have

$$\frac{1}{10} = .0001100110011001\ldots = \frac{1}{2^4}\left(1 + \frac{9}{16} + \frac{9}{16^2} + \frac{9}{16^3} + \cdots\right).$$

Since these series are infinite, we cannot write 1/10 exactly as a double precision floating point number. We can only write 1/10 approximately as such a number. It turns out that the closest such number is

$$2^{-4}\Big(1+\frac{9}{16}+\frac{9}{16^2}+\frac{9}{16^3}+\cdots+\frac{9}{16^{12}}+\frac{10}{16^{13}}\Big).$$

(Note that 52 binary digits corresponds to $52/4 = 13$ digits in base 16.) Matlab displays the binary representation of any double precision floating point number with the `format hex` command. Consecutive groups of four binary digits are rewritten as base sixteeen (hexadecimal) digits. In the hexadecimal representation of a number, the letters `a,b,c,d,e,f` correspond to $10, 11, 12, 13, 14, 15$. The hexadecimal representation of 1/10 is then

`3fb999999999999a`

The first three hexadecimal digits `3fb` represent the three-digit number $11+15\cdot 16+3\cdot(16^2) = 1019$. That is, these three digits represent the exponent of the number 1/10, since

$$1019 - 1023 = -4.$$

As anticipated, the remaining thirteen digits `999999999999a` represent the mantissa of the hexadecimal representation of 1/10 described above.

This rounding error can propagate through other operations. For example, `.3/.1` does not evaluate to 3, since the numerator is slightly smaller than .3 and the denominator is slightly larger than .1. In order to see that `.3/.1`, we can either evaluate `.3/.1 <3` or check the hexadecimal representation of `.3/.1` in Matlab. Here are some examples of exact hexadecimal representations of numbers:

Real Number	Matlab Command	Hexadecimal Representation
2^{-1074}	`realmin*eps`	`0000000000000001`
2^{-1022}	`realmin`	`0010000000000000`
2^{-52}	`eps`	`3cb0000000000000`
-2^{-52}	`-eps`	`bcb0000000000000`
$(2-2^{-52})\cdot 2^{1023}$	`realmax`	`7fefffffffffffff`
$-(2-2^{-52})\cdot 2^{1023}$	`-realmax`	`ffefffffffffffff`
0	0	`0000000000000000`
0	-0	`8000000000000000`

To explain the hexadecimal representation of `realmax`, note that the first three digits `7fe` represent an exponent of $7\cdot 16^2 + 15\cdot 16 + 14 = 2046$ and $2046 - 1023 = 1023$. Similarly, in the hexadecimal representation of `eps`, the first three digits `3cb` represent an exponent of $3\cdot 16^2 + 12\cdot 16 + 11 = 971$ and $971 - 1023 = 52$.

The `floatgui` command in the Matlab NCM package is a plot of all positive numbers in a floating point number system where the mantissa uses t bits of storage, and the exponent is bounded between *emin* and *emax*.

Exercise 2.3. Let $\mathcal{F}$ be the set of all positive double precision floating point numbers (except for `NaN`s and `Inf`s), that have the exponent `7fe` (in their hexadecimal representation in Matlab). (For example, after entering the command `format hex` in Matlab, we can see that the number `realmax` is in $\mathcal{F}$, since its hexadecimal representation in Matlab is `7fefffffffffffff`)

- How many elements are in $\mathcal{F}$? That is, what is the cardinality $|\mathcal{F}|$ of $\mathcal{F}$.

- What fraction of elements of $\mathcal{F}$ are in the interval $[2^{1023}, 2^{1024})$?
- What fraction of elements of $\mathcal{F}$ are in the interval $[2^{1023}, \frac{3}{2}2^{1023})$?
- Using e.g. Matlab's `rand` function, write a program that estimates the fraction of $x \in \mathcal{F}$ that satisfy the Matlab expression `x * (1/x)==1`. (It would take a pretty long time to check how many elements of $\mathcal{F}$ satisfy this equation, so you should not do that.)

Warning: Matlab's `rand` function tries to find a uniformly random chosen number in the interval $(0, 1)$ and then round it to the nearest floating point number. This operation is different than choosing a floating point number uniformly over all (positive) floating point numbers with a fixed exponent. (This is the point of the second and third items of this exercise, and the point of the `floatgui` program.) For this reason, your answer to the last part of the question should be much different from the output of the program: `x=rand(1,1000); sum(x.*(1./x)==1)/1000` .

2.1. **Floating Point Arithmetic and Loss of Significance.**

Definition 2.4 (**Floating Point Addition**)**.** Let x, y be positive normal numbers, as defined in Definition 2.1. Then the addition of x and y is defined as follows.

- Represent each of x and y as binary numbers of the form
$$x = (1.a_1a_2\cdots a_{52})\cdot 2^{e_x}, \qquad y = (1.\widetilde{a}_1\widetilde{a}_2\cdots \widetilde{a}_{52})\cdot 2^{e_y}.$$
(Here e_x, e_y are integers and $a_1, \ldots, a_{52}, \widetilde{a}_1, \ldots, \widetilde{a}_{52} \in \{0, 1\}$.)
- Write both x and y using the same exponent. For example, if $e_x \geq e_y$, we write
$$x = (1.a_1a_2\cdots a_{52})\cdot 2^{e_x}, \qquad y = (.0\cdots 01\widetilde{a}_1\widetilde{a}_2\cdots \widetilde{a}_{52})\cdot 2^{e_x}.$$
- Add the digits x and y componentwise with carrying rules. (Since the numbers have the same exponent, you can use the addition and carry rules you learned in grade school.) We then get
$$x + y = (1.c_1\cdots c_k)\cdot 2^{e_x},$$
for some positive integer $k \geq 52$. (In the case $e_x = e_y$, we might need to change the exponent in this step to write $x + y$ itself as a floating point number.)
- In the case $k > 52$, round the result from the previous step to a floating point number such as
$$(1.c_1\cdots c_{52})\cdot 2^{e_x}.$$
(Truncating to 52 decimal places corresponds to "rounding down.") (Matlab will round to the nearest floating point number, and it will round towards zero in case of a tie. For example `1+eps/2` returns 1, `1+eps/1.9` is equal to `1+eps`, and `-1-(eps/2)` returns -1.)

(According to the above definition, we might need to take k very large in order to perform addition. However, Matlab only requires $k \leq 55$ bits to store y during the addition step.)

Example 2.5. Suppose for simplicity we have a floating point arithmetic system in base ten with three digits stored in the mantissa, and we want to add
$$x = 1.312 \times 10^3, \qquad y = 1.929 \times 10^2.$$
We first write
$$x = 1.312 \times 10^3, \qquad y = .1929 \times 10^3.$$

Adding componentwise leads to

$$x + y = 1.5049 \times 10^3.$$

Since the arithmetic system only stores three digits, the final computed value of $x + y$ is the rounded answer

$$1.505 \times 10^3.$$

In certain implementations, extra unnecessary bits might be discarded in the computation described in Definition 2.4, e.g. adding $x = 2^{50}$ and $y = 2^{-50}$ naïvely might require about 100 bits of storage for y when we write both x and y using the same exponent, but such storage is not really needed since the addition of x and y is just x. (In this case, adding y to x does not change the digits of x at all.)

Remark 2.6. Floating point subtraction is defined in a similar way to addition. Multiplication and division are even simpler, since Step 2 of Definition 2.4 is not needed. For example, to multiply, just multiply the mantissas and add the exponents.

Proposition 2.7. *Let x, y be positive normal numbers, as defined in Definition 2.1. Assume that $x + y < 2^{1024}$. Let $\mathrm{fl}(x + y)$ denote the double precision floating point representation of $x + y$. Then there exists $\delta \in \mathbb{R}$ with $|\delta| \leq 2^{-52}$ such that*

$$\mathrm{fl}(x + y) = (x + y)(1 + \delta).$$

Proof. This follows from the last part of Definition 2.4. □

Definition 2.8. Let x be a real number, and let $x^* \in \mathbb{R}$ be a computed value of x (such as $\mathrm{fl}(x)$). We define the **absolute error** of x^* to be

$$|x - x^*|.$$

If $x \neq 0$, we define the **relative error** of x^* to be

$$\frac{|x - x^*|}{|x|}$$

So, Proposition 2.7 says that the relative error of the computation $\mathrm{fl}(x + y)$ relative to $x + y$ is

$$\frac{|\mathrm{fl}(x + y) - (x + y)|}{x + y} \leq 2^{-52},$$

whenever $x, y > 0$ are normal double precision floating number numbers with $x + y < 2^{1024}$. Iterating Proposition 2.7 k times gives

Proposition 2.9. *Let $x_1, \ldots, x_k$ be positive normal numbers, as defined in Definition 2.1. Assume that $\sum_{i=1}^k x_i < 2^{1024}$. Then the relative error of $\mathrm{fl}\left(\sum_{i=1}^k x_i\right)$ relative to $\sum_{i=1}^k x_i$ is at most*

$$(1 + 2^{-52})^{k-1} - 1 \approx (k - 1)2^{-52}.$$

To justify the approximation, note that the binomial theorem implies that

$$(1 + 2^{-52})^{k-1} = \sum_{j=0}^{k-1} \binom{k-1}{j} (2^{-52})^j = 1 + (k - 1)2^{-52} + (1/2)(k - 1)(k - 2)(2^{-52})^2 + \cdots$$

If k is small (say $k < 2^{20}$), then the term $(1/2)(k - 1)(k - 2)(2^{-52})^2$ is much smaller than $(k - 1)2^{-52}$. The same comment applies for the remaining terms in the sum.

Exercise 2.10. Do the following plot in Matlab.

```
x = 0.988:.0001:1.012;
y = x.^7-7*x.^6+21*x.^5-35*x.^4+35*x.^3-21*x.^2+7*x-1;
plot(x,y)
```

This is the function $y(x) = (x-1)^7$ for $x \in [.988, 1.012]$. Does the plot look like a polynomial? Explain why or why not.

Exercise 2.11. Suppose we want to solve the linear system of equations

$$17x_1 + 5x_2 = 22,$$
$$1.7x_1 + .5x_2 = 2.2.$$

Note that $(x_1, x_2) = (1, 1)$ is a solution to this system of equations.

Matlab can numerically solve this system with the following program

```
A = [17 5; 1.7 0.5];
b = [22; 2.2];
x = A\b
```

- What is the solution x that is output from the program?
- Is the output of the program an actual solution of the original system of equations?
- What is the determinant of A? What does Matlab output from the command `det(A)`?

Warning: for a 2×2 matrix A and a scalar $t > 0$, we have $\det(tA) = t^2\det(A)$. So, the value of a determinant does not necessarily say anything about how well we can solve a linear system of equations of the form $Ax = b$.

Exercise 2.12. The sin function, like other special functions such as $\cos, \exp, \log$, etc., cannot be computed exactly on a computer. A common way to compute these special functions is via power series. Recall that sin has the following power series that is absolutely convergent for all $x \in \mathbb{R}$:

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

With this power series in mind, run the following program when $x = \pi/2, 11\pi/2, 21\pi/2$ and $31\pi/2$. (Before you run the program, set x to a specific value.)

```
s = 0;
t = x;
n = 1;
while s+t ~= s;
    s = s + t;
    t =-x.^2/((n+1)*(n+2)).*t;
    n = n + 2;
end
```

When the program terminates, the value of s is the computed value of $\sin(x)$. For each value of x stated above, answer the following:

- What is the absolute error of the computation of $\sin(x)$?
- How many terms of the power series were used in the computation of $\sin(x)$?
- What is the largest term in the power series expansion of $\sin(x)$? (Hint: consider using the `max` command)

2.1.1. *Subtraction and Loss of Significance.* Analogues of Propositions 2.7 and 2.9 hold for multiplication and division. For example

Proposition 2.13. *Let x, y be positive normal numbers, as defined in Definition 2.1. Let $\odot$ denote multiplication or division. Assume that $2^{-1022} < x \odot y < 2^{1024}$. Let $\mathrm{fl}(x \odot y)$ denote the double precision floating point representation of $x \odot y$. Then there exists $\delta \in \mathbb{R}$ with $|\delta| \leq 2^{-52}$ such that*

$$\mathrm{fl}(x \odot y) = (x \odot y)(1 + \delta).$$

Unfortunately Proposition 2.9 can not hold for a succession of addition and subtraction. To see why, suppose for simplicity we have a floating point arithmetic system in base ten with three digits stored in the mantissa, and we want to subtract

$$x = 1.234 \times 10^0, \qquad \text{and} \qquad y = 1.233 \times 10^0.$$

When we perform the subtraction $x - y$, we get

$$0.001 \times 10^0$$

The final answer must be a floating point number, so the output of the program is

$$1.000 \times 10^{-3}.$$

Since x and y shared the most significant digits in their mantissas, the subtraction $x - y$ had only one significant digit. Then the returned value of $x - y$ has zero significant digits in the mantissa. Since significant digits were lost in the mantissa, this issue is known as a **loss of significance**.

For this single subtraction, no error has actually occurred, since $x - y = 10^{-3}$. However, combining other operations with subtractions can cause substantial errors. For example, the expression

$$(1 + 2^{-53}) - 1$$

returns the value 0 in Matlab, when it should be equal to 2^{-53}. The absolute error of 2^{-53} is quite small, but the relative error is not even defined. Even more alarming, the expression

$$2^{53}((1 + 2^{-53}) - 1)$$

returns the value 1 in Matlab, when it should be equal to 0. That is, there is an absolute error of 1. This observation leads to the following heuristic.

Heuristic for Floating Point Arithmetic: Subtractions are dangerous, but addition, multiplication and division are generally safe (concerning relative errors).

The relative safety of addition, multiplication and division follows from the analogues of Propositions 2.7 and 2.9. The danger of using subtraction can be formalized in the following statement.

Proposition 2.14. *Let x and y be positive normal double precision floating point numbers with $x > y$. (Since $x > y$, $1 - y/x > 0$.) Let p, q be nonnegative integers such that*

$$2^{-q} \leq 1 - \frac{y}{x} \leq 2^{-p}.$$

Let d be the number of zeros at the end of the mantissa in the computation of $x - y$. (We could say d is the number of significant binary digits that are lost in computing $x - y$.) Then

$$p \leq d \leq q.$$

Proof. Let $1 \leq s, t < 2$ and let m, n be integers such that

$$x = s2^m, \qquad y = t2^n.$$

Write $y = t2^{n-m}2^m$ so that $x - y = (s - t2^{n-m})2^m$. Since $x > y$, $s - t2^{n-m} > 0$, so that $s - t2^{n-m}$ is a mantissa representation of $x - y$. This mantissa satisfies

$$s - t2^{n-m} = s\Big(1 - \frac{t2^n}{s2^m}\Big) = s\Big(1 - \frac{y}{x}\Big).$$

Since $1 \leq s < 2$, our assumption implies that

$$2^{-q} \leq s - t2^{n-m} < 2 \cdot 2^{-p}.$$

That is, the mantissa's binary representation starts with at least p zeros, and at most q zeros. □

Example 2.15. Near zero, the function

$$f(x) = \sqrt{x^2 + 1} - 1$$

exhibits some loss of significance errors, as the following plot shows.

```
x = linspace(-10^(-7),10^(-7),1000);
plot(x,sqrt(x.^2 +1) -1,'r',x,(x.^2)./(sqrt(x.^2 +1)+1),'b');
legend('original function','rewritten function')
```

However, by multiplying and dividing by $\sqrt{x^2 + 1} + 1$, we can rewrite f as

$$f(x) = \frac{x^2}{\sqrt{x^2 + 1} + 1}, \qquad \forall\, x \in \mathbb{R},$$

which avoids the loss of significance of the previous formula.

Exercise 2.16. Suppose we want to compute the quantity

$$x - \sin(x)$$

for any real $x \in \mathbb{R}$. For x near zero, there will be a loss of significance error, so we should perhaps try to find a better way to compute this quantity.

- Find the loss of significance (i.e. the number of zero bits at the end of the binary mantissa) when $x - \sin(x)$ is computed directly in double precision floating point arithmetic in Matlab, when $x = 2^{-25}$.
- Find the loss of significance (i.e. the number of zero bits at the end of the mantissa) when $x - \sin(x)$ is computed as

  $$\frac{x^3}{3!} - \frac{x^5}{5!},$$

  when $x = 2^{-25}$. (Your answer can be off by one or two from the true value.)
- Estimate the relative error when $x - \sin(x)$ is computed as

  $$\frac{x^3}{3!} - \frac{x^5}{5!},$$

  when $x = 2^{-25}$. (Your answer does not have to exactly correct. It is okay to be approximately correct.)

Exercise 2.17. This exercise examines an unstable recurrence computation.

Consider the following recursion with $x_1 := 1$ and $x_2 := 1/3$.

$$x_{n+1} = \frac{13}{3}x_n - \frac{4}{3}x_{n-1}, \qquad \forall\, n \geq 2.$$

- Verify that the recurrence is solved by $x_n := (1/3)^{n-1}$ for all $n \geq 1$.
- Using Matlab, solve for x_{40}. For example, use

```
x(1)=1; x(2)=1/3;
for i=3:40
    x(i)=(13/3)*x(i-1) - (4/3)*x(i-2);
end
x(40)
```

  Is the answer what you expected to get? (Hint: examine a logarithmically scaled plot in the y-axis, using `semilogy(abs(x))`.)
- With a different initial condition, the above recurrence can have other solutions. To find them, rewrite the recurrence as

$$\begin{pmatrix} 13/3 & -4/3 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_n \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix}, \qquad \forall\, n \geq 2.$$

  Then note that the eigenvalues of the matrix $A := \begin{pmatrix} 13/3 & -4/3 \\ 1 & 0 \end{pmatrix}$ are $1/3$ and 4, so iterating the recurrence shows that

$$\begin{pmatrix} 13/3 & -4/3 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} x_2 \\ x_1 \end{pmatrix} = \begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix}, \qquad \forall\, n \geq 1.$$

  Since A has two distinct eigenvalues, it is diagonalizable, so if $\begin{pmatrix} x_2 \\ x_1 \end{pmatrix}$ is written as a linear combination $a_1v_1 + a_2v_2$ of the corresponding eigenvectors $v_1, v_2 \in \mathbb{R}^2$ of A, the recurrence becomes

$$a_1(1/3)^{n-1}v_1 + a_2 4^{n-1}v_2 = \begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix}, \qquad \forall\, n \geq 1.$$

- Show that in the case $x_1 = 1$ and $x_2 = 1/3$, we have $a_2 = 0$. However, small numerical errors that occur in the computation of the recurrence correspond to a_2 being computed to be nonzero. Explain how this relates to the logarithmic plot `semilogy(abs(x))` you examined above.

2.2. **Simulation of Random Variables.**

Remark 2.18. A Monte Carlo simulation simulates a large number samples from some random quantity. For example, the command `rand(1,1000)` generates a row vector that simulates 1000 numbers that are equally likely to take any value in $(0, 1)$. And the command `round(rand(1,1000))` represents a row vector of 1000 fair coin flips, since each entry of the vector should have probability $1/2$ of taking the value 0 or 1.

In a Monte Carlo simulation, one often sums the results of n samples and then divides by n. For example, the Law of Large Numbers says that, for a large number (such as 10000), `mean(round(rand(1,10000)))` should be close to $1/2$. That is, roughly half of 10000 coin

flips will be heads, and roughly half of these flips will be tails. (Though actually it is unlikely that exactly 5000 of the coin flips will be heads.)

Exercise 2.19 (Numerical Integration)**.** Consider the function

$$f(t) := t^3 + 1.$$

In this case, we can easily compute

$$\int_0^1 f(t)dt = \frac{5}{4}.$$

Sometimes, especially in computer graphics applications, integrals are too complicated to compute directly, so we instead use randomness to estimate the integral. That is, we pick n random points in $[0, 1]$, and average the values of f at these points, as in the following program.

```
n=10^5;
f= @(t) t.^3 +1;
mean(f(rand(1,n)))
```

Using this program with $n = 10^5, 10^6, 10^7$ and 10^8, report the estimated values for the integral of f, along with their relative errors.

Now, compute the exact value of $\int_3^5 \log x dx$, and modify the above program to give estimates for the value of this integral and report relative errors, using a number of points n where $n = 10^5, 10^6, 10^7$ and 10^8.

Remark 2.20. When Matlab or other computer programs generate "random numbers" using e.g. `rand` or `randn`, these numbers are not actually random or independent. These numbers are **pseudorandom**. That is, functions such as `rand` output numbers in a deterministic way, but these numbers behave as if they were random. All "random" numbers generated by computers are actually pseudorandom, and this includes slot machines at casinos, video games, etc. So, when using Monte Carlo simulation as we did above, we should be careful about interpreting our results, since it is generally impossible to take random samples on a computer.

And, theoretically, if you knew enough about the random number generator that a slot machine is using, you could predict its output.

2.3. **Additional Comments.** Classical numerical analysis often bounds the numerical errors of a numerical algorithm, e.g. that estimates the integral of a function.

Modern numerical analysis also examines the behavior of algorithms that work well with noisy data (i.e. algorithms that work well in the average case, rather than the worst case). Sometimes we can even guarantee the performance of an algorithm when it is given adversarial data (i.e. some inputs to the algorithm can be chosen arbitrarily).

For more details on the use of "guard bits" and "sticky bits" in implementation of e.g. addition in Matlab, see Numerical Computing with IEEE Floating Point Arithmetic, Michael Overton

3. Solving One Variable Equations

3.1. **Introduction.** Suppose we want to find the zero of a continuous function $f\colon [a,b] \to \mathbb{R}$ If $f(a)$ and $f(b)$ have different signs, the Intermediate Value Theorem guarantees that f has a zero in the interval $[a,b]$. Then a zero can be found by a divide and conquer strategy.

Algorithm 3.1 (**Divide and Conquer Zero Finding**).

Input: $a, b \in \mathbb{R}$, $a < b$, $f\colon [a,b] \to \mathbb{R}$ a continuous function (e.g. specified as a function in Matlab) where $f(a)$ and $f(b)$ have different signs, and a tolerance level $\delta > 0$.

Output: a floating point value $x \in \mathbb{R}$ that is close to $y \in \mathbb{R}$ with $f(y) = 0$.

Initialize $a_0 := a$, $b_0 := b$, $\gamma := b - a$, $n := 0$.

While $\gamma > \delta$, do the following.

Let $c := (a_n + b_n)/2$.

If $\operatorname{sign}(c) = \operatorname{sign}(a_n)$ (then f changes sign on the right half of the current interval, so the next interval is the right half of the previous interval).

Define $a_{n+1} := c$, $b_{n+1} := b_n$.

Else (then f changes sign on the left half of the current interval, so the next interval is the left half of the previous interval).

Define $b_{n+1} := c$, $a_{n+1} := a_n$.

End

Increase the value of n by 1.

Divide the value of γ by 2.

When the while loop terminates, define $x := a_{n+1}$.

Proposition 3.2. *Let $a, b > 0$. Let $f\colon [a,b] \to \mathbb{R}$ be a continuous function such that $f(a)$ and $f(b)$ have different signs. After $n \leq 52$ iterations, Algorithm 3.1 outputs $x \in \mathbb{R}$ such that*

$$|x - y| \leq 2^{-n}(1 + 2^{-52})^{2n} |a - b|,$$

where $y \in [a,b]$ is a zero of f that is guaranteed to exist by the Intermediate Value Theorem.

Proof. At the n^{th} iteration of Algorithm 3.1, we have an interval $[a_n, b_n]$ such that the signs of $f(a_n)$ and $f(b_n)$ are not the same. So, each interval $[a_n, b_n]$ contains a zero y. So, if $x := a_n$, then

$$|x - y| \leq |b_n - a_n| \leq 2^{-n} |b - a|.$$

The last inequality follows by induction on n and $|b_n - a_n| \leq (1/2)\,|b_{n-1} - a_{n-1}|$, for all $n \geq 1$, which follows by the definition of Algorithm 3.1. Finally, each iteration of the algorithm involves one addition and one division of positive numbers, hence the $(1 + 2^{-52})^{2n}$ term from Proposition 2.7. □

Proposition 3.2 says that n iterations of Algorithm 3.1 results in about n bits of accuracy in the mantissa of a zero of a continuous function f. Put another way, the accuracy of the zero-finding Algorithm 3.1 (as measured by a number of bits) is linear in the number of iterations of the algorithm. A natural question is: can we do better? For example, is it possible to have about n^2 bits of accuracy in the mantissa of the zero of f, after n iterations of an algorithm? Perhaps surprisingly, the answer is yes.

3.2. **Newton's Method.**

Example 3.3 (**The Babylonian Square Root Algorithm**)**.** Suppose we want to find the closest double precision floating point number to $\sqrt{2}$. The function $f(x) := x^2 - 2$ has a zero at $x = \sqrt{2}$, so we could use Algorithm 3.1 with $a = 1/2, b = 3/2$ to approximate $\sqrt{2}$ with around 52 iterations. However, this is not what computers and calculators do, since there is a much faster way to approximate $\sqrt{2}$.

The following recursively defined sequence gets closer and closer to the square root of 2.

Define $a_1 = 1$, and for any integer $n \geq 2$,

$$a_n = \frac{1}{2}\left(a_{n-1} + \frac{2}{a_{n-1}}\right).$$

We use the following Matlab program to compute the recurrence:

```
x=1;
for i=1:7
    x=(1/2)*(x+2/x)
end
```

The corresponding output is the following:

$$\begin{array}{ll} a_1 & 1 \\ a_2 & 1.5 \\ a_3 & 1.416666666666667 \\ a_4 & 1.414215686274510 \\ a_5 & 1.414213562374690 \\ a_6 & 1.414213562373095 \\ a_7 & 1.414213562373095 \\ a_8 & 1.414213562373095 \end{array}$$

The algorithm only required five iterations to find a 52 binary digit approximation to $\sqrt{2}$.

Indeed, it can be shown that a_n gets arbitrarily close to $\sqrt{2}$ as n tends to infinity. We will make this statement more precise below.

If we instead defined $a_n = \frac{1}{2}\left(a_{n-1} + \frac{M}{a_{n-1}}\right)$, then this sequence will get closer and closer to $\sqrt{M}$, where $M \geq 0$. This phenomenon can be explained by **Newton's Method**. We have

$$a_n = a_{n-1} - \frac{1}{2}a_{n-1} + \frac{M}{2a_{n-1}} = a_{n-1} - \frac{(a_{n-1}^2 - M)}{2a_{n-1}} = a_{n-1} - \frac{f(a_{n-1})}{f'(a_{n-1})},$$

where $f(x) = x^2 - M$. This iterative scheme searches for a zero of the function f. Given the x-value a_{n-1}, the linear approximation of f at a_{n-1} is $f'(a_{n-1})(x - a_{n-1}) + f(a_{n-1})$. And we define a_n as the x-intercept (i.e. zero) of this line. That is, we define a_n so that

$$0 = f'(a_{n-1})(a_n - a_{n-1}) + f(a_{n-1}).$$

Solving this equation for a_n gives the above equality.

$$a_n = a_{n-1} - \frac{f(a_{n-1})}{f'(a_{n-1})}.$$

Remark 3.4. The Babylonian square root algorithm is typically used by computers when you ask for the square root of a number. That is, the computer will compute something like a_{10} for a given number M, and then return a_{10} as the square root of the number M.

Exercise 3.5. Using the Divide and Conquer Algorithm to approximate the value of $\sqrt{3}$ with the function $f(x) := x^2 - 3$ by starting your search on the interval $[1, 2]$. Report how many iterations the algorithm takes until it no longer makes any progress (i.e. once the algorithm can no longer create a smaller interval to search for a zero of f.)

Then, use the Babylonian square root algorithm to approximate the value of $\sqrt{3}$, starting with $a_1 = 1$. Report how many iterations the algorithm takes until it no longer makes any progress (i.e. once the sequence generated by the algorithm becomes constant.)

Algorithm 3.6. Newton's Method, a general way to find the roots of a differentiable function $f\colon \mathbb{R} \to \mathbb{R}$.

(1) Choose any point $x_1 \in \mathbb{R}$.
(2) Compute the tangent line of f at x_1: $y(x) = f'(x_1)(x - x_1) + f(x_1)$.
(3) Find x_2 such that $y(x_2) = 0$. This is the intersection of the tangent line $y(x)$ with the x-axis. Note that x_2 satisfies
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}.$$
(4) Return to step (2), but replace x_1 with x_2. More generally, at the n^{th} iteration of the algorithm, compute the tangent line of f at x_n in step (2), and then find an x_{n+1} in step (3) which is a zero of the tangent line. So, in general we iterate the following equation.
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Newton's Method requires some assumptions to converge to an actual zero of a function. As we will see in the exercises below, there are some relatively simple examples where Newton's Method produces large errors.

Theorem 3.7. *Let $f\colon \mathbb{R} \to \mathbb{R}$ be a function with two continuous derivatives, with $f'(y) > 0$ and $f''(y) > 0$ for all $y \in \mathbb{R}$ (so that f is strictly increasing and strictly convex). Assume $\exists$ $x \in \mathbb{R}$ with $f(x) = 0$. Starting from any point $x_1 \in \mathbb{R}$, the sequence $x_1, x_2, \ldots,$ generated by Newton's Method will converge to x. (Also x is the only zero of f.)*

Proof. (Since $f' > 0$, we can freely divide by f' below.) For any $k \geq 1$, define $e_k := x_k - x$. Then
$$e_{k+1} = x_{k+1} - x = x_k - x - \frac{f(x_k)}{f'(x_k)} = e_k - \frac{f(x_k)}{f'(x_k)} = \frac{e_k f'(x_k) - f(x_k)}{f'(x_k)}. \qquad (*)$$
Since f has two continuous derivatives, Taylor's Theorem applies at x_k, and there exists $\xi_k \in \mathbb{R}$ such that
$$0 = f(x) = f(x_k - e_k) = f(x_k) - e_k f'(x_k) + (1/2)e_k^2 f''(\xi_k).$$
Plugging this into $(*)$, we get
$$e_{k+1} = \frac{1}{2}\frac{f''(\xi_k)}{f'(x_k)} e_k^2.$$

In particular, $e_{k+1} \geq 0$ for all $k \geq 1$, so that $x_k \geq x$ for all $k \geq 2$. Since f is strictly increasing $f(x_k) \geq f(x) = 0$ for all $k \geq 2$. Then $(*)$ implies that $e_{k+1} \leq e_k$ for all $k \geq 1$. So, the sequence $e_2, e_3, \ldots$ is decreasing and nonnegative, and the sequence $x_2, x_3, \ldots$ is decreasing and bounded below by x. Therefore $y := \lim_{k\to\infty} x_k$ exists. Define

$$\phi(t) := t - (f'(t))^{-1} f(t), \qquad \forall\, t \in \mathbb{R}.$$

Since ϕ is a composition of continuous functions, ϕ is continuous. Also, by definition of ϕ, we have $\phi(x_k) = x_{k+1}$. We therefore have

$$\phi(y) = \lim_{k\to\infty} \phi(x_k) = \lim_{k\to\infty} x_{k+1} = y.$$

Since $\phi(y) = y$, we have $f(y)/f'(y) = 0$. Since $f' \neq 0$, we conclude that $f(y) = 0$.

