Numerical Methods
An Inquiry-Based Approach

Eric Sullivan
Department of Mathematics
Carroll College, Helena, MT
esullivan@carroll.edu

Content Last Updated: May 20, 2019

Contents

Chapter 0

To the Student and the Instructor

> "Any creative endeavor is built in the ash heap of failure."
> –Michael Starbird

This document contains lecture notes, classroom activities, code, examples, and challenge problems specifically designed for an introductory semester of numerical analysis. The content herein is written and maintained by Dr. Eric Sullivan of Carroll College. Problems were either created by Eric Sullivan, the Carroll Mathematics Department faculty, part of NSF Project Mathquest, or come from other sources and are either cited directly or cited in the LaTeX source code for the document.

0.1 An Inquiry Based Approach

Problem 0.1 (Setting The Stage)**.** Let's start the book off right away with a problem designed for groups, discussion, disagreement, and deep critical thinking. This problem is inspired by Dana Ernst's first day IBL activity titled: Setting the Stage.

- Get in groups of size 3-4.
- Group members should introduce themselves.
- For each of the questions that follow I will ask you to:
  1. **Think** about a possible answer on your own
  2. **Discuss** your answers with the rest of the group
  3. **Share** a summary of each group's discussion

Questions:

Question #1: What are the goals of a university education?

Question #2: How does a person learn something new?

Question #3: What do you reasonably expect to remember from your courses in 20 years?

Question #4: What is the value of making mistakes in the learning process?

Question #5: How do we create a safe environment where risk taking is encouraged and productive failure is valued?

▲

This material is written with an Inquiry-Based Learning (IBL) flavor. In that sense, this document could be used as a stand-alone set of materials for the course but these notes are not a *traditional textbook* containing all of the expected theorems, proofs, code, examples, and exposition. The students are encouraged to work through problems and homework, present their findings, and work together when appropriate. You will find that this document contains collections of problems with only minimal interweaving exposition. It is expected that you do every one of the problems and then use other more traditional texts as a backup when you are stuck. Let me say that again: this is not the only set of material for the course. Your brain, your peers, and the books linked in the next section are your best resources when you are stuck.

To learn more about IBL go to http://www.inquirybasedlearning.org/about/. The long and short of it is that the students in the class are the ones that are doing the work; proving theorems, writing code, working problems, leading discussions, and pushing the pace. The instructor acts as a guide who only steps in to redirect conversations or to provide necessary insight. If you are a student using this material you have the following jobs:

1. **Fight!** You will have to fight hard to work through this material. The fight is exactly what we're after since it is ultimately what leads to innovative thinking.

2. **Screw Up!** More accurately, don't be afraid to screw up. You should write code, work problems, and prove theorems then be completely unafraid to scrap what you've done and redo it from scratch. Learning this material is most definitely a non-linear path.

3. **Collaborate!** You should collaborate with your peers with the following caveats:

   (a) When you are done collaborating you should go your separate ways. When you write your solution you should have no written (or digital) record of your collaboration.

   (b) The internet is not a collaborator. Use of the internet to help solve these problems robs you of the most important part of this class; the chance for original thought.

4. **Enjoy!** Part of the fun of IBL is that you get to experience what it is like to think like a true mathematician / scientist. It takes hard work but ultimately this should be fun!

0.2 Online Texts and Other Resources

If you are looking for online textbooks for numerical methods or numerical analysis I can point you to a few of my favorites. Some of the following online resources may be a good place to help you when you're stuck but they will definitely say things a bit differently. Use these resources wisely.

- Holistic Numerical Methods http://nm.mathforcollege.com/
  The Holistic Numerical Methods book is probably the most complete free reference that I've found on the web. This should be your source to look up deeper explanations of problems, algorithms, and code.
- Scientific Computing with MATLAB http://gribblelab.org/scicomp/scicomp.pdf
- Tea Time Numerical Analysis http://lqbrin.github.io/tea-time-numerical/

0.3 To the Instructor

If you are an instructor wishing to use these materials then I only ask that you adhere to the Creative Commons license. You are welcome to use, distribute, and remix these materials for your own purposes. Thanks for considering my materials for your course! Let me know if you have questions, edits, or suggestions: esullivan@carroll.edu.

0.4 Special Thanks

I would first like to thank Dr. Kelly Cline for being brave enough to teach a course that he loves out of a rough draft of my notes. Your time, suggested edits, and thoughts for future directions of the book were, and are, greatly appreciated. You've taught me a lot in the short time that we've worked together. Thanks! Second, I would like to thank my wife, Johnanna, for simply being awesome. I would like to thank the institution of Carroll College for seeing this project as a worthy academic pursuit even though the end result is not a book or publication in the traditional sense. Finally, I would like to thank all of my colleagues and students, both past and present, in the math department at Carroll. The suggestions, questions, struggles, and triumphs of these folks are what have shaped this work into something that I'm proud of and that I hope will be a useful resource for future students and instuctors.

Chapter 1

Introductory Topics

> "In the 1950s and 1960s, the founding fathers of [Numerical Analysis] discovered that inexact arithmetic can be a source of danger, causing errors in results that 'ought' to be right."
>
> Lloyd N. Trefethen (2006)

The field of Numerical Analysis is really the study of how to take mathematical problems and perform them efficiently and accurately on a computer. There are some problems where numerical analysis doesn't make much sense (e.g. finding an algebraic derivative of a function) but for many problems a numerical method that gives an approximate answer is both more efficient and more versatile than any analytic technique. For example, if we needed to solve the differential equation $\frac{dy}{dt} = \sin(y^2) + t$ the nonlinear nature of the problem makes it hard to work with analytically but computational methods that result in a plot of an approximate solution can be made very quickly and likely give enough of a solution to be usable. Similarly, efficient algorithms to do the basic operations of linear algebra (e.g. Gaussian elimination, matrix factorization, or eigenvalue decomposition) are highly sought after and useful when the matrices used for a model are very large.

In this chapter we will discuss some of the basic underlying ideas behind the scenes in Numerical Analysis, and the essence of the above quote will be part of the focus of this chapter. Particularly, we need to know how a computer stores numbers and when that storage can get us into trouble. On a more mathematical side, we offer a brief review of the Taylor Series in this chapter. The Taylor Series underpins many of our approximation methods in this class; so much so that we could easily rename the course: *Applied Taylor Series*. Finally, at the end of this chapter we provide several coding exercises that will help you to develop your programming skills. It is expected that you know some of the basics of MATLAB programming before beginning this class. If you need to review the basics then see the Scientific Computing with MATLAB text cited in the bibliography of these notes [8]. Trust me, you'll have more than just the basics by the end.

Let's begin.

1.1 Base 2 and Binary Arithmetic

Problem 1.1. By hand (no computers!) compute the first 50 terms of this sequence with the initial condition $x_0 = 1/10$.

$$x_{n+1} = \begin{cases} 2x_n, & x_n \in [0, \frac{1}{2}] \\ 2x_n - 1, & x_n \in (\frac{1}{2}, 1] \end{cases}$$

▲

Problem 1.2. Now use Excel and MATLAB to do the computations. Do you get the same answers? ▲

Problem 1.3. It seems like the computer has failed you! What do you think happened on the computer and why did it give you a different answer? What, do you suppose, is the cautionary tale hiding behind the scenes with this problem? ▲

Problem 1.4. Now what happens with this problem when you start with $x_0 = 1/8$? Why does this new initial condition work better? ▲

A computer stores numbers using base 2, called a binary number system. Let's first discuss our more familiar base 10 system. What do the digits in the number "135" really mean? A moment's reflection likely reveals that

$$135 = 100 + 30 + 5 = 1 \times 10^2 + 3 \times 10^1 + 5 \times 10^0.$$

We use what is commonly called a *positional number system*. In such a system, the position of the number indicates the power of the base, and the value of the digit itself tells you the multiplier of that power. The number "48329" can therefore be interpreted as

$$48329 = 40000 + 8000 + 300 + 20 + 9 = 4 \times 10^4 + 8 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 9 \times 10^0.$$

Now let's switch to the number system used by computers: the binary number system. In a binary number system the base is 2 so the only allowable digits are 0 and 1. In binary (base-2), the number "101101" can be interpreted as

$$101101_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

(where the subscript "2" indicates the based to the reader). If we put this back into base 10 (so that we can read it more comfortably) we get

$$101101_2 = 32 + 0 + 8 + 4 + 0 + 1 = 45_{10}.$$

Problem 1.5. Convert to following numbers from base 10 to base 2 or visa versa.

- Write 12_{10} in binary
- Write 11_{10} in binary
- Write 23_{10} in binary

- Write 11_2 in base 10
- What is 100101_2 in base 10?

▲

Problem 1.6. Converting base 2 numbers to base 10 is relatively easy – expand in powers of 2 and add up the results. Converting base 10 numbers to base 2 is a little bit more interesting. Describe the technique that you used to convert the numbers 12, 11, and 23 into base 2 in the previous problem. ▲

Example 1.7. Convert the number 137 from base 10 to base 2.
Solution:

$$\begin{aligned} 137_{10} &= 128 + 8 + 1 \\ &= 2^7 + 2^3 + 2^0 \\ &= 1\times 2^7 + 0\times 2^6 + 0\times 2^5 + 0\times 2^4 + 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 1\times 2^0 \\ &= 10001001_2 \end{aligned}$$

Next we'll work with fractions and decimals. For example, let's take the base 10 number 5.341_{10} and expand it out to get

$$5.341_{10} = 5 + \frac{3}{10} + \frac{4}{100} + \frac{1}{1000} = 5\times 10^0 + 3\times 10^{-1} + 4\times 10^{-2} + 1\times 10^{-3}.$$

We can do a similar thing with binary decimals.

Example 1.8. Convert 11.01011_2 to base 10.
Solution:

$$\begin{aligned} 11.01011_2 &= 2 + 1 + \frac{0}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} + \frac{1}{32} \\ &= 1\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4} + 1\times 2^{-5} \\ &= 3.34375_{10}. \end{aligned}$$

Problem 1.9. Convert the following numbers from base 10 to binary.

(a) What is 1/2 in binary?

(b) What is 1/8 in binary?

(c) What is 4.125 in binary?

(d) What is 0.15625 in binary?

▲

Problem 1.10. Convert the base 10 decimal 0.635 to binary using the following steps. Explain why each step gives the binary digit that it does.

(a) Multiply 0.635 by 2. The whole number part of the result is the first binary digit to the right of the decimal point.

(b) Take the result of the previous multiplication and ignore the digit to the left of the decimal point. Multiply the remaining decimal by 2. The whole number part is the second binary decimal digit.

(c) Repeat the previous step until you have nothing left, until a repeating pattern has revealed itself, or until your precision is *close enough*.

▲

Problem 1.11. Based on your previous problem write an algorithm that will convert base-10 decimals (less than 1) to base decimal expansions. ▲

Problem 1.12. Use the "`=if( )`" command in Excel to create a spreadsheet that uses the process from the previous problem to turn a base 10 decimal (less than 1) into a binary decimal. ▲

Problem 1.13. Use a `for loop` and `if-else` statements in MATLAB to write a script that converts a base 10 decimal (less than 1) into a binary decimal. ▲

Problem 1.14. Convert the base 10 fraction 1/10 into binary. Use your solution to fully describe what went wrong in problems 1.1 - 1.3? ▲

1.2 Floating Point Arithmetic

Everything stored in the memory of a computer is a number, but how does a computer actually store a number. More specifically, since computers only have finite memory we would really like to know the full range of numbers that are possible to store in a computer.

Problem 1.15. Consider the number $x = -129.15625$ (in base 10). As we've seen this number can be converted into binary. Indeed

$$x = -123.15625_{10} = -1111011.00101_2$$

(you should check this).

(a) If a computer needs to store this number then first they put in the binary version of scientific notation. In this case we write

$$x = -1.\_\_\_\_\_ \times 2\text{——}$$

(b) Based on the fact that every binary number (other than 0) can be written in this way, what three things do you suppose a computer needs to store for any given number?

(c) Using your answer to part (b), what would a computer need to store for the binary number $x = 10001001.1100110011_2$?

▲

For any number x we can write

$$x = (-1)^s \times (1+m) \times 2^E$$

where $s \in \{0,1\}$ is called the *sign bit* and m is a binary number such that $0 \leq m < 1$.

Definition 1.16. For a number $x = (-1)^s \times (1+m) \times 2^E$ stored in a computer, the number m is called the **mantissa** or the **significand**, s is known as the sign bit, and E is known as the exponent.

Example 1.17. What are the mantissa, sign bit, and exponent for the numbers 7_{10}, -7_{10}, and $(0.1)_{10}$?
Solution:

- For the number $7_{10} = 111_2 = 1.11 \times 2^2$ we have $s = 0$, $m = 0.11$ and $E = 2$.
- For the number $-7_{10} = 111_2 = -1.11 \times 2^2$ we have $s = 1$, $m = 0.11$ and $E = 2$.
- For the number $\frac{1}{10} = 0.000110011001100\cdots = 1.100110011 \times 2^{-4}$ we have $s = 0$, $m = 0.100110011\cdots$, and $E = -4$.

In the last part of the previous example we saw that the number $(0.1)_{10}$ is actually a repeating decimal in base-2. This means that in order to completely represent the number $(0.1)_{10}$ in base-2 we need infinitely many decimal places. Obviously that can't happen since we are dealing with computers with finite memory. Over the course of the past several decades there have been many systems developed to properly store numbers. The IEEE standard that we now use is the accumulated effort of many computer scientists, much trial and error, and deep scientific research. We now have three standard precisions for storing numbers on a computer: single, double, and extended precision. The double precision standard is what most of our modern computers use.

Definition 1.18 (Computer Precision Standards)**.** There are three standard precisions for storing numbers in a computer.

- A **single-precision** number consists of 32 bits, with 1 bit for the sign, 8 for the exponent, and 23 for the significand.
- A **double-precision** number consists of 64 bits with 1 bit for the sign, 11 for the exponent, and 52 for the significand.
- An **extended-precision** number consists of 80 bits, with 1 bit for the sign, 15 for the exponent, and 64 for the significand.

Definition 1.19. Machine precision is the gap between the number 1 and the next larger floating point number. Often it is represented by ϵ. To clarify, the number 1 can always be stored in a computer system exactly and if ϵ is machine precision for that computer then $1 + \epsilon$ is the next largest number that can be stored with that machine.

For all practical purposes the computer cannot tell the difference between two numbers if the difference is smaller than machine precision. This is of the utmost important when you want to check that something is "zero" since a computer just cannot know the difference between 0 and ϵ. As a side note: You can determine the working precision in MATLAB by typing "eps" in the command line.

Problem 1.20. To make all of these ideas concrete let's play with a small computer system where each number is stored in the following format:

s	E	$b_1 b_2 b_3$

The first entry is a bit for the sign (0= + and 1 = −). The second entry, b_e is for the exponent, and we'll assume in this example that the exponent can be 0, 1, or −1. The three bits on the right represent the significand of the number. Hence, every number in this number system takes the form

$$(-1)^s \times (1 + 0.b_1 b_2 b_3) \times 2^E$$

- What is the smallest positive number that can be represented in this form?
- What is the largest positive number that can be represented in this form?
- What is the machine precision in this number system?
- What would change if we allowed $b_e \in \{-2, -1, 0, 1, 2\}$?

▲

Problem 1.21. What are the largest and smallest numbers that can be stored in single and double precicision? ▲

Problem 1.22. What is machine epsilon for single and and double precision? ▲

Problem 1.23. A typical computer number:

0	E=4	10011001100000000000000

What is this number? Is it stored in single or double precision? ▲

Problem 1.24. Explain the behavior of the sequence from the first problem in these notes using what you know about how computers store numbers in double precision.

$$x_{n+1} = \begin{cases} 2x_n, & x_n \in [0, \frac{1}{2}] \\ 2x_n - 1, & x_n \in (\frac{1}{2}, 1] \end{cases} \quad \text{with} \quad x_0 = \frac{1}{10}$$

In particular, now that you know about how numbers are stored in a computer, how long do you expect it to take until the truncation error creeps into the computation? ▲

More can be said about floating point numbers such as how we store infinity, how we store NaN, and how we store 0. The Wikipedia page for floating point arithmetic might be an interesting read for the curious student.

1.3 The Taylor Series

Now we turn our attention in this introduction to something more mathematical. The Taylor Series is a mathematical tool that is widely used in approximation theory, and since a computer can only make approximations we need ways to approximate functions so that a computer can understand them. The Taylor Series will be just the right tool for our purposes.

Consider the function $f(x) = e^x$. Euler's number

$$e = 2.718281828459045\cdots$$

is irrational and impossible for a computer to represent directly. How, do you suppose, does a computer actually *understand* a function like e^x (or any other transcendental function for that matter)?
Answer: Polynomials!
Polynomials are some of the simplest types of functions since they involve very basic mathematical operations: really just addition and multiplication (since subtraction and division are just *special* addition and multiplication).

1.3.1 Polynomial Approximation

Let's get a feel for how we approximate functions like $f(x) = e^x$ with a simple exercise. In the following exercise we will build a polynomial function with certain properties to *match* the exponential function.

Problem 1.25. Let $f(x) = e^x$.

(a) Find a linear function of the form $g(x) = a_0 + a_1 x$ such that $g(0) = f(0)$ and $g'(0) = f'(0)$.

(b) Find a quadratic function of the form $g(x) = a_0 + a_1 x + a_2 x^2$ such that $g(0) = f(0)$, $g'(0) = f'(0)$, and $g''(0) = f''(0)$.

(c) Find a polynomial of order n that matches the function $f(x) = e^x$ such that $g^{(k)}(0) = f^{(k)}(0)$ for all $k \leq n$.