Finally, f is strictly increasing, so x is the only zero of f, and we conclude that $x = y$. $\square$

Exercise 3.8. Let $f(x) := x^2 - 2$ for all $x \in \mathbb{R}$. Let $x_1 > 0$. Let $x_1, x_2, \ldots$ be the sequence generated by Newton's method for f, started at x_1. Prove that $x_1, x_2, \ldots$ converges to $\sqrt{2}$.

Theorem 3.9. *Let $f\colon \mathbb{R} \to \mathbb{R}$ be a function satisfying $f(0) = 0$, $f'(0) \neq 0$, and such that f has two continuous derivatives in some interval of the form $[-r, r]$ with $r > 0$. Let $K > 1$ and suppose that*

$$K^{-1} \leq |f'(x)| \leq K, \qquad |f''(x)| \leq K, \qquad \forall\, x \in [-r, r]. \tag{1}$$

Assume that $|x_1| \leq \frac{1}{8} K^{-4} \max(r^3, r^{-3})$.

Then the sequence $x_1, x_2, \ldots$ from Newton's Method converges to 0.

Proof. **Step 1**. We will first show that if $x_k, x_{k-1} \in [-r, r]$ for some $k \geq 2$,

$$|x_{k+1} - x_k| \;\leq\; \frac{K^2}{2} |x_k - x_{k-1}|^2 .$$

Since f has two continuous derivatives, Taylor's Theorem applies at x_{k-1}. Let $x_{k-1} + y \in [-r, r]$. Then there exists $\xi \in [-r, r]$ such that

$$f(x_{k-1} + y) = f(x_{k-1}) + y f'(x_{k-1}) + (1/2) y^2 f''(\xi). \qquad (*)$$

So, using the definition of x_k, and using $(*)$ and $x_k \in [-r, r]$,

$$\begin{aligned} x_{k+1} - x_k = -\frac{f(x_k)}{f'(x_k)} &= -\frac{f\left(x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}\right)}{f'(x_k)} \\ &\overset{(*)}{=} -\frac{f(x_{k-1}) - \frac{f(x_{k-1})}{f'(x_{k-1})} f'(x_{k-1}) + (1/2)\left(-\frac{f(x_{k-1})}{f'(x_{k-1})}\right)^2 f''(\xi)}{f'(x_k)} \\ &= -\frac{f''(\xi)}{2f'(x_k)} \left(\frac{f(x_{k-1})}{f'(x_{k-1})}\right)^2 = -\frac{f''(\xi)}{2f'(x_k)} (x_k - x_{k-1})^2 . \end{aligned}$$

Note also that $f' \neq 0$ on $[-r, r]$ by (1), so we can freely divide by f', since $x_k, x_{k-1} \in [-r, r]$. Now, using the above equality and our derivative bounds,

$$|x_{k+1} - x_k| \leq \frac{1}{2} \sup_{\xi \in [-r,r], y \in [-r,r]} |f''(\xi)|\, |f'(y)|^{-1} |x_k - x_{k-1}|^2 \overset{(1)}{\leq} \frac{K^2}{2} |x_k - x_{k-1}|^2 .$$

Step 2. We now show that $|x_2 - x_1| \leq K^2 |x_1|$.

As in Step 1, f is twice differentiable, so we apply a single term Taylor's expansion at $x = 0$. Let $x \in [-r, r]$. Then $\exists\, \xi \in [-r, r]$ such that $f(x) = xf'(\xi)$. Therefore, with $x = x_1$,

$$|x_2 - x_1| = \left|\frac{f(x_1)}{f'(x_1)}\right| = \left|x_1 \frac{f'(\xi)}{f'(x_1)}\right| \overset{(1)}{\leq} K^2 |x_1|.$$

For simplicity, we now consider $r = 1$.

Step 3 Let $\varepsilon > 0$. If $x_1 \in [-\varepsilon K^{-4}, \varepsilon K^{-4}]$, then Step 2 implies that $|x_2 - x_1| \leq K^{-2}\varepsilon$ and $|x_2| \leq 2\varepsilon$. So, if $\varepsilon < 1/8$, then $|x_1| \leq 1/2$, $|x_2| \leq 3/4$, and Step 1 implies that $|x_3 - x_2| \leq (1/2)K^{-2}\varepsilon^2$. So, if $\varepsilon < 1/8$, then $|x_3 - x_2| \leq 1/8$, so $|x_3| \leq 7/8$.

We will prove by induction on $k \geq 3$ that $|x_{k-1}| \leq (1-2^{-k})$ and $|x_k - x_{k-1}| \leq 2^{-k+2}\varepsilon^{2^{k-2}}K^{-2}$ if $\varepsilon < 1/8$. The case $k = 3$ is already proven so we prove the inductive step when $k \geq 3$. By the inductive assumption, and $2^{k-2} \geq k$

$$|x_k - x_{k-1}| \leq 2^{-k+2}8^{-2^{k-2}} \leq 2^{-k+2}8^{-k} \leq 2^{-k-1}.$$

$$|x_k| \leq (1 - 2^{-k}) + 2^{-k-1} = (1 - 2^{-k-1}).$$

So, Step 1 implies, with $-2k + 4 \leq -k + 1$,

$$|x_{k+1} - x_k| \leq (1/2)2^{-2k+4}\varepsilon^{2^{k-1}}K^{-2} \leq (1/2)2^{-k+1}\varepsilon^{2^{k-1}}K^{-2} \leq 2^{-k+1}\varepsilon^{2^{k-1}}K^{-2}.$$

We have therefore completed the inductive step. We conclude in particular that

$$|x_k - x_{k-1}| \leq 2^{-k+2}(1/8)^{2^{k-2}}K^{-2} \leq 2^{-k}K^{-2}. \qquad (*)$$

Step 4. It follows from Step 3 that the sequence $x_1, x_2, \ldots$ is Cauchy, since for any $1 < m < n$, we have

$$|x_n - x_m| = \left|\sum_{k=m+1}^{n} (x_k - x_{k-1})\right| \leq \sum_{k=m+1}^{n} |x_k - x_{k-1}| \overset{(*)}{\leq} K^{-2} \sum_{k=m+1}^{\infty} 2^{-k} = K^{-2}2^{-m}.$$

Denote x as the limit of the sequence $x_1, x_2, \ldots$. Define

$$\phi(x) := x - (f'(x))^{-1}f(x), \qquad \forall\, x \in [-r, r].$$

Since $K^{-1} \leq |f'(y)| \leq K$ for all $y \in [-r, r]$, $f'(y) \neq 0$ for all $y \in [-r, r]$. So, $\phi(x)$ and $\phi(x_k)$ are defined for all $k \geq 1$. Moreover, ϕ is a composition of continuous functions, so it is continuous. Also, by definition of ϕ, we have $\phi(x_k) = x_{k+1}$. We therefore have

$$\phi(x) = \lim_{k\to\infty} \phi(x_k) = \lim_{k\to\infty} x_{k+1} = x.$$

Since $\phi(x) = x$, we have $f(x)/f'(x) = 0$. Since $f' \neq 0$ on $[-r, r]$, we conclude that $f(x) = 0$.

Since f is continuously differentiable on $[-r, r]$ and $K^{-1} \leq |f'(y)| \leq K$ for all $y \in [-r, r]$, the Intermediate Value Theorem implies that f is equal to zero on $[-r, r]$ only at the point $y = 0$. We therefore conclude that $x = 0$.

Step 5. We have shown that if $r = 1$ and if $|x_1| \leq K^{-4}/8$, then $x_1, x_2, \ldots \in [-r, r]$ converges to $x = 0$, where $f(0) = 0$. It remains to remove the assumption that $r = 1$.

For the more general case that $r \neq 1$, define $F(x) := f(rx)$, and note that $F'(x) = rf'(xr)$ and $F''(x) = r^2 f''(xr)$. So the assumed derivative bounds on f imply that

$$K^{-1}r \leq |F'(x)| \leq Kr, \qquad |f''(x)| \leq Kr^2, \qquad \forall\, x \in [-1, 1].$$

So, defining $J := \max(Kr^2, K/r^2)$, we have

$$J^{-1} \leq |F'(x)| \leq J, \qquad |f''(x)| \leq J, \qquad \forall\, x \in [-1, 1].$$

We will show that Newton's Method for f started at $x_1 \in [-r, r]$ is somehow equivalent to starting Newton's method for F at $y_1 := x_1/r$. Also, Newton's Method applied to F on $[-1, 1]$ says

$$y_{k+1} = y_k - \frac{F(y_k)}{F'(y_k)} = y_k - \frac{f(ry_k)}{rf'(ry_k)} = \frac{1}{r}\Big(ry_k - \frac{f(ry_k)}{f'(ry_k)}\Big).$$

That is, for all $k \geq 1$,

$$ry_{k+1} = ry_k - \frac{f(ry_k)}{f'(ry_k)}.$$

So, Newton's method applied to F results in the sequence $y_1, y_2, \ldots$, and this is equivalent to applying Newton's method for f, resulting in the sequence $ry_1, ry_2, \ldots$.

Lastly, the assumption that $|y_1| \leq \frac{J^{-4}}{8}$ translates to $|x_1| \leq r\frac{J^{-4}}{8} = (r/8)K^{-4}\max(r^2, r^{-2})$. □

Exercise 3.10. Consider the function

$$f(x) := \operatorname{sign}(x)\sqrt{|x|}, \qquad \forall\, x \in \mathbb{R}.$$

This function has only one zero at $x = 0$.

Either by hand or a computer, use Newton's method to try to find a zero, using an initial guess such as $1, 1/2$ or $1/4$.

What happens? Describe the output of Newton's Method. Does the sequence of iterates $x_0, x_1, \ldots$ converge to 0? Explain why or why not. (Does the convergence theorem 3.9 apply?)

Exercise 3.11. Consider the function

$$f(x) := x^{1/3}, \qquad \forall\, x \in \mathbb{R}.$$

This function has only one zero at $x = 0$.

Using a computer, use Newton's method to try to find a zero, using an initial guess such as $1, 1/2$ or $1/4$.

What happens? Describe the output of Newton's Method. Does the sequence of iterates $x_0, x_1, \ldots$ converge to 0? Explain why or why not. (Does the convergence theorem 3.9 apply?)

Exercise 3.12. Consider the function

$$f(x) := x^2 + 1/100, \qquad \forall\, x \in \mathbb{R}.$$

This function has no zeros, but it is nearly zero at $x = 0$. For instructive purposes, we can still apply Newton's Method to see what happens.

Using a computer, use Newton's method to try to find a zero of f, using an initial guess of $1/10$.

What happens? Describe the output of Newton's Method. Explain what has happened.

4. Numerical Linear Algebra

4.1. Review of Linear Algebra.

Definition 4.1 (**Linear combination**). Let V be a vector space over a field $\mathbb{F}$. Let $u_1, \ldots, u_n \in V$ and let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. Then $\sum_{i=1}^n \alpha_i u_i$ is called a **linear combination** of the vector elements $u_1, \ldots, u_n$.

Definition 4.2 (**Linear dependence**)**.** Let V be a vector space over a field $\mathbb{F}$. Let S be a subset of V. We say that S is **linearly dependent** if there exists a finite set of vectors $u_1, \ldots, u_n \in S$ and there exist $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ which are not all zero such that $\sum_{i=1}^{n} \alpha_i u_i = 0$.

Definition 4.3 (**Linear independence**)**.** Let V be a vector space over a field $\mathbb{F}$. Let S be a subset of V. We say that S is **linearly independent** if S is not linearly dependent.

Example 4.4. The set $S = \{(1,0),(0,1)\}$ is linearly independent in $\mathbb{R}^2$. The set $S \cup (1,1)$ is linearly dependent in $\mathbb{R}^2$, since $(1,0) + (0,1) - (1,1) = 0$.

Definition 4.5 (**Span**)**.** Let V be a vector space over a field $\mathbb{F}$. Let $S \subseteq V$ be a finite or infinite set. Then the **span** of S, denoted by $\operatorname{span}(S)$, is the set of all finite linear combinations of vectors in S. That is,

$$\operatorname{span}(S) = \left\{ \sum_{i=1}^{n} \alpha_i u_i \colon n \in \mathbb{N}, \alpha_i \in \mathbb{F}, u_i \in S, \forall\, i \in \{1, \ldots, n\} \right\}.$$

Remark 4.6. We define $\operatorname{span}(\emptyset) := \{0\}$.

Theorem 4.7 (**Span as a Subspace**)**.** *Let V be a vector space over a field $\mathbb{F}$. Let $S \subseteq V$. Then* $\operatorname{span}(S)$ *is a subspace of V such that $S \subseteq \operatorname{span}(S)$. Also, any subspace of V that contains S must also contain* $\operatorname{span}(S)$.

Definition 4.8 (**Normed Linear Space**)**.** Let $\mathbb{F}$ denote either $\mathbb{R}$ or $\mathbb{C}$. Let V be a vector space over $\mathbb{F}$. A **normed linear space** is a vector space V equipped with a **norm**. A norm is a function $V \to \mathbb{R}$, denoted by $\|\cdot\|$, which satisfies the following properties.

(a) For all $v \in V$, for all $\alpha \in \mathbb{F}$, $\|\alpha v\| = |\alpha|\,\|v\|$. (Homogeneity)
(b) For all $v \in V$ with $v \neq 0$, $\|v\|$ is a positive real number; $\|v\| > 0$. And $v = 0$ if and only if $\|v\| = 0$. (Positive definiteness)
(c) For all $v, w \in V$, $\|v + w\| \leq \|v\| + \|w\|$. (Triangle Inequality)

Definition 4.9 (**Complex Conjugate**)**.** Let $i := \sqrt{-1}$. Let $x, y \in \mathbb{R}$, and let $z = x + iy \in \mathbb{C}$. Define $\overline{z} := x - iy$. Define $|z| := \sqrt{x^2 + y^2}$. Note that $|z|^2 = z\overline{z}$.

Definition 4.10 (**Inner Product**)**.** Let $\mathbb{F}$ denote either $\mathbb{R}$ or $\mathbb{C}$. Let V be a vector space over $\mathbb{F}$. An **inner product space** is a vector space V equipped with an **inner product**. An inner product is a function $V \times V \to \mathbb{F}$, denoted by $\langle \cdot, \cdot \rangle$, which satisfies the following properties.

(a) For all $v, v', w \in V$, $\langle v + v', w \rangle = \langle v, w \rangle + \langle v', w \rangle$. (Linearity in the first argument).
(b) For all $v, w \in V$, for all $\alpha \in \mathbb{F}$, $\langle \alpha v, w \rangle = \alpha \langle v, w \rangle$. (Homogeneity in the first argument)
(c) For all $v \in V$, if $v \neq 0$, then $\langle v, v \rangle$ is a positive real number; $\langle v, v \rangle > 0$. (Positivity)
(d) For all $v, w \in V$, $\langle v, w \rangle = \overline{\langle w, v \rangle}$. (Conjugate symmetry)

Exercise 4.11. Using the above properties, show the following things.

(e) For all $v, v', w \in V$, $\langle w, v + v' \rangle = \langle w, v \rangle + \langle w, v' \rangle$. (Linearity in the second argument)
(f) For all $v, w \in V$, for all $\alpha \in \mathbb{F}$, $\langle v, \alpha w \rangle = \overline{\alpha} \langle v, w \rangle$.
(g) For all $v \in V$, $\langle v, 0 \rangle = \langle 0, v \rangle = 0$.
(h) $\langle v, v \rangle = 0$ if and only if $v = 0$.

Remark 4.12. If $\mathbb{F} = \mathbb{R}$, then property (d) says that $\langle v, w \rangle = \langle w, v \rangle$.

Lemma 4.13. *Let $\langle , \rangle$ be an inner product on a vector space V. Then the function $\|\cdot\| : V \to \mathbb{R}$ defined by $\|v\| := \sqrt{\langle v, v\rangle}$ is a norm on V.*

Definition 4.14 (**Orthogonal Vectors**)**.** Let V be an inner product space, and let $v, w \in V$. We say that v, w are **orthogonal** if $\langle v, w\rangle = 0$.

Definition 4.15 (**Orthogonal Set, Orthonormal Set**)**.** Let V be an inner product space and let $(v_1, \ldots, v_n)$ be a collection of vectors in V. The set of vectors $(v_1, \ldots, v_n)$ is said to be **orthogonal** if $\langle v_i, v_j\rangle = 0$ for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$. If additionally $\langle v_i, v_i\rangle = 1$ for all $i \in \{1, \ldots, n\}$, the set of vectors $(v_1, \ldots, v_n)$ is called **orthonormal**.

Corollary 4.16. *Let V be an inner product space, and let $v_1, \ldots, v_n \in V$ be an orthonormal set of vectors. Then*

$$\left\| \sum_{i=1}^{n} \alpha_i v_i \right\|^2 = \sum_{i=1}^{n} |\alpha_i|^2 .$$

Corollary 4.17. *Any set of orthonormal vectors is linearly independent.*

Definition 4.18 (**Orthonormal Basis**)**.** Let V be an inner product space. An **orthonormal basis** of V is a collection $(v_1, \ldots, v_n)$ of orthonormal vectors that is also a basis for V.

Corollary 4.19. *Let V be an n-dimensional inner product space. Let $(v_1, \ldots, v_n)$ be an orthonormal set in V. Then $(v_1, \ldots, v_n)$ is an orthonormal basis of V.*

Theorem 4.20. *Let V be an inner product space. Let $(v_1, \ldots, v_n)$ be an orthonormal basis of V. Then, for any $v \in V$, we have*

$$v = \sum_{i=1}^{n} \langle v, v_i\rangle v_i.$$

Definition 4.21 (**Unit Vector**)**.** Let V be a normed linear space, and let $v \in V$. If $\|v\| = 1$, we say that v is a **unit vector**.

Remark 4.22. Let $v \neq 0$. Then $v/\|v\|$ is a unit vector.

Theorem 4.23 (**Gram-Schmidt Orthogonalization**)**.** *Let $v_1, \ldots, v_n$ be a linearly independent set of vectors in an inner product space V. Then we can create an orthogonal set of vectors in V as follows. Define*

$$\begin{aligned} w_1 &:= v_1. \\ w_2 &:= v_2 - \left\langle v_2, \frac{w_1}{\|w_1\|} \right\rangle \frac{w_1}{\|w_1\|}. \\ w_3 &:= v_3 - \left\langle v_3, \frac{w_1}{\|w_1\|} \right\rangle \frac{w_1}{\|w_1\|} - \left\langle v_3, \frac{w_2}{\|w_2\|} \right\rangle \frac{w_2}{\|w_2\|}. \end{aligned}$$

And so on. In general, for $k \in \{2, \ldots, n\}$, define

$$w_k := v_k - \sum_{j=1}^{k-1} \left\langle v_k, \frac{w_j}{\|w_j\|} \right\rangle \frac{w_j}{\|w_j\|}.$$

Then for each $k \in \{1, \ldots, n\}$, $(w_1, \ldots, w_k)$ is an orthogonal set of nonzero vectors in V. Also, $\operatorname{span}(w_1, \ldots, w_k) = \operatorname{span}(v_1, \ldots, v_k)$ for each $k \in \{1, \ldots, n\}$. Finally, note that the set $(w_1/\|w_1\|, \ldots, w_n/\|w_n\|)$ is an orthonormal set of vectors in V with the same span as $v_1, \ldots, v_n$.

Definition 4.24 (**Transpose**)**.** Let A be an $m \times n$ matrix with entries A_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$. Then the **transpose** A^T of A is defined to be the $n \times m$ matrix with entries $(A^T)_{ij} := A_{ji}$, $1 \leq i \leq n$, $1 \leq j \leq m$.

Exercise 4.25. Let A be an $m \times n$ matrix. Let B be an $\ell \times m$ matrix. Show that $(BA)^T = A^T B^T$.

Remark 4.26. If A is an $n \times n$ invertible matrix, then $I_n^T = (AA^{-1})^T = (A^{-1})^T A^T$, so A^T is also invertible.

Definition 4.27 (**Adjoint of a Matrix**)**.** Let A be an $m \times n$ matrix with $A_{jk} \in \mathbb{C}$, $1 \leq j \leq m$, $1 \leq k \leq n$. The **adjoint** of A, denoted by A^*, is an $n \times m$ matrix with entries $(A^*)_{jk} := \overline{A_{kj}}$, $1 \leq j \leq n$, $1 \leq k \leq m$.

Definition 4.28 (**Normal Matrix**)**.** Let A be an $n \times n$ matrix with values in $\mathbb{C}$. We say that A is **normal** if $AA^* = A^*A$.

Definition 4.29 (**Self-Adjoint Matrix**)**.** Let $\mathbb{F}$ denote $\mathbb{R}$ or $\mathbb{C}$. A square matrix A with elements in $\mathbb{F}$ is said to be **self-adjoint** if $A = A^*$. The term **Hermitian** is a synonym of self-adjoint.

Definition 4.30 (**Unitary Matrix/ Orthogonal Matrix**)**.** Let A be an $n \times n$ matrix with elements in $\mathbb{C}$. We say that A is **unitary** if $AA^* = A^*A = I$. In the case that A has real entries and $AA^T = A^T A = I$, we say that A is **orthogonal**.

Definition 4.31. A matrix A is said to be **diagonalizable** if there exists an invertible matrix Q and a diagonal matrix D such that $A = QDQ^{-1}$. That is, a matrix A is diagonalizable if and only if it is similar to a diagonal matrix.

Definition 4.32 (**Eigenvector and Eigenvalue**)**.** Let V be a vector space over a field $\mathbb{F}$. Let $T\colon V \to V$ be a linear transformation. An **eigenvector** of T is a nonzero vector $v \in V$ such that, there exists $\lambda \in \mathbb{F}$ with $T(v) = \lambda v$. The scalar λ is then referred to as the **eigenvalue** of the eigenvector v.

Theorem 4.33 (**The Spectral Theorem for Normal Matrices**)**.** *Let $\mathbb{F}$ denote either $\mathbb{R}$ or $\mathbb{C}$. Let A be an $n \times n$ matrix with entries in $\mathbb{F}$. Then there exists an orthonormal basis of $\mathbb{F}^n$ consisting of eigenvectors of A. In particular, A is diagonalizable with $A = QDQ^{-1}$, where the columns of Q are eigenvectors of A and $QQ^* = Q^*Q = I$.*

Theorem 4.34 (**The Spectral Theorem for Self-Adjoint Matrices**)**.** *Let $\mathbb{F}$ denote either $\mathbb{R}$ or $\mathbb{C}$. Let A be an $n \times n$ matrix with entries in $\mathbb{F}$. Then there exists an orthonormal basis of F^n consisting of eigenvectors of A. In particular, A is diagonalizable with $A = QDQ^{-1}$, where the columns of Q are eigenvectors of A and $QQ^* = Q^*Q = I$. Moreover, all eigenvalues of A are real, i.e. D has real entries.*

Theorem 4.35 (**The Spectral Theorem for Unitary Matrices**)**.** *Let $\mathbb{F}$ denote either $\mathbb{R}$ or $\mathbb{C}$. Let A be an $n \times n$ matrix with entries in $\mathbb{F}$. Then there exists an orthonormal basis*

of F^n consisting of eigenvectors of A. In particular, A is diagonalizable with $A = QDQ^{-1}$, where the columns of Q are eigenvectors of A. Moreover, all eigenvalues of T have absolute value 1.

We will discuss algorithms for Spectral Theorems in Section 4.7.

4.2. **Row Operations.** We begin our discussion of row operations on matrices with some examples.

Example 4.36 (**Type 1: Interchange two Rows**)**.** For example, we can swap the first and third rows of the matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix}$$

to get

$$\begin{pmatrix} 0 & 8 \\ 3 & 5 \\ 1 & 2 \end{pmatrix}.$$

Define

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

Note that

$$E \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix} = \begin{pmatrix} 0 & 8 \\ 3 & 5 \\ 1 & 2 \end{pmatrix}.$$

Remark 4.37. E as defined above is invertible. In fact, $E = E^{-1}$. In general, if E is the $n \times n$ matrix that swaps two rows of an $n \times n$ matrix A, then EA is A with those two rows swapped. So $EEA = A$ for all $n \times n$ matrices A, so $EE = I_n$, i.e. E is invertible.

Example 4.38 (**Type 2: Multiply a row by a nonzero scalar**)**.** For example, let's multiply the second row of the following matrix by 2.

$$\begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix}.$$

We then get

$$\begin{pmatrix} 1 & 2 \\ 6 & 10 \\ 0 & 8 \end{pmatrix}.$$

Define

$$E := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Note that

$$E \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 6 & 10 \\ 0 & 8 \end{pmatrix}$$

Remark 4.39. E as defined above has inverse

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

In general, suppose E corresponds to multiplying the i^{th} row of a given matrix by $\alpha \in \mathbb{F}$, $\alpha \neq 0$. Then E is a matrix with ones on the diagonal, except for the i^{th} entry on the diagonal, which is α. And all other entries of E are zero. Then, we see that E^{-1} exists and is a matrix with ones on the diagonal, except for the i^{th} entry on the diagonal, which is α^{-1}. And all other entries of E^{-1} are zero. In particular, E is invertible.

Example 4.40 (**Adding one row to another**)**.** Let's add two copies of the first row of the following matrix to the third row.

$$\begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix}.$$

We then get

$$\begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 2 & 12 \end{pmatrix}.$$

Define

$$E := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}.$$

Note that

$$E \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 5 \\ 2 & 12 \end{pmatrix}.$$

Remark 4.41. E as defined above has inverse

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

That is, adding 2 copies of row one to row three is inverted by adding -2 copies of row one to row three. In a similar way, a general row addition operator is seen to be invertible.

Remark 4.42 (**Summary of Row Operations**)**.** The three row operations (Type 1, Type 2, and Type 3) are all invertible.

Remark 4.43 (**Solving Systems of Linear Equations**)**.** Let A be an $m \times n$ matrix, let $x \in \mathbb{R}^n$ be a variable vector, and let $b \in \mathbb{R}^m$ be a known vector. Consider the system of linear equations

$$Ax = b.$$

Let E be any elementary row operation. Since E is invertible, finding a solution x to the system $Ax = b$ is equivalent to finding the solution x to the system $EAx = Eb$. By applying many elementary row operations, you have seen in a previous course how to solve the system $Ax = b$. That is, you continue to apply elementary row operations $E_1, \ldots, E_k$ such that

$E_1 \cdots E_k A$ in **row-echelon** form, and you then solve $E_1 \cdots E_k Ax = E_1 \cdots E_k b$. A matrix B is in row-echelon form if each row is either zero, or its left-most nonzero entry is 1, with zeros below the 1.

Remark 4.44 (**Inverting a Matrix**)**.** Let A be an invertible $n \times n$ matrix. You learned in a previous course an algorithm for inverting A using elementary row operations. Below, we will prove that this algorithm works.

Remark 4.45 (**Column Operations**)**.** In the above discussion, we could have also used column operations instead of row operations. Column operations would then correspond to multiplying the matrices E on the right side, rather than the left side. The invertibility of column operations would therefore still hold.

Definition 4.46 (**Rank**)**.** The **rank** of a matrix A is equal to the dimension of the space spanned by the columns of A.

Proposition 4.47. *Let A be a real $n \times n$ matrix. Then A is invertible if and only if A has rank n.*

Lemma 4.48. *Let A be a matrix in row-echelon form. Then the rank of A is equal to the number of nonzero rows of A.*

Theorem 4.49. *Let A be an $m \times n$ matrix of rank r. Then, there exist a finite number of elementary row and column operations which, when applied to A, produce the matrix*

$$\begin{pmatrix} I_{r\times r} & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{pmatrix}.$$

Proof. We first use row reduction to put A into row-echelon form. So, after this row reduction, the first r rows of A have some zeros, and then a 1 with zeros below this 1. And the remaining $m - r$ rows are all zero. (In case $r = 0$, then we are done, so we may assume that $r > 0$.) Now, the first row of A has some zeros, then a 1 with zeros below this 1. So, by adding copies of the column that contains the entry 1 to each column to the right, the remaining entries of the first row can be made to be zero. And we still keep our matrix in row-echelon form. Now, the second row of A has some zeros, then a 1 with zeros above and below this 1. So, by adding copies of the column that contains this entry 1 to each column to the right, the remaining entries of the second row can be made to be zero. And once again, our matrix is still in row-echelon form. We then continue this procedure. The first r rows then each have exactly one entry of 1, and all remaining entries in the matrix are zero. By swapping columns as needed, A is then put into the required form, as desired. □

Corollary 4.50 (**A Factorization Theorem**)**.** *Let A be an $m \times n$ matrix of rank r. Then, there exists an $m \times m$ matrix B and an $n \times n$ matrix C such that B is the product of a finite number of elementary row operations, C is the product of a finite number of elementary column operations, and such that*

$$A = B \begin{pmatrix} I_{r\times r} & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{pmatrix} C.$$

Lemma 4.51. *Let A be an $m \times n$ matrix. Let B be an $m \times m$ invertible matrix, and let C be an $n \times n$ invertible matrix. Then*

$$\operatorname{rank}(A) = \operatorname{rank}(BA) = \operatorname{rank}(AC) = \operatorname{rank}(BAC).$$

Lemma 4.52. *Let A be an $m \times n$ matrix with rank r. Then A^T also has rank r.*

Lemma 4.53. *Let A be an $n \times n$ matrix. Then A is invertible if and only if it is the product of elementary row and column operations.*

Remark 4.54. Suppose A is an invertible matrix, and we have elementary row operations $E_1, \ldots, E_j$ such that

$$E_1 \cdots E_j A = I_n.$$

Multiplying both sides by A^{-1} on the right,

$$E_1 \cdots E_j I_n = A^{-1}.$$

So, to compute A^{-1} from A, it suffices to find row operations that turn A into the identity. And we then apply these operations to I_n to give A^{-1}. This is the algorithm for computing the inverse A^{-1} that you learned in a previous class.

4.3. **Multiplying Matrices.**

Example 4.55 (**Multiplying Matrices**)**.** The naïve way to multiply two real $n \times n$ matrices requires approximately n^a arithmetic operations where $a = 3$. (The output matrix has n^2 entries, and each entry requires at most $2n$ arithmetic operations, so a total of $2n \cdot n^2$ operations could be needed.) However, there seems to be some redundancy in all of these operations, so one might hope to improve the number of required operations. In fact, $a < 2.3728639$ also possible [Gal14] (building upon Coppersmith-Winograd, Stothers, and Williams.) I do not think the algorithm with such a value of a has been implemented in practice, since the implied constants in its analysis are quite large, and apparently the algorithm does not parallelize. On the other hand, Strassen's algorithm has been implemented, and it has $a = \log 7 / \log 2 \approx 2.807$.