▲

Problem 1.26. Repeat Problem 1.25 with the function $f(x) = \sin(x)$. ▲

Problem 1.27. Repeat Problem 1.25 with the function $f(x) = \cos(x)$. ▲

Problem 1.28. Now let's graphically explore the results that you got from Problems 1.25, 1.26, and 1.27. You should have found the following results (check your work from the previous problems):

- The exponential function:

$$f(x) = e^x \approx 1 + x \qquad \text{(First order approximation)}$$
$$f(x) = e^x \approx 1 + x + \frac{x^2}{2} \qquad \text{(Second order approximation)}$$
$$f(x) = e^x \approx 1 + x + \frac{x^2}{2} + \frac{x^3}{6} \qquad \text{(Third order approximation)}$$
$$f(x) = e^x \approx 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots + \frac{x^n}{n!} \qquad (n^{th} \text{ order approximation})$$

- The sine function

$$f(x) = \sin(x) \approx x \qquad \text{(First order approximation)}$$
$$f(x) = \sin(x) \approx x - \frac{x^3}{6} \qquad \text{(Third order approximation)}$$
$$f(x) = \sin(x) \approx x - \frac{x^3}{6} + \cdots + \frac{(-1)^n x^{2n+1}}{(2n+1)!} \qquad (n^{th} \text{ order approximation})$$

- The cosine function

$$f(x) = \cos(x) \approx 1 - \frac{x^2}{2} \qquad \text{(Second order approximation)}$$
$$f(x) = \cos(x) \approx 1 - \frac{x^2}{2} + \frac{x^4}{24} \qquad \text{(Fourth order approximation)}$$
$$f(x) = \cos(x) \approx 1 - \frac{x^2}{2} + \cdots + \frac{(-1)^n x^{2n}}{(2n)!} \qquad (n^{th} \text{ order approximation})$$

(a) Using any convenient graphing tool (e.g. Desmos, a graphing calculator, MATLAB, etc) make a plot of the exponential function along with the first, second, and third order approximations. What do you notice and what do you wonder?

(b) Repeat part (a) with the sine function.

(c) Repeat part (a) with the cosine function.

(d) Graphically test the following conjecture:

> **Conjecture:** *If I add more and more terms to the exponential, sine, or cosine polynomial approximation then the resulting polynomial will match the actual function better and better over a larger domain.*

Based on the graphs that you made do you think that this conjecture is true or false?

▲

Problem 1.29. In Problem 1.28 you should have confirmed that the Taylor series for the exponential, sine, and cosine functions appear to get better and better on larger and larger domains as the degree of the polynomial increases. Now we'll test the conjecture in general:

> **Conjecture:** *Will the Taylor polynomial always be a better approximation over a larger domain when we add more terms?*

Below you will find several functions along with their polynomial approximations. Test the above conjecture graphically on these approximations.

(a) $f(x) = \dfrac{1}{1-x} \approx 1 + x + x^2 + x^3 + x^4 + \cdots x^n.$

(b) $f(x) = \ln(1+x) \approx x - \dfrac{x^2}{2} + \dfrac{x^3}{3} - \dfrac{x^4}{4} + \cdots + \dfrac{x^n}{n}.$

(c) $f(x) = \arctan(x) = x - \dfrac{x^3}{3} + \dfrac{x^5}{5} - \dfrac{x^7}{7} + \cdots + \dfrac{(-1)^n x^{2n+1}}{2n+1}.$

▲

Now we'll formally state the actual definition of a Taylor Series.

Definition 1.30 (Taylor Series)**.** If f is infinitely smooth (has infinitely many derivatives) then f can be expressed as an infinite sum of power functions

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

in some neighborhood of $x = a$.

It is sometimes easier to look at the definition of the Taylor series without the summation notation.

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a)^1 + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \frac{f^{(4)}(a)}{4!}(x-a)^4 + \cdots.$$

Example 1.31. Verify that the Taylor series for $f(x) = e^x$ is exactly what we found in Problems 1.25 and 1.28.
Solution: First note that $f(x) = e^x$ has the beautiful property that $f^{(n)}(x) = e^x$. That is, every derivative of the exponential function is just the exponential function again. If

we take $a = 0$ in the definition of the Taylor Series then we see that

$$\begin{aligned} e^x &= f(0) + \frac{f'(0)}{1!}(x-0)^1 + \frac{f''(0)}{2!}(x-0)^2 + \frac{f'''(0)}{3!}(x-0)^3 + \frac{f^{(4)}(0)}{4!}(x-0)^4 + \cdots \\ &= e^0 + e^0 x + \frac{e^0}{2}x^2 + \frac{e^0}{3!}x^3 + \frac{e^0}{4!}x^4 + \cdots \\ &= 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots. \end{aligned}$$

Problem 1.32. We have encountered 6 different Taylor series thus far:

$$e^x, \quad \sin(x), \quad \cos(x), \quad \frac{1}{1-x}, \quad \ln(1+x), \quad \text{and} \quad \arctan(x).$$

Using the definition of the Taylor series, verify each of the formulas that we have used. You can see a verification for e^x in Example 1.31. ▲

One should keep in mind that the Taylor Series for a function may not make sense on the whole real line even if the function's domain is the entire real line. You should have taken note of this in problem 1.29 (if you didn't then go back to problem 1.29). The domain on which the Taylor series makes sense is called the **domain of convergence**. We can also talk about a Taylor series having a **radius of convergence** where the radius is the distance from the center (the a in Definition 1.30) to the first point where the Taylor series does not converge.

To determine the radius of convergence for a Taylor Series we should recall the Ratio Test from Calculus

Theorem 1.33 (The Ratio Test from Calculus)**.** Let $\sum_{n=0}^{\infty} a_n$ be an infinite series. Suppose that

$$\lim_{n\to\infty} \frac{|a_{n+1}|}{|a_n|} = r.$$

(a) If $0 < r < 1$ then the infinite series converges.

(b) If $r > 1$ then the infinite series diverges.

(c) If $r = 1$ then the ratio test is inconclusive.

For the purposes of Taylor series we need a more robust statement of the Ratio Test. Notice that in Theorem 1.33 we are only considering an infinite series of numbers, not an infinite series of functions. The following corollary gives a more thorough statement of the Ratio Test as it applies to Taylor Series.

Corollary 1.34 (The Ratio Test for Taylor Series)**.** Let $f(x)$ be given as a Taylor Series

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n.$$

The series converges for all x such that

$$\lim_{n\to\infty}\left|\frac{f^{(n+1)}(a)(x-a)^{n+1}}{(n+1)!}\right| \Big/ \left|\frac{f^{(n)}(a)(x-a)^{k}}{n!}\right| < 1.$$

Simplifying the fractions gives

$$\lim_{n\to\infty}\frac{|f^{(n+1)}(a)(x-a)|}{|f^{(n)}(a)(n+1)|} < 1.$$

The values of x that satisfy this limit are called the *domain of convergence* for the Taylor Series.

Example 1.35. Find the domain of convergence of the Taylor Series for the exponential function $f(x) = e^x$.
Solution:
For simplicity we will take the Taylor series to be centered at $a = 0$. We know that

$$f(x) = e^x = \sum_{k=0}^{\infty}\frac{x^k}{k!}$$

so we need to determine the following limit.

$$\lim_{k\to\infty}\left|\frac{x^{k+1}}{(k+1)!}\frac{k!}{x^k}\right|.$$

Simplifying the fractions inside the absolute values gives

$$\lim_{k\to\infty}\left|\frac{x}{(k+1)}\right|$$

which for every $x \in \mathbb{R}$ we see that

$$\lim_{k\to\infty}\left|\frac{x}{(k+1)}\right| = 0 < 1.$$

Therefore, for every value of $x \in \mathbb{R}$ we know that the Taylor Series for e^x converges to the actual function. Hence the "=" sign that we used in the beginning of this solution is actually valid for every x. Be careful since this is not true for all functions.

Example 1.36. Find the Taylor series centered at $a = 0$ for the function $f(x) = \frac{1}{1-x}$ and determine the domain of convergence.
Solution:

Recall from the definition of the Taylor series that

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n.$$

Hence we need to find a pattern in the sequence of derivatives $f^{(n)}(0)$ first.

$$\begin{aligned}
f(x) = (1-x)^{-1} &\implies f(0) = 1 \\
f'(x) = (1-x)^{-2} &\implies f'(0) = 1 \\
f''(x) = 2(1-x)^{-3} &\implies f''(0) = 2 \\
f'''(x) = 6(1-x)^{-4} &\implies f'''(0) = 6 = 3! \\
f^{(4)}(x) = 24(1-x)^{-5} &\implies f^{(4)}(0) = 24 = 4! \\
f^{(5)}(x) = 120(1-x)^{-6} &\implies f^{(5)}(0) = 120 = 5! \\
&\vdots \\
f^{(n)}(x) = (-1)^n n!(1-x)^{-n-1} &\implies f^{(n)}(0) = n!
\end{aligned}$$

Therefore the Taylor series for $f(x) = \frac{1}{1-x}$ is

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} \frac{n!}{n!} x^n$$

which can clearly simplify to

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + x^4 + \cdots$$

(a beautifully simple Taylor series for kind of a complicated function).

To find the domain of convergence we note that the absolute value of the ratio of successive terms in the series is

$$\left| \frac{x^{n+1}}{x^n} \right| = |x|.$$

Hence

$$\lim_{n \to 0} \left| \frac{x^{n+1}}{x^n} \right| = |x|$$

and we see from the ratio test that $|x| < 1$ is the domain of convergence. In other words, the Taylor series only makes sense for x values on the inverval $-1 < x < 1$.

A deeper look at this function reveals a deeper insight: The domain of convergence is the distance from the center of the series, $a = 0$, to the neareest vertical asymptote at $x = 1$.

Problem 1.37. Write MATLAB code to show successive approximations of the function $f(x) = e^x$ on the domain $-1 < x < 1$ using a Taylor series centered at $a = 0$. Write your code

so that it animates through the approximations. Once your code is working, modify it to do the same for $f(x) = \sin(x)$ centered at $a = 0$ and for $f(x) = \cos(x)$ centered at $a = 0$. ▲

Problem 1.38. Consider the function $f(x) = \frac{1}{1+x^2}$.

(a) Build the Taylor Series for this function centered at $a = 0$.

(b) Using the previous example as a guide find the domain of convergence for the Taylor Series.

(c) Modify your MATLAB code from the previous problem to demonstrate the fact that $f(x) = \frac{1}{1+x^2}$ does not have an infinite domain of convergence.

▲

1.3.2 Truncation Error in Taylor Series

The great thing about the Taylor Series is that it allows for the approximation of smooth functions as polynomials – and polynomials are easily dealt with on a computer. The down side is that the sum is infinite. Hence, every time we use a Taylor series on a computer we are actually going to be using a truncated Taylor Series where we only take a finite number of terms.

Problem 1.39. What is the absolute truncation error when calculating e^1 with the Taylor Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

using the following values for n? Recall that $e^1 \approx 2.718281828459045$.

n	Value from the Taylor Series	Absolute Error
0	1	1.718281828459045…
1	1+1=2	0.718281828459045…
2	$1 + 1 + 1^2/2 = 2.5$	0.218281828459045…
3	$1 + 1 + 1^2/2 + 1^3/6 = 2.\bar{6}$	
4		
5		
6		

▲

Problem 1.40. Based on your answers to the previous problem, how many terms do you need in the Taylor Series for e^x to approximation e to two decimal places? ▲

Remember that when using a computer we cannot ever use an infinite series. Thus, every time we use a Taylor Series approximation we will naturally be using a truncated version of the infinite series. This means that we need a good tool to measure the error that we make when doing so.

To set the stage, let's say that we truncate a Taylor series at the n^{th} term. Therefore there is some remainder left over in the infinity of terms from the $(n+1)^{st}$ term on and we can write the series as

$$f(x) = P_n(x) + R_n(x)$$

where $P_n(x)$ is just the n^{th} order polynomial coming from the 0^{th} to the n^{th} terms of the Taylor Series. The remainder is the subject of the next theorem.

Theorem 1.41 (Taylor's Theorem)**.** Let f, f', f'', $\ldots$, $f^{(n)}$ be continuous *near* a and let $f^{(n+1)}(x)$ exist for all x *near* $x = a$. Then there is a number ξ between x and a such that

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n(x) \tag{1.1}$$

where the remainder function $R_n(x)$ is given as

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}. \tag{1.2}$$

Often times we are using Taylor series that are centered at $a = 0$ so for simplicity we restate Taylor's theorem here with $a = 0$.

Corollary 1.42 (Taylor's Theorem at $a = 0$)**.** Let f, f', f'', $\ldots$, $f^{(n)}$ be continuous *near* a and let $f^{(n+1)}(x)$ exist for all x *near* $x = 0$. Then there is a number ξ between x and 0 such that

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots + \frac{f^{(n)}(0)}{n!}x^n + R_n(x) \tag{1.3}$$

where the remainder function $R_n(x)$ is given as

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}x^{n+1} \tag{1.4}$$

Example 1.43. The first three terms of the Taylor series for $f(x) = e^x$ centered at $x = 0$ are

$$f(x) = e^x \approx 1 + x + \frac{x^2}{2}.$$

Use Taylor's theorem to approximate the error in this approximation when $x \approx 1$.
Solution: The remainder function gives us that there exists a number ξ such that

$0 < \xi < 1$ and the remainder in the Taylor series is

$$R_3(x) = \frac{f^{(3)}(\xi)}{3!}(x-0)^3 = \frac{e^\xi}{3!}x^3.$$

Therefore $R_3(x) \leq \frac{e^1}{3!} \cdot 1^3 = \frac{e}{6} \approx 0.45$. In Figure 1.1 we see that the error is indeed less than this. Indeed, $f(1) = 2.718281828459045\cdots$ and $g(1) = 2.5$ so the actual error is about $0.218 < 0.45$.

If we extend this example a bit we can see that the absolute error and the approximated error (as found with Taylor's Theorem) converge to each other rather quickly.

n	Value from the Taylor Series	Absolute Error	Approximate Error
0	1	1.71828	2.71828
1	2	0.71828	1.35914
2	2.5	0.21828	0.45305
3	2.66666	0.05162	0.11326
4	2.70833	0.00995	0.02265
5	2.71806	0.00023	0.00054
6	2.71825	0.00003	0.00006

Example 1.44. The Taylor series for $g(x) = \frac{1}{1-x}$ is given in Example 1.36. What is the maximum error that can occur when approximating $f(c)$ for $c \in (-1,1)$ with a fifth-order polynomial?
Solution: We know that $g(x) \approx \frac{1}{1-x} = 1+x+x^2+x^3+x^4+x^5$ for $x \in (-1,1)$ and from Taylor's remainder theorem we know that $R_5(x) = g^{(6)}(\xi)x^{5+1}/(5+1)!$. The sixth derivative of g is $g^{(6)}(x) = 6!(1-x)^{-7}$ and therefore the largest that $g^{(6)}\xi$ can be for $\xi \in (-1,1)$ is $6!$. Therefore the remainder term simplifies to $R_5(x) = x^6$. Finally, the largest that $R_5(x)$ can be on $x \in (-1,1)$ is 1 so the maximum error is 1.

Long story short: Taylor's theorem gives a way to bound the amount of error that you can make when using a truncated Taylor series.

Problem 1.45. The *engineer's approximation* to the sine function is:

$$\text{For } x \text{ close to } 0, \sin(x) \approx x.$$

Obviously the word *close* is relative. Use Taylor's theorem to determine how much error is being made with the *engineer's approximation* if you want to calculate $\sin(0.5)$? See Figure 1.2. ▲

Problem 1.46. No computational software actually *knows* functions like the exponential function or the sine function. Instead, they have a way to calculate values for these functions based on Taylor series. If we want to calculate a value for $e^{0.5}$ on a computer, how many terms in the Taylor series do we need so that the truncation error is less than machine precision? ▲

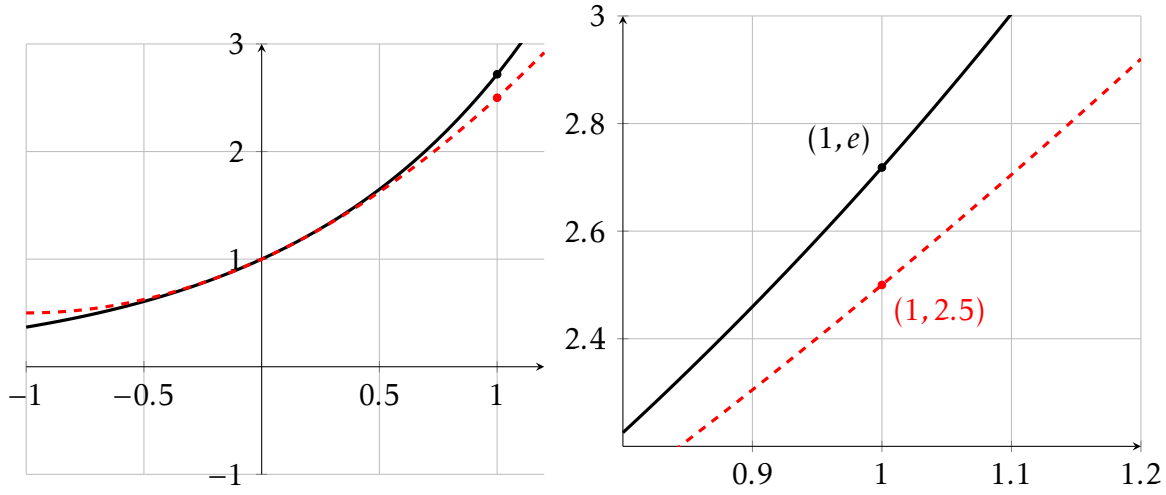


Figure 1.1. The function $f(x) = e^x$ and a second order Taylor approximation. The solid black curve is $f(x) = e^x$ and the dashed red curve is the Taylor approximation. The right-hand plot shows a zoomed in view near the point $x = 1$.