Example 4.56 (**Computing Determinants**)**.** Let $n > 0$ be an integer. Suppose we want to compute the determinant of a real $n \times n$ matrix A with entries A_{ij}, $i, j \in \{1, \ldots, n\}$. An inefficient but straightforward way to do this is to directly use a definition of the determinant. Let S_n denote the set of all permutations on n elements. For any $\sigma \in S_n$, let $\operatorname{sign}(\sigma) := (-1)^j$, where σ can be written as a composition of j transpositions (Exercise: this quantity is well-defined). (A transposition $\sigma \in S_n$ satisfies $\sigma(i) = i$ for at least $n-2$ elements of $\{1, \ldots, n\}$.) Then

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \prod_{i=1}^{n} A_{i\sigma(i)}.$$

This sum has $|S_n| = n!$ terms. So, if we use this formula to directly compute the determinant of A, in the worst case we will need to perform at least $(n+1) \cdot n!$ arithmetic operations. This is quite inefficient. We know a better algorithm from linear algebra class. We first perform row operations on A to make it upper triangular. Suppose B is an $n \times n$ real matrix such that BA represents one single row operation on A (i.e. adding a multiple of one row to another row, or swapping the positions of two rows). Then there are real $n \times n$ matrices $B_1, \ldots, B_m$ such that

$$B_1 \cdots B_m A \qquad (*)$$

is an upper triangular matrix. The matrices $B_1, \ldots, B_m$ can be chosen to first eliminate the left-most column of A under the diagonal, then the second left-most column entries under

the diagonal, and so on. That is, we can choose $m \leq n(n-1)/2$, and each row operation involves at most $3n$ arithmetic operations. So, the multiplication of $(*)$ uses at most

$$3mn \leq 2n^3$$

arithmetic operations. The determinant of the upper diagonal matrix $(*)$ is then the product of its diagonal elements, and

$$\det(B_1 \cdots B_m A) = \det(B_1) \cdots \det(B_m) \det(A).$$

That is,

$$\det(A) = \frac{\det(B_1 \cdots B_m A)}{\det(B_1) \cdots \det(B_m)}.$$

So, $\det(A)$ can be computed with at most $2n^3 + m + n \leq 4n^3 = O(n^3)$ arithmetic operations. Can we do any better?

It turns out that this is possible. Indeed, if it is possible to multiply two $n \times n$ real matrices with $O(n^a)$ arithmetic operations for some $a > 0$, then it is possible to compute the determinant of an $n \times n$ matrix with $O(n^a)$ arithmetic operations.

Remark 4.57. Interestingly, computing the permanent of a matrix

$$\operatorname{per}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i\sigma(i)}$$

is $\#\mathbf{P}$-complete, so we expect this quantity cannot be computed using a polynomial number (in n, e.g. n^3) of arithmetic operations on a computer, even though we can do this for the determinant. However, for any $\varepsilon > 0$, there is a $(1+\varepsilon)$ polynomial time randomized approximation algorithm for computing the permanent of a matrix with nonnegative entries [JSV04]. That is, for any $\varepsilon > 0$, and $0 < \delta < 1$ there is a randomized algorithm such that the following holds. For any $n \times n$ matrix A of nonnegative real numbers, the algorithm runs in time that is polynomial in $1/\varepsilon, n$, and $\log(1/\delta)$, and with probability at least $1 - \delta$ the algorithm outputs a real number p such that

$$p \leq \operatorname{per}(A) \leq (1+\varepsilon)p.$$

On the other hand, for any constant c, the problem of approximating the permanent of an arbitrary matrix A is $\#\mathbf{P}$-hard [Aar11].

4.4. Gaussian Elimination, LU Factorization, Ax=b.

Theorem 4.58 (Gaussian Elimination/ LU Factorization). *Let $\mathbb{F}$ denote $\mathbb{R}$ or $\mathbb{C}$. Let A be an $n \times n$ matrix with values in $\mathbb{F}$. Then there exist $n \times n$ matrices P, L, U such that*

$$PA = LU,$$

where L is lower triangular with ones on its diagonal and values in $\mathbb{F}$, U is upper triangular with values in $\mathbb{F}$, and P is a permutation matrix (i.e. P is the identity matrix with its rows permuted). Moreover, P, L, U can be computed with at most $5n^3$ arithmetic operations.

Remark 4.59. Even in the case $n = 1$, we can see the LU factorization is not unique.

Proof. We first apply a permutation matrix P_1 to A such that the top left entry of P_1A has largest absolute value in its first column. In the case that the first column of A is zero, let L_1 be the identity. Otherwise, let L_1 denote the (lower triangular) row operation matrix such that L_1P_1A has zeros below the top left entry. We now iterate this procedure. Apply a permutation matrix P_2 to L_1P_1A that fixes the first row, and such that the second diagonal entry has largest absolute value among the lowest $n-1$ entries in the second column. When all entries below and including the diagonal are zero in the second column, let L_2 be the identity. Otherwise, let L_2 denote the (lower triangular) row operation matrix (that fixes the first row) such that $L_2P_2L_1P_1A$ has zeros below the diagonal. We continue this procedure. We arrive at an upper triangular matrix U such that

$$L_{n-1}P_{n-1}\cdots L_1P_1A = U.$$

For each $1 \leq k \leq n-1$, define $L'_k := P_{n-1}\cdots P_{k+1}L_kP_{k+1}\cdots P_{n-1}$. Note that $L'_1, \ldots, L'_{n-1}$ are lower triangular. To see this, we first write

$$P_{k+1}L_kP_{k+1} = P_{k+1}(L_k - I + I)P_{k+1} = P_{k+1}(L_k - I)P_{k+1} + I.$$

Now, $L_k - I$ is nonzero only in its k^{th} column below the k^{th} row, and P_{k+1} only permutes rows below the k^{th} row. So, $P_{k+1}(L_k - I)$ also is only nonzero in its k^{th} column below the k^{th} row. Then multiplying on the right by P_{k+1} permutes the columns to the right of the k^{th} column (i.e. it has no effect); in block form we have

$$L_k = \begin{pmatrix} I_{k-1} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & v_k & I_{n-k} \end{pmatrix}, \qquad P_{k+1} = \begin{pmatrix} I_k & 0 \\ 0 & * \end{pmatrix}.$$

Therefore,

$$\begin{aligned} P_{k+1}(L_k - I)P_{k+1} &= \begin{pmatrix} I_k & 0 \\ 0 & * \end{pmatrix} \begin{pmatrix} 0_{k-1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & v_k & 0_{n-k} \end{pmatrix} \begin{pmatrix} I_k & 0 \\ 0 & * \end{pmatrix} \\ &= \begin{pmatrix} 0_{k-1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & v'_k & 0_{n-k} \end{pmatrix} \begin{pmatrix} I_k & 0 \\ 0 & * \end{pmatrix} = \begin{pmatrix} 0_{k-1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & v'_k & 0_{n-k} \end{pmatrix}. \end{aligned}$$

We conclude $P_{k+1}L_kP_{k+1}$ is lower triangular. By a similar argument, $P_{k+2}P_{k+1}L_kP_{k+1}P_{k+2}$ is lower triangular. Iterating this argument, we conclude that $L'_1, \ldots, L'_{n-1}$ are all lower triangular.

By definition of $L'_1, \ldots, L'_{n-1}$, we have

$$U = L_{n-1}P_{n-1}\cdots L_1P_1A = L'_{n-1}\cdots L'_1P_{n-1}\cdots P_1A.$$

Finally, define $L' := L'_{n-1}\cdots L'_1$, and define $P := P_{n-1}\cdots P_1$. We have

$$U = L'PA.$$

So, let $L := (L'_1)^{-1}\cdots(L'_{n-1})^{-1}$, so that $LL' = I$, and

$$LU = LL'PA = PA.$$

(Alternatively, once we have the expression for L', we could just solve directly for its inverse, which is straightforward since L' is lower triangular, and then define $L := (L')^{-1}$ directly.)

Each iteration requires at most n^2 arithmetic operations, and there are $n-1$ such iterations, so completing the computation requires at most n^3 arithmetic operations. $\square$

In the NCM package, the LU factorization is demonstrated with the command `lugui`, with partial pivoting.

Exercise 4.60. Write your own program in Matlab that finds the LU decomposition of a given $n \times n$ real matrix (without using any matrix decomposition programs in Matlab). Then apply your program to the matrix

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 5 & 1 & 0 & 2 \\ -6 & 0 & 1 & 2 & 0 \\ 3 & 0 & 4 & 0 & -1 \end{pmatrix}.$$

Exercise 4.61. Suppose A is an $n \times n$ matrix of the form

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \cdots & 1 & 1 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{pmatrix}$$

Find an LU decomposition of A. Hint: use

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ -1 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \cdots & 1 & 0 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{pmatrix}.$$

What U do you get? Explain why, when n is large, this LU decomposition would lead to a large loss of significance.

Exercise 4.62. Suppose A is an $n \times n$ real matrix with rank n. Let L_1, L_2 be real $n \times n$ lower triangular matrices, and let U_1, U_2 be real $n \times n$ upper triangular matrices. Suppose

$$A = L_1 U_1 = L_2 U_2.$$

Show that there exists a real $n \times n$ diagonal matrix D with nonzero diagonal entries such that

$$L_1 = L_2 D, \qquad U_1 = D^{-1} U_2.$$

In this sense, the LU factorization of A is unique, up to multiplication by D.

Exercise 4.63. Let L be a complex $n \times n$ lower triangular matrix with nonzero diagonal entries. Describe an algorithm that computes L^{-1} with at most $5n^3$ arithmetic operations.

Then, write a program in Matlab that finds L^{-1} for such an $n \times n$ matrix (without using any built-in matrix inversion things in Matlab), and apply your program when L is

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \\ -6 & 0 & 1 & 2 & 0 \\ 1 & 0 & 4 & 0 & 3 \end{pmatrix}.$$

Finally, verify that your computed version of L^{-1} satisfies $LL^{-1} = L^{-1}L = I$ (at least approximately).

(Hint: L^{-1} is also lower triangular. Starting with the top row of L^{-1} and working your way down, what entries must L^{-1} have? Try starting from the diagonal entries and then moving to the left, one entry at a time.)

(Hint: it might be helpful to access portions of a row of a matrix L in Matlab. For example, if $1 \leq j < i \leq n$ are integers, the command `L(i, j+1: i)` is the i^{th} row of L starting from entry $j+1$ and ending at enetry i. And `L(j+1 : i, j)` is the j^{th} column of L, starting at entry $j+1$ and ending at entry i.)

Exercise 4.64 (**Matrix Inversion**)**.** There are a few standard algorithms that invert an invertible matrix. One such algorithm uses the LU decomposition. Suppose A is an invertible matrix. Using the LU decomposition, show that $PA = LU$, with L, U invertible, L lower triangular, U upper triangular, and P a permutation matrix, so that $A = P^T LU$. Then, with Exercise 4.63, describe an algorithm for computing A^{-1} as $U^{-1}L^{-1}P$ that uses at most $20n^3$ arithmetic operations.

As we mentioned in Remark 4.43, the linear system $Ax = b$ can be solved (if a solution exists) by performing elementary row operations on A. These elementary row operations are encoded in the LU decomposition, so the LU decomposition can solve a linear system of equations.

Theorem 4.65 (**Solving Systems of Linear Equations**)**.** *Let A be a complex $n \times n$ matrix. Let $b \in \mathbb{C}^n$. Consider the equation*

$$Ax = b$$

where $x \in \mathbb{C}^n$ is unknown. Let $PA = LU$ be the LU decomposition of A that we found (together with an algorithm) in Theorem 4.58. If a solution $x' \in \mathbb{C}^n$ exists to the equation $Ax' = b$, then some $x \in \mathbb{C}^n$ satisfying $Ax = b$ can be found by solving the following two triangular systems, whose solutions exist

- *First solve for $y \in \mathbb{C}^n$ in the equation $Ly = Pb$,*
- *Then solve for $x \in \mathbb{C}^n$ in the equation $Ux = y$, and output x,*

Remark 4.66. Solving linear triangular systems is easy. Note $LUx = PAx = Pb$.

Proof. First, note that L is lower triangular with ones on its diagonal, so a solution y exists to $Ly = Pb$ (i.e. L is invertible). Since P is invertible, $x \in \mathbb{C}^n$ satisfies $PAx = Pb$ if and only if $Ax = b$. Since $Ly = Pb$, and $PA = LU$, we have $PAx = LUx = Ly$. Since L is invertible, we have $Ux = y$. That is, $Ax = b$ is solvable for x if and only if $Ux = y$ is solvable for x. We assumed that some $x' \in \mathbb{C}^n$ solves $Ax' = b$, so we deduce this same x' solves $Ux' = y$. □

Remark 4.67. Recall that if A is a real $n \times n$ matrix with rank n, and if $b \in \mathbb{R}^n$, then the equation $Ax = b$ can always be solved for some unique $x \in \mathbb{R}^n$. More generally, if A perhaps does not have full rank, then $Ax = b$ can be solved only if b is in the span of the columns of A. In this case, the solution x might not be unique. For this reason, solving for $Ux = y$ in Theorem 4.65 might result in a non-unique solution x to $Ax = b$.

Exercise 4.68. Let

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 5 & 1 & 0 & 2 \\ -6 & 0 & 1 & 2 & 0 \\ 3 & 0 & 4 & 0 & -1 \end{pmatrix}.$$

Using the LU decomposition of A, solve the equation

$$Ax = b,$$

using Matlab code (without using any built-in linear algebra solvers) where $b = (2, 4, 3, 1, 5)^T$. (Note that solving a linear system such as $Ly = b$ when L is lower triangular should be relatively straightforward, by e.g. first solving for y_1, then y_2, and so on.)

Definition 4.69. A self-adjoint $n \times n$ matrix A is said to be **positive semidefinite** if all eigenvalues of A are nonnegative. If additionally all eigenvalues of A are positive, A is called **positive definite**.

Theorem 4.70. *Let A be an $n \times n$ self-adjoint matrix with values in $\mathbb{C}$. Then the following are equivalent.*

(i) *All eigenvalues of A are nonnegative.*
(ii) *There exists an $n \times n$ matrix B with values in $\mathbb{C}$ such that $A = B^*B$.*
(iii) *For any $x \in \mathbb{C}^n \setminus \{0\}$, we have $x^*Ax \geq 0$.*

Moreover, strict equality holds in (i) and (iii) if and only if all eigenvalues of A are positive.

Proof. We will show that (i) implies (ii), (ii) implies (iii), and (iii) implies (i), thereby obtaining the equivalence of all three conditions. In all cases, since A is self-adjoint, the Spectral Theorem 4.34 says $A = QDQ^*$ with D an $n \times n$ diagonal matrix with real entries and Q is unitary $n \times n$. If (i) holds, D has nonnegative entries, $A = Q\sqrt{D}\sqrt{D}Q^* = (Q\sqrt{D})(Q\sqrt{D})^*$, so (ii) holds with $B := (Q\sqrt{D})^*$. If (ii) holds and $x \in \mathbb{C}^n \setminus \{0\}$, then $x^*Ax = x^*B^*Bx = (Bx)^*Bx = \|Bx\|^2 \geq 0$, so (iii) holds. If (iii) holds and if $v \in \mathbb{C}^n \setminus \{0\}$ is an eigenvector of A with eigenvalue $\lambda \in \mathbb{R}$, then $\lambda \|v\|^2 = \lambda v^*v = v^*Av \geq 0$. We conclude that $\lambda \geq 0$ since $v \neq 0$ implies $\|v\| \neq 0$ so that (i) holds. □

Corollary 4.71. *Let $\mathbb{F}$ denote $\mathbb{R}$ or $\mathbb{C}$. Let A be a self-adjoint positive definite $n \times n$ matrix with elements in $\mathbb{F}$. Then there exist $n \times n$ matrices L, U with elements in $\mathbb{F}$ such that*

$$A = LU,$$

where L is lower triangular, U is upper triangular. Moreover, L and U can be computed with at most $5n^3$ arithmetic operations.

Proof. We repeat the proof of Theorem 4.58. Observe first that the top left entry of A must be positive by Theorem 4.70(iii), so we can take $P_1 = I$. Similarly, every other P_k can be taken to be the identity matrix. We prove this by contradiction. Suppose for example that

at step k of the proof of Theorem 4.58, we find that all entries in the k^{th} column of the matrix below and including the k^{th} entry are all zero. If this occurs, then the top left $k \times k$ minor of $L_k \cdots L_1 A$ has a zero row in its k^{th} row. So the vector $x \in \mathbb{R}^n$ with a 1 in the k^{th} entry and a zero in its other entries satisfies

$$\begin{aligned} 0 < x^T L_k \cdots L_1 A L_1^T \cdots L_k^T x &= x^T \begin{pmatrix} A_k & * \\ 0 & 0 \end{pmatrix} L_1^T \cdots L_k^T x \\ &= x^T \begin{pmatrix} 0 & * \\ 0 & 0 \end{pmatrix} L_1^T \cdots L_k^T x = x^T \begin{pmatrix} 0 & * \\ 0 & 0 \end{pmatrix} x = 0. \end{aligned}$$

The first inequality used Theorem 4.70. The penultimate equality used that $L_1^T \cdots L_k^T$ is upper triangular. The last equality used the definition of x. With this contradiction, we conclude we can choose $P = I$ in Theorem 4.58.

□

Exercise 4.72. Write a computer program on your own that finds the LU factorization of the matrix

$$A = \begin{pmatrix} 6 & 0 & -4 & 0 \\ 0 & 7 & 0 & -1 \\ -4 & 0 & 6 & 0 \\ 0 & -1 & 0 & 7 \end{pmatrix}.$$

4.5. **QR Decomposition.** Due to examples of LU factorizations such as Exercise 4.61, the LU decomposition (or equivalently, Gaussian elimination) might not be the best method for solving linear systems of equations. There is another matrix factorization, the QR decomposition, which sometimes behaves better for solving linear systems of equations. The QR decomposition also has other applications not directly related to solving linear systems.

We will construction the QR decomposition iteratively with the following lemma. The idea of the QR decomposition is that, rather than using row operations to force a column of a matrix to have many zeros, we will apply a (complex) rotation to the matrix to force a column to have many zeros.

Lemma 4.73. *Let $e_1 = (1, 0, \ldots, 0)^T$. Let $w \in \mathbb{C}^n$. Then there exists $v \in \mathbb{C}^n$ with $\|v\| = 1$ and there exists $\alpha \in \mathbb{C}$ such that*

$$(I - 2vv^*)w = \alpha e_1.$$

Moreover, $I - 2vv^*$ is a unitary matrix, and we can choose $\alpha := -\|w\| e^{\sqrt{-1}\theta}$ where $\theta \in [0, 2\pi)$ satisfies $w_1 = |w_1| e^{\sqrt{-1}\theta}$, i.e. θ is the angle w_1 makes with the positive real axis.

Proof. First, note that the matrix $I - 2vv^*$ is unitary since $(I - 2vv^*)v = v - 2v\|v\|^2 = -v$ and for any x perpendicular to v, we have $v^*x = 0$ so $(I - 2vv^*)x = x$, i.e. there is an orthonormal basis of $\mathbb{C}^n$ that diagonalizes $I - 2vv^*$ with all eigenvalues 1 or -1.

If $w = 0$, choose $\alpha = 0$, so below, assume $w \neq 0$. Now, we will choose α so that $vv^*w = \frac{1}{2}(w - \alpha e_1)$. Also, we will choose $v := \frac{w - \alpha e_1}{\|w - \alpha e_1\|}$ (we will choose α so the denominator is nonzero). Then

$$vv^*w = \frac{(w - \alpha e_1)(w - \alpha e_1)^*}{\|w - \alpha e_1\|^2} w = (w - \alpha e_1) \frac{\|w\|^2 - \overline{\alpha} w_1}{\|w\|^2 - \alpha \overline{w_1} - w_1 \overline{\alpha} + |\alpha|^2}.$$

We want to choose α so this quantity is $\frac{1}{2}(w - \alpha e_1)$, i.e. we choose α so that

$$2\Big(\|w\|^2 - \overline{\alpha} w_1 \Big) = \|w\|^2 - \alpha \overline{w_1} - w_1 \overline{\alpha} + |\alpha|^2 . \qquad (**)$$

That is, we choose α so that

$$\|w\|^2 = -\alpha \overline{w_1} + w_1 \overline{\alpha} + |\alpha|^2 = 2\mathrm{Im}(w_1 \overline{\alpha}) + |\alpha|^2 .$$

We can then choose $\alpha \in \mathbb{C}$ so that the imaginary part of $w_1 \overline{\alpha}$ is zero, and such that $|\alpha| = \|w\|$. For example, if $w_1 = re^{i\theta}$ where $i = \sqrt{-1}$, $r > 0$ and $\theta \in [0, 2\pi)$, then we can choose

$$\alpha := - \|w\| e^{i\theta},$$

so that $(**)$ holds and $\overline{\alpha} w_1 = -r \|w\|$. We now verify (recalling $w \neq 0$) that

$$\|w - \alpha e_1\|^2 = 2(\|w\|^2 - \overline{\alpha} w_1) = 2(\|w\|^2 + \|w\| r) > 0,$$

so that we have not divided by zero in the definition of α, so that $(*)$ holds as required. □

Theorem 4.74 (QR Factorization). *Let $\mathbb{F}$ denote $\mathbb{R}$ or $\mathbb{C}$. Let A be an $m \times n$ matrix with values in $\mathbb{F}$, with $m \geq n$. Then there exists an $m \times m$ unitary matrix Q (with values in $\mathbb{F}$) and an $m \times n$ upper triangular matrix R (with values in $\mathbb{F}$) such that*

$$A = QR.$$

Moreover, Q is obtained by applying $n-1$ unitary matrices to the columns of A, and at most $8n^2m$ arithmetic operations are required to compute the matrices Q and R.

Proof. Let w be the first column of A. We would like to apply a rotation (i.e. a unitary matrix) to A that rotates w into a vector with all entries zero except for the first entry. The unitary matrix will be $I - 2vv^*$ for some $v \in \mathbb{C}^m$ with $\|v\| = 1$. This is possible by Lemma 4.73. We then have a unitary matrix $U_1 = I - 2vv^*$ such that

$$U_1 A = \begin{pmatrix} a_1 & * \\ 0 & A_2 \end{pmatrix},$$

where A_2 is an $(m-1) \times (n-1)$ matrix. We can then iterate this procedure, finding a vector $v_2 \in \mathbb{C}^{m-1}$ such that $I - 2v_2 v_2^*$ and such that $(I - 2v_2 v_2^*)A_2$ has zeros below the first entry of its first column. Define then

$$U_2 := \begin{pmatrix} 1 & 0 \\ 0 & I - 2v_2 v_2^* \end{pmatrix}.$$

Then $U_2 U_1 A$ is upper triangular with zeros below its first two diagonal entries. After iterating this procedure $n-1$ times, we have found $m \times m$ unitary matrices $U_1, \ldots, U_{n-1}$ such that $R := U_{n-1} \cdots U_1 A$ is upper triangular. Define $Q := U_1^* \cdots U_{n-1}^*$, so that $A = QR$.

Since $(I - 2vv^*)w = w - 2v(v^*w) = w - v(2)(v^*w)$, computing $(I - 2vv^*)w$ requires at most $4m$ arithmetic operations, so that $(I - 2vv^*)A$ requires at most $4nm$ operations. Iterating $n-1$ times, we require at most $4n^2m$ operations to compute R. Since $(I - 2vv^*)^* = I - 2vv^*$, each of the unitary matrices $U_1, \ldots, U_{n-1}$ is also self-adjoint, so $Q = U_1 \cdots U_{n-1}$, and doing that multiplication also requires at most $4n^2m$ operations, for a total of at most $8n^2m$ operations. □

The above algorithm using Lemma 4.73 is preferred in practice. The QR decomposition can also be constructed via the Gram-Schmidt orthogonalization. However, the subtractions in the Gram-Schmidt procedure lead to loss of significance errors and instability.

Proof. Denote the columns of A as $A_1, \ldots, A_n$, and denote the output of Theorem 4.23 as $Q_1, \ldots, Q_n$. Let Q denote the matrix with columns $Q_1, \ldots, Q_n$. Define $r_{ij} := \langle A_j, Q_i \rangle$. Since $Q_1, \ldots, Q_n$ are an orthonormal basis of $\mathbb{C}^n$, we have by Theorem 4.20, for each $1 \leq j \leq n$,

$$A_j = \sum_{i=1}^{n} \langle A_j, Q_i \rangle Q_i = \sum_{i=1}^{n} r_{ij} Q_i.$$

That is, $a_{kj} = \sum_{i=1}^{n} q_{ki} r_{ij}$ for all $1 \leq j \leq m$, $1 \leq k \leq n$. That is, $A = QR$.

By the definition of the Gram-Schmidt procedure $r_{ii} > 0$ for all $1 \leq i \leq n$. Also, by definition of the Gram-Schmidt Orthogonalization, if $1 \leq j < i \leq n$, then Q_i is orthogonal to A_j, so that $r_{ij} = \langle Q_i, A_j \rangle = 0$. That is, R is upper triangular.

Lastly, note that the k^{th} step of the Gram-Schmidt procedure uses at most $5km$ arithmetic operations, so n total steps results in at most $n(n+1)(5/2)m$ arithmetic operations, with one final normalization step using at most $2nm$ operations, for a total of at most $5n^2m$ operations. □

Lemma 4.75. *Let A be a complex $n \times n$ matrix with rank n. Then there is a unique way to write a QR decomposition $A = QR$ where R has nonnegative diagonal entries.*

Proof. Suppose $A = QR = Q_0 R_0$. Since A has rank n, R is invertible. Then $Q_0^* Q = R_0 R^{-1}$. The matrix on the left is unitary and the matrix on the right is upper triangular with nonnegative diagonal entries. So, we must have $R_0 R^{-1} = I$, so that $R_0 = R$ and similarly $Q = Q_0$. □

Theorem 4.70(ii) says that a self-adjoint positive definite matrix can be written as $A = B^*B$, and this factorization is useful for many applications. However, Theorem 4.70(ii) was not constructive. Below, we describe an algorithm for finding this decomposition.

Theorem 4.76 (Cholesky Decomposition). *Let $\mathbb{F}$ denote $\mathbb{R}$ or $\mathbb{C}$. Let A be an $n \times n$ self-adjoint positive definite matrix with values in $\mathbb{F}$. Then there exists an $n \times n$ upper triangular matrix B with elements in $\mathbb{F}$ such that*

$$A = B^*B.$$

Moreover, $B = \sqrt{(U^*)^{-1}L}U$ where $A = LU$ is an LU decomposition of A (from Corollary 4.71), so that B can be computed from Corollary 4.71 and Exercise 4.63 below. Lastly, the Cholesky decomposition $A = B^*B$ is unique up to multiplication of B by a diagonal matrix with entries with absolute value 1.

Proof. From Corollary 4.71, we can write $A = LU$ where L is lower triangular, and U is upper triangular. Since A is self-adjoint, we have

$$LU = A = A^* = U^*L^*.$$

Since A is positive definite, U, L are invertible (otherwise $0 < \det(A) = \det(L)\det(U) = 0$). So,

$$(U^*)^{-1}L = L^*U^{-1}.$$

The matrix on the left is lower triangular and the matrix on the right is upper triangular. Therefore, both of these matrices must be diagonal. That is, there is a diagonal matrix D such that $D = (U^*)^{-1}L$, i.e. $L = U^*D$. Then

$$A = LU = U^*DU. \qquad (\ddagger)$$

Then $D = (U^*)^{-1}AU^{-1} = (U^{-1})^*AU^{-1}$. From Theorem 4.70, for any $x \in \mathbb{F}^n \setminus \{0\}$,

$$x^*Dx = (U^{-1}x)^*AU^{-1}x > 0.$$

So, D is diagonal with positive values. Defining $\sqrt{D}$ as the diagonal matrix with entries the square roots of the entries of D, (‡) becomes

$$A = U^*\sqrt{D}\sqrt{D}U = (\sqrt{D}U)^*\sqrt{D}U.$$

We then define $B := \sqrt{D}U$. Since $B = \sqrt{(U^*)^{-1}L}U$, Corollary 4.71 and Exercise 4.63 complete the algorithm for computing B.

To see the uniqueness of the Cholesky decomposition, write $A = B^*B = C^*C$ with C, B lower triangular. Then $(CB^{-1})^*CB^{-1} = I$, so that CB^{-1} is a lower triangular, unitary matrix, i.e. $CB^{-1} = D$ where D is a diagonal matrix with entries with absolute value 1, so that $C = DB$. □

4.6. **Matrix Norms as a Measure of Error.** Norms are used in numerical analysis to bound the errors in matrix computations.

An $n \times m$ matrix can be treated as a vector of length nm, so that a set of matrices can be equipped with a vector norm.

However, there are also other natural norms on the set of matrices, e.g. ones related to eigenvalues or singular values of the matrix, which we describe further below.

Definition 4.77 (**Singular Values**)**.** Let A be an $m \times n$ complex matrix. Then the **singular values** of A are the square roots of the eigenvalues of A^*A. (Theorem 4.70 says the eigenvalues of A^*A are nonnegative.)

Definition 4.78 (**Vector ℓ_p Norms**)**.** Let $1 \le p < \infty$. Let $x \in \mathbb{C}^n$. Define the ℓ_p norm of x to be

$$\|x\|_p := \Big(\sum_{i=1}^{n} |x_i|^p\Big)^{1/p}.$$

Also define the ℓ_∞ norm of x to be

$$\|x\|_\infty := \max_{1\le i\le n} |x_i|.$$

Proposition 4.79. *The ℓ_p norm is a norm for any $1 \le p \le \infty$, i.e. the definition of a norm from Definition 4.8 holds.*

Proof. We will show the triangle inequality holds. The case $p = \infty$ follows from the scalar triangle inequality, so assume $1 \le p < \infty$. Let $x, y \in \mathbb{C}^n$. We need to show that $\|x + y\|_p \le \|x\|_p + \|y\|_p$. By scaling, we may assume $\|x\|_p = 1 - t$, $\|y\|_p = t$, for some $t \in (0, 1)$ (zeros and infinities being trivial). Define $v := x/(1 - t)$, $w := y/t$. Then by convexity of $s \mapsto |s|^p$ on $\mathbb{R}$,

$$|(1 - t)v_i + tw_i|^p \le (1 - t)\,|v_i|^p + t\,|w_i|^p, \qquad \forall\, 1 \le i \le n$$

Summing over $1 \le i \le n$, we get

$$\|x + y\|_p^p \le (1 - t)\,\|v\|_p^p + t\,\|w\|_p^p = (1 - t) + t = 1.$$

So, $\|x + y\|_p \le 1 = \|x\|_p + \|y\|_p$. □

Definition 4.80 (**Standard Inner Product**)**.** Let $x, y \in \mathbb{C}^n$. Define the standard inner product of x and y by

$$\langle x, y \rangle := \sum_{i=1}^{n} x_i \overline{y_i}.$$

One can check that the standard inner product is an inner product, i.e. Definition 4.10 holds in this case.