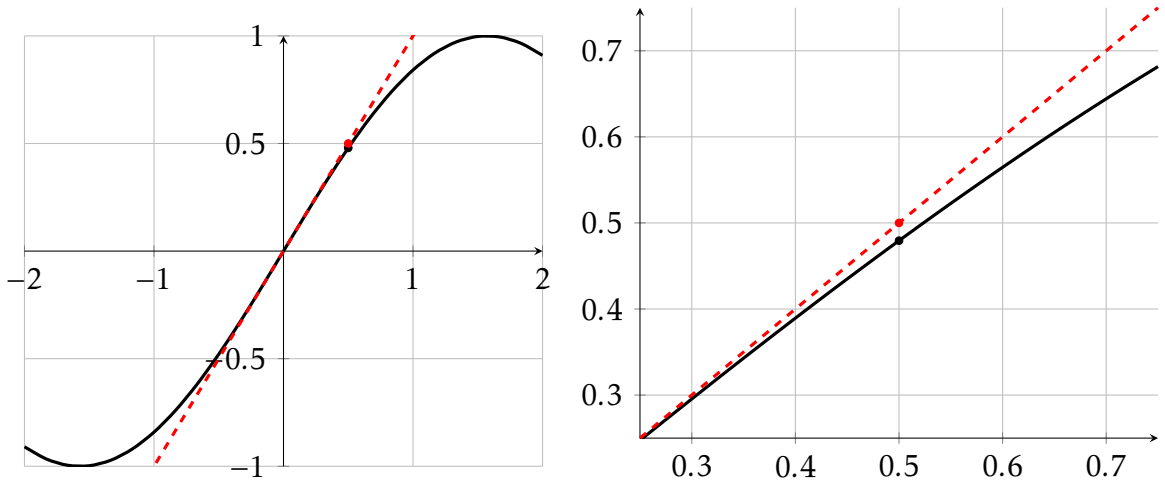


Figure 1.2. The function $f(x) = \sin(x)$ and a second order Taylor approximation. The solid black curve is $f(x) = \sin(x)$ and the dashed red curve is the Taylor approximation. The right-hand plot shows a zoomed in view near the point $x = 0.5$.

Example 1.47. How many terms of the Taylor Series for e^x (centered at $a = 0$) are needed to calculate e to within 10 decimal places?
Solution:

Recall that if $f(x) = e^x$ then

$$f(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!},$$

and this Taylor Series representation converges for all $x \in \mathbb{R}$. From Taylor's Theorem we know that the remainder term takes the form

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} x^{n+1}$$

where $\xi \in (0,1)$. For the exponential function we know that $f^{(n+1)}(\xi) = f(\xi) = e^{\xi}$. Furthermore, we are trying to approximate the error at $x = 1$ so the remainder term is

$$R_n(1) = \frac{e^{\xi}}{(n+1)!}.$$

We want to find n such that $R_n(1) < 10^{-10}$. Observe that since $\xi \in (0,1)$ we know that $e^{\xi} < e$. Hence we need n such that $e \cdot 10^{10} < (n+1)!$. Examining the orders of magnitude for the factorial sequence we see that $n \geq 13$ since $(13+1)! \approx 9 \times 10^{10} > e \cdot 10^{10}$.

Indeed, if we use 13 terms in the Taylor Series for $f(x) = e^x$ we get

$$f(1) \approx 2.71828182844676,$$

and this approximation is correct to the 10^{th} decimal digit.

Example 1.48. Calculate e to within machine precision.
Solution:
Modifying the previous example we see that we need n such that

$$R_n(1) = \frac{e^{\xi}}{(n+1)!} < 10^{-16}.$$

Examining the orders of magnitude of the factorial sequence we see that we need $n \geq 18$ terms in the Taylor Series to have a value of e that is accurate to within machine precision.

1.4 Exercises

Several of the exercises in this section are designed just to get you coding. If you are having trouble getting started on the coding please see Appendix B.

Problem 1.49. For each of the following commands tell what the command is asking MATLAB to do and why the answer is *wrong*.

(a) `sqrt(2)^2 == 2`

(b) `(1/49)*49 == 1`

(c) `exp(log(3)) == 3`

▲

Problem 1.50. If we list all of the numbers below 10 that are multiples of 3 or 5 we get 3, 5, 6, and 9. The sum of these multiples is 23. Write code to find the sum of all the multiples of 3 or 5 below 1000. Your code needs to run error free and output only the sum. Consult some of the examples in Appendix B if you're stuck with the coding. ▲

Problem 1.51. Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

$$1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

By considering the terms in the Fibonacci sequence whose values do not exceed four million, write code to find the sum of the even-valued terms. Your code needs to run error free and output only the sum. Consult some of the examples in Appendix B if you're stuck with the coding. ▲

Problem 1.52. Write computer code that will draw random numbers from the unit interval $[0,1]$, distributed uniformly, until the sum of the numbers that you draw is greater than 1. Keep track of how many numbers you draw. Then write a loop that does this process many many times. On average, how many numbers do you have draw until your sum is larger than 1?
Hint #1: In MATLAB you should use the `rand(1,1)` command to draw a single number from a uniform distribution with bounds $[0,1]$.
Hint #2: You should do this more than 1,000,000 times to get a good average . . . and the number that you get should be familiar! ▲

Problem 1.53. Use the ratio test to determine the radius of convergence for the functions $\sin(x)$, $\cos(x)$, $\ln(1+x)$, and $\arctan(x)$. Show all of your work and support your answer with plots demonstrating your findings graphically. ▲

Problem 1.54 (modified from [1])**.** In the 1999 movie *Office Space*, a character creates a program that takes fractions of cents that are truncated in a bank's transactions and deposits them to his own account. This is idea has been attempted in the past and not well that banks look for this sort of thing. In this problem you will build a simulation of the program to see how long it takes to become a millionaire.

Assumptions:

- Assume that you have access to 50,000 bank accounts.
- Assume that the account balances are uniformly distributed between \$100 and \$100,000.
- Assume that the annual interest rate on the accounts is 5% and the interest is compounded daily and added to the accounts, except that fractions of cents are truncated.
- Assume that your `illegal` account initially has a \$0 balance.

Your Tasks:

(a) Explain what the following two lines of MATLAB code do.

```
accounts = 100 + (100000-100) * rand(50000,1);
accounts = floor(100*accounts)/100;
```

(b) By hand (no computer) write the mathematical steps necessary to increase the accounts by (5/365)% per day, truncate the accounts to the nearest penny, and add the truncated amount into an account titled "`illegal`".

(c) Write code to complete your plan from part (b).

(d) Using a `while` loop, iterate over your code until the illegal account has accumulated \$1,000,000. How long does it take?

▲

Problem 1.55 (modified from [1])**.** In the 1991 Gulf War, the Patriot missle defense system failed due to roundoff error. The troubles stemmed from a computer that performed the tracking calculations with an internal clock whose integer values in tenths of a second were converted to seconds by multiplying by a 24-bit binary approximation to $\frac{1}{10}$:

$$0.1_{10} \approx 0.00011001100110011001100_2.$$

(a) Convert the binary number above to a fraction by hand (common denominators would be helpful).

(b) The approximation of $\frac{1}{10}$ given above is clearly not equal to $\frac{1}{10}$. What is the absolute error in this value?

(c) What is the time error, in seconds, after 100 hours of operation?

(d) During the 1991 war, a Scud missile traveled at approximately Mach 5 (3750 mph). Find the distance that the Scud missle would travel during the time error computed in (c).

▲

Problem 1.56 (Approximating π)**.** In this problem we will use Taylor Series to build approximations for the irrational number π.

(a) Write the Taylor series centered at $a = 0$ for the function $f(x) = \frac{1}{1+x}$. You may want to refer back to Examples 1.35 and 1.36 to get started.

(b) Use the ratio test to determine the domain of convergence for this Taylor series.

(b) Substitute t^2 for x to get a Taylor series for $g(t) = \frac{1}{1+t^2}$.

(c) Integrate both sides from $t = 0$ to $t = y$ to get a Taylor series for $h(y) = \arctan(y)$.

(d) Use the fact that $\arctan(1) = \pi/4$ along with your answer to part (c) to approximate π to 10 decimal digits of accuracy. Use Taylor's theorem to prove that you have the correct accuracy.

▲

Problem 1.57. In this problem we will prove the famous (and the author's favorite) formula

$$e^{i\theta} = \cos(\theta) + i\sin(\theta).$$

This is known as Euler's formula after the famous mathematician Leonard Euler. Show all of your work for the following tasks.

(a) Write the Taylor series for the functions e^x, $\sin(x)$, and $\cos(x)$.

(b) Replace x with $i\theta$ in the Taylor expansion of e^x. Recall that $i = \sqrt{-1}$ so $i^2 = -1$, $i^3 = -i$, and $i^4 = 1$. Simplify all of the powers of $i\theta$ that arise in the Taylor expansion.

(c) Gather all of the real terms and all of the imaginary terms together. Factor the i out of the imaginary terms. What do you notice?

(d) Use your result from part (c) to prove that $e^{i\pi} + 1 = 0$.

▲

Problem 1.58. Create a MATLAB function that accepts an anonymous function handle $f(x)$ and an integer N and gives as an output an animation of successive Taylor approximations of the function up to the N^{th} term.

```
function TaylorAnimation(f , N)
```

Write a test script that calls your function on several different infinitely differentiable functions. Your test script will look something like the following.

```
clear; clc; clf;
f = @(x) sin(x)
N = 50;
TaylorAnimation(f,N)
```

▲

Problem 1.59 (Jenny's Phone Number)**.** My favorite prime number is 8675309. Yep. Jenny's phone number is prime! Write a MATLAB script that verifies this fact. You cannot use the built-in MATLAB function `isprime( )`. Consult some of the examples in Appendix B if you're stuck with the coding. ▲

Problem 1.60. Write a function called `MyPrimeChecker` that accepts an integer and returns a binary variable: 0 = not prime, 1 = prime.

```
function Primality = MyPrimeChecker(n)
```

Next write a MATLAB script to find the sum of all of the prime numbers less than 1000. Consult some of the examples in Appendix B if you're stuck with the coding. ▲

Problem 1.61. The sum of the squares of the first ten natural numbers is,

$$1^2 + 2^2 + \cdots + 10^2 = 385$$

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + \cdots + 10)^2 = 55^2 = 3025$$

Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is $3025 - 385 = 2640$.