Theorem 4.81 (**Hölder's Inequality**)**.** *Let $1 \leq p \leq \infty$, and let q be dual to p (so $1/p + 1/q = 1$). Let $x, y \in \mathbb{C}^n$. Then*

$$|\langle x, y \rangle| \leq \|x\|_p \|y\|_q .$$

*The case $p = q = 2$ recovers the **Cauchy-Schwarz** inequality:*

$$|\langle x, y \rangle| \leq \|x\|_2 \|y\|_2 .$$

Proof. By scaling, we may assume $\|x\|_p = \|y\|_q = 1$ (zeros and infinities being trivial). Also, the case $p = 1, q = \infty$ follows from the triangle inequality, so we assume $1 < p < \infty$. From concavity of the log function, we have

$$|x_i y_i| = (|x_i|^p)^{1/p} (|y_i|^q)^{1/q} \leq \frac{1}{p} |x_i|^p + \frac{1}{q} |y_i|^q , \qquad \forall\, 1 \leq i \leq n.$$

Summing over $1 \leq i \leq n$, we get $|\langle x, y \rangle| \leq \frac{1}{p} + \frac{1}{q} = 1 = \|x\|_p \|y\|_q$. □

Corollary 4.82 (**Duality for ℓ_p Norms**)**.** *Let $1 \leq p \leq \infty$, and let q be dual to p (so $1/p + 1/q = 1$). Let $y \in \mathbb{C}^n$. Then*

$$\|y\|_p = \sup_{x \in \mathbb{R}^n :\, \|x\|_q \leq 1} |\langle x, y \rangle| .$$

Proof. Hölder's inequality, Theorem 4.81, implies that $\sup_{x \in \mathbb{R}^n :\, \|x\|_q \leq 1} |\langle x, y \rangle| \leq \|y\|_p$. To get the other corresponding inequality, consider $x \in \mathbb{C}^n$ defined by $x_i := \|y\|_p^{-(p-1)} \, \overline{y_i} \, |y_i|^{p-2} \, 1_{y_i \neq 0}$ for all $1 \leq i \leq n$. Since $1/p + 1/q = 1$, $p + q = pq$, and $q(p-1) = p$, so

$$\|x\|_q^q = \|y\|_p^{-q(p-1)} \sum_{i=1}^{n} |y_i|^{q(p-1)} = \|y\|_p^{-p} \sum_{i=1}^{n} |y_i|^p = \|y\|_p^{p-p} = 1.$$

$$\langle x, y \rangle = \|y\|_p^{-(p-1)} \sum_{i=1}^{n} |y_i|^p = \|y\|_p .$$

Therefore, $\sup_{x \in \mathbb{R}^n :\, \|x\|_q \leq 1} |\langle x, y \rangle| \geq \|y\|_p$. □

A natural class of matrix norms is defined in analogy with Corollary 4.82.

Definition 4.83. Let A be an $m \times n$ complex matrix. Let $1 \leq p, q \leq \infty$. Define the $p \to q$ norm of A to be

$$\|A\|_{p \to q} := \sup_{x \in \mathbb{C}^n :\, \|x\|_p \leq 1} \|Ax\|_q .$$

Proposition 4.84. *The $p \to q$ norm is a norm for any $1 \leq p, q \leq \infty$, i.e. the definition of a norm from Definition 4.8 holds.*

Proof. We will show the triangle inequality holds. Let A, B be $m \times n$ complex matrices. Fix $x \in \mathbb{C}^n$ with $\|x\|_p \leq 1$. From the triangle inequality for the ℓ_q norm, $\|(A+B)x\|_q \leq \|Ax\|_q + \|Bx\|_q$. Taking the supremum over x on both sides, we get $\|A+B\|_{p\to q} \leq \|A\|_{p\to q} + \|B\|_{p\to q}$. □

The supremum definition makes these norms difficult to compute directly. Still, in certain cases, Corollary 4.82 can give simpler expressions for these norms.

Exercise 4.85. Let A be an $m \times n$ complex matrix. Show the following

- $\|A\|_{1\to 1} = \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}|$.
- $\|A\|_{\infty\to\infty} = \max_{1\leq i\leq n} \sum_{j=1}^{m} |a_{ij}|$.
- $\|A\|_{2\to 2}^2$ is equal to the largest eigenvalue of AA^* (or of A^*A). That is, $\|A\|_{2\to 2}$ is the largest singular value of A.
- For any $1 \leq p, q \leq \infty$, $\|AB\|_{p\to q} \leq \|A\|_{p\to q} \|B\|_{p\to q}$.

Theorem 4.86 ([GVL13, Theorem 3.3.1])**.** *Let $\|\cdot\|$ denote any matrix norm. Let A be an $n \times n$ complex matrix. Assume that A has an LU factorization of the form $A = LU$, and the diagonal entries of L and U are all positive. Assume*

- *All operations use normal floating point numbers.*
- *For any floating point numbers x, y used in the algorithm, for any operation $\odot$ among addition, subtraction, multiplication, and division, we have*

$$\mathrm{fl}(x \odot y) = x \odot_{\mathrm{fl}} y.$$

  Here $\odot_{\mathrm{fl}}$ denotes the floating point implementation of an operation such as addition.
- *For any floating point numbers x, y used in the algorithm, for any operation $\odot$ among addition, subtraction, multiplication, and division, there exists $\delta \in \mathbb{R}$ with $|\delta| \leq \varepsilon$ such that*

$$x \odot_{\mathrm{fl}} y = (x \odot y)(1+\delta).$$

Then the algorithm in Theorem 4.58 outputs a factorization $\widetilde{L}, \widetilde{U}$ such that

$$\|A - \widetilde{L}\widetilde{U}\| \leq 3(1-(1+2^{-52})^n)\Big(\|A\| + \|L\|\,\|U\|\Big).$$

Recall from Exercise 4.61 that $\|L\|$ or $\|U\|$ can be exponentially large in n, in which case this theorem has limited significance.

4.7. **Eigenvalues and the Power Method.**

Exercise 4.87 (The Power Method). This exercise gives an algorithm for finding the eigenvectors and eigenvalues of a symmetric matrix. In modern statistics, this is often a useful thing to do. The Power Method described below is not the best algorithm for this task, but it is perhaps the easiest to describe and analyze.

Let A be an $n \times n$ real symmetric matrix. Let $\lambda_1 \geq \cdots \geq \lambda_n$ be the (unknown) eigenvalues of A, and let $v_1, \ldots, v_n \in \mathbb{R}^n$ be the corresponding (unknown) eigenvectors of A such that $\|v_i\| = 1$ and such that $Av_i = \lambda_i v_i$ for all $1 \leq i \leq n$.

Given A, our first goal is to find v_1 and λ_1. For simplicity, assume that $1/2 < \lambda_1 < 1$, and $0 \leq \lambda_n \leq \cdots \leq \lambda_2 < 1/4$. Suppose we have found a vector $v \in \mathbb{R}^n$ such that $\|v\| = 1$ and $|\langle v, v_1\rangle| > 1/n$. (An exercise more suitable for a probability class shows that a randomly

chosen v satisfies this property, with probability at least 1/2.) Let k be a positive integer. Show that

$$A^k v$$

approximates v_1 well as k becomes large. More specifically, show that for all $k \geq 1$,

$$\left\| A^k v - \langle v, v_1 \rangle \lambda_1^k v_1 \right\|^2 \leq \frac{n-1}{16^k}.$$

(Hint: use the spectral theorem for symmetric matrices.)

Since $|\langle v, v_1 \rangle| \, \lambda_1^k > 2^{-k}/n$, this inequality implies that $A^k v$ is approximately an eigenvector of A with eigenvalue λ_1. That is, by the triangle inequality,

$$\left\| A(A^k v) - \lambda_1 (A^k v) \right\| \leq \left\| A^{k+1} v - \langle v, v_1 \rangle \lambda_1^{k+1} v_1 \right\| + \lambda_1 \left\| \langle v, v_1 \rangle \lambda_1^k v_1 - A^k v \right\| \leq 2 \frac{\sqrt{n-1}}{4^k}.$$

Moreover, by the reverse triangle inequality,

$$\left\| A^k v \right\| = \left\| A^k v - \langle v, v_1 \rangle \lambda_1^k v_1 + \langle v, v_1 \rangle \lambda_1^k v_1 \right\| \geq \frac{1}{n} 2^{-k} - \frac{\sqrt{n-1}}{4^k}.$$

In conclusion, if we take k to be large (say $k > 10 \log n$), and if we define $z := A^k v$, then z is approximately an eigenvector of A, that is

$$\left\| A \frac{A^k v}{\|A^k v\|} - \lambda_1 \frac{A^k v}{\|A^k v\|} \right\| \leq 4 n^{3/2} 2^{-k} \leq 4 n^{-4}.$$

And to approximately find the first eigenvalue λ_1, we simply compute

$$\frac{z^T A z}{z^T z}.$$

That is, we have approximately found the first eigenvector and eigenvalue of A.

Remarks. To find the second eigenvector and eigenvalue, we can repeat the above procedure, where we start by choosing v such that $\langle v, v_1 \rangle = 0$, $\|v\| = 1$ and $|\langle v, v_2 \rangle| > 1/(10\sqrt{n})$. To find the third eigenvector and eigenvalue, we can repeat the above procedure, where we start by choosing v such that $\langle v, v_1 \rangle = \langle v, v_2 \rangle = 0$, $\|v\| = 1$ and $|\langle v, v_3 \rangle| > 1/(10\sqrt{n})$. And so on.

Google's PageRank algorithm uses the power method to rank websites very rapidly. In particular, they let n be the number of websites on the internet (so that n is roughly 10^9). They then define an $n \times n$ matrix C where $C_{ij} = 1$ if there is a hyperlink between websites i and j, and $C_{ij} = 0$ otherwise. Then, they let B be an $n \times n$ matrix such that B_{ij} is 1 divided by the number of 1's in the i^{th} row of C, if $C_{ij} = 1$, and $B_{ij} = 0$ otherwise. Finally, they define

$$A = (.85)B + (.15)D/n$$

where D is an $n \times n$ matrix all of whose entries are 1.

The power method finds the eigenvector v_1 of A, and the size of the i^{th} entry of v_1 is proportional to the "rank" of website i.

Exercise 4.88. Consider the following symmetric real matrix

$$A = \begin{pmatrix} 5 & 1 & -2 & 3 & 1 \\ 1 & 3 & 6 & 0 & 0 \\ -2 & 6 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 & 2 \\ 1 & 0 & 1 & 2 & 3 \end{pmatrix}.$$

Using the power method (i.e. by examining large powers of A in Matlab), find the largest eigenvalue $\lambda \in \mathbb{R}$ of A and a corresponding eigenvector $v \in \mathbb{R}^5$ with $\|v\|_2 = 1$.

Note that $(A - \lambda vv^T)v = Av - \lambda v = 0$, and if w is any other eigenvector of A, then $(A - \lambda vv^T)w = Aw$. Using this observation, apply the power method to $A - \lambda vv^T$ to find the second largest eigenvalue of A.

Finally, compare your results with the built-in Matlab function `eigs`.

4.8. **Eigenvalues and the QR Algorithm.** The power method just described can find the first few eigenvalues and eigenvectors of a self-adjoint matrix relatively quickly. However, finding all eigenvalues and eigenvectors with this method can be costly. Thankfully, there is an efficient way to find all eigenvalues and eigenvectors of a matrix simultaneously, using a cleverly chosen sequence of QR decompositions.

Algorithm 4.89 (**QR Algorithm for Eigenvalues**)**. Input**: A symmetric $n \times n$ real matrix A (or a self-adjoint complex matrix A), a number of iterations k.
Output: An $n \times n$ matrix D' whose diagonal entries approximate the eigenvalues of A.

Define $A_0 := A$. For each $1 \leq j \leq k$, do the following.

- Write A_{j-1} in its QR Factorization as $A_{j-1} =: Q_j R_j$ (using the algorithm from Theorem 4.74, which uses either Householder reflections (i.e. Lemma 4.73) or Gram-Schmidt orthogonalization).
- Define $A_j := R_j Q_j$.

Output $D' := A_k$.

Theorem 4.90. *Let A be a real symmetric $n \times n$ positive definite matrix with distinct eigenvalues $\lambda_1 > \cdots > \lambda_n > 0$. (From the spectral theorem 4.34, write $A = QDQ^{-1}$ where Q is an orthogonal matrix.) Assume that D is ordered so that $D_{ii} = \lambda_i$ for all $1 \leq i \leq n$. Assume that Q^T has an LU decomposition $Q^T = LU$ where the diagonal entries of U are positive.*

Then as $k \to \infty$, the sequence of matrices $A_1, A_2, \ldots$ in Algorithm 4.89 converges to D, and $Q_1 \cdots Q_k$ converges to Q.

Proof. Note that $A = A_0 = Q_1 R_1$ and

$$A^2 = Q_1 R_1 Q_1 R_1 = Q_1 A_1 R_1 = Q_1 Q_2 R_2 R_1.$$

More generally, we can prove by induction on k that

$$A^k = Q_1 \cdots Q_k R_k \cdots R_1. \qquad (\ddagger)$$

Since $A = QDQ^{-1}$, $A^k = QD^kQ^{-1}$, so recalling $Q^T = LU$, so $L = Q^T U^{-1}$,

$$QD^k L D^{-k} = QD^k Q^T U^{-1} D^{-k} = A^k U^{-1} D^{-k} \overset{(\ddagger)}{=} Q_1 \cdots Q_k R_k \cdots R_1 U^{-1} D^{-k}. \qquad (**)$$

Recall that the diagonal entries of L are all 1. For any $1 \leq i, j \leq n$, note that

$$(D^k L D^{-k})_{ij} = \begin{cases} 1 & \text{, if } i = j \\ L_{ij}\left(\frac{\lambda_i}{\lambda_j}\right)^k & \text{, if } i > j \\ 0 & \text{, otherwise.} \end{cases}$$

Since $\lambda_i < \lambda_j$ when $i > j$, $D^k L D^{-k}$ converges to the identity matrix as $k \to \infty$. So, $Q D^k L D^{-k}$ converges to Q as $k \to \infty$, and $(**)$ implies that $Q_1 \cdots Q_k R_k \cdots R_1 U^{-1} D^{-k}$ converges to Q as $k \to \infty$. The matrix Q itself has a unique QR factorization as $Q = Q \cdot I$ with nonnegative diagonal entries on the second term (by Lemma 4.75). So, as $k \to \infty$, $Q_1 \cdots Q_k$ converges to Q (the diagonal entries of D and D^{-1} are positive). Multiplying both sides by Q_k^{-1}, $Q_1 \cdots Q_{k-1}$ converges to $Q Q_k^{-1}$ as $k \to \infty$ as well. So, as $k \to \infty$, Q_k converges to I. From Algorithm 4.89, $A_k = Q_k R_k$. It also follows by induction that A_k is symmetric and similar to $A = A_0$ (we know $A = A_0$ is symmetric, and Algorithm 4.89 says $A_k = R_k Q_k = Q_k^T Q_k R_k Q_k = Q_k^T A_{k-1} Q_k$.) Since $A_k = Q_k R_k$, A_k is symmetric, and Q_k converges to I as $k \to \infty$, it follows that A_k converges to a diagonal matrix as $k \to \infty$. Since A_k is similar to A, as $k \to \infty$, A_k converges to a diagonal matrix whose elements are the eigenvalues of A. Since $A_k = (Q_1 \cdots Q_k)^T A (Q_1 \cdots Q_k)$ and $Q_1 \cdots Q_k$ converges to Q as $k \to \infty$, we must have A_k converging to D as $k \to \infty$, since $A = Q D Q^T$, i.e. $Q^T A Q = D$. □

Remark 4.91. This argument shows that A_k converges exponentially fast to a diagonal matrix whose elements are the eigenvalues of A, in theory.

Exercise 4.92. Consider the following symmetric real matrix

$$A = \begin{pmatrix} 5 & 1 & -2 & 3 & 1 \\ 1 & 3 & 6 & 0 & 0 \\ -2 & 6 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 & 2 \\ 1 & 0 & 1 & 2 & 3 \end{pmatrix}.$$

Using the QR algorithm, find all eigenvalues and eigenvectors of A.

Finally, compare your results with the built-in Matlab function `eigs`.

In order to decrease the computation time in the QR algorithm, the matrix A can be pre-processed into a similar tridiagonal matrix. This step can be done by a modification of the QR factorization. If A is a symmetric real $n \times n$ matrix, and $w \in \mathbb{R}^{n-1}$ is the lowest $n-1$ entries of the first column of A, then Lemma 4.73 says we can find $v \in \mathbb{C}^{n-1}$ such that the $(n-1) \times (n-1)$ unitary matrix $I - 2vv^*$ satisfies $(I - 2vv^*)w = \alpha e_1$. So, if

$$Q := \begin{pmatrix} 1 & 0 \\ 0 & I - 2vv^* \end{pmatrix},$$

then QA has a first column with zeros below its first two entries. But then QAQ^T also has a first column with zeros below its first two entries, since multiplying on the right by Q^T has no effect on those zero entries. Since A is symmetric, QAQ^T then must have zeros in its

first row after its first two entries. In summary, QAQ^T has the form

$$QAQ^T = \begin{pmatrix} * & * & 0 & \cdots & 0 \\ * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & \cdots & * \end{pmatrix}$$

We can then iterate this procedure, letting w be the lowest $n-2$ entries of the second column of QAQ^T. After $n-1$ iterations, we obtain a tridiagonal matrix QAQ^T. The NCM package command `eigsvdgui` illustrates this procedure.

4.9. **Least Squares.** In Theorem 4.65, we used the LU decomposition of a square matrix A to solve the equation $Ax = b$, if a solution x exists. If A does not have full rank, then a solution to the equation $Ax = b$ might not exist. In such a case, we still might want to find a vector x that "most closely" solves the equation $Ax = b$. More precisely, we want to find a vector x that minimizes the quantity $\|Ax - b\|_2$, or equivalently, $\|Ax - b\|_2^2$.

Definition 4.93 (Least Squares Problem). Let A be an $m \times n$ real matrix. Let $b \in \mathbb{R}^m$. The **least squares** problem asks for a vector $x \in \mathbb{R}^n$ minimizing

$$\|Ax - b\|_2^2 .$$

Example 4.94. Suppose we have data points $(a_1, b_1), \ldots, (a_m, b_m) \in \mathbb{R}^2$ and we would like to find the "best fit" line to the data. More specifically, we would like to find $x_0, x_1 \in \mathbb{R}$ such that the linear function

$$f(t) := x_0 + x_1 t, \qquad \forall\, t \in \mathbb{R}$$

minimizes the sum of squared differences

$$\sum_{i=1}^{m} [f(a_i) - b_i]^2 = \sum_{i=1}^{m} [x_0 + x_1 a_i - b_i]^2.$$

We can rewrite this sum of squares in matrix form as

$$\|Ax - b\|^2$$

where $x = \binom{x_0}{x_1}$, $b = (b_1, \ldots, b_m)^T$, and

$$A = \begin{pmatrix} 1 & a_1 \\ 1 & a_2 \\ \vdots & \vdots \\ 1 & a_m \end{pmatrix}.$$

So, finding the "best fit" line is a special case of the least squares minimization problem.

More generally, let $k \geq 1$ be an integer, and suppose we would like to we would like to find the "best fit" degree k polynomial to the data, i.e. we would like to find $x_0, \ldots, x_k \in \mathbb{R}$ such that the polynomial

$$f(t) := x_0 + x_1 t + \cdots + x_k t^k, \qquad \forall\, t \in \mathbb{R}$$

minimizes the sum of squared differences

$$\sum_{i=1}^{m}[f(a_i)-b_i]^2=\sum_{i=1}^{m}[x_0+x_1a_i+\cdots+x_ka_i^k-b_i]^2.$$

We can rewrite this sum of squares in matrix form as

$$\|Ax-b\|^2$$

where $x=(x_0,\ldots,x_k)^T$, $b=(b_1,\ldots,b_m)^T$, and A is a Vandermonde matrix

$$A=\begin{pmatrix}1 & a_1 & a_1^2 & \cdots & a_1^k\\ 1 & a_2 & a_2^2 & \cdots & a_2^k\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & a_m & a_m^2 & \cdots & a_m^k\end{pmatrix}.$$

As before, finding the "best fit" degree k polynomial is a special case of the least squares minimization problem.

Lemma 4.95. *Let A be an $m\times n$ real matrix. Let $b\in\mathbb{R}^m$.*
The vector $x\in\mathbb{R}^n$ minimizes $\|Ax-b\|$ if and only if $A^TAx=A^Tb$.

Proof. Let $t\in\mathbb{R}$ and consider the function $f\colon\mathbb{R}\to\mathbb{R}$ defined by

$$f(t):=\|A(x+ty)-b\|_2^2=\|Ax-b+tAy\|_2^2=\|Ax-b\|_2^2+2ty^TA^T(Ax-b)+t^2\|Ay\|_2^2.$$

This function is quadratic in t, and a minimum occurs at $t=0$ if and only if $y^TA^T(Ax-b)=0$ for all $y\in\mathbb{R}^n$, i.e. when $A^T(Ax-b)=0$. □

Exercise 4.96. Let A be an $m\times n$ real matrix with $m\geq n$. Show that A has rank n if and only if A^TA is positive definite.
(Hint: A^TA is always positive semidefinite.)

Lemma 4.95 and Exercise 4.96 together imply the following.

Proposition 4.97. *Let A be an $m\times n$ real matrix with $m\geq n$. Suppose A has rank n. Then there is a unique solution of the least squares problem given by*

$$x=(A^TA)^{-1}A^Tb.$$

*In this case, the matrix $(A^TA)^{-1}A^T$ is called the **pseudoinverse** of A. (When A is a non-square matrix, it will not have an inverse, but $(A^TA)^{-1}A^TA=I$.)*

Explicitly inverting a matrix is generally not advisable. For this reason, when m,n are large, it is not a good idea to use Proposition 4.97 and its explicit formula for the x minimizing the least squares problem. We can instead minimize a full rank least squares problem using the two methods described below.

Theorem 4.98 (Least Squares via Cholesky Decomposition). *Let A be an $m\times n$ real matrix with $m\geq n$. Suppose A has rank n. The unique solution of the least squares problem can be found in the following way*

- *Find the Cholesky decomposition L^*L of A^*A, where L is an $n\times n$ lower triangular real matrix (using the algorithm from Theorem 4.76, which uses an LU factorization of A^*A.)*
- *Solve the equation $L^*y=A^*b$ for $y\in\mathbb{R}^n$.*

- *Solve the equation $Lx = y$ for $x \in \mathbb{R}^n$.*

Proof. From Lemma 4.95 and Proposition 4.97, there is a unique solution $x \in \mathbb{R}^n$ to the equation $A^*Ax = A^*b$. By assumption, we have $L^*Lx = A^*b$. Since A^*A is positive definite by Exercise 4.96, L is invertible, so there is a unique solution y to the equation $L^*y = A^*b$. Similarly, there is a unique solution x' to $Lx' = y$. Once these equations are solved, we have $A^*b = L^*y = L^*Lx'$. That is, $x = x'$. □

Theorem 4.99 (**Least Squares via QR Decomposition**). *Let A be an $m \times n$ real matrix with $m \geq n$. Suppose A has rank n. The unique solution of the least squares problem can be found in the following way*

- *Find the QR decomposition QR of A, (using the algorithm from Theorem 4.74, which uses either Householder reflections (i.e. Lemma 4.73) or Gram-Schmidt orthogonalization.)*
- *Solve the equation $Rx = Q^*b$ for $x \in \mathbb{R}^n$.*

Proof. From Lemma 4.95 and Proposition 4.97, there is a unique solution $x \in \mathbb{R}^n$ to the equation $A^*Ax = A^*b$. Since $A^*A = R^*Q^*QR = R^*R$, we can rewrite the first equation as $R^*Rx = R^*Q^*b$. Since R is $n \times n$ and upper triangular, and A has rank n, R is invertible, so we can rewrite the equation as $Rx = Q^*b$. □

Exercise 4.100. Suppose we have data points $(0,1), (1,3), (2,3), (3,5), (4,2) \in \mathbb{R}^2$ denoted as $\{(a_i, b_i)\}_{i=1}^5$. Find the line that best fits the data. That is, find the line $f\colon \mathbb{R} \to \mathbb{R}$ that minimizes the sum of squared differences $\sum_{j=1}^5 |f(a_i) - b_i|^2$. Use either a Cholesky decomposition or a QR decomposition, with your own method written in Matlab (i.e. don't use any built-in Matlab matrix decomposition functions).

Then, find the best fit degree two polynomial, and the best fit degree three polynomial to these data points.

4.10. **Singular Value Decomposition (SVD).** Recall the definition of singular values in Definition 4.77.

Remark 4.101. If $m = n$, if A is self-adjoint, and if $\lambda \in \mathbb{R}$ is an eigenvalue of A with eigenvector $v \in \mathbb{C}^n$, then $A^*Av = A^2v = \lambda^2 v$, so that $|\lambda|$ is a singular value of A.

The Spectral Theorem 4.33 4.34 says that a square matrix A can be written as $A = QDQ^{-1}$ under some assumptions. In general, not every matrix can be written in this way. However, a different decomposition, known as the singular value decomposition, can be applied to any matrix.

Theorem 4.102 (**Singular Value Decomposition (SVD)**). *Let $\mathbb{F}$ denote $\mathbb{R}$ or $\mathbb{C}$. Let A be an $m \times n$ matrix with values in $\mathbb{F}$ with $m \leq n$. Then there exists a $p \times p$ diagonal matrix D with positive entries ($p \leq m$), an $m \times m$ unitary matrix U, an $n \times n$ unitary matrix V, each with elements in $\mathbb{F}$ such that*

$$A = U \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} V.$$

Moreover, $AA^* = U \begin{pmatrix} D^2 & 0 \\ 0 & 0 \end{pmatrix} U^*$ and $V = \begin{pmatrix} (D^{-1}, 0)U^*A \\ \cdots \end{pmatrix}$ where D has no zero entries. (And U, D can be obtained from the QR Algorithm 4.89 applied to AA^*.) (In the case $p = m$ we have $A = U(D, 0)V$, $AA^* = UD^2U^*$, etc.)

Proof. Theorem 4.70 implies that AA^* and A^*A are self-adjoint positive semidefinite. The Spectral Theorem 4.34 implies that unitary $m \times m$ U and diagonal $p \times p$ D with positive entries exists such that $p \leq m$ and

$$AA^* = U \begin{pmatrix} D^2 & 0 \\ 0 & 0 \end{pmatrix} U^*. \qquad (\ddagger)$$

We can apply the QR Algorithm 4.89 and Theorem 4.90 to AA^* to compute U, D in the factorization $AA^* = U \begin{pmatrix} D^2 & 0 \\ 0 & 0 \end{pmatrix} U^*$ (at least when all entries of D are positive and distinct).

Recall D is diagonal with positive entries so D^{-1} exists. Define $Z := (D^{-1}, 0)U^*A$. Then

$$ZZ^* = (D^{-1}, 0)U^*AA^*U\begin{pmatrix} D^{-1} \\ 0 \end{pmatrix} = (D^{-1}, 0)\begin{pmatrix} D^2 & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} D^{-1} \\ 0 \end{pmatrix} = D^{-1}D^2D^{-1} = I.$$

By its definition, Z is an $m \times n$ matrix with m orthogonal rows. Since $m \leq n$, we can add extra rows to Z as necessary to obtain an $n \times n$ matrix V with orthogonal rows.

Finally, observe that

$$\begin{aligned} U\begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} V &= U\begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} Z \\ \cdots \end{pmatrix} = U\begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} (D^{-1}, 0)U^*A \\ \cdots \end{pmatrix} = U\begin{pmatrix} D \\ 0 \end{pmatrix}(D^{-1}, 0)U^*A \\ &= U\begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix}U^*A = U\Big(I - \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}\Big)U^*A = A - U\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}U^*A = A. \end{aligned}$$

In the last line we used $U\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}U^*A = 0$, which follows since

$$U\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}U^*AA^* \overset{(\ddagger)}{=} U\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} D^2 & 0 \\ 0 & 0 \end{pmatrix}U^* = U0U^* = 0. \qquad (**)$$

Therefore,

$$U\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}U^*A\Big[U\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}U^*A\Big]^* = \Big[U\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}U^*AA^*\Big](\cdots) \overset{(**)}{=} 0,$$

so that $U\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}U^*A = 0$. We have therefore shown the existence of the SVD. Lastly, in order to conclude that V is a unitary matrix, we have used Lemma 4.103 below. □

Lemma 4.103. *Let C be an $n \times n$ complex matrix such that $CC^* = I$. Then $C^*C = I$.*

Proof. By assumption, we have $C^*CC^*C = C^*C$, i.e. $(C^*C)^2 = C^*C$. Since C^*C is a self-adjoint positive semidefinite matrix, the spectral theorem 4.34 implies that there is a unitary $n \times n$ matrix U and a diagonal matrix D with nonnegative diagonal entries such that $C^*C = UDU^*$. Since $(C^*C)^2 = C^*C$, we have

$$UD^2U^* = UDU^*.$$

That is, $D^2 = D$. Since D is diagonal with nonnegative diagonal entries, we conclude that $D = I$, so that $C^*C = UIU^* = UU^* = I$. □

Exercise 4.104.

- Give an example of a real 2×2 matrix A with a non-unique singular value decomposition. That is, find a diagonal 2×2 matrix D, unitary 2×2 matrices $U_1 \neq U_2$, unitary 2×2 matrices $V_1 \neq V_2$ such that
$$A = U_1 D V_1 = U_2 D V_2.$$
- Give an example of a real 2×3 matrix A with a non-unique singular value decomposition. That is, find a diagonal 2×2 matrix D, unitary 2×2 matrices $U_1 \neq U_2$, unitary 3×3 matrices $V_1 \neq V_2$ such that
$$A = U_1 (D,\, 0) V_1 = U_2 (D,\, 0) V_2.$$

4.11. **Additional Comments.** We noted in Theorem 4.86 a performance guarantee of the LU factorization. Yet this guarantee can be quite bad in the worst case, due to the example from Exercise 4.61. On the other hand, there are various known average-case guarantees for the LU factorization, such as: https://arxiv.org/abs/2206.01726

We remarked above that the matrix $p \to q$ norms can be difficult to compute exactly for certain p, q. For more precise computational hardness statements, see e.g. https://arxiv.org/abs/1802.07425 and https://epubs.siam.org/doi/10.1137/S0097539704441629.