Write code to find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum. Your code needs to run error free and output only the difference. ▲

Problem 1.62. The prime factors of 13195 are $5, 7, 13$ and 29. Write code to find the largest prime factor of the number 600851475143? Your code needs to run error free and output only the largest prime factor. ▲

Problem 1.63. The number 2520 is the smallest number that can be divided by each of the numbers from 1 to 10 without any remainder. Write code to find the smallest positive number that is evenly divisible by all of the numbers from 1 to 20? ▲

Problem 1.64. The following iterative sequence is defined for the set of positive integers:

$$n \to \frac{n}{2} \quad \text{(n is even)}$$
$$n \to 3n + 1 \quad \text{(n is odd)}$$

Using the rule above and starting with 13, we generate the following sequence:

$$13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Write code to determine which starting number, under one million, produces the longest chain. NOTE: Once the chain starts the terms are allowed to go above one million. ▲

Problem 1.65. Find the domain of convergence for the Taylor series for each of the following Taylor series. Center each Taylor series at $a = 0$.

$$f_1(x) = \sin(x)$$
$$f_2(x) = \cos(x)$$
$$f_3(x) = \frac{1}{1+x}$$
$$f_4(x) = \frac{1}{1-x}$$
$$f_5(x) = \frac{1}{1+x^2}$$
$$f_6(x) = \ln(1+x)$$

▲

Chapter 2

Numerical Algebra

Let's play a game!

Problem 2.1. Find a partner in the room to play a two-person game. Choose someone to go first and follow these steps to play.

Step #1: The first player secretly chooses a positive real number. For simplicity let's choose something less than 100.

Step #2: The first player chooses non-negative real numbers a and b so that the secret number is between a and b.

Step #3: The first player tells the second player "*my number is between a and b*".

Step #4: The second player makes a guess at the secret number. We'll call that number c.

Step #5: The first player responds with one of the following three responses.

- If c is within 0.01 of the secret number then say "*you win!*"
- If the secret number is between a and c then say "*the secret number is between a and c.*"
- If the secret number is between c and b then say "*the secret number is between c and b.*"

Step #6: Repeat steps 4 and 5 until the second player guesses the secret number to within 0.01 or you have repeated more than 20 times. The second player wins if he/she guesses the number in 20 or fewer steps.

▲

Problem 2.2. Discuss with your partner what the optimal strategy for the second player is for the game in Problem 2.1. ▲

Problem 2.3. Now we want to write MATLAB code so you can play the game from Problem 2.1 against the computer. The following incomplete code should get you started. Fill in the missing pieces of the code and play the game several times to be sure that it works. What strategy do you find yourself using to play the game? You can get a copy of the partial code HERE.

```
%% Secret Number Guesser Game
clear; clc; format compact;
secret = 100*rand(1,1); % random secret number between 0 and 100
tolerance = 0.01;
a = randi(floor(secret),1); % initial lower bound integer below the secret
b = randi( [ceil(secret),101] , 1); % initial upper bound integer above the secret
c = 1000; % dummy initial value for c
n = 1; % initialize the game counter
while abs( secret - c ) > tolerance
    fprintf('Guess #%g.\n',n)
    fprintf('The secret number is between %g and %g.\n',a,b)
    c = input('What is your guess for the secret number? ');
    if abs( secret - c ) < tolerance
        fprintf('  \n') % respond appropriately here
        ... % write more code here if necessary
    elseif secret < (c-tolerance)
        ... % write appropriate code here in this case
    elseif secret > (c+tolerance)
        ... % write appropriate code here in this case
    end
    if n>=20
        fprintf('   \n') % respond appropriately here
        break % this will break out of the loop
    end
    n=n+1; % increment the game counter
end
```

▲

2.1 Introduction to Root Finding

In this chapter we want to solve algebraic equations using a computer. Consider the equation $\ell(x) = r(x)$ (where ℓ and r stand for left and right respectively). To solve this equation we can first rewrite it by subtracting the right-hand side from the left to get

$$\ell(x) - r(x) = 0.$$

For example, if we want to solve $3\sin(x) + 9 = x^2 - \cos(x)$ then this is the same as solving $(3\sin(x) + 9) - (x^2 - \cos(x)) = 0$. Hence, we can define a function $f(x)$ as $f(x) = \ell(x) - r(x)$ and observe that every algebraic equation can be written as: "if $f(x) = 0$, find x". We illustrate this idea in Figure 2.1. On the left-hand side of Figure 2.1 we see the solutions to the equation $3\sin(x) + 9 = x^2 - \cos(x)$, and on the right-hand side of Figure 2.1 we see the solutions to the equation $(3\sin(x) + 9) - \left(x^2 - \cos(x)\right) = 0$. From the plots it is apparent that the two equations have the same solutions: $x_1 \approx -2.55$ and $x_2 \approx 2.88$.

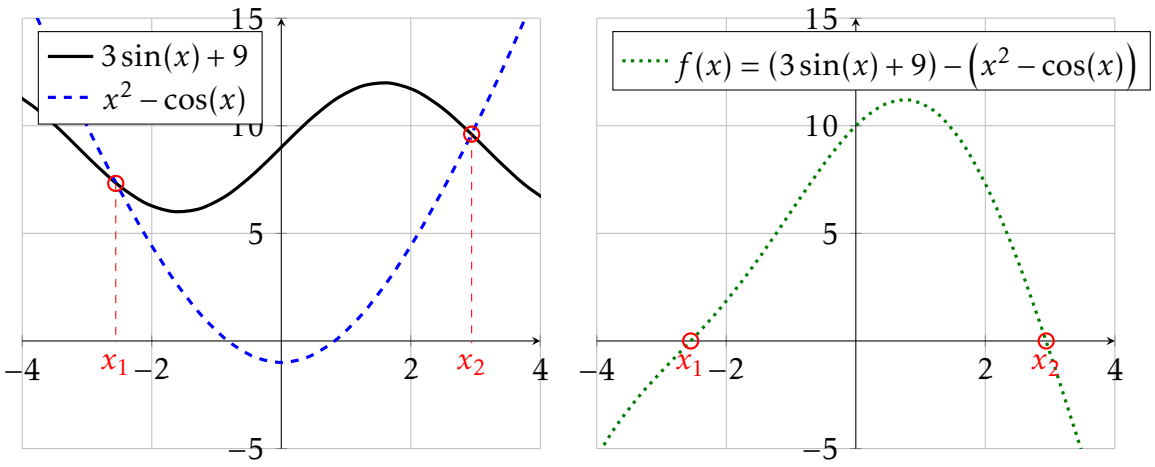


Figure 2.1. The left-hand plot shows two nonlinear functions, $\ell(x) = \sin(x) + 9$ and $r(x) = x^2 - \tan(x)$, with their intersection points marked. The right-hand plot shows the equivalent problem formed by solving $\ell(x) - r(x) = 0$.

We now have one way to view every algebraic equation-solving problem. As we'll see in this chapter, if $f(x)$ has certain properties then different numerical techniques for solving the equation will apply – and some will be much faster and more accurate than others. The following sections give several different techniques for solving algebraic equations of the form $f(x) = 0$.

2.2 The Bisection Method

We'll start the mathematical discussion with a theorem from Calculus.

Theorem 2.4 (The Intermediate Value Theorem (IVT))**.** If $f(x)$ is a continuous function on the closed interval $[a,b]$ and y_* lies between $f(a)$ and $f(b)$, then there exists some point $x_* \in [a,b]$ such that $f(x_*) = y_*$.

Problem 2.5. Draw a picture of what the intermediate value theorem says graphically. ▲

Problem 2.6. If $y_* = 0$ the intermediate value theorem gives us important information about solving equations. What does it tell us? ▲

Corollary 2.7. If $f(x)$ is a continuous function on the closed interval $[a,b]$ and if $f(a)$ and $f(b)$ have opposite signs then from the Intermediate Value Theorem we know that there exists some point $x_* \in [a,b]$ such that _____.

Theorem 2.4 and its corollary are *existence theorems* in the sense that they tell us that some point exists. The annoying thing about mathematical existence theorems is that they typically don't tell us *how* to find the point that is guaranteed to exist – annoying. The following algorithm, known as the Bisection Method, uses the Intermediate Value Theorem to systematically approximate the point x_* to solve the algebraic equation $f(x_*) = 0$.

Problem 2.8. Discussion: What does the game from Problem 2.1 have in common with the Intermediate Value Theorem and its corollary? Use your optimal strategy from the game along with the corollary to the IVT to propose a strategy to find roots of a continuous function. Write psuedo-code for your solution strategy. ▲

Algorithm 2.9 (The Bisection Method)**.** Assume that $f(x)$ is continuous on the closed interval $[a,b]$. To make approximations of the solutions to the equation $f(x) = 0$, do the following:

1. Check to see if $f(a)$ and $f(b)$ have opposite signs (why is this important?).
2. Compute the midpoint of the closed interval, $m = \frac{a+b}{2}$, and evaluate $f(m)$.
3. Compare the signs of $f(a)$ vs $f(m)$ and $f(b)$ vs $f(m)$. Replace one of the endpoints of the closed interval $[a,b]$ with $m = (a+b)/2$. (Which one do you replace and why?)
4. Repeat steps 2 and 3 and stop when $f(m)$ is *close enough* to zero.

Problem 2.10. Draw a picture illustrating what the Bisection Method does to approximate solutions to the algebraic equation $f(x) = 0$. ▲

Problem 2.11. We want to write a MATLAB function for the Bisection Method. Instead of jumping straight into the code we should ALWAYS write pseudo-code first. It is often helpful to write pseudo-code as comments in your MATLAB file. Use the template below to complete your pseudo-code.

```
function root = Bisection(f , a , b , tol)
% The input parameters are
% f is an anonymous function handle
% a is the lower guess
% b is the upper guess
% tol is an optional tolerance for the accuracy of the root

% if the user doesn't define a tolerance we need code to create a default

% check that there is a root between a and b
% if not we should return an error and break the code

% next calculate the midpoint m = (a+b)/2

% start a while loop
    % in the while loop we need an if statement
    % if ...
    % elseif ...
    % elseif ...

    % we should check that the while loop isn't running away

% end the while loop
% define the root
```

▲

Problem 2.12. Now use the pseudo-code as structure to complete a MATLAB function for the Bisection Method. Also write a test script that verifies that your function works properly. Be sure that it can take an anonymous function handle as an input along with an initial lower bound, an initial upper bound, and an optional error tolerance. The output should be only 1 single number: the root.

```
function root=Bisection(f , a , b , tol)
```

▲

Problem 2.13. Test your Bisection Method code on the following algebraic equations.

1. $x^2 - 2 = 0$ on $x \in [0, 2]$
2. $\sin(x) + x^2 = 2\ln(x) + 5$ on $x \in [0, 5]$ (be careful! make a plot first)
3. $(5 - x)e^x = 5$ on $x \in [0, 5]$

▲

Problem 2.14. Let $f(x)$ be a continuous function on the interval $[a, b]$ and assume that $f(a) \cdot f(b) < 0$. If we want to approximate the solution to the equation $f(x) = 0$ to within δ how many iterations will the bisection method need?
Hint: If we want an approximation within δ then the width of the interval is 2δ. ▲

Problem 2.15. How many iterations of the bisection method are necessary to approximate $\sqrt{3}$ to within $10^{-3}, 10^{-4}, \ldots, 10^{-15}$ using the initial interval $[a, b] = [0, 2]$? ▲

2.3 The Regula Falsi Method

The bisection method is one of many methods for performing root finding on a continuous function. The next algorithm takes a slightly different approach.

Algorithm 2.16 (The Regula Falsi Method)**.** Assume that $f(x)$ is continuous on the interval $[a, b]$. To make approximations of the solutions to the equation $f(x) = 0$, do the following:

1. Check to see if $f(a)$ and $f(b)$ have opposite signs so that the intermediate value theorem guarantees a root on the interval.
2. Write the equation of the line connecting the points $(a, f(a))$ and $(b, f(b))$. Fill in the blanks in the point slope form.

$$y - \_\_\_\_\_ = \_\_\_\_\_ \cdot (x - \_\_\_\_\_)$$

3. Find the x intercept of the linear function that you wrote in the previous step. Call this point $x = c$.

$$c = \_\_\_\_\_$$

4. Just as we did with the bisection method, compare the signs of $f(a)$ vs $f(c)$ and $f(b)$ vs $f(c)$. Replace one of the endpoints with c. Which one do you replace and why?
5. Repeat steps 2 - 4, and stop when $f(c)$ is *close enough* to zero.

Problem 2.17. Draw a picture of what the Regula Falsi method does to approximate a root. ▲

Problem 2.18. Give sketches of functions where the Regula Falsi method will perform faster than the Bisection method and visa versa. Justify your thinking with several pictures and be prepared to defend your answers. ▲

Problem 2.19. Create a new MATLAB function called `RegulaFalsi` and write comments giving pseudo-code for the Regula-Falsi method. ▲

Problem 2.20. Use your pseudo-code to create a MATLAB function that implements the Regula Falsi method, and write a test script that verifies that your function works properly. Your function should accept an anonymous function handle as an input along with an initial lower bound, an initial upper bound, and an optional error tolerance. The output should be only 1 single number: the root.

```
function root=RegulaFalsi(f , a , b , tol)
```
▲

2.4 Newton's Method

We now investigate a calculus-based method (originally proposed by Isaac Newton and later modified by Joseph Raphson) for solving the algebraic equation $f(x) = 0$. In very basic terms, this method involves iteratively finding tangent lines to a differentiable curve and locating where those tangent lines intersect the horizontal axis.

Algorithm 2.21 (Newton's Method)**.** The Newton-Raphson method for solving algebraic equations can be described as follows:

1. Check that f is a twice differentiable function on a given domain and find a way to guarantee that f has a root on that domain (this step happens by hand, not on the computer).
2. Pick a starting point x_0 in the domain
3. Write the equation of a tangent line to f at x_0. Fill in the blanks in the point-slope form.
$$y - \_\_\_\_\_ = \_\_\_\_\_ \cdot (x - \_\_\_\_\_)$$
4. Find the x intercept of the equation of the tangent line and call this new point x_1.
$$x_1 = \_\_\_\_\_$$
5. Now iterate the process by replacing the labels "x_1" and "x_0" in the previous step with x_{n+1} and x_n respectively.
$$x_{n+1} = \_\_\_\_\_$$
6. Iterate step 5 until $f(x_n)$ is *close* to zero.

Problem 2.22. Draw a picture of what Newton's method does graphically. ▲

Problem 2.23. There are several reasons why Newton's method could fail. Work with your partners to come up with a list of all of the reasons. Support each of your reasons with a sketch or an algebraic example. ▲

Problem 2.24. Create a new MATLAB function called `Newton` and write comments giving pseudo-code for Newton's method. This version of Newton's method will accept the function and the first derivative so you don't need to set aside any code for calculating the derivative. ▲

Problem 2.25. Write a MATLAB function for Newton's method. Your function needs to accept an anonymous function handle, the derivative of $f(x)$ as an anonymous function handle, an initial guess, and an optional error tolerance. The only output should be the solution to the equation that you are solving. Write a test script to verify that your Newton's method code indeed works.

```
function soln = Newton(f , df , x0 , tol)
```
▲

Problem 2.26. The previous problem required that you calculate the derivative ahead of time for Newton's method. While this is relatively easy with a computer algebra system it might be easier just to have your function compute the derivative for you. Write a new MATLAB function for Newton's method that accepts the function as an anonymous function handle, the initial point, and an optional error tolerance. Your function must compute the derivative symbolically behind the scenes.

```
function soln = NewtonSymbolic(f, x0, tol)
```
▲

Problem 2.27. In Problem 2.23 you constructed several examples of where Newton's method will fail. Modify your Newton's method code to catch these special cases and warns the user before the method fails. Test your new code with several of the special cases showing that indeed you were able to catch them properly and give the user appropriate feedback. ▲

Problem 2.28. Newton's Method is known to have a *quadratic convergence rate*. This means that

$$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|^2}$$

will be constant where x_* is the root that we're hunting for. This implies that if we plot the error in the new iterate on the y-axis and the error in the old iterate on the x axis of a log-log plot then we will see a constant slope of 2. (stop and verify why this would be true).

In this problem we're going to build a numerical experiment to verify that Newton's method indeed has quadratic convergence. Modify your Newton's method code so that it outputs all of the iterations instead of just the final root. Once you have the iterations compute the error between the approximations and the exact root. For simplicity let's solve the equation $x^2 - 2 = 0$. Plot the sequence of error approximations with the iterate e_k on the x-axis and the iterate e_{k+1} on the y-axis of a log-log plot.
Your plot command will look something like:

```
loglog(error(1:end-1),error(2:end),'b*')
```

where `error` is a vector containing all of the errors. Quadratic convergence means that at every iteration the error should decrease by roughly 2 orders of magnitude. How can you see this in your plot? ▲

Problem 2.29. Repeat the previous problem with the bisection and regula falsi methods. Plot the errors of all three methods on the same plot and discuss the rates of convergence for the three methods. ▲

2.5 Quasi-Newton Methods

Newton's method requires that you have a function and a derivative of that function. The conundrum here is that sometimes the derivative is cumbersome or impossible to obtain but you still want to have the great quadratic convergence exhibited by Newton's method.

Recall that Newton's method is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

If we replace $f'(x_n)$ with an approximation of the derivative then we may have a method that is *close* to Newton's method in terms of convergence rate but is less troublesome to compute. Any method that replaces the derivative in Newton's method with an approximation is called a **Quasi-Newton Method**.