For very large matrices, SVD decompositions are difficult to compute efficiently, due to the need to multiply large matrices to compute the SVD. To alleviate this issue, we can try to reduce the dimension of the matrices while preserving their structure. For more on this topic see e.g. https://arxiv.org/abs/0909.4061 or https://en.wikipedia.org/wiki/Johnson Lindenstrauss lemma

The LU and QR decomposition solve linear systems of equations relatively quickly, though one could try to decrease their computation times even more. For more on this topic, see e.g. https://arxiv.org/abs/2007.10254.

5. Interpolation

5.1. **Polynomial Interpolation.** In Example 4.94, we found the best fit polynomial of fixed degree to data points. This polynomial might not take any particular prescribed values; it instead minimizes a sum of squared errors. For some applications, it is desirable to have a polynomial that *exactly* takes certain prescribed values. Such a polynomial is called an **interpolating** polynomial. Under certain assumptions, this polynomial will be unique.

Theorem 5.1. *Let $a_0, \ldots, a_n$ be distinct real numbers and let $b_0, \ldots, b_n \in \mathbb{R}$. Then there exists a unique polynomial $p_n \colon \mathbb{R} \to \mathbb{R}$ of degree at most n such that*
$$p_n(a_i) = b_i, \qquad \forall\ 0 \leq i \leq n.$$
Moreover, the coefficients of p_n can be found by solving a linear system of $n+1$ equations in $n+1$ unknowns.

Proof. We first show uniqueness. Suppose p_n and q_n are both polynomials satisfying the above conditions. Then $p_n - q_n$ is a polynomial of degree at most n that vanishes at $n+1$ points. The Fundamental Theorem of Algebra, Theorem 5.2, then implies that $p_n - q_n = 0$, so $p_n = q_n$, proving uniqueness.

We now show existence. For each $0 \leq j \leq n$, denote
$$f_j(x) := \prod_{i \in \{0,\ldots,n\} \colon\, i \neq j} \frac{x - a_i}{a_j - a_i}, \qquad \forall\, x \in \mathbb{R}.$$

Note that $f_j(a_i) = 0$ for all $0 \le i \le n$ with $i \ne j$ and $f_j(a_j) = 1$. So, the polynomial

$$p_n(x) := \sum_{j=0}^{n} b_j f_j(x), \qquad \forall\, x \in \mathbb{R}$$

satisfies $p_n(a_i) = b_i$ for all $0 \le i \le n$. Since each f_j has degree at most n, p_n has degree at most n. Finally, the requirement $p_n(a_i) = b_i \ \forall\ 0 \le i \le n$ for $p_n(t) = x_0 + x_1 t + \cdots + x_n t^n$ can be written as $Ax = b$ where

$$A = \begin{pmatrix} 1 & a_0 & a_0^2 & \cdots & a_0^n \\ 1 & a_1 & a_1^2 & \cdots & a_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^n \end{pmatrix}, \qquad x = \begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix}, \qquad b = \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix}.$$

□

Since A is an $(n+1) \times (n+1)$ matrix, to show that $Ax = b$ has a unique solution a priori, it suffices to show that A has rank $n+1$, i.e. that $\det(A) \ne 0$. This follows from Exercise 5.3 below.

Theorem 5.2 (Fundamental Theorem of Algebra). *Let $p\colon \mathbb{R} \to \mathbb{R}$ be a polynomial of degree $n \ge 1$. Then there exist $\alpha, z_1, \ldots, z_n \in \mathbb{C}$ such that*

$$p(z) = \alpha \prod_{i=1}^{n} (z - z_i), \qquad \forall\, z \in \mathbb{C}.$$

Exercise 5.3. Let $a_0, \ldots, a_n \in \mathbb{R}$. Show that

$$\det \begin{pmatrix} 1 & a_0 & a_0^2 & \cdots & a_0^n \\ 1 & a_1 & a_1^2 & \cdots & a_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^n \end{pmatrix} = \prod_{1 \le i < j \le n} (a_j - a_i).$$

(Hint: Denote the columns of this matrix as $A_0, \ldots, A_n$. Perform elementary column operations in the following order (which do not change the value of the determinant, since they correspond to multiplying by determinant 1 matrices). First, replace A_n with $A_n - a_0 A_{n-1}$. Then, replace A_{n-1} with $A_{n-1} - a_0 A_{n-2}$, and so on. The first show of the resulting matrix should be 1 followed by zeros.)

Remark 5.4 (Lagrange Form of Interpolating Polynomial). Theorem 5.1 says that $n+1$ points in the plane with distinct x-values can be interpolated in a unique way by a degree n polynomial. In the proof of Theorem 5.1, we showed that we can find p_n by solving a linear system, or by expressing p_n as a sum of more basic functions. The latter procedure is called the Lagrange form of the interpolating polynomial. Let $a_0, \ldots, a_n$ be distinct real numbers. For each $0 \le j \le n$, denote

$$f_j(x) := \prod_{i \in \{0,\ldots,n\}\colon\, i \ne j} \frac{x - a_i}{a_j - a_i}, \qquad \forall\, x \in \mathbb{R}.$$

Note that $f_j(a_i) = 0$ for all $0 \le i \le n$ with $i \ne j$ and $f_j(a_j) = 1$. So, the polynomial

$$p_n(x) := \sum_{j=0}^{n} b_j f_j(x), \qquad \forall\, x \in \mathbb{R}$$

satisfies $p_n(a_i) = b_i$ for all $0 \le i \le n$. Since p_n has degree at most n, Theorem 5.1 says p_n is the only polynomial with these properties.

Remark 5.5 (**Newton Form of Interpolating Polynomial**). Let $a_0, \ldots, a_n$ be distinct real numbers. For each $0 \le j \le n$, denote

$$g_j(x) := \prod_{i=0}^{j-1} (x - a_i), \qquad \forall\, x \in \mathbb{R}.$$

Within the set of degree n polynomials, the polynomials $g_0, \ldots, g_n$ are linearly independent, so there exist constants $c_0, \ldots, c_n \in \mathbb{R}$ such that the degree n interpolating polynomial p_n satisfies

$$p_n(x) = \sum_{j=0}^{n} c_j g_j(x), \qquad \forall\, x \in \mathbb{R}.$$

These constants can be found recursively. For example, since we assume $p_n(a_i) = b_i$ for all $0 \le i \le n$, we have

$$b_0 = p_n(a_0) = \sum_{j=0}^{n} c_j g_j(a_0) = c_0 \cdot 1.$$

$$b_1 = p_n(a_1) = \sum_{j=0}^{n} c_j g_j(a_1) = c_0 \cdot 1 + c_1(a_1 - a_0).$$

$$b_2 = p_n(a_2) = \sum_{j=0}^{n} c_j g_j(a_2) = c_0 \cdot 1 + c_1(a_2 - a_0) + c_2(a_2 - a_0)(a_2 - a_1).$$

and so on. In general, we find that $c_0 = b_0$ and for all $1 \le j \le n$.

$$c_j = \frac{b_j - \sum_{k=0}^{j-1} c_k \prod_{i=0}^{k-1} (a_j - a_i)}{\prod_{i=0}^{j-1} (a_j - a_i)}.$$

Theorem 5.6 (**Mean Value Theorem**). *Let $a < b$ and let $f\colon [a, b] \to \mathbb{R}$ be a differentiable function. Then there exists $c \in (a, b)$ such that*

$$\frac{f(b) - f(a)}{b - a} = f'(c).$$

In particular, if $f(a) = f(b)$, then $f'(c) = 0$.

If an interpolating polynomial p_n agrees with the values of a function f at $n + 1$ distinct points, then we can also bound the error between f and p_n for any other point x.

Theorem 5.7 (**Polynomial Interpolation Error**). *Let $a < b$. Let f be $n + 1$ times continuously differentiable in $[a, b]$. Let p_n be a polynomial of degree at most n that interpolates*

f at $n+1$ distinct points $a_0, \ldots, a_n \in [a,b]$ (i.e. $f(a_i) = p_n(a_i)\ \forall\ 0 \le i \le n$). Then, for any $x \in [a,b]$, there exists $y_x \in (a,b)$ such that

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(y_x) \cdot \prod_{i=0}^{n} (x - a_i).$$

Proof. Fix $x \in [a,b]$. If $x = a_i$ for some $0 \le i \le n$, then both sides of the equality are zero, and the conclusion holds. So, assume $x \neq a_i$ for all $0 \le i \le n$. Define

$$q(x) := \prod_{i=0}^{n} (x - a_i), \qquad g := f - p_n - \alpha q, \qquad (\ddagger)$$

where $\alpha \in \mathbb{R}$ is chosen so that $g(x) = 0$ (recalling that x is fixed), i.e. $\alpha = (f(x) - p_n(x))/q(x)$.

By its definition, g is $n+1$ times continuously differentiable and g vanishes at $n+2$ distinct points $x, a_0, \ldots, a_n$. From the Mean Value Theorem 5.6, g' has at least $n+1$ distinct zeros in (a,b). Again from the Mean Value Theorem 5.6, g'' has at least n distinct zeros in (a,b). And so on. We conclude that $g^{(n+1)}$ has at least one zero in (a,b), which we denote as y_x. Since p_n is a polynomial of degree at most n, $p_n^{(n+1)} = 0$. Also, $q^{(n+1)} = (n+1)!$, so

$$0 = g^{(n+1)}(y_x) \overset{(\ddagger)}{=} f^{(n+1)}(y_x) - \alpha(n+1)!.$$

Solving for α and substituting into $(\ddagger)$ completes the proof. □

In a typical situation in polynomial interpolation, we will have some bound on a higher derivative of f. In the error bound in Theorem 5.7, we are then free to choose the points $a_0, \ldots, a_n$ that minimize the quantity

$$\prod_{i=0}^{n} (x - a_i) = x^{n+1} - x^n(a_0 + \cdots + a_n) + \cdots.$$

Notice that the coefficient of the highest degree part of this polynomial is 1. That is, this polynomial is **monic**.

It turns out that the degree $n+1$ monic polynomial that minimizes

$$\max_{-1 \le x \le 1} |p(x)|$$

among all degree $n+1$ monic polynomials p is a well-studied polynomial of the following form.

Definition 5.8 (Chebyshev Polynomial). For any integer $n \ge 0$, define a function $T_n \colon [-1,1] \to \mathbb{R}$ by

$$T_n(x) := \cos(n \cos^{-1}(x)), \qquad \forall\, x \in [-1,1].$$

Proposition 5.9. *For any $n \ge 0$, T_n from Definition 5.8 is a polynomial of degree n on $[-1,1]$, and these polynomials satisfy the recurrence,*

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad \forall\, x \in [-1,1], \qquad \forall\, n \ge 1.$$

Moreover, the n^{th} degree term of T_n is 2^{n-1}, i.e. $T_n(x) = 2^{n-1}x^n + \cdots$ for all $n \ge 1$.

Proof. Let $\theta \in [0, \pi]$. From the cosine addition formula, we have

$$\cos(n+1)\theta = \cos\theta\cos n\theta - \sin\theta\sin n\theta,$$

$$\cos(n-1)\theta = \cos\theta\cos n\theta + \sin\theta\sin n\theta.$$

Adding these and rearranging,

$$\cos(n+1)\theta = 2\cos\theta\cos n\theta - \cos(n-1)\theta. \qquad (\ddagger)$$

Let $x \in [-1, 1]$ and let $\theta := \cos^{-1} x$, so that $x = \cos\theta$. Let $n \geq 1$. Using Definition 5.8, we rewrite $(\ddagger)$ as

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad \forall\, x \in [-1, 1].$$

Since $T_0(x) = 1$ and $T_1(x) = x$ for all $x \in [-1, 1]$ by Definition 5.8, it follows from this recurrence that T_n is a polynomial of degree at most n. The first part of the recurrence implies that T_n has degree equal to n, and the n^{th} order term has a coefficient of 2^{n-1}. □

Theorem 5.10 (**Minimizing Property of Chebyshev Polynomials**). *Let $q_n \colon \mathbb{R} \to \mathbb{R}$ be a polynomial of degree $n \geq 1$ such that $q_n(x) = x^n + \cdots$, i.e. p_n is a monic polynomial (it has a coefficient 1 on its n^{th} degree term). Then*

$$\max_{-1 \leq x \leq 1} |q_n(x)| \geq 2^{1-n}.$$

Moreover, equality holds when $q_n = 2^{1-n}T_n$.

Proof. For any $n \geq 1$ and for any $0 \leq j \leq n$, define $c_j := \cos(j\pi/n)$. We note the following two properties that follow directly from Definition 5.8: $|T_n(x)| \leq 1$ for all $x \in [-1, 1]$, and $T_n(c_j) = (-1)^j$, for all $0 \leq j \leq n$, for all $n \geq 1$. Define $h_n := 2^{1-n}T_n$. Then

(i) $|h_n(x)| \leq 2^{1-n}$ for all $x \in [-1, 1]$.
(ii) $h_n(c_j) = (-1)^j 2^{1-n}$, for all $0 \leq j \leq n$, for all $n \geq 1$.

We now argue by contradiction. Assume that $\max_{-1 \leq x \leq 1} |q_n(x)| < 2^{1-n}$. Then

$$(-1)^j q_n(c_j) \leq |q_n(c_j)| < 2^{1-n} \overset{(ii)}{=} (-1)^j h_n(c_j).$$

That is,

$$(-1)^j [h_n(c_j) - q_n(c_j)] > 0, \qquad \forall\, 0 \leq j \leq n.$$

Since $h_n - q_n$ is a polynomial, it is continuous, so the Intermediate Value Theorem 5.11 says that $h_n - q_n$ has n zeros in $[-1, 1]$. However, since q_n and h_n are both monic with degree n, $h_n - q_n$ has degree at most $n - 1$. This contradicts the Fundamental Theorem of Algebra, Theorem 5.2. We conclude that $\max_{-1 \leq x \leq 1} |q_n(x)| \geq 2^{1-n}$. □

Theorem 5.11 (**Intermediate Value Theorem**). *Let $a < b$ and let $f \colon [a, b] \to \mathbb{R}$ be a continuous function. Then f takes every value between $f(a)$ and $f(b)$. In particular, if $f(a)$ and $f(b)$ have opposite signs, then there exists $c \in [a, b]$ such that $f(c) = 0$.*

Remark 5.12. We showed that a degree n monic polynomial p_n satisfies $\max_{-1 \leq x \leq 1} |q_n(x)| \geq 2^{1-n}$, with equality when $q_n = 2^{1-n}T_n$. If we are given $a < b$ and we want the same result for $[a, b]$ instead of $[-1, 1]$, we just translate and dilate. For example, for any $t \in \mathbb{R}$, if q_n is a monic polynomial of degree n, then

$$\max_{-1+t \leq x \leq 1+t} |q_n(x)| \geq 2^{1-n}$$

and equality holds only when $q_n(x) = 2^{1-n}T_n(x-t)$ for all $x \in [-1+t, 1+t]$. (The polynomial $2^{1-n}T_n(x-t)$ is still monic.) And for any $s > 0$, if $s^{-n}q_n$ is a monic polynomial of degree n, then

$$\max_{\frac{-1+t}{s} \le x \le \frac{1+t}{s}} |q_n(x)| \ge 2^{1-n}$$

and equality holds only when $q_n(x) = 2^{1-n}T_n(s(x-t))$ for all $x \in \left[\frac{-1+t}{s}, \frac{1+t}{s}\right]$. (The polynomial $2^{1-n}T_n(s(x-t))$ is not monic, but the coefficient on its highest degree term is s^n.)

Combining Theorems 5.10 (in the $n+1$ case) and 5.7 yields the following.

Theorem 5.13 (**Chebyshev Node Interpolation Error**). *Let f be $n+1$ times continuously differentiable in $[-1,1]$. Let p_n be a polynomial of degree at most n that interpolates f at the $n+1$ points*

$$a_j := \cos((j+1/2)\pi/(n+1)), \quad 0 \le j \le n.$$

Then, for all $x \in [-1,1]$,

$$|f(x) - p_n(x)| \le \frac{1}{2^n(n+1)!} \max_{|y|\le 1} \left|f^{(n+1)}(y)\right|.$$

Exercise 5.14. Let $f(x) := e^{2x}$ for all $x \in \mathbb{R}$. Let p_n be a polynomial of degree n that interpolates f on $[-1,1]$ at the $n+1$ roots of the Chebyshev polynomial T_{n+1} on $[-1,1]$. Find the smallest n such that

$$|f(x) - p_n(x)| < 10^{-6}, \qquad \forall\, x \in [-1,1]$$

Exercise 5.15. The nodes $\{\cos((j+1/2)\pi/(n+1))\}_{j=0}^n$ were shown to be optimal for interpolation error on $[-1,1]$. One might guess that choosing equally spaced points for interpolation might lead to comparable errors, but this is not the case. Consider the function

$$f(x) := \frac{1}{1+25x^2}, \qquad \forall\, x \in [-1,1].$$

Using Matlab, plot various interpolating polynomials p_n of this function on $[-1,1]$ using equally spaced nodes. Verify experimentally that $\|f - p_n\|_\infty$ does not go to zero as $n \to \infty$. (In fact, these numbers go to infinity!)

Then, plot the interpolating polynomials of f on $[-1,1]$ using the Chebyshev nodes $\{\cos(j\pi/n)\}_{j=0}^n$. Verify experimentally that $\lim_{n\to\infty} \|f - p_n\|_\infty = 0$.

5.2. **Hermite Interpolation.** In the previous section, we described how to find a polynomial with values specified at particular points. Sometimes we might also want to specify the values of the derivatives of the polynomial.

Theorem 5.16. *Let $a_0, \ldots, a_n$ be distinct real numbers and let $b_0, \ldots, b_n, c_0, \ldots, c_n \in \mathbb{R}$. Then there exists a unique polynomial $p\colon \mathbb{R} \to \mathbb{R}$ of degree at most $2n+1$ such that*

$$p(a_i) = b_i, \quad p'(a_i) = c_i, \qquad \forall\, 0 \le i \le n.$$

Moreover, the coefficients of p can be found by solving a linear system of $2n+2$ equations in $2n+2$ unknowns.

Proof. For any $0 \le j \le n$, denote

$$f_j(x) := (x - a_j) \cdot \prod_{i \in \{0,\ldots,n\}\colon\, i \ne j} \left(\frac{x - a_i}{a_j - a_i}\right)^2, \qquad \forall\, x \in \mathbb{R}.$$

$$g_j(x) := \alpha_j f_j(x) + \prod_{i\in\{0,\dots,n\}:\, i\neq j} \left(\frac{x-a_i}{a_j-a_i}\right)^2, \qquad \forall\, x\in\mathbb{R}.$$

Here $f_j(a_i) = f_j'(a_i) = 0$ for all $i \neq j$, $f_j(a_j) = 0$ and $f_j'(a_j) = 1$. So, we can choose $\alpha_j \in \mathbb{R}$ such that $g_j'(a_j) = 0$ and $g_j(a_j) = 1$. Note that changing α_j does not change the value of $g_j(a_j)$ (which is 1). Also, $g_j(a_i) = g_j'(a_i) = 0$ for all $i \neq j$.

The $2n+2$ functions $\{f_0,\dots,f_n,g_0,\dots,g_n\}$ are therefore linearly independent in the vector space V of polynomials of degree at most $2n+1$. Since V has dimension $2n+2$, $\{f_0,\dots,f_n,g_0,\dots,g_n\}$ is a basis for V. So, we can write the polynomial p uniquely as

$$p = \sum_{i=0}^n \frac{b_i}{g_i(a_i)} g_i + \sum_{i=0}^n \frac{c_i}{f_i'(a_i)} f_i = \sum_{i=0}^n b_i g_i + \sum_{i=0}^n c_i f_i.$$

The constraints for $p_n(t) = x_0 + x_1 t + \cdots + x_{2n+1}t^{2n+1}$ can be written as $Ax = b$ where

$$A = \begin{pmatrix} 1 & a_0 & a_0^2 & a_0^3 & \cdots & a_0^{2n+1} \\ 1 & a_1 & a_1^2 & a_1^3 & \cdots & a_1^{2n+1} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & a_n & a_n^2 & a_n^3 & \cdots & a_n^{2n+1} \\ 0 & 1 & 2a_0 & 3a_0^2 & \cdots & (2n+1)a_0^{2n} \\ 0 & 1 & 2a_1 & 3a_1^2 & \cdots & (2n+1)a_1^{2n} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 1 & 2a_n & 3a_n^2 & \cdots & (2n+1)a_n^{2n} \end{pmatrix}, \qquad x = \begin{pmatrix} x_0 \\ \vdots \\ x_{2n+1} \end{pmatrix}, \qquad b = \begin{pmatrix} b_0 \\ \vdots \\ b_n \\ c_0 \\ \vdots \\ c_n \end{pmatrix}.$$

□

The proof of Theorem 5.7 can be repeated yielding the following.

Theorem 5.17 (Hermite Interpolation Error). *Let $a < b$. Let f be $2n+2$ times continuously differentiable in $[a,b]$. Let p be a polynomial of degree at most $2n+1$ such that $f = p$ and $f' = p'$ at distinct points $a_0,\dots,a_n \in [a,b]$. Then, for any $x \in [a,b]$, there exists $y_x \in (a,b)$ such that*

$$f(x) - p(x) = \frac{1}{(2n+2)!} f^{(2n+2)}(y_x) \cdot \prod_{i=0}^n (x-a_i)^2.$$

Proof. Fix $x \in [a,b]$. If $x = a_i$ for some $0 \le i \le n$, then both sides of the equality are zero, and the conclusion holds. So, assume $x \neq a_i$ for all $0 \le i \le n$. Define

$$q(x) := \prod_{i=0}^n (x-a_i)^2, \qquad g := f - p - \alpha q, \qquad (\ddagger)$$

where $\alpha \in \mathbb{R}$ is chosen so that $g(x) = 0$ (recalling that x is fixed), i.e. $\alpha = (f(x) - p_n(x))/q(x)$.

By its definition, g is $2n+1$ times continuously differentiable and g vanishes at $n+2$ distinct points $x, a_0,\dots,a_n$. From the Mean Value Theorem 5.6, g' has at least $n+1$ distinct zeros in (a,b), and besides those zeros, g' also vanishes at $a_0,\dots,a_n$. So, g' has at least $2n+2$ distinct zeros. Again from the Mean Value Theorem 5.6, g'' has at least $2n+1$ distinct zeros

in (a, b). And so on. We conclude that $g^{(2n+2)}$ has at least one zero in (a, b), which we denote as y_x. Since p is a polynomial of degree $2n + 1$, $p^{(2n+1)} = 0$. Also, $q^{(2n+2)} = (2n + 2)!$, so

$$0 = g^{(2n+2)}(y_x) \overset{(\ddagger)}{=} f^{(2n+2)}(y_x) - \alpha(2n + 2)!.$$

Solving for α and substituting into $(\ddagger)$ completes the proof. □

5.3. **Spline Interpolation.**

6. Numerical Implementation of Calculus

6.1. **Numerical Partial Differentiation.** If $f\colon \mathbb{R} \to \mathbb{R}$ is differentiable at $x \in \mathbb{R}$, then

$$f'(x) = \lim_{h\to 0} \frac{f(x + h) - f(x)}{h}.$$

So, when h is close to zero, we can use the following formula to numerically approximate derivatives on a compute:

$$f'(x) \approx \frac{f(x + h) - f(x)}{h}. \qquad (*)$$

One could also use other approximations to $f'(x)$ such that $\frac{f(x+h)-f(x-h)}{2h}$.

If $f''(x)$ exists, we similarly have

$$f'(x) = \lim_{h\to 0} \frac{f'(x + h) - f'(x)}{h}.$$

So, when h is small, we have

$$\begin{aligned} f''(x) &\approx \frac{f'(x + h) - f'(x)}{h} \overset{(*)}{\approx} \frac{f(x + 2h) - f(x + h) - [f(x + h) - f(x)]}{h^2} \\ &= \frac{f(x + 2h) - 2f(x + h) + f(x)}{h^2}. \end{aligned}$$

Since $f(x) \approx f(x - h)$ when h is small, we have the more symmetric approximation

$$f''(x) \approx \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}.$$

One can similarly derive approximation formula for higher derivatives.

Partial derivatives can be approximated analogously.

6.2. **Numerical Integration.**

Example 6.1 (**Trapezoidal Rule**)**.** Suppose we have $N + 1$ equally spaced points on the interval $[a, b]$. We label these points as $a = x_0 < x_1 < \cdots < x_N = b$. Note that $x_i - x_{i-1} = (b - a)/N$ for each $1 \le i \le N$. We approximate the area under the curve of f by a set of trapezoids. Recall that a trapezoid of with w and heights h_1, h_2 has area $w(h_1 + h_2)/2$. We will particularly approximate the area under f by the trapezoids of width $x_i - x_{i-1}$ and heights $f(x_i), f(x_{i-1})$, for each $1 \le i \le N$. The total area of all of these trapezoids is then

$$T_N = (x_1 - x_0)\frac{f(x_1) + f(x_0)}{2} + (x_2 - x_1)\frac{f(x_2) + f(x_1)}{2} + \cdots + (x_N - x_{N-1})\frac{f(x_N) + f(x_{N-1})}{2}.$$

Using $x_i - x_{i-1} = (b-a)/N$ for each $1 \le i \le N$, the total area of the trapezoids is equal to

$$T_N = \frac{b-a}{N}\left(f(x_0)/2 + f(x_1) + f(x_2) + f(x_3) + \cdots + f(x_{N-1}) + f(x_N)/2\right).$$

Remark 6.2. Another way of describing the trapezoidal rule is that we have approximated our function f by a piecewise linear function g, and we then use the integral of g to approximate the integral of f.

Remark 6.3 (**Midpoint Rule**)**.** If is also possible to approximate a function f by its Riemann sums (as in the midpoint rule). In particular, the Midpoint Rule approximates a function f by a Riemann sum of equally spaced points, where we evaluate the function f at the midpoint of each rectangle. That is, we approximate the integral of f by

$$M_N = \frac{b-a}{N}\left(f\left(\frac{x_0+x_1}{2}\right) + f\left(\frac{x_1+x_2}{2}\right) + \cdots + f\left(\frac{x_{N-1}+x_N}{2}\right)\right).$$

Another way of describing the Midpoint Rule is that we approximate f by a piecewise constant function g, and we then use the integral of g to approximate the integral of f.

Lemma 6.4. *Let $f\colon [a,b] \to \mathbb{R}$. Assume that f'' exists and is continuous on $[a,b]$. Then*

$$\left|\int_a^b f(x)dx - (b-a)\frac{f(a)+f(b)}{2}\right| \le \frac{(b-a)^3}{12}\max_{y\in[a,b]}|f''(y)|.$$

Proof. Let $s, t \in \mathbb{R}$. Integrate by parts twice to get

$$\begin{aligned}
&\int_a^b f(x)dx = \int_a^b f(x)\frac{d}{dx}(x-s)dx = f(x)(x-s)|_{x=a}^{x=b} - \int_a^b f'(x)(x-s)dx \\
&= f(b)(b-s) - f(a)(a-s) - \int_a^b f'(x)\frac{d}{dx}[t + (x-s)^2/2]dx \\
&= f(b)(b-s) - f(a)(a-s) - f'(x)[t+(x-s)^2/2]|_{x=a}^{x=b} + \int_a^b f''(x)[t+(x-s)^2/2]dx
\end{aligned}$$

Choosing $s := (a+b)/2$, we get $f(b)(b-s) - f(a)(a-s) = (b-a)(f(a)+f(b))/2$. We can also choose t so that the f' terms sum to zero. That is, we choose t so that

$$t := -\frac{(b-s)^2}{2} = -\frac{(b-a)^2}{8}.$$

(Note that $(a-s)^2 = (b-s)^2$, so both f' terms are now multiplied by zero.) Rearranging,

$$\begin{aligned}
\left|\int_a^b f(x)dx - (b-a)\frac{f(a)+f(b)}{2}\right| &\le \max_{y\in[a,b]}|f''(y)|\int_a^b \left|t + (x-s)^2/2\right| dx \\
&= \max_{y\in[a,b]}|f''(y)|\int_a^b \left|-\frac{(b-a)^2}{8} + \frac{1}{2}\Big(x - \frac{a+b}{2}\Big)^2\right| dx.
\end{aligned}$$

The parabola inside the absolute values is nonpositive, so

$$\begin{aligned}\left|\int_a^b f(x)dx-(b-a)\frac{f(a)+f(b)}{2}\right| &\le \max_{y\in[a,b]}|f''(y)|\int_a^b \frac{(b-a)^2}{8}-\frac{1}{2}\Big(x-\frac{a+b}{2}\Big)^2dx\\ &= \max_{y\in[a,b]}|f''(y)|\left[\frac{(b-a)^3}{8} \qquad -\frac{1}{6}\Big(x-\frac{a+b}{2}\Big)^3|_{x=a}^{x=b}\right]\\ &= \max_{y\in[a,b]}|f''(y)|\left[\frac{(b-a)^3}{8}-\frac{1}{6}2\frac{(b-a)^3}{8}\right]\\ &= \max_{y\in[a,b]}|f''(y)|\,(b-a)^3\frac{1}{8}(1-1/3).\end{aligned}$$

□

Lemma 6.5. *Let $f\colon[a,b]\to\mathbb{R}$. Assume that f'' exists and is continuous on $[a,b]$. Then*

$$\left|\int_a^b f(x)dx-(b-a)f((a+b)/2)\right|\le\frac{1}{24}\max_{y\in[a,b]}|f''(y)|\,(b-a)^3.$$

Proof. From Taylor's Theorem (or the Mean Value Theorem), for any $x\in[a,b]$, there exists $y_x\in[a,b]$ such that

$$f(x)=f((a+b)/2)+(x-(a+b)/2)f'((a+b)/2)+\frac{1}{2}(x-(a+b)/2)^2f''(y_x).$$

Integrating both sides,

$$\int_a^b f(x)dx=(b-a)f((a+b)/2)+\frac{1}{2}\int_a^b(x-(a+b)/2)^2f''(y_x)dx.$$

Rearranging and using $\int_a^b(x-(a+b)/2)^2dx=(1/12)(b-a)^3$, we get

$$\begin{aligned}\left|\int_a^b f(x)dx-(b-a)f((a+b)/2)\right| &\le \frac{1}{2}\max_{y\in[a,b]}|f''(y)|\int_a^b(x-(a+b)/2)^2dx\\ &= \frac{(b-a)^3}{24}\max_{y\in[a,b]}|f''(y)|\,.\end{aligned}$$

□

Theorem 6.6 (**Error Bounds for Trapezoid and Midpoint Rules**). *Let $f\colon[a,b]\to\mathbb{R}$ with $a<b$, $a,b\in\mathbb{R}$. Assume that f'' exists and is continuous. Then*

$$\left|\int_a^b f(x)dx-T_N\right|\le\frac{(b-a)^3}{12N^2}\max_{y\in[a,b]}|f''(y)|\,.$$

$$\left|\int_a^b f(x)dx-M_N\right|\le\frac{(b-a)^3}{24N^2}\max_{y\in[a,b]}|f''(y)|\,.$$

Example 6.7. Let's compute both the trapezoid and midpoint rules for a function, and verify that these error rates are correct. We will estimate $\int_1^2\sqrt{x}dx$ with $N=6$. Since $b=2$ and $a=1$, we have $(b-a)/N=1/6$. The points $x_0,\dots,x_N$ are $1,7/6,8/6,9/6,10/6,11/6,2$. And

$$T_6=\frac{1}{6}\left(\sqrt{1}/2+\sqrt{7/6}+\sqrt{4/3}+\sqrt{3/2}+\sqrt{5/3}+\sqrt{11/6}+\sqrt{2}/2\right)\approx 1.218612.$$

$$M_6 = \frac{1}{6}\left(\sqrt{13/12} + \sqrt{15/12} + \sqrt{17/12} + \sqrt{19/12} + \sqrt{21/12} + \sqrt{23/12}\right) \approx 1.219121.$$

In this case, we can compute the integral exactly:

$$\int_1^2 \sqrt{x}dx = (2/3)(2^{3/2} - 1^{3/2}) = (2/3)(2^{3/2} - 1) \approx 1.218951.$$

For $f(x) = \sqrt{x}$, we have $f'(x) = (1/2)x^{-1/2}$ and $f''(x) = (-1/4)x^{-3/2}$, so for $1 \le x \le 2$, we have $|f''(x)| \le 1/4$. So, we can verify our Theorem for Error Bounds as follows

$$.000339 \approx \left|\int_1^2 \sqrt{x}dx - T_6\right| \le \frac{K(b-a)^3}{12N^2} = \frac{(1/4)(1)}{12(36)} = \frac{1}{1728} \approx .000579.$$

$$.000170 \approx \left|\int_1^2 \sqrt{x}dx - M_6\right| \le \frac{K(b-a)^3}{24N^2} = \frac{(1/4)(1)}{24(36)} = \frac{1}{3456} \approx .000289.$$

So, in this case, our error bound is actually not too much larger than the actual error between the integral and its approximations.

The following problem is fairly typical when we try to evaluate an integral with a computer. We want to approximate a certain integral, and we want to guarantee that our approximation is a certain distance from the correct answer.

Example 6.8. Find an integer N such that T_N approximates $\int_0^3 e^{-x^2}dx$ within an absolute error of 10^{-5}.

We first estimate the second derivative of $f(x) = e^{-x^2}$. Then $f'(x) = (-2x)e^{-x^2}$ and $f''(x) = (4x^2 - 2)e^{-x^2}$. Let $0 \le x \le 3$. Then $|4x^2 - 2| \le 34$ and $|e^{-x^2}| \le 1$, so $|f''(x)| \le 34$ for $0 \le x \le 3$. Using the error bound for T_N, we want to find N such that

$$\frac{34(b-a)^3}{12N^2} < 10^{-5}.$$

Since $(b-a) = 3$, we want to find N such that

$$N > \sqrt{\frac{34(3^3)(10^5)}{12}} \approx 2765.9$$

Therefore, choosing $N = 2766$ suffices.

So far, we have seen the Midpoint Rule, which approximates f by a piecewise constant function g and then computes the integral of g as an approximation to the integral of f. We also saw the Trapezoid Rule, which approximates f by a piecewise linear function g and then computes the integral of g as an approximation to the integral of f. We now take this idea one step further. With **Simpson's Rule**, we approximate f by a piecewise quadratic function g and then compute the integral of g as an approximation to the integral of f. After some analysis (which we omit), Simpson's rule S_N for an even N has the following formula

$$S_N = \frac{1}{3}T_{N/2} + \frac{2}{3}M_{N/2}.$$

Substituting the formulas for $T_{N/2}$ and $M_{N/2}$ into this formula, we get

$$S_N = \frac{(b-a)}{3N}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{N-2}) + 4f(x_{N-1}) + f(x_N)\right).$$

We then get the below error bound for Simpson's Rule

Lemma 6.9. *Let $f\colon [a,b] \to \mathbb{R}$ with $a < b$, $a, b \in \mathbb{R}$. Assume that $f^{(4)}$ exists and is continuous. Then*

$$\left| \int_a^b f(x)dx - \frac{b-a}{6}\Big(f(b) + f(a) + 4f(0)\Big) \right| \leq \frac{(b-a)^5}{180 \cdot 16} \max_{y\in[a,b]} \left|f^{(4)}(y)\right|.$$

Proof. Assume for simplicity for now that $b = -a$, $b > 0$. Let p_2 be the (unique) degree two polynomial such that $p_2(0) = f(0)$, $p_2(a) = f(a)$, and $p_2(b) = f(b)$. Assume also for now that f is an even function, i.e. $f(x) = f(-x)$ for all $x \in [a,b]$. It then follows that p_2 is also even, and $p_2'(0) = f'(0) = 0$. Fix $x \in (a,b)$ with $x \neq 0$. Define

$$g(t) := f(t) - p_2(t) - \alpha t^2(t+b)(t-b), \qquad (\ddagger)$$

where $\alpha \in \mathbb{R}$ is chosen so that $g(x) = 0$ (recalling that x is fixed), i.e.

$$\alpha = \frac{f(x) - p_2(x)}{x^2(x+b)(x-b)}. \qquad (**)$$

By its definition, g is four times continuously differentiable and g vanishes at four distinct points $x, a, 0, b$. From the Mean Value Theorem 5.6, g' has at least three distinct zeros in (a,b) that are nonzero. Moreover,

$$g'(0) = f'(0) - p_2'(0) - 0 = 0.$$

So, g' has at least four zeros in (a,b). Again from the Mean Value Theorem 5.6, g'' has at least three distinct zeros in (a,b). And so on. We conclude that $g^{(4)}$ has at least one zero in (a,b), which we denote as y_x. Since p_2 is a polynomial of degree 2, $p_2^{(3)} = 0$. So,

$$0 = g^{(4)}(y_x) \overset{(\ddagger)}{=} f^{(4)}(y_x) - \alpha \cdot 4! \overset{(**)}{=} f^{(4)}(y_x) - 24\frac{f(x) - p_2(x)}{x^2(x+b)(x-b)}.$$

That is,

$$f(x) - p_2(x) = \frac{1}{24}x^2(x+b)(x-b)f^{(4)}(y_x).$$

Integrating, taking absolute values, and using Exercise 6.10 with $h := (b-a)/2 = b = -a$,

$$\begin{aligned}\left| \int_a^b f(x)dx - \frac{b-a}{6}\Big(f(b) + f(a) + 4f(0)\Big) \right| &\leq \frac{1}{24} \max_{y\in[a,b]} \left|f^{(4)}(y)\right| \int_a^b \left|x^2(x+b)(x-b)\right| dx \\ &= \frac{1}{24} \max_{y\in[a,b]} \left|f^{(4)}(y)\right| 2\int_0^b x^2(x+b)(b-x)dx.\end{aligned}$$

Finally,

$$\int_0^b x^2(x+b)(b-x)dx = \int_0^b [x^2b^2 - x^4]dx = b^5\Big(\frac{1}{3} - \frac{1}{5}\Big) = b^5\frac{2}{15}.$$

So,

$$\begin{aligned}\left| \int_a^b f(x)dx - \frac{b-a}{6}\Big(f(b) + f(a) + 4f(0)\Big) \right| &\leq \frac{1}{12}\frac{2}{15}b^5 \max_{y\in[a,b]} \left|f^{(4)}(y)\right| = \frac{b^5}{90} \max_{y\in[a,b]} \left|f^{(4)}(y)\right| \\ &= \frac{(b-a)^5}{90 \cdot 2^5} \max_{y\in[a,b]} \left|f^{(4)}(y)\right|.\end{aligned}$$

This inequality holds under the assumption that f is even. For a general f, we can always write $f(x) = [f(x) + f(-x)]/2 + (f(x) - f(-x))/2$, as a sum of an even function and

an odd function. The odd function integrates to zero and contributes zero to the sum $f(b) + f(a) + 4f(0)$. So, this inequality holds for all f. □

Exercise 6.10. Let $h > 0$. Let $f\colon [-h, h] \to \mathbb{R}$ be continuous. Let p_2 be the (unique) degree two polynomial such that

$$p_2(h) = f(h), \qquad p_2(-h) = f(-h), \qquad p_2(0) = f(0).$$

Show that

$$\int_{-h}^{h} p_2(t)dt = \frac{h}{3}\Big(f(h) + f(-h) + 4f(0)\Big).$$

Theorem 6.11 (**Error Bound for Simpson's Rule**). *Let $f\colon [a, b] \to \mathbb{R}$ with $a < b$, $a, b \in \mathbb{R}$. Assume that $f^{(4)}$ exists and is continuous. Then*

$$\left|\int_a^b f(x)dx - S_N\right| \leq \frac{(b-a)^5}{180N^4} \max_{y\in[a,b]} \left|f^{(4)}(y)\right|.$$

Note the extra factor of 2 occurs here since S_N uses $2N - 1$ nodes.

Example 6.12. We continue our above example, and this time we use Simpson's rule. We estimate $\int_1^2 \sqrt{x}dx$ with $N = 6$. Since $b = 2$ and $a = 1$, we have $(b-a)/N = 1/6$. The points $x_0, \ldots, x_N$ are $1, 7/6, 8/6, 9/6, 10/6, 11/6, 2$. And

$$S_6 = \frac{1}{18}\left(\sqrt{1} + 4\sqrt{7/6} + 2\sqrt{4/3} + 4\sqrt{3/2} + 2\sqrt{5/3} + 4\sqrt{11/6} + \sqrt{2}\right) \approx 1.21895013.$$

As before, we can compute the integral exactly

$$\int_1^2 \sqrt{x}dx = (2/3)(2^{3/2} - 1^{3/2}) = (2/3)(2^{3/2} - 1) \approx 1.21895142.$$

For $f(x) = \sqrt{x}$, we have $f'(x) = (1/2)x^{-1/2}$, $f''(x) = (-1/4)x^{-3/2}$, $f'''(x) = (3/8)x^{-5/2}$, and $f^{(4)}(x) = -(15/16)x^{-7/2}$. So for $1 \leq x \leq 2$, we have $\left|f^{(4)}(x)\right| \leq 15/16$. So, we can verify our Theorem for Error Bounds as follows

$$.00000129 \approx \left|\int_1^2 \sqrt{x}dx - S_6\right| \leq \frac{K(b-a)^5}{180N^4} = \frac{(15/16)(1)}{180(6^4)} = \frac{1}{248832} \approx .00000402.$$

Note that in this case, Simpson's rule is roughly 100 times more accurate than the Midpoint or Trapezoid rules, even though we used the same number of sample points.

Exercise 6.13 (**Adaptive Quadrature**). The NCM package function `quadtx` is a simplified version of Matlab's built-in integration function `quad`. (To view the code of `quadtx` use the command `edit quadtx`. Similarly, `edit quad` should show you the source code for the function `quad`.) For example, the command `quadtx(@(x)(cos(x))^2,0,4*pi)` approximates $\int_0^{4\pi} (\cos(x))^2 dx$. More generally, the program `quadtx` starts by evaluating the given function $f\colon [a, b] \to \mathbb{R}$ with two different Simpson's rule evaluations (one using three points, and another using five points, each equally spaced on the interval). (In the code, these two evaluations are denoted `Q1` and `Q2`.)

If these two different Simpson's rule evaluations are closer than 10^{-6} (the default value of `tol`), then the program believes it has succeeded in estimating $\int_a^b f(x)dx$. So, the program outputs a combination of these two Simpson's rule evaluations, which happens to be a sixth order Newton-Cotes formula (in the code this is `Q2 + (Q2 - Q1)/15`).

If these two different Simpson's rule evaluations are not closer than 10^{-6}, then `quadtx` repeats the above Simpson's rule procedure on a smaller subinterval, and then iterates. This is done via a recursive call to the function `quadtxstep`. Note the recursive nature of this program, since the function `quadtxstep` calls itself. Also, note that `varargin` is used often in `quadtx`. This command allows a variable number of arguments to be input to a function.

The recursive use of Simpson's rule can be visualized with the `quadgui` command, after clicking "auto." Function evaluations are depicted as blue dots, and the total number of function evaluations is displayed at the top of the plot.

- Run the programs `quadtx(@(x)x.^3,0,1)` and `quadgui(@(x)x.^3,0,1)`. How many function evaluations are used to estimate $\int_0^1 x^3 dx$? What is the absolute error of the estimation?
- Run the programs `quadgui(@(x)x.^5,0,1)` and `quadgui(@(x)x.^5,0,1,10^(-8))`. (Also use `quadtx` with the same arguments.) In each case, how many function evaluations are used to estimate $\int_0^1 x^5 dx$? What is the absolute error of the estimation?
- Run the program `quadtx(@(x)(cos(x))^2,0,4*pi)`. How many function evaluations are used to estimate $\int_0^{4\pi} (\cos(x))^2 dx$? What is the absolute error of the estimation? Explain what happened. Does `quad(@(x)(cos(x))^2,0,4*pi)` produce the same output? Explain why or why not.
- Describe a nonnegative function $f\colon [0,1] \to [0,1]$ such that the Matlab built-in command `quad` has the same error as in the previous part of this problem. That is, find f such that the command `quad(@(x) f(x),0,1)` outputs 1 and has absolute error at least 1/10.

Exercise 6.14.

- Using the textbook program `quadtx`, try to integrate the function $\frac{1}{3x-1}$ from $x = 0$ to $x = 1$. Do you get an error? If so, explain why the error happened.
- Find a function $f\colon [0,1] \to [0,\infty)$ such that $\lim_{x\to 0^+} f(x) = \infty$ and with $\int_0^1 f(x)dx$ finite. Can the programs `quadtx` and `quad` evaluate your integral with good relative accuracy?
- Find an interval $[a,b]$ and a function $f\colon [a,b] \to \mathbb{R}$ that exceeds the maximum function evaluation count (i.e. produces the maximum function count warning) both for `quadtx` and `quad` when trying to estimate $\int_a^b f(x)dx$.

The trapezoid rule approximates f by piecewise degree one polynomial interpolations (i.e. piecewise linear functions). The integral of the approximating function then approximates the integral of f.

Simpson's rule approximates f by piecewise degree two polynomial interpolations.

One can similarly approximate f by piecewise degree three (or higher) polynomial interpolations. The resulting formulas are known as Newton-Coates formulas.

Alternatively, we can approximate the function f itself on n nodes by a degree $n+1$ interpolating polynomial p_n on the nodes $a_0, \ldots, a_n$. If we do that, then we obtain an error bound from Theorem 5.7 of the form

$$\left| \int_a^b f(x)dx - \int_a^b p_n(x)dx \right| \leq \frac{1}{(n+1)!} \max_{y\in[a,b]} \left| f^{(n+1)}(y) \right| \int_a^b \left| \prod_{i=0}^{n} (x - a_i) \right| dx.$$

As in Theorem 5.13, we would then like to choose the nodes $a_0, \dots, a_n$ to minimize the only term that depends on those nodes, i.e. $\int_a^b |\prod_{i=0}^n (x - a_i)|\, dx$. Such a minimization can be found in the following Theorem.

Theorem 6.15 (**Interpolation Integration Error**). *Let q_n be a monic polynomial of degree at most n. Then*

$$\int_{-1}^{1} |q_n(x)|\, dx \geq \int_{-1}^{1} \left|2^{-n} U_n(x)\right| dx,$$

where $U_n(x) := \frac{\sin((n+1)\cos^{-1} x)}{\sin\cos^{-1} x}$ is the n^{th} ***Chebyshev polynomial of the second kind****.*

Note that the zeros of U_n are $a_i := \cos((i+1)\pi/(n+1))$, $0 \leq i \leq n-1$.

6.3. **Gaussian Quadrature.** Let $c, d \in \mathbb{R}$ with $c < d$, and let $f\colon [c,d] \to \mathbb{R}$ be continuous. Let $w\colon [c,d] \to \mathbb{R}$ be continuous and positive. Let $a_0, \dots, a_n \in [c,d]$ be distinct. The set of polynomials of degree at most n on $[c,d]$ is a vector space of dimension $n+1$. Then there exist constants $b_0, \dots, b_n$ such that

$$\int_c^d f(x)w(x)dx = \sum_{i=0}^{n} b_i f(a_i) \qquad (*)$$

for all polynomials f of degree at most n. In particular, we can choose

$$b_i := \int_c^d w(x) \prod_{j \in \{0,\dots,n\}\colon j \neq i} \frac{x - a_j}{a_i - a_j} dx \qquad (**).$$

As long as the nodes $a_0, \dots, a_n$ are distinct and f has degree at most n, the equality $(*)$ always holds. It turns out we can even choose particular nodes $a_0, \dots, a_n$ such that $(*)$ holds when f has degree at most $2n+1$.

Proposition 6.16 (**Gaussian Quadrature**). *Let $c, d \in \mathbb{R}$ with $c < d$, and let $f\colon [c,d] \to \mathbb{R}$ be continuous. Let $w\colon [c,d] \to \mathbb{R}$ be continuous and positive. Let q be a nonzero polynomial of degree $n+1$ that is w-orthogonal to all degree n polynomials, i.e.*

$$\int_c^d f(x)q(x)w(x)dx = 0, \qquad \text{if } \deg(f) \leq n.$$

Then the zeros $a_0, \dots, a_n$ of q all lie in $[c,d]$ and are distinct. Define $b_0, \dots, b_n$ by $(**)$. Then for any polynomial g of degree at most $2n+1$,

$$\int_c^d g(x)w(x)dx = \sum_{i=0}^{n} b_i g(a_i).$$

Proof. Let g have degree at most $2n+1$. From the Euclidean division algorithm (i.e. long division), we can write

$$g = qp + r,$$

where p, r are polynomials of degree at most n. Then $g(a_i) = r(a_i)$ for all $0 \leq i \leq n$, so our assumption means

$$\int_c^d g(x)w(x)dx = \int_c^d r(x)w(x)dx \overset{(*)}{=} \sum_{i=0}^{n} b_i r(a_i) = \sum_{i=0}^{n} b_i g(a_i).$$

For the zero property, we argue by contradiction. Suppose q changes sign at most n times in $[c,d]$. Denote these zeros as $z_1 < \cdots < z_m$ with $m \leq n$. Then $\prod_{i=1}^m (x - z_i)$ has the same sign changes as q, so $\int_c^d q(x) \prod_{i=1}^m (x - z_i) w(x) dx \neq 0$ or $q = 0$. Either case contradicts the assumption on q. □

Remark 6.17. A polynomial q with the property from Proposition 6.16 (i.e. q being w-orthogonal to all polynomials of degree at most n) can be found by the Gram-Schmidt orthogonalization process, Theorem 4.23. Let V be the vector space consisting of all real polynomials on $[c,d]$. The vectors $v_0, \ldots, v_n$ are defined to be the monomials $x^0, \ldots, x^n$. As functions on $[c,d]$, these vectors are linearly independent. For any two continuous $f, g \colon [c,d] \to \mathbb{R}$, we then use the inner product

$$\langle f, g \rangle := \int_c^d f(x) g(x) w(x) dx.$$

(Exercise: show this is an inner product, i.e. it satisfies Definition 4.10.) The Gram-Schmidt process, then produces polynomials $p_0, \ldots, p_n$ that are an orthonormal basis of V. In particular, for any polynomial f of degree at most $n-1$, $\langle p_n, f \rangle = 0$. This is the assumption of Proposition 6.16.

Example 6.18. For any $x \in (-1,1)$, let $w(x) := \sqrt{1-x^2}$. We show that the Chebyshev polynomials of the second kind are w-orthogonal to each other. In particular, we can choose $q := U_{n+1}$ in Proposition 6.16 for this w. We have

$$\begin{aligned}
\langle U_n, U_m \rangle &= \int_{-1}^1 U_n(x) U_m(x) w(x) dx \\
&= -\int_{\pi}^0 U_n(\cos(\theta)) U_m(\cos(\theta)) \sin(\theta) \sin(\theta) d\theta, \qquad \text{substituting } x = \cos(\theta) \\
&= \int_0^{\pi} \frac{\sin((n+1)\theta) \sin((m+1)\theta) \sin^2(\theta)}{\sin^2(\theta)} d\theta \\
&= \int_0^{\pi} \sin((n+1)\theta) \sin((m+1)\theta) d\theta = 0.
\end{aligned}$$

To get the last equality, let $\theta \in [0, \pi]$. From the cosine addition formula, we have

$$\cos(n+m)\theta = \cos n\theta \cos m\theta - \sin n\theta \sin m\theta,$$

$$\cos(n-m)\theta = \cos n\theta \cos m\theta + \sin n\theta \sin m\theta.$$

Subtracting these and rearranging,

$$2 \sin n\theta \sin m\theta = \cos(n-m)\theta - \cos(n+m)\theta.$$

So $n \neq m$ implies the integral of this quantity is zero.

Gaussian quadrature works exactly for polynomials of low degree. Perhaps surprisingly, Gaussian quadrature also works well for arbitrary continuous functions, as we now show.

Theorem 6.19 (Gaussian Quadrature Convergence). *Let $c, d \in \mathbb{R}$ with $c < d$, and let $f \colon [c,d] \to \mathbb{R}$ be continuous. Let $w \colon [c,d] \to \mathbb{R}$ be continuous and positive. Let q_n be*

a nonzero polynomial of degree $n+1$ that is w-orthogonal to all degree n polynomials. Let $a_{0n}, \ldots, a_{nn}$ be the zeros of q_n (that all lie in $[c,d]$ by Proposition 6.16). Define

$$b_{in} := \int_c^d w(x) \prod_{j \in \{0,\ldots,n\}:\, j \neq i} \frac{x - a_{jn}}{a_{in} - a_{jn}} dx, \qquad \forall\, 0 \leq i \leq n.$$

Then

$$\lim_{n\to\infty} \sum_{i=0}^{n} b_{in} f(a_{in}) = \int_c^d f(x) w(x) dx.$$

Proof. Fix $\varepsilon > 0$. From the Weierstrass Approximation Theorem 6.20, there exists a polynomial $p\colon \mathbb{R} \to \mathbb{R}$ such that $|f(x) - p(x)| < \varepsilon$ for all $x \in [c,d]$. Then there exists N such that if $n \geq N$, then the n^{th} order Gaussian quadrature formula will be equal to the integral of p. (We just need $2N \geq \deg(p)$ for this to happen, by Proposition 6.16.) For any such $n \geq N$, we have by the triangle inequality

$$\begin{aligned}
&\left| \int_c^d f(x)w(x)dx - \sum_{i=0}^{n} b_{in} f(a_{in}) \right| \\
&\quad = \left| \int_c^d f(x)w(x)dx - \sum_{i=0}^{n} b_{in} p(a_{in}) + \sum_{i=0}^{n} b_{in} p(a_{in}) - \sum_{i=0}^{n} b_{in} f(a_{in}) \right| \\
&\quad = \left| \int_c^d (f(x) - p(x))w(x)dx + \sum_{i=0}^{n} b_{in} p(a_{in}) - \sum_{i=0}^{n} b_{in} f(a_{in}) \right| \\
&\quad \leq \left| \int_c^d (f(x) - p(x))w(x)dx \right| + \sum_{i=0}^{n} |b_{in}|\, |p(a_{in}) - f(a_{ij})| \\
&\quad \leq \varepsilon \int_c^d w(x)dx + \varepsilon \sum_{i=0}^{n} b_{in} = 2\varepsilon \int_c^d w(x)dx.
\end{aligned}$$

Here we used $b_{in} > 0$ for all $0 \leq i \leq n$. To see this, fix $0 \leq j \leq n$, let $r(x) := q(x)/(x - a_{jn}) = \prod_{i \in \{0,\ldots,n\}:\, i \neq j} (x - a_{in})$, note $\deg(r^2) \leq 2n$, so

$$\int_c^d (r(x))^2 w(x)dx = \sum_{i=0}^{n} b_{in} [r(a_{in})]^2 = b_{jn} (r(a_{jn}))^2.$$

We also used $\int_c^d 1 \cdot w(x)dx = \sum_{i=0}^{n} b_{in}$.

In summary, for any $\varepsilon > 0$, there exists N such that, for all $n \geq N$, we have

$$\left| \int_c^d f(x)w(x)dx - \sum_{i=0}^{n} b_{in} f(a_{in}) \right| \leq 2\varepsilon \int_c^d w(x)dx.$$

The conclusion follows. □

Theorem 6.20 (Weierstrass approximation theorem). *Let $c, d \in \mathbb{R}$ with $c < d$, and let $f\colon [c,d] \to \mathbb{R}$ be continuous. Let $\varepsilon > 0$. Then there exists a polynomial $p\colon \mathbb{R} \to \mathbb{R}$ such that*

$$\max_{x \in [c,d]} |f(x) - p(x)| < \varepsilon.$$

Theorem 6.21 (**Gaussian Quadrature Error Bound**). *Let $c, d \in \mathbb{R}$ with $c < d$, and let $f\colon [c,d] \to \mathbb{R}$ be continuous. Let $w\colon [c,d] \to \mathbb{R}$ be continuous and positive. Let q be a nonzero polynomial of degree $n+1$ that is w-orthogonal to all degree n polynomials. Let $a_0, \dots, a_n$ be the zeros of q (that all lie in $[c,d]$ by Proposition 6.16). Define*

$$b_i := \int_c^d w(x) \prod_{j \in \{0,\dots,n\}\colon j \neq i} \frac{x - a_j}{a_i - a_j} dx, \qquad \forall\, 0 \leq i \leq n.$$

Then there exists $y \in [c,d]$ such that

$$\int_c^d f(x)w(x)dx - \sum_{i=0}^n b_i f(a_i) = \frac{1}{(2n+2)!} f^{(2n+2)}(y) \int_c^d w(x) \prod_{i=0}^n (x - a_i)^2 dx.$$

Proof. Using Hermite interpolation (Theorem 5.16), let p be a polynomial of degree at most $2n+1$ such that $p(a_i) = f(a_i)$ and $p'(a_i) = f'(a_i)$ for all $0 \leq i \leq n$. Theorem 5.17 says: for any $x \in [c,d]$, there exists $y_x \in (c,d)$ such that

$$f(x) - p(x) = \frac{1}{(2n+2)!} f^{(2n+2)}(y_x) \cdot \prod_{i=0}^n (x - a_i)^2. \qquad (*)$$

Integrating both sides, and using Proposition 6.16 (and $\deg(p) \leq 2n+1$), we get

$$\int_c^d f(x)w(x)dx - \sum_{i=0}^n b_i p(a_i) = \frac{1}{(2n+2)!} \int_c^d f^{(2n+2)}(y_x) w(x) \prod_{i=0}^n (x - a_i)^2 dx.$$

Using the definition of p on the left, and also the mean value theorem for integrals (if $g \geq 0$ is continuous, then $\int_c^d g(x)h(x)dx = g(y) \int_c^d h(x)dx$ for some $y \in [c,d]$),

$$\int_c^d f(x)w(x)dx - \sum_{i=0}^n b_i f(a_i) = \frac{1}{(2n+2)!} f^{(2n+2)}(y) \int_c^d w(x) \prod_{i=0}^n (x - a_i)^2 dx.$$

(Note that $f^{(2n+2)}(y_x)$ is a continuous function of x by $(*)$.) □

Note that if we used the usual interpolating polynomial in the above proof, then the error term in $(*)$ would not have a square on the right side, so the mean value theorem for integrals would not apply in that case.

Exercise 6.22. Recall that we defined $U_0, U_1, \dots$ to be the Chebyshev polynomials of the second kind, where $U_n(x) := \frac{\sin((n+1)\cos^{-1} x)}{\sin \cos^{-1} x}$ for any $x \in (-1,1)$.

- Show that $U_0, U_1, \dots$ satisfy the recursion
$$U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x), \qquad \forall\, n \geq 1, \quad \forall\, x \in (-1,1),$$
where $U_0(x) = 1$ and $U_1(x) = 2x$.
- Show that
$$\frac{d}{dx} T_n(x) = nU_n(x), \qquad \forall\, n \geq 1, \quad \forall\, x \in (-1,1),$$
where $T_0, T_1, \dots$ are the Chebyshev polynomials (of the first kind).

Exercise 6.23. For any $x \in (-1,1)$, let $w(x) := 1/\sqrt{1-x^2}$. Let P_n be the set of polynomials of degree at most n on $[-1,1]$.

- Show that the Chebyshev polynomials $T_0, \ldots, T_n$ are a w-orthogonal basis of P_n.
- Give an explicit formula for the nodes of the Gaussian quadrature that uses $n+1$ nodes and this w.

Exercise 6.24. It is known that

$$\pi = \int_{-1}^{1} \frac{2}{1+x^2} dx.$$

(You can compute this integral using $(d/dx)\tan^{-1}(x) = 1/(1+x^2)$.)

- In Matlab, program your own trapezoid rule to estimate the integral $\int_{-1}^{1} \frac{2}{1+x^2} dx$. Plot the relative error of the integral estimate versus the number n of points used in the trapezoid rule. Do you see any evidence of numerical errors? (You might need to take n to be quite large, e.g. around $30,000$.)
- Use `quadtx` to estimate $\int_{-1}^{1} \frac{2}{1+x^2} dx$. Make a table recording the integral estimates for various tolerance values, including the estimated integral value `Q`, the function evaluate count `fcount`, and the relative error. (For example, consider tolerances of the form 10^{-k} where $k \in \{1, 2, 3, \ldots, 12\}$.) The first two rows of the table might look like this:

```
    tol            Q          fcount     Relative Error
*************************************************************
  1e-001      3.14211764705882       9    1.671e-004
  1e-002      3.14211764705882       9    1.671e-004
```

  Does the relative error decrease as the tolerance decreases?

Exercise 6.25. This exercise investigates Matlab's standard routine for evaluating double integrals. We are particularly interested in computational time. Suppose we integrate the function

$$f(x,y) = \sqrt{1-(x^2+y^2)}$$

over the unit disc $\{(x,y) \in \mathbb{R}^2 \colon x^2+y^2 \leq 1\}$ for decreasing tolerances with the following program.

```
fprintf('    tol       estimated Q      relerror   Computation Time (s)\n')
fprintf('************************************************\n');
for k=1:12
    tol=10^(-k);            %selected tolerance
    tic;                       %this command starts a timer
    Q=dblquad(@(x,y) (sqrt(1-(x.^2+y.^2))).*(x.^2+y.^2<=1),-1,1,-1,1,tol);
    comptime=toc;             % 'toc' records the time elapsed since 'tic'
    actual = (2/3)*pi;        % actual value of the integral
    relerror=abs(Q-actual)/actual;
    fprintf('%8.0e %21.14f %3.2e %7.3f\n', ...
        tol,Q,relerror,comptime);
end
```

- How does the computation time vary with the tolerance?

- Now, modify this program to integrate the function
$$g(x,y) = 1 + \sqrt{1-(x^2+y^2)}.$$
over the unit disc $\{(x,y) \in \mathbb{R}^2 \colon x^2 + y^2 \leq 1\}$. How does the computation time vary with the tolerance? If you observe long computation times, try to explain why they have occurred.

Exercise 6.26. Estimate the integral
$$\int_0^1 \frac{\cos(x)}{\sqrt{x-x^2}} dx.$$
Use whatever built in Matlab functions you want to use (more is perhaps better than less). Do your best to justify what the correct answer is.

(Hint: consider making the substitution $s = 2x - 1$ and then $s = \cos\theta$.)

7. Numerical Solution of ODEs

7.1. **Introduction.** In linear algebra, we discuss extensively equations of the form $Ax = b$. The matrix A and vector b are given, and an important issue is existence and uniqueness of vectors x satisfying $Ax = b$. A priori, it is not obvious whether or not a vector x satisfies $Ax = b$. Indeed, there are examples of A, b where the equation $Ax = b$ has no solution (e.g. if A is a square matrix with determinant zero, and b is not in the column space of A). And there are examples of A, b where $Ax = b$ has infinitely many solutions (e.g. if we have one solution x, then we can add to x any vector in the null space of A). Finally, sometimes the solution x exists and is unique (e.g. when A is invertible).

The situation for differential equations is loosely analogous. An example of an ordinary differential equation is an equation for a real function y of the form $y'(t) = f(t, y(t))$ for all $t \in [a, b]$. Here f is given and "solving" the equation amounts to finding a y satisfying the equation. Sometimes a solution is unique, and sometimes it is not (e.g. the equation $y'(t) = 1$ for all $t \in \mathbb{R}$ has multiple solutions. For any $c \in \mathbb{R}$, the equation $y(t) := t + c$ satisfies $y'(t) = 1$. However, if we assume that $y(0)$ is fixed to be e.g. equal to zero, then $y(t) := t$ is the only solution.) Sometimes a solution exists, and sometimes it does not. For example, consider the equation
$$y'(t) = 1 + (y(t))^2, \qquad t \in \mathbb{R}.$$
The function $y(t) = \tan(t)$ satisfies this equation and $y(0) = 0$, since $\tan'(t) = \sec^2(t)$ and $1 + \tan^2(t) = \sec^2(t)$. However, this solution only exists for $t \in (-\pi/2, \pi/2)$. It cannot be extended to a continuous function on a larger interval.

Definition 7.1 (**Ordinary Differential Equation, Initial Value Problem, One Variable**). Let $a, b \in \mathbb{R}$ with $a < b$. Let $f \colon [a, b] \times \mathbb{R} \to \mathbb{R}$. Let $t_0 \in [a, b]$ and let $y_0 \in \mathbb{R}$. An **initial value problem** is the following example of an ordinary differential equation.
$$\begin{cases} \frac{dy}{dt}(t) &= f(t, y(t)), \qquad \forall\, t \in [a, b] \\ y(t_0) &= y_0. \end{cases}$$

Since ordinary differential equations often model physical phenomena, the existence and uniqueness of solutions is an important issue. If a solution is not unique, then a physical model using ODEs might not uniquely represent the physical process the ODE is modeling.

If a solution does not exist, then perhaps the model does not appropriately describe the physical process.

Theorem 7.2 (**Local Existence, Peano**). *Suppose there exists $\varepsilon > 0$ such that f is continuous in the region*

$$R := \{(t, y) \in \mathbb{R}^2 \colon |t - t_0| \leq \varepsilon, \quad |y - y_0| \leq \varepsilon\}.$$

Let $m := \max_{(y,t)\in R} |f(y, t)|$. Then the initial value problem from Definition 7.1 has a solution y in the region $\{t \in [a, b] \colon |t - t_0| \leq \varepsilon \min(1, 1/m)\}$.

The assumption of Theorem 7.2 unfortunately is insufficient to prove uniqueness of a solution $y(t)$. For example, the initial value problem

$$\begin{cases} \frac{dy}{dt}(t) &= (y(t))^{2/3}, \qquad \forall\, t \in [0, 1] \\ y(0) &= 0, \end{cases}$$

satisfies the hypothesis of Theorem 7.2 with $f(t, y) := y^{2/3}$, $t_0 = y_0 = 0$, $\varepsilon = 1$, and $m = 1$. But $y(t) := 0$ solves the problem, as does $y(t) := (t/3)^3$, since $y'(t) = (t/3)^2 = (y(t))^{2/3}$.

In order to get uniqueness of the initial value problem, we need slightly stronger assumptions on f.

Theorem 7.3 (**Short-Time Existence and Uniqueness, Picard**). *Suppose there exists $\varepsilon > 0$ such that f and $\partial f/\partial y$ are continuous in the region*

$$R := \{(t, y) \in \mathbb{R}^2 \colon |t - t_0| \leq \varepsilon, \quad |y - y_0| \leq \varepsilon\}.$$

Then the initial value problem from Definition 7.1 has a unique solution y in the region $\{t \in [a, b] \colon |t - t_0| \leq \varepsilon \min(1, 1/m)\}$.

In the example $y'(t) = 1 + (y(t))^2$, we have $f(t, y) = 1 + y^2$, $\partial f/\partial y = 2y$, so Theorem 7.3 applies, giving a short-time existence and uniqueness for solutions of the initial value problem. However, as we saw above, the solution does not exist for all times t since $y(t)$ approaches infinity at a finite time $t = \pi/2$. If we want some condition guaranteeing long-time existence and uniqueness, this condition must eliminate the example $f(t, y) = 1 + y^2$. The issue with long-time existence here is that $\partial f/\partial y$ can become quite large when y is large. Put another way, $f(t, y) - f(t, z)$ can be much larger than $|y - z|$. If we eliminate this possibility, it turns out we can get long-time existence and uniqueness.

Theorem 7.4 (**Long-Time Existence and Uniqueness, Picard**). *Suppose f is uniformly Lipschitz continuous in the y variable, i.e. there exists $L > 0$ such that*

$$|f(t, y) - f(t, z)| \leq L\,|y - z|\,, \qquad \forall\, t \in [a, b],\, \forall\, y, z \in \mathbb{R}.$$

Then the initial value problem from Definition 7.1 has a unique solution y in the region $t \in [a, b]$.

Note that $f(t, y) := y^{2/3}$ is not Lipschitz continuous, so Theorem 7.3 does not apply in this case.

We will be constructing approximate solutions of differential equations. So, it is important that a small perturbation to a solution $y(t)$ at some fixed time t will not change the behavior of y too much at other times. The following Theorem proves this continuity property.

Theorem 7.5 (**Continuity of Solutions**). *Suppose f is uniformly Lipschitz continuous in the y variable, i.e. there exists $L > 0$ such that*

$$|f(t,y) - f(t,z)| \leq L\,|y - z|\,, \qquad \forall\, t \in [a,b],\, \forall\, y, z \in \mathbb{R}.$$

Fix $t_0 \in [a,b]$. Suppose $z\colon \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ is a function such that, for any $y_0 \in \mathbb{R}$, $t \mapsto z(t,y_0)$ is a solution of the initial value problem from Definition 7.1 (i.e. with initial value $z(t_0,y_0) = y_0$). Then z is continuous.

7.1.1. *Euler's Method.* Very simple looking differential equations do not have explicit solutions in terms of elementary functions. So, solving differential equations numerically is fairly important. The simplest way to "solve" a differential equation on a computer is Euler's method. That is, we just take the differential equation and replace any derivative by its discrete counterpart, as in Section 6.1.

Example 7.6. Consider the initial value problem

$$\begin{cases} \frac{dy}{dt}(t) & = 1 + (y(t))^2, \qquad \forall\, t \in [a,b] \\ y(0) & = 0. \end{cases}$$

As we mentioned above, the unique solution to this problem is $y(t) = \tan(t)$, valid for all $t \in (-\pi/2, \pi/2)$. If we did not know this ahead of time, we could try to solve this equation on a computer by discretizing it as

$$\begin{cases} \frac{y(t+h)-y(t)}{h} & \approx 1 + (y(t))^2, \qquad \forall\, t \in [a,b] \\ y(0) & = 0, \end{cases}$$

for some small $h > 0$. We can then solve for $y(t)$ recursively. Solving for $y(t+h)$, we have the recursion

$$y(t+h) \approx y(t) + h(1 + (y(t))^2).$$

For example, if we know $y(0) = 0$, then $y(h) = y(0) + h(1 + (y(0))^2)$, $y(2h) = y(h) + h(1 + (y(h))^2)$, and so on. We can then plot our approximate solution in Matlab.

```
n=500;
a=0;
b=pi/2 - .01;
h=(b-a)/n;
y=zeros(1,n);
for i=1:n-1
    y(i+1)= y(i)+ h*(1+(y(i))^2 );
end
t=linspace(a,b,n);
plot(t, y, t, tan(t));
legend('approximate solution', 'exact solution');
title("Solutions of the equation y'=1+y^2 with h="+num2str(h));
```

Taylor's Theorem 7.7 implies that $|f(x+h) - [f(x) + hf'(x)]|$ is approximately of size h^2. We therefore say that the **local truncation error** of Euler's method in the above example is of order h^2. Since we expect that the global truncation error is roughly the sum of about $1/h$ local errors, we say that Euler's method has **global truncation error** of h (since $h^2/h = h$).

When the global truncation error is of size h^n, we say the differential equation method is of **order** n. That is, Euler's method is an order one method for solving differential equations.

Using more terms in the Taylor expansion in Euler's method would then result in smaller local and global truncation error.

Theorem 7.7 (**Taylor Series, with integral remainder**). *Let $f\colon \mathbb{R} \to \mathbb{R}$ have $n+1$ continuous derivatives. Let $t, h \in \mathbb{R}$. Then*

$$f(t+h) = f(t) + \frac{f'(t)}{1!}h + \cdots + \frac{f^{(n)}(t)}{n!}h^n + \frac{1}{n!}\int_t^{t+h} (t+h-s)^n f^{(n+1)}(s)ds.$$

Consequently,

$$\left| f(t+h) - \left[f(t) + \frac{f'(t)}{1!}h + \cdots + \frac{f^{(n)}(t)}{n!}h^n \right] \right| \leq |h|^{n+1} \max_{s\in[t,t+h]} \left| f^{(n+1)}(s) \right|.$$

Proof. The case $n = 0$ follows from the Fundamental Theorem of Calculus. We now induct on n. Assume that the case $n-1$ holds. Integrating by parts we see that

$$\begin{aligned}
&\frac{1}{(n-1)!}\int_t^{t+h} (t+h-s)^{n-1} f^{(n)}(s)ds \\
&= -\frac{1}{n!}[(t+h-s)^n f^{(n)}(s)]_{s=t}^{s=t+h} + \frac{1}{n!}\int_t^{t+h} (t+h-s)^n f^{(n+1)}(s)ds \\
&= \frac{1}{n!}h^n f^{(n)}(t) + \frac{1}{n!}\int_t^{t+h} (t+h-s)^n f^{(n+1)}(s)ds
\end{aligned}$$

For the final, inequality, note that

$$\left| \int_t^{t+h} (t+h-s)^n f^{(n+1)}(s)ds \right| \leq \int_t^{t+h} |h|^n \left| f^{(n+1)}(s) \right| ds \leq |h|^{n+1} \cdot \max_{s\in[t,t+h]} \left| f^{(n+1)}(s) \right|.$$

□

Example 7.8. Consider the initial value problem

$$\begin{cases} \frac{dy}{dt}(t) &= 1 + (y(t))^2, \qquad \forall\, t \in [a,b] \\ y(0) &= 0. \end{cases}$$

Above, we discretized this differential equation as

$$y(t+h) - y(t) \approx hy'(t).$$

$$y(t+h) \approx y(t) + hy'(t) = y(t) + h(1 + (y(t))^2).$$

If we add another term in the Taylor expansion, and use $y''(t) = 2y(t)y'(t) = 2y(t)(1 + (y(t))^2)$, we would instead get

$$y(t+h) \approx y(t) + hy'(t) + \frac{1}{2}h^2 y''(t) = y(t) + h\Big(1 + (y(t))^2 + \frac{h}{2}(2y(t)(1 + (y(t))^2))\Big).$$

Adding yet another term in the Taylor expansion, and using

$$y'''(t) = 2y(t)y''(t) + 2(y'(t))^2 = 4(y(t))^2(1 + (y(t))^2) + 2(1 + (y(t))^2)^2,$$

we get

$$\begin{aligned} y(t+h) &\approx y(t) + hy'(t) + \frac{1}{2}h^2y''(t) + \frac{1}{6}h^3y'''(t) \\ &= y(t) + h\Big(1 + (y(t))^2 + \frac{h}{2}\Big[2y(t)(1 + (y(t))^2) \\ &\qquad + \frac{h}{3}\Big(4(y(t))^2(1 + (y(t))^2) + 2(1 + (y(t))^2)^2\Big)\Big]\Big). \end{aligned}$$

We can then plot our approximate solutions in Matlab.

```
n=100;
a=0;
b=pi/2 - .01;
h=(b-a)/n;
x=zeros(1,n);  y=x; z=x;
for i=1:n-1
    x(i+1)= x(i)+ h*(1+(x(i))^2 );
    y(i+1)= y(i)+ h*(1+(y(i))^2 + (h/2)* 2*(y(i))*(1+(y(i))^2)  );
    z(i+1)= z(i)+ h*(1+(z(i))^2 + (h/2)*( 2*(z(i))*(1+(z(i))^2) ...
     + (h/3)*(4*(z(i))^(2) *(1+(z(i))^(2)) + 2*(1+(z(i))^(2))^(2))));
end
t=linspace(a,b,n);
plot(t, x, t, y, t, z, t, tan(t));
xerror = sum(abs(x-tan(t)))/sum(abs(tan(t)));
yerror = sum(abs(y-tan(t)))/sum(abs(tan(t)));
zerror = sum(abs(z-tan(t)))/sum(abs(tan(t)));
legend("1st order Euler, relerror "+num2str(xerror), ...
    "2nd order Euler, relerror "+num2str(yerror), ...
    "3rd order Euler, relerror "+num2str(zerror),'exact solution');
title("Solutions of the equation y'=1+y^2 with h="+num2str(h));
```

Though the local truncation error of the three term Taylor expansion should be smaller than the two term expansion, there seems to be little difference between them.

Example 7.9. By introducing extra variables, we can numerically solve ODEs with more than one derivative. Consider the following ODE that models the planetary motion of a planet-moon system together with a star. Unless otherwise specified, all functions below are functions of a real variable t.

The planet has mass $\mu := .012277471$, the star has mass $\widehat{\mu} := 1 - \mu$, and the moon has negligible mass. All three bodies are moving in the same plane. (I could not find details on the coordinates, but I think (u_1, u_2) are the position coordinates of the moon, the planet is considered to be stationary at the origin, and the sun is considered to be very far away to the extreme negative u_1 direction.)

$$
\begin{aligned}
u_1'' &= u_1 + 2u_2' - \widehat{\mu}\frac{u_1+\mu}{d_1} - \mu\frac{u_1-\widehat{\mu}}{d_2} \\
u_2'' &= u_2 - 2u_1' - \widehat{\mu}\frac{u_2}{d_1} - \mu\frac{u_2}{d_2} \\
d_1 &= ((u_1+\mu)^2 + u_2^2)^{3/2} \\
d_2 &= ((u_1-\widehat{\mu})^2 + u_2^2)^{3/2},
\end{aligned}
$$

with initial condition

$$
\begin{aligned}
&u_1(0) = .994, \quad u_2(0) = 0, \quad u_1'(0) = 0, \\
&u_2'(0) = -2.00158510637908252240537862224.
\end{aligned}
$$

We can convert this differential equation into one that just uses single derivatives by defining

$$
y_1 := u_1, \quad y_2 := u_2, \quad y_3 := u_1', \quad y_4 := u_2'.
$$

We can then rewrite the system as

$$
\begin{aligned}
y_1' &= y_3 \\
y_2' &= y_4 \\
y_3' &= y_1 + 2y_4 - \widehat{\mu}\frac{y_1+\mu}{d_1} - \mu\frac{y_1-\widehat{\mu}}{d_2} \\
y_4' &= y_2 - 2y_3 - \widehat{\mu}\frac{y_2}{d_1} - \mu\frac{y_2}{d_2} \\
d_1 &= ((y_1+\mu)^2 + y_2^2)^{3/2} \\
d_2 &= ((y_1-\widehat{\mu})^2 + y_2^2)^{3/2},
\end{aligned}
$$

with initial condition

$$
\begin{aligned}
&y_1(0) = .994, \quad y_2(0) = 0, \quad y_3(0) = 0, \\
&y_4(0) = -2.00158510637908252240537862224.
\end{aligned}
$$

The ODE can be solved in Matlab in the following way

```
function planetaryode
tspan = [0 30];
y0 = [.994 0 0 -2.00158510637908252240537862224];
[t,y] = ode23(@(t,y) myode(t,y), tspan, y0);
plot(y(:,1),y(:,2));
end

function dydt = myode(t,y)
mu=.012277471;
d(1) = ((y(1) + mu).^2 + (y(2)).^2 ).^(3/2);
d(2) = ((y(1) -(1-mu)).^2 + (y(2)).^2 ).^(3/2);

dydt = zeros(4,1);
dydt(1) = y(3);
```

```
dydt(2) = y(4);
dydt(3)= y(1)+2*y(4)- (1-mu)* (y(1)+mu)./d(1)  -mu *(y(1)- (1-mu))./ d(2);
dydt(4)= y(2)-2*y(3)- (1-mu)* (y(2))./d(1)  -mu *(y(2))./ d(2);
end
```

7.2. **Runge-Kutta Methods.** In the previous section, we used Taylor expansions to give first, second, and third order Euler methods for a specific initial value problem. One downside of the higher order Euler method is that we need to explicitly compute some derivatives of f. Instead of computing these derivatives explicitly, we can approximate them with their own difference quotients. This is the idea of Runge-Kutta methods.

The first order Runge-Kutta method is identical to Euler's method. So, we begin with a derivation of the second order Runge-Kutta method for the differential equation

$$y_t(t) = f(t, y(t)).$$

Differentiation both sides and applying the chain rule, we have

$$y_{tt} = f_t + f_y y_t = f_t + f_y f, \qquad y_{ttt} = f_{tt} + f_{ty} y_t + f_y y_{tt} + f_{yt} y_t + f_{yy} y_t^2.$$

We can then write the Taylor expansion of y as

$$\begin{aligned} y(t+h) &= y(t) + hy'(t) + \frac{h^2}{2} y''(t) + O(h^3) \\ &= y(t) + hf + \frac{h^2}{2}(f_t + f_y f) + O(h^3) \\ &= y(t) + \frac{h}{2} f + \frac{h}{2}(f + hf_t + hf_y f) + O(h^3). \end{aligned}$$

The last term can be rewritten using a Taylor expansion of f

$$f(t+h, y+hf) = f(t,y) + hf_t + hf f_y + O(h^2).$$

Substituting this into our expression for $y(t+h)$, we get

$$\begin{aligned} y(t+h) &= y(t) + \frac{h}{2} f + \frac{h}{2}(f(t+h, y+hf) + O(h^2)) + O(h^3) \\ &= y(t) + \frac{h}{2}\Big(f(t,y) + f(t+h, y+hf)\Big) + O(h^3). \end{aligned}$$

Definition 7.10 (**Second Order Runge-Kutta Method**)**.** Here $t_0 \in [a,b]$. Consider the initial value problem

$$\begin{cases} \frac{dy}{dt}(t) &= f(t, y(t)), \qquad \forall\, t \in [a,b] \\ y(t_0) &= y_0. \end{cases}$$

The **second order Runge-Kutte Method** for solving this initial value problem is an iterative (recursive) method that proceeds as follows. First, define a step size $h > 0$. Let $t := t_0$. Then, compute the following recursion for $y(t+h), y(t+2h), y(t+3h)$, and so on.

$$y(t+h) := y(t) + \frac{h}{2}\Big(f(t,y) + f(t+h, y+hf(t, y(t)))\Big).$$

As suggested by its name, the second order Runge-Kutte method is a second order numerical method for solving the initial value problem from Definition 7.1. There is an analogous fourth order Runge-Kutte method, but we will not give its derivation.

Definition 7.11 (**Fourth Order Runge-Kutta Method**). Let $t_0 \in [a,b]$. Consider the initial value problem

$$\begin{cases} \frac{dy}{dt}(t) &= f(t,y(t)), \qquad \forall\, t \in [a,b] \\ y(t_0) &= y_0. \end{cases}$$

The **fourth order Runge-Kutte Method** for solving this initial value problem is an iterative (recursive) method that proceeds as follows. First, define a step size $h > 0$. Let $t := t_0$. Then, compute the following recursion for $y(t+h), y(t+2h), y(t+3h)$, and so on.

$$\begin{cases} a_1 &= hf(t,y(t)) \\ a_2 &= hf(t+h/2, y(t)+a_1/2) \\ a_3 &= hf(t+h/2, y(t)+a_2/2) \\ a_4 &= hf(t+h, y(t)+a_3) \end{cases}$$

$$y(t+h) := y(t) + \frac{1}{6}\Big(a_1 + 2a_2 + 2a_3 + a_4\Big).$$

Example 7.12. Consider the initial value problem

$$\begin{cases} \frac{dy}{dt}(t) &= 1 + (y(t))^2, \qquad \forall\, t \in [a,b] \\ y(0) &= 0. \end{cases}$$

Let's compare the performance of our Euler methods with the second and fourth order Runge-Kutta methods.

```
n=1000;
a=0;
b=pi/2 - .01;
h=(b-a)/n;
x=zeros(1,n);  y=x; z=x; r=x; k=x;
f= @(y) 1+y.^2;
for i=1:n-1
    x(i+1)= x(i)+ h*(1+(x(i))^2 );      % 1st order Euler, then 2nd,3rd
    y(i+1)= y(i)+ h*(1+(y(i))^2 + (h/2)* 2*(y(i))*(1+(y(i))^2)  );
    z(i+1)= z(i)+ h*(1+(z(i))^2 + (h/2)*( 2*(z(i))*(1+(z(i))^2) ...
     + (h/3)*(4*(z(i))^(2) *(1+(z(i))^(2)) + 2*(1+(z(i))^(2))^(2))));
    r(i+1) = r(i)+ (h/2)* ( f(r(i)) + f(r(i)+ h *f(r(i))) );  % 2nd RK
    a1= h*f(k(i));
    a2= h* f(k(i)+ a1/2);
    a3= h* f(k(i)+ a2/2);
    a4= h* f(k(i)+a3);
    k(i+1)= k(i)+(1/6)*(a1+2*a2+2*a3+a4);  %4th RK
end
t=linspace(a,b,n);
plot(t, x, t, y, t, z, t, r, t, k, t, tan(t));
xerror = sum(abs(x-tan(t)))/sum(abs(tan(t)));
yerror = sum(abs(y-tan(t)))/sum(abs(tan(t)));
zerror = sum(abs(z-tan(t)))/sum(abs(tan(t)));
rerror = sum(abs(r-tan(t)))/sum(abs(tan(t)));
```

```
kerror = sum(abs(k-tan(t)))/sum(abs(tan(t)));
legend("1st order Euler, relerror "+num2str(xerror), ...
    "2nd order Euler, relerror "+num2str(yerror), ...
    "3rd order Euler, relerror "+num2str(zerror), ...
    "2nd order RK, relerror "+num2str(rerror), ...
    "4th order RK, relerror "+num2str(kerror), ...
    'exact solution');
title("Solutions of the equation y'=1+y^2 with h="+num2str(h));
```

Though the local truncation error of the three term Taylor expansion should be smaller than the two term expansion, there seems to be little difference between them.

Exercise 7.13. Consider the following initial value problem

$$y'(t) = \sqrt{|t|}, \qquad \forall\, t \in [-2, 2], \qquad y(-2) = -1.$$

- Verify that there is a solution to this initial value problem of the form
$$f(t) = \begin{cases} \frac{2\sqrt{-t}t}{3} + c & , \text{if } t < 0 \\ \frac{2t^{3/2}}{3} + c & , \text{if } t \geq 0. \end{cases}$$
Moreover, find the correct value of c.
- Using Matlab, compute a numerical solution of this initial value problem with Euler's method and different step sizes h, on the interval $[-2, 2]$.
- Should you be concerned with your results from Euler's method? For what value of t is the absolute error the highest? Naïvely, one might measure the error of Euler's method, by just examining the final endpoint of the interval $y(2)$, and comparing the computed value with the exact value. Is this sensible in this example?

Exercise 7.14. This exercise begins by investigating some solutions of the Lorentz equations which are defined for $y\colon \mathbb{R} \to \mathbb{R}^3$ by

$$y'(t) = A(t)y(t)$$
$$y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{pmatrix} \qquad \text{where} \qquad A(t) = \begin{pmatrix} -\beta & 0 & y_2(t) \\ 0 & -\sigma & \sigma \\ -y_2(t) & \rho & -1. \end{pmatrix}$$

Here $\sigma = 10$, $\beta = \frac{8}{3}$, and we vary the value of ρ. (Since A depends on $y_2(t)$, the differential equation $y' = Ay$ is nonlinear, leading to some interesting behavior.) Solutions of this system display various periodicities that give the appearance of a particle orbiting around two different points. These two points are called attractors. Since the behavior of the solution seems rather unpredictable, the solution y is called chaotic. In our investigation, the initial value of y is started near one of the two attractors, i.e. with $\eta = +\sqrt{\beta(\rho - 1)}$ there is an attractor at the point $r_1 = (\rho - 1, \eta, \eta) \in \mathbb{R}^3$ and we start at the initial value $r_1 + (0, 0, 3) \in \mathbb{R}^3$.

- In varying our parameter ρ, using the program `lorenzgui`, check the values of ρ which produce non-chaotic orbits. Then, we label the periodicity of the orbits with a series of pluses and minuses, where a plus denotes a circuit around one attractor and a minus denotes a circuit around the other attractor.

For example, if the solution y goes around attractor one, then attractor two, then one, then two, etc. label this periodicity as $+-$.

- Run the Matlab demo function `orbitode` which demonstrates the use of events in an ODE solver. This solution of this problem gives the orbit of a small body around two larger bodies. (Think for example about a spacecraft's orbit under the influence of the Earth and moon's gravity.) What roles to the variables `te`, `ye`, `y`, `ie` play in the program?
- Mimic the program `orbitode` and implement an event function in the Lorenz problem. Using this event function, find the periods of the periodic orbits which occur for different ρ values. That is, for a given ρ value, find the smallest nonzero time T such that $y(t+T) = y(t)$. (This equality will probably not hold exactly, so you should probably check for the smallest nonzero time T such that $|y(t+T) - y(t)| < 10^{-6}$ for all larger t, or using some other small number other than 10^{-6}).

Exercise 7.15. This exercise investigates the numerical solutions of the following systems of equations which appears in mathematical ecology as a predator prey system:

$$\frac{dr}{dt} = 2r - \alpha r f$$

$$\frac{df}{dt} = -f + \alpha r f$$

where $r(0) = r_0$, $f(0) = f_0$, t is time, $r(t)$ is the number of rabbits, $f(t)$ is the number of foxes, and α is a positive constant. This system has no known analytical solution, but it is known that the solutions for r and f are periodic, with the same period.

Use the following code if you wish.

```
function [t,y]=predpreymod(alpha,rstart,fstart,tstop,tolpick)
%Volterrra-Lotka predator prey model
%alpha is parameter in the eqn
%
% the eqn is of the form
% [r';f']=A*[r;f]
%
% alpha= parameter in the differential equations
% rstart= starting population of rabbits
% fstart= starting population of foxes
% tstop= time at which to stop the calculation
% tolpick= relative tolerance chosen to give the calculator
%
% e.g. predpreymod(1,10,5,10,1.e-6)

y0=[rstart;fstart];                 %initial cond
tspan=[0,tstop];                    %span of time
opts=odeset('reltol',tolpick,'outputfcn',@odephas2,'events',@events);
F=@(t,y) [2*y(1)-alpha*y(2)*y(1); alpha*y(1)*y(2)-y(2)]; %defining eqn

[t,y,te,ye,ie]=ode23(F,tspan,[rstart; fstart],opts);    %solve the eqn
```

```
%add title to phase plot created by matlab
usetitle=strcat('Original LV Phase Plane Plot alpha=',num2str(alpha), ...
     '--period~',num2str(mode(diff(te))), ...
     '--r_0:',num2str(rstart), ...
     '--f_0:',num2str(fstart));
title(usetitle);
xlabel('Rabbit Population');
ylabel('Fox Population');

%post processing
figure;
plot(t,y(:,1),'r-',t,y(:,2),'b-');
legend('Rabbit Population','Fox Population','Location','Northwest');
xlabel('Time');
ylabel('Population');
usetitle=strcat('Original LV Pred Prey Model alpha=',num2str(alpha), ...
     '--period~',num2str(mode(diff(te))), ...
     '--r_0:',num2str(rstart), ...
     '--f_0:',num2str(fstart));
title(usetitle);

    function [value,isterminal,direction]=events(t,y)
    %determine the period by tracking the min/max points of either of the
    %curves

        ydot=F(t,y);
        value=ydot(1);  %track the change in the rabbit population
        isterminal=0;   %do not stop at critical values
        direction=1;    %track local minima
    end
end
```

- Compute some solutions for different values of r_0, f_0 and α. For example, once you save the function file as `predpreymod.m` in the current directory, you could call this function with the command `predpreymod(1,1,1,10,10^-5)`
- You might observe some periodicity behavior in these plots. To examine this behavior, consider a particular substitution. Note that $(r, f) = (\frac{1}{\alpha}, \frac{2}{\alpha})$ is a stable equilibrium point (i.e. when r, f take these values at one time, they take these values at all future times, since $r'(t) = f'(t) = 0$ in this case). With this in mind, we adjust our functions by this value as follows. For any $t \in \mathbb{R}$, define

$$u(t) := r(t) - \frac{1}{\alpha}$$

$$v(t) := f(t) - \frac{2}{\alpha}$$

Then we have $u' = r'$, $v' = f'$ and ignoring the uv terms in our equations, derive the approximations

$$u' \approx -v, \qquad v' \approx 2u.$$

Taken together these equations yield

$$v'' \approx 2u' \approx -2v.$$

The equation $v'' = -2v$ is exactly the equation for a harmonic oscillator (notice if we solve for v we necessarily solve for u). For instance, an analytic solution is

$$v = A\cos(\sqrt{2}t + \phi)$$

where v has period $\sqrt{2}\pi \approx 4.44288$. In fact, all solutions will have such a period, and indeed, this is almost exactly the period observed for our third example above. Therefore, these manipulations give a valid analysis of the original system, where we observe oscillatory phenomena. Moreover, from this formal analysis, we can trust the solutions produced by the computer, and not have to worry that they are an artifact.

- Now investigate solutions to a system which is a modification of our previous equations

$$\frac{dr}{dt} = 2\left(1 - \frac{r}{R}\right) - \alpha r f$$
$$\frac{df}{dt} = -f + \alpha r f$$

where we are in the same situation as before except now R is a constant, and R essentially represents the maximum allowable population of rabbits. Compare the solutions of this equation to solutions of our previous equation under similar conditions.

Exercise 7.16. We now investigate solutions to equations which describe the flight of a projectile with wind. In this case, we model a cannonball shot from the origin in the direction of the positive x-axis. We also treat the x-axis as the ground, so we consider the flight of the projectile done when the projectile hits the ground. The equations are as follows

$$x'(t) = v\cos(\theta), \qquad y'(t) = v\sin(\theta)$$
$$\theta'(t) = -\frac{g}{v}\cos(\theta), \qquad v'(t) = -\frac{D}{m} - g\sin(\theta), \qquad \forall\, t \geq 0,$$

where $\theta = \theta(t)$ is the angle that the velocity vector makes with the x-axis, $x(t)$ and $y(t)$ are the usual spacial coordinates, $v(t)$ the speed and $D(t)$ is a function of time representing the drag with

$$D(t) = \frac{c\rho s}{2}\left((x'(t) - w(t))^2 + (y'(t))^2\right)$$

where $w(t)$ is also a function of time representing the wind, $c = .2$ is the drag coefficient, $\rho = 1.29$ kg/m^3 is the density of the air, and $s = .25$m^2 is the projectile's cross-sectional area. Also, in the above equations we have $v_0 = 50$ m/s the initial speed, $m = 15$ kg the weight of the cannonball and $g = 9.81$m/s^2 acceleration due to gravity.

- Plot a range of different initial angles for trajectories on one plot, where one plot corresponds to one wind function. Then, report the attributes of the trajectory of maximal distance, with its angle in degrees noted. When making these plots, consider the following four wind functions

(1) $w(t) = 0$.
(2) $w(t) = -10$.
(3) $w(t) = 10$ if $\lfloor t \rfloor$ is even, and zero otherwise.
(4) w(t) is a Gaussian random variable with mean 0, standard deviation 10.

In Matlab, item (3) is `10*(\sim mod(floor(t),2))*(t >= 0)`, and item (4) is `10*randn`

- In each plot, report (in the following order) information for the maximal projectile: the function w, the initial angle of flight in degrees, the flight time, the distance of the projectile, the impact speed (keeping in mind an initial speed of 50m/s), and the number of steps taken by the solver in making the calculation. Choose a relative tolerance of $1.e-6$ in all of our trials so that we are able to compare the performance of the calculations under each of the wind functions.
- Which wind function requires the most computation time?

7.3. **Multistep Methods.** For the initial value problem from Definition 7.1, we considered recursive solutions on a computer. We begin with a step size $h > 0$ and a value $y_0 := y(t_0)$, and y_n is a computed approximation to the value of $y(nh + t_0)$. A **multistep method** is a recursion of the form

$$a_k y_n + a_{k-1} y_{n-1} + \cdots + a_0 y_{n-k} = h\Big(b_k f_n + b_{k-1} f_{n-1} + \cdots + b_0 f_{n-k}\Big), \forall\, n \geq k.$$

where f_n is the computed approximation to the value of $f(nh+t_0, y(nh+t_0))$, and $a_0, \ldots, a_k$, $b_0, \ldots, b_k$ are fixed constants determined by the particular method.

For example, Euler's method can be written as $y_{n+1} = y_n + hf(y_n)$, so that $a_1 = 1$, $a_0 = -1$, $b_0 = 1$.

Example 7.17. Suppose $y\colon [0, \infty) \to \mathbb{R}$ satisfies $y(0) := 10$, and y satisfies the following ODE:

$$y'(t) = -20y(t), \qquad \forall\, 0 \leq t \leq 100.$$

Note that an exact solution is $y(t) := 10e^{-20t}$. Euler's method with step size h satisfies

$$y_{n+1} = y_n + h(-20)y_n = y_n(1 - 20h), \qquad \forall\, n \geq 0.$$

When $y_0 := 10$, this recursion has a unique solution

$$y_n = 10(1 - 20h)^n, \qquad \forall\, n \geq 0.$$

Note that if $h > 1/10$, then $\lim_{n\to\infty} |y_n| = \infty$ while $\lim_{t\to\infty} y(t) = 0$. So, choosing a small step size is very important for solving this ODE via Euler's method.

Theorem 7.5 implies that small errors in the computed solution of an ODE do not affect the accuracy of the result, *for short times*. However, when we compute solutions for long time intervals, the accumulated errors could become quite large. Unfortunately, large accumulated errors can actually occur. And the occurrence of such errors can be characterized in terms of the coefficients $a_0, \ldots, a_k, b_0, \ldots, b_k$. Define

$$p(z) := a_k z^k + a_{k-1} z^{k-1} + \cdots + a_1 z + a_0, \qquad \forall\, z \in \mathbb{C}.$$

$$q(z) := b_k z^k + b_{k-1} z^{k-1} + \cdots + b_1 z + b_0, \qquad \forall\, z \in \mathbb{C}.$$

Example 7.18. Recall that Euler's method with step size h satisfies

$$y_{n+1} - y_n = hf_n, \qquad \forall\, n \geq 0.$$

In this case, we have $k = 1$, $a_1 = 1$, $a_0 = -1$, $b_0 = 1$ and

$$p(z) = z - 1, \qquad q(z) = 1.$$

Note that p has a single zero at the value $z = 1$, $p(1) = 0$, and $p'(1) = 1 = q(1)$.

Theorem 7.19 (Convergence of Multistep Methods). *Suppose we have an initial value problem (from Definition 7.1) on an interval $t \in [t_0, t_1]$ such that the function $f(t, y)$ satisfies the Lipschitz assumption of Theorem 7.4. Let $y(t, h)$ denote the computed approximation to $y(t)$ with step size h for a given multistep method. Assume that $\lim_{h\to 0^+} y(t_0 + jh, h) = y(t_0)$ for all $0 \leq j < k$. Then the multistep method converges ($\lim_{h\to 0^+} y(t, h) = y(t)$ for all $t_0 \leq t \leq t_1$) if and only if the following two conditions hold:*

- *All roots of p lie in the set $\{z \in \mathbb{C} \colon |z| \leq 1\}$, and any root on the circle $\{z \in \mathbb{C} \colon |z| = 1\}$ has multiplicity one. (Stability)*
- *$p(1) = 0$ and $p'(1) = q(1)$. (Consistency)*

Proof. We only show the forward implication. We begin with the first condition. Consider the initial value problem $y'(t) = 0$ for all $t \geq 0$ with $y(0) = 0$ (so that $t_0 = 0$). Since $f = 0$, the multistep method simplifies to the recursion

$$a_k y_n + a_{k-1} y_{n-1} + \cdots + a_0 y_{n-k} = 0, \forall\, n \geq k.$$

Evidently, one solution of this recursion is $y_n := h\lambda^n$ for all $n \geq k$, where λ is a root of p. For all $0 \leq j < k$, $y_j := h\lambda^j \to 0 = y(t_0)$ as $h \to 0$. So, the choice $y(jh, h) := h\lambda^j$ for all $0 \leq j < k$ is an initial condition that satisfies the hypothesis of our theorem. So, if $|\lambda| > 1$, then in perfect arithmetic, with our initial condition as stated, the computed output of the algorithm at time $t = nh$ will be

$$|y(nh, h)| = |y_n| = h\,|\lambda|^n = h\,|\lambda|^{t/h} \to \infty, \quad \text{as } h \to 0^+.$$

(This limit is equal to $\lim_{n\to\infty} \frac{t}{n}\,|\lambda|^n = \infty$.) That is, if $|\lambda| > 1$, the convergence of the multistep method cannot occur, i.e. $\lim_{h\to 0^+} y(t, h) \neq y(t)$ for all $t > 0$.

Similarly, if $|\lambda| = 1$ and $p'(\lambda) = 0$ (so that λ has multiplicity at least two), then a solution of the multistep recursion is $y_n := hn\lambda^n$ for all $n \geq k$. The choice $y(jh, h) := hj\lambda^j$ for all $0 \leq j < k$ is an initial condition that satisfies the hypothesis of our theorem. And in perfect arithmetic, with our initial condition as stated, the computed output of the algorithm at time $t = nh \neq 0$ will be

$$|y(nh, h)| = |y_n| = hn\,|\lambda|^n = t \neq 0.$$

As $h \to 0^+$, convergence cannot occur, i.e. $\lim_{h\to 0^+} y(t, h) \neq y(t)$ for all $t > 0$.

We now consider the second stated condition. Consider the initial value problem $y'(t) = 0$ for all $t \geq 0$ with $y(0) = 1$ (so that $t_0 = 0$). Since $f = 0$, the multistep method simplifies to the recursion

$$a_k y_n + a_{k-1} y_{n-1} + \cdots + a_0 y_{n-k} = 0, \forall\, n \geq k.$$

A solution of this recursion is then $y_0 = \cdots = y_{k-1} = 1$ and then $y_k, y_{k+1}, \ldots$ are found by applying the recursion. If the multistep method converges, then e.g. y_k converges to 1 as $h \to 0^+$, so that

$$a_k + \cdots + a_0 = 0.$$

That is, $p(1) = 0$.

To see that $p'(1) = q(1)$, consider the initial value problem $y'(t) = 1$ for all $t \geq 0$ with $y(0) = 0$ (so that $t_0 = 0$). Since $f = 1$, the multistep method simplifies to the recursion

$$a_k y_n + a_{k-1} y_{n-1} + \cdots + a_0 y_{n-k} = h\Big(b_k + \cdots + b_0\Big), \forall\, n \geq k.$$

Since $p(1) = 0$, the first item of this Theorem implies that $p'(1) \neq 0$. A solution of the recursion is then $y_n = (n+k)h \cdot q(1)/p'(1)$ for all $n \geq k$, since

$$\begin{aligned}
&a_k y_n + a_{k-1} y_{n-1} + \cdots + a_0 y_{n-k} \\
&\quad = h\frac{q(1)}{p'(1)}\Big(a_k(n+k) + a_{k-1}(n+k-1) + \cdots + a_0 n\Big) \\
&\quad = nh\frac{q(1)}{p'(1)}\Big(a_k + \cdots + a_0\Big) + h\frac{q(1)}{p'(1)}\Big(a_k k + a_{k-1}(k-1) + \cdots + a_1\Big) \\
&\quad = nh\frac{q(1)}{p'(1)}p(1) + h\frac{q(1)}{p'(1)}p'(1) = 0 + hq(1) = h\Big(b_k + \cdots + b_0\Big).
\end{aligned}$$

So, the choice $y(jh, h) := (j+k)h \cdot q(1)/p'(1)$ for all $0 \leq j < k$ is an initial condition that satisfies the hypothesis of our theorem. So, in perfect arithmetic, with our initial condition as stated, the computed output of the algorithm at time $t = nh$ will be

$$y(nh, h) = y_n = (n+k)h\frac{q(1)}{p'(1)} = (t + kh)\frac{q(1)}{p'(1)} \to t\frac{q(1)}{p'(1)}, \quad \text{as } h \to 0^+.$$

Since we assume convergence occurs, $\lim_{h\to 0^+} y(t, h) = y(t) = t$ for all $t > 0$. (Recall the solution of the initial value problem is $y(t) = t$.) We therefore conclude that $q(1)/p'(1) = 1$, so that $q(1) = p'(1)$. □

Exercise 7.20. Fix $\lambda > 0$. Suppose $y\colon [0, \infty) \to \mathbb{R}$ satisfies $y(0) := 10$, and y satisfies the following ODE:

$$y'(t) = f(y(t)) := -\lambda y(t), \qquad \forall\, 0 \leq t \leq 100.$$

Note that an exact solution is $y(t) := 10e^{-\lambda t}$. In this exercise, we will try out different iterative methods for solving this ODE.

Consider setting $\lambda = 20, 200$ or 2000 and set the step size h to be $1, .1, .01$ and $.001$ in the following solution methods.

(a) $y_{n+1} = y_n + hf(y_n)$ (Euler's)
(b) $y_{n+1} = y_n + hf(y_{n+1})$ (backwards Euler's) (You should solve for y_{n+1}.)
(c) $y_{n+1} = y_n + hf(y_n + hf(y_n))$ (predictor-corrector Euler's)
(d) $y_{n+1} = y_n + hf(y_n + \frac{h}{2}f(y_n))$ (modified Euler's)
(e) $y_{n+1} = y_{n-1} + 2hf(y_n)$ (Nystrom's midpoint)

For each iterative method and for each value of λ and h, report the computed value of $y(100)$, and compare this value to the actual value $10e^{-100\lambda}$ (which is basically zero). Describe which methods perform the best, and which methods perform the worst. How do the results compare to theoretical error bounds? For example, for a multistep method (which is the case for (a) and (e)), is the method stable and consistent?

Exercise 7.21. Find the values of h where the recursions satisfy stability for the numerical methods (c), (d) and (e) from Exercise 7.20, and also for

(f) $y_{n+1} = y_n + \frac{h}{2}[f(y_n) + f(y_{n+1})]$ (Trapezoid rule)

Exercise 7.22. We investigate solutions of the differential equation

$$y'(t) = -1000(y(t) - \sin(t)) + \cos(t), \quad \forall\, t > 0, \quad y(0) = 1.$$

Using Matlab's equation solver find the exact solution. For example, the syntax for solving $y'(t) = ty(t)$, $y(0) = 2$ would be

```
syms y(t)
dsolve(diff(y(t),t)==t*y(t) , y(0)==2)
```

Compare the performance of the textbook function `ode23tx` to Matlab's stiff solver `ode23s`. How does each method perform?

Denote $g(t) := \sin(t)$, $\lambda := 1000$, and consider the following rewritten version of the differential equation (without the initial condition):

$$y'(t) = -\lambda(y(t) - g(t)) + g'(t)$$

If we introduce the variable $z(t) = y(t) - g(t)$ we then have

$$\lambda z(t) = z'(t)$$

and if we look at our exact solution when $y(0) = 1$, we see that z kind of measures the distance between some general solution and the "attracting" solution $y(t) = \sin(t)$. Therefore, since we have $\lambda = 1000$ in this case, we see that this distance to the attracting solution changes very rapidly. This is exactly why our problem is stiff. Solutions move towards $\sin(t)$ at roughly 1000 times their distance from it. This is much faster than the "stable" solution $y(t) = \sin(t)$ varies (all its derivatives are bounded by 1). Therefore, we have an analytical explanation of exactly why the problem is stiff.

Exercise 7.23. Consider the following system of equations

$$y_1'(t) = -1000y_1(t) + y_2(t), \qquad y_2'(t) = 999y_1(t) - 2y_2(t).$$
$$y_1(0) = 1, \qquad y_2(0) = 0.$$

(a) Using Matlab, solve this equation exactly. Describe the behavior of y_1, y_2 and y_1/y_2 as $t \to \infty$.
(b) Solve the system using Euler's and backwards Euler's methods up to the time $T = 10$. In both cases find (approximately) the largest h which produces an absolute error E_T at time T where $E_T < 10^{-5}$ (this absolute error is the maximum of the errors of the two variables $y_1(T)$ and $y_2(T)$).
(c) Repeat part (b) for Euler's method, changing your error tolerance to $E_T < 10^{-6}$.
(d) Plot $\log(E_T)$ as a function of n, for Euler's method. Try to explain the origin of any interesting behavior of this plot.

Exercise 7.24. This exercise investigates the double pendulum. We have m_1, m_2 as the masses of our two bobs, θ_1, θ_2 the angle each rod makes with the $y - axis$ (0 indicates the rod is hanging straight down, an angle with a small and positive value indicates the rod is pointing into the lower right quadrant, etc.) and ℓ_1, ℓ_2 are the lengths of the rods. The positions of the bobs (x_1, y_1) and (x_2, y_2), where

$$x_1 = \ell_1 \sin(\theta_1), \qquad y_1 = -\ell_1 \cos(\theta_1)$$
$$x_2 = \ell_1 \sin(\theta_1) + \ell_2 \sin(\theta_2), \qquad y_2 = -\ell_1 \cos(\theta_1) - \ell_2 \cos(\theta_2)$$

and physics leads to a pair of second-order, nonlinear ODEs that describe the motion of the pendulum. Perform your investigation with the `swinger` program in the NCM package

(a) When the initial angle of the double pendulum is small, note that it acts like a normal pendulum. How large can you take the initial angle while maintaining this behavior?
(b) The initial orbit of the `swinger` function is interesting because it is periodic. It keeps repeating the same thing over and over, rather than being chaotic like most all other initial conditions. Try to observe some other initial conditions that lead to periodic behavior. (To do this, click on different points on the plot.)
(c) If you run `swinger` for a while, click `stop` and then type in the command line `get(gcf,'userdata')` the initial angles θ_1, θ_2 (the last ones that were chosen) are returned.
(d) Now, attempt to change the way the initial conditions of the system are chosen, by specifying the initial angles θ_1 and θ_2, instead of specifying the initial values of x_2, y_2. In the case where θ_1 and θ_2 differ, one sees by drawing a parallelogram that if the position of x_2 is specified, then the initial value of x_1 has two possible values. From this parallelogram, we see that the angles θ_1 and θ_2 are simply swapped between each case, corresponding to the two choices of x_1. Therefore, in the Matlab code that you write, you can choose either of these two orderings of the angles θ_1, θ_2.
(e) Now, change the function to handle different masses. That is, extend the main function definition to accept m_1,m_2 as the respective weights for the masses of the bobs.
(f) Modify the `swinger` function so that lengths other than $\ell_1 = \ell_2 = 1$ are possible
(g) Modify the function so that we can change g, the acceleration due to gravity.
(h) Alter the program so that it solves its differential equation use the textbook function `odetx`.
(i) Is the ODE solved by `swinger` stiff? (The book defines stiffness as a "wide disparity in the time scales of the components of the vector solution.")

Exercise 7.25. Complete NCM Problem 7.21, from here. This exercise investigates an atmospheric simulation.

7.4. **Boundary-Value Problems.** Boundary value problems govern the flows of fluids, heat, electricity and magnetism. Those applications require multivariable calculus, so we will not discuss them much here.

Definition 7.26 (**Ordinary Differential Equation, Boundary Value Problem, One Variable**)**.** Let $a, b \in \mathbb{R}$ with $a < b$. Let $f\colon [a,b] \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Let $\alpha, \beta \in \mathbb{R}$. A **boundary value problem** is the following example of an ordinary differential equation.

$$\begin{cases} y''(t) &= f(t, y(t), y'(t)), \qquad \forall\, t \in [a,b] \\ y(a) &= \alpha \\ y(b) &= \beta. \end{cases}$$

The first equality can be written as a single derivative condition applied to a vector:

$$\frac{d}{dt}\begin{pmatrix} y'(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} f(t, y(t), y'(t)) \\ y'(t) \end{pmatrix}, \qquad \forall\, t \in [a,b].$$

However, we cannot rewrite Definition 7.26 as an initial value problem as in Definition 7.1, since doing so would require specifying some initial values of e.g.

$$\begin{pmatrix} y'(a) \\ y(a) \end{pmatrix}.$$

In fact, our first method for solving a boundary value problem will fix these initial values, and temporarily ignore the constraint on $y(b)$.

7.5. **Shooting Methods.** A **shooting method** for the boundary value problem 7.26 solves a sequence of initial value problems for any $\eta \in \mathbb{R}$:

$$\begin{cases} \frac{d}{dt}\begin{pmatrix} y'(t) \\ y(t) \end{pmatrix} &= \begin{pmatrix} f(t, y(t), y'(t)) \\ y'(t) \end{pmatrix}, \qquad \forall\, t \in [a,b] \\ \begin{pmatrix} y'(a) \\ y(a) \end{pmatrix} &= \begin{pmatrix} \eta \\ \alpha \end{pmatrix} \end{cases} \qquad (*)$$

Define then $g\colon \mathbb{R} \to \mathbb{R}$ so that $g(\eta) := y(b) - \beta$ (note that y depends on η), and find a zero of the function g. If $g(\eta) = 0$, then the function y satisfying $(*)$ also solves the boundary value problem 7.26.

7.6. **Finite Difference Methods.**

Exercise 7.27. In this exercise, we will solve the following boundary value problem in three different ways. Let $y\colon [0,1] \to \mathbb{R}$ satisfy

$$y''(t) = (y(t))^2 - 1, \quad \forall\, t \in [0,1], \qquad y(0) = 0, \quad y(1) = 1.$$

(a) First solve this problem using the shooting method. That is, ignore temporarily the condition $y(1) = 1$, and instead impose the initial value conditions

$$y'(0) = \eta, \qquad y(0) = 0.$$

Denote the solution y which depends on t and η as $y(t, \eta)$. Then, create a Matlab function f, defined to be

$$f(\eta) := y(1, \eta) - 1$$

and look for the zero of this function. If you find an η such that $f(\eta) = 0$, then the solution y satisfies the original boundary value problem.

(b) Now observe that we can re-write the differential equation as

$$\frac{d}{dt}\left(\frac{(y')^2}{2} - \frac{y^3}{3} + y\right) = 0$$

and if we assume y is continuously differentiable, then this means

$$\frac{(y')^2}{2} - \frac{y^3}{3} + y = K \qquad (*)$$

is a constant function K. Since $y(0) = 0$, we can solve $(*)$ for $y'(0)$ to get $y'(0) = \pm\sqrt{2K}$ (but trusting the validity of our previous result, we assume that $y'(0) =$

$+\sqrt{2K}$). Solving $(*)$ for $\frac{1}{y'(t)}$ and recalling that $(d/dt)y^{-1}(t) = 1/y'(y^{-1}(t))$, we can integrate $1/y'(t)$ to obtain the inverse function of y, denoted as $t(y)$:

$$t(y) = \int_{s=0}^{s=y} \frac{1}{\sqrt{2(K + \frac{s^3}{3} - s)}} ds. \qquad (**)$$

and since we want to impose the condition $y(1) = 1$ we want equivalently that $t(1) = 1$ so that we must find the zero of the following equation

$$g(K) = \left(\int_{s=0}^{s=1} \frac{1}{\sqrt{2(K + \frac{s^3}{3} - s)}} ds \right) - 1$$

and this will solve our problem. That is, finding such a K will find the inverse function of y via $(**)$, so that y is then obtained from $(**)$, since y is the inverse of $t(y)$.

(c) Now try a finite difference method, choosing $n+1$ equal subintervals of length $h = \frac{1}{n+1}$ which turns the equation $y'' = y^2 - 1$ into a system involving n unknowns

$$y_{i+1} - 2y_i + y_{i-1} = h^2(y_i^2 - 1), \quad \forall\, i = 1, \ldots, n$$

where $y_0 = 0$ and $y_{n+1} = 1$. In matrix form this is

$$Ay + b = h^2(y^2 - 1)$$

where A has -2's on the diagonal and $1's$ on the superdiagonal and subdiagonal and zeros elsewhere, $b_n = 1$ and $b_i = 0$ for $i < n$, and y^2 denotes y with each of its entries squared. In this particular instance, we write

$$Ay = h^2(y^2 - 1) - b \qquad (‡)$$

choosing an initial guess for y, and repeatedly solve the linear system, improving our guess more each time until we are sufficiently close to the solution. That is, if y is fixed on the right side of $(‡)$, then we solve for x the linear system $Ax = h^2(y^2-1)-b$. We then solve the linear system $Az = h^2(x^2 - 1) - b$ for z, and so on.

Exercise 7.28. Now consider the boundary value problem

$$y''(t) = \frac{-t(y'(t) + \pi \sin(\pi t))}{d} - \pi^2 \cos(\pi t), \quad \forall\, t \in [-1, 1], \qquad y(-1) = -2, \quad y(1) = 0. \qquad (*)$$

Here $d > 0$ is a fixed parameter.

(a) First try solving this problem with the shooting method for fairly small d (e.g. try $d = .1$, $d = .01$ and $d = .001$). What is the smallest value of d for which the shooting method is able to find a solution y with $|y(1)| < 10^{-3}$? Denote this value of d as $\widehat{d}$.

(b) Now try to manipulate your initial guesses to improve the performance of the `fzero` function used in the shooting method. If f denotes the function used in our definition of the shooting method, we want to have $|f(1)| < 10^{-2}$. With $d > 0$ fixed, let $\eta(d)$ denote the value of η that `fzero` returns when trying to solve $f(\eta) = 0$. Let $d_k := \widehat{d} \cdot (.8)^k$ for any $k \geq 0$. Then, try to use $\eta(d_k)$ as an initial guess in `fzero` when you apply the shooting method in the case $d = d_{k+1}$. Does this method work well for values of d smaller than $\widehat{d}$?

(c) This ODE is linear in the following sense. If y and $\widetilde{y}$ are solutions of the differential equation $(*)$ with only the initial value specified (i.e. where $y(1)$ and $\widetilde{y}(1)$ are not fixed), then $ay + (1-a)\widetilde{y}$ also satisfies $(*)$ for any $a \in \mathbb{R}$. So, if we use two different shootings y and $\widetilde{y}$ with differential initial derivatives $y'(-1)$ and $\widetilde{y}'(-1)$, why can we not just make a linear combination of them fit any of our desired solutions? That is, why can we not just pick a particular a such that $z := ay+(1-a)\widetilde{y}$ satisfies $z(1) = 0$? This is exactly what the book suggests we do. Can we do this on the computer to solve $(*)$ when d is small (e.g. $d = .001$)?

(d) Since the ODE is linear, try to use a single iteration finite difference method. Do this first with evenly spaced nodes, and then estimate the L^∞ error for the $n-$node solution by comparing it with the $2n$-node solution. For select d values, plot the minimal solution on $n = 2^j$ nodes where the estimated error E_n is less than 10^{-2}.

(e) Try using unevenly spaced nodes in a finite difference method. In particular, try to use more nodes near $t = 0$. Do the unevenly spaced nodes perform better than the evenly spaced case?

7.7. **Collocation.**

Exercise 7.29. Consider the following two-point boundary value problem

$$\begin{cases} u''(t) + p(t)u'(t) + q(t)u(t) = w(t), & \forall\, t \in [a,b] \\ u(a) = \alpha \qquad u(b) = \beta \end{cases}$$

which we solve by the method of collocation. Use $n-2$ collocation points labelled t_i for $i = 1, \ldots, n-2$ equally spaced on $[a,b]$. Obtain a linear system of n equations with n unknowns $c_1, \ldots, c_n$

$$\begin{cases} \displaystyle\sum_{j=1}^{n} c_j \left(Lv_j\right)(t_i) = w(t_i) \qquad (1 \le i \le n-2) \\ \displaystyle\sum_{j=1}^{n} c_j v_j(a) = \alpha \qquad \sum_{j=1}^{n} c_j v_j(b) = \beta \end{cases}$$

Where

$$L(v_j)(t) = v_j''(t) + p(t)v_j'(t) + q(t)v_j(t)$$

and

$$v_j(t) = B\left(\frac{t-a}{h} - j + 2\right)$$

and B is the standard cubic $B-$*spline*.

After obtaining the solution $u = \sum_{j=1}^{n} c_j v_j$, then also examine the residual, $Lu - w$ on our original knots, and also the midpoints of the intervals $[t_i, t_{i+1}]$, $i = 1, \ldots, n-3$. To test our program, we consider the system

$$\begin{cases} u''(t) + \sin(t)u'(t) + (t^2+2)u(t) = e^{t-3} \\ u(2.6) = 7 \qquad u(5.1) = -3 \end{cases}$$

8. Appendix: Notation

Let n, m be a positive integers. Let A, B be sets contained in a universal set Ω.

$\mathbb{N} = \{1, 2, \ldots\}$ denotes the set of natural numbers

$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ denotes the set of integers

$\mathbb{Q} = \{a/b \colon a, b, \in \mathbb{Z}, b \neq 0\}$ denotes the set of rational numbers

$\mathbb{R}$ denotes the set of real numbers

$\mathbb{C} = \{a + b\sqrt{-1} \colon a, b \in \mathbb{R}\}$ denotes the set of complex numbers

$\in$ means "is an element of." For example, $2 \in \mathbb{R}$ is read as "2 is an element of $\mathbb{R}$."

$\forall$ means "for all"

$\exists$ means "there exists"

$\mathbb{R}^n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in \mathbb{R} \, \forall \, 1 \leq i \leq n\}$

$f \colon A \to B$ means f is a function with domain A and range B. For example,

$f \colon \mathbb{R}^2 \to \mathbb{R}$ means that f is a function with domain $\mathbb{R}^2$ and range $\mathbb{R}$

$\emptyset$ denotes the empty set

$A \subseteq B$ means $\forall \, a \in A$, we have $a \in B$, so A is contained in B

$A \setminus B := \{a \in A \colon a \notin B\}$

$A^c := \Omega \setminus A$, the complement of A in Ω

$A \cap B$ denotes the intersection of A and B

$A \cup B$ denotes the union of A and B

$A \Delta B := (A \setminus B) \cup (B \setminus A)$

$\mathbf{P}$ denotes a probability law on Ω

Let $n \geq m \geq 0$ be integers. We define

$$\binom{n}{m} := \frac{n!}{(n-m)!m!} = \frac{n(n-1)\cdots(n-m+1)}{m(m-1)\cdots(2)(1)}.$$

Let $a_1, \ldots, a_n$ be real numbers. Let n be a positive integer.

$$\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_{n-1} + a_n.$$

$$\prod_{i=1}^{n} a_i = a_1 \cdot a_2 \cdots a_{n-1} \cdot a_n.$$

$\min(a_1, a_2)$ denotes the minimum of a_1 and a_2.

$\max(a_1, a_2)$ denotes the maximum of a_1 and a_2.

The min of a set of nonnegative real numbers is the smallest element of that set. We also define $\min(\emptyset) := \infty$.

Let $A \subseteq \mathbb{R}$.

$\sup A$ denotes the supremum of A, i.e. the least upper bound of A.

$\inf A$ denotes the infimum of A, i.e. the greatest lower bound of A.

$1_A \colon \Omega \to \{0, 1\}$, denotes the indicator function of A, so that

$$1_A(\omega) = \begin{cases} 1 & \text{, if } \omega \in A \\ 0 & \text{, otherwise.} \end{cases}$$

Let $g, h \colon \mathbb{R} \to \mathbb{R}$. Let $t \in \mathbb{R}$.

$(g * h)(t) = \int_{-\infty}^{\infty} g(x)h(t-x)dx$ denotes the convolution of g and h at $t \in \mathbb{R}$

References

[Aar11] Scott Aaronson, *A linear-optical proof that the permanent is #p-hard*, Electronic Colloquium on Computational Complexity (ECCC) **18** (2011), 43.

[Gal14] François Le Gall, *Powers of tensors and fast matrix multiplication*, Preprint, arXiv:1401.7714. ISAAC 2014., 2014.

[GVL13] G.H. Golub and C.F. Van Loan, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, 2013.

[JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda, *A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries*, J. ACM **51** (2004), no. 4, 671–697.

USC Mathematics, Los Angeles, CA

E-mail address: stevenmheilman@gmail.com