Algorithm 2.30 (Secant Method)**.** To solve $f(x) = 0$:

1. Determine if there is a root *near* an arbitrary starting point x_0.
2. Pick a second starting point *near* x_0. Call this second starting point x_1. Note well that the points x_0 and x_1 should be close to each other. (Why?)
   (The choice here is different than for the bisection method)
3. Use the backward difference

   $$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

   to approximate the derivative of f at x_n.
4. Perform the Newton-type iteration

   $$x_{n+1} = x_n - \frac{f(x_n)}{\left(\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}\right)}$$

   until $f(x_n)$ is *close enough* to zero. Notice that the new iteration simplifies to

   $$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}.$$

Problem 2.31. Draw several pictures showing what the Secant method does pictorially. ▲

Problem 2.32. Write MATLAB code for solving algebraic equations of the form $f(x) = 0$ with the Secant method. You should ALWAYS start by writing pseudo-code as comments in your MATLAB file. Your function should accept an anonymous function handle, two starting points, and an optional error tolerance. Also write a test script that clearly shows that your code is working.

```
function soln = SecantMethod(f, x0, x1, tol)
```
▲

Problem 2.33. Choose a non-trivial algebraic equation for which you know the solution and write a script to empirically determine the convergence rate of the Secant method. You may want to look back at 2.28. ▲

2.6 Exercises

Problem 2.34. In this problem you will demonstrate that all of your root finding codes work. At the beginning of this chapter we proposed the algebraic equation solving problem

$$3\sin(x) + 9 = x^2 - \cos(x).$$

Write a MATLAB script that calls upon your Bisection, Regula Falsi, Newton, and Secant, methods one at a time to find the positive solution to this equation. Your script needs to output the solutions in a clear and readable way so you can tell which answer can from which root finding algorithm. Clearly all of your root finding routines will be stored as MATLAB functions so be sure to turn in all of these functions along with this problem. ▲

Problem 2.35. Compare the number of iterations necessary for convergence to within 10^{-8} for both the bisection method and the regula falsi method on several test problems. Find example problems where bisection converges faster and examples where regula falsi converges faster. Write a test script that clearly indicates to the user which equation was being solved, which endpoints were used, and which root finding technique performed faster. ▲

Problem 2.36 (Modified from [1])**.** An artillery officer wishes to fire his cannon on an enemy brigade. He wants to know the angle to aim the cannon in order to strike the target. Follow the steps below to arrive at an approximate answer.

(a) Solve the simple differential equation $v_y'(t) = -g$ by hand where $v_y(t)$ is the vertical velocity of the canon and gravity is given as $g \approx 9.8\text{m/s}^2$. We don't know the initial velocity so just use $v(0) = v_0$ and hence $v_y(0) = v_0\sin(\theta)$. Note: your answer will have a "v_0" and a "$\sin(\theta)$" in it.

(b) Solve the differential equation $s_y'(t) = v_y(t)$ by hand for the position function $s_y(t)$. Assume that $s_y(0) = 0$. Your answer will still have a "v_0" and a "$\sin(\theta)$" in it.

(c) Solve $s_y(t) = 0$ by hand for t in terms of v_0 and θ to find a function for the amount of time the projectile takes to reach the ground.

(d) In the absence of air resistance the projectile will have a constant velocity in the horizontal direction. Solve the differential equation $s_x'(t) = v_0\cos(\theta)$ for the horizontal position function $s_x(t)$.

(e) The range function $R(v_0, \theta)$ can be found by substituting the time from part (c) into the horizontal position function in part (d). Find the function $R(v_0, \theta)$.

(f) For a certain projectile and canon the initial velocity is $v_0 = 126\text{m/s}$. We want to give the artillery officer a distance, d, and have them calculate the angle to hit the target. Write MATLAB code to approximate θ in the equation

$$R(126, \theta) = d.$$

(Hint: remember that we can rewrite the equation $R(126,\theta) = d$ in the form $f(\theta) = 0$ for some function f so that we can actually use our numerical root finding techniques. See the discussion at the very beginning of this chapter.)

Report a table of values of the form shown below and provide an appropriate plot showing your results. Clearly some distances will be out of range so be sure to clearly indicate the range of the weapon.

Distance (d meters)	Angle (θ)
0	
25	
50	
100	
⋮	

▲

Problem 2.37. Consider our three primary root finding methods: Bisection, Newton's method, and the Secant method.

(a) For each method give a mathematical situation where you might want to use the method and give a mathematical situation where you might not want to use the method.

(b) For each of these methods give one example where the method will fail.

Support your arguments with proper mathematics. You are welcome to argue graphically but be sure to fully explain your graphical reasoning. ▲

Problem 2.38. Write MATLAB code that produces a loglog plot that compares the convergence rates for the bisection method, the regula-falsi method, Newton's method, and the secant method on a problem where we know the exact answer. See problem 2.28 to get you started. ▲

Problem 2.39. A *fixed point* of a function $f(x)$ is a point that solves the equation $f(x) = x$. Fixed points are interesting in iterative processes since fixed points don't change under repeated application of the function f.

For example, consider the function $f(x) = x^2 - 6$. The fixed points of $f(x)$ can be found by solving the equation $x^2 - 6 = x$ which, when simplified algebraically, is $x^2 - x - 6 = 0$. Factoring the left-hand side gives $(x-3)(x+2) = 0$ which implies that $x = 3$ and $x = -2$ are fixed points for this function. That is, $f(3) = 3$ and $f(-2) = -2$. Notice, however, that finding fixed points is identical to a root finding problem.

(a) Use a numerical root-finding algorithm to find the fixed points of the function $f(x) = x^2 - 6$ on the interval $[0, \infty)$.

(b) Find the fixed points of the function $f(x) = \sqrt{\frac{8}{x+6}}$.

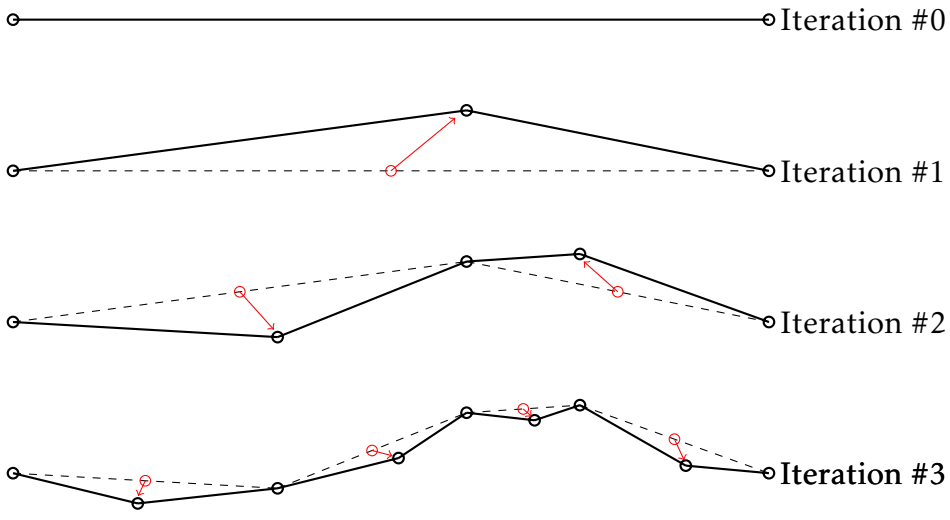
▲

Problem 2.40. In Single Variable Calculus you studied methods for finding local and global extrema of functions. You likely recall that part of the process is to set the first derivative to zero and to solve for the independent variable (remind yourself why you're doing this). The trouble with this process is that it may be very very challenging to do the algebraic solve by hand. This is a perfect place for Newton's method or any other root finding techinque!
Find the local extrema for the function $f(x) = x^3(x-3)(x-6)^4$ and explicitly demonstrate, without the use of a plot, that you have found all of the local extrema. ▲

Challenge 2.41 (modified from [1])**.** Implement the following algorithm to create a *fractal coastline* – a collection of line segments that looks like a realistic coastline as viewed from above.

1. Begin with one straight line segment from the point $(0,0)$ to the point $(1,0)$. Make a plot of this line segment using the `plot( )` command in MATLAB.
2. Repeat the following until your image looks like a coastline.
   (a) Find the midpoint of every line segment
   (b) For each midpoint create a new point by adding a random 2×1 vector to that midpoint. The largest allowable size of the random vector needs to be smaller in each iteration.
   (c) Connect the points with the plot command. You may want to put a pause command in your code so you can see the evolution of your coastline

An example of the first few iterations is shown in the figures below where the red arrows indicate the random 2×1 vectors, the dashed lines indicate the previous iteration, the red circles indicate the midpoints, and the black circles indicate the new points. The solid black line is the result of the algorithm at the end of the iteration. Write MATLAB code to implement this algorithm where the resulting animation shows the evolution of the coastline. ▲



Chapter 3

Numerical Calculus

In this brief chapter we discuss techniques for approximating the two primary computations in calculus: taking derivatives and evaluating definite integrals. Throughout this chapter we will make heavy use of Taylor's Theorem to build these approximations. At the end of the chapter we'll examine a numerical technique for solving optimization problems without explicitly finding derivatives.

3.1 Numerical Differentiation

In this section we'll build several approximation of first and second derivatives. The idea for each of these approximation is:

- Partition the interval $[a,b]$ into N points.
- Approximate the derivative at the point $x \in [a,b]$ by using linear combinations of $f(x-h)$, $f(x)$, $f(x+h)$, and/or other points in the partition.

Partitioning the interval into discrete points turns the continuous problem of finding a derivative at every real point in $[a,b]$ into a discrete problem where we calculate the approximate derivative at finitely many points in $[a,b]$. Figure 3.1 shows a depiction of the partition as well as making clear that h is the separation between each of the points in the partition. Note that in general the points in the partition do not need to be equally spaced, but that is the simplest place to start.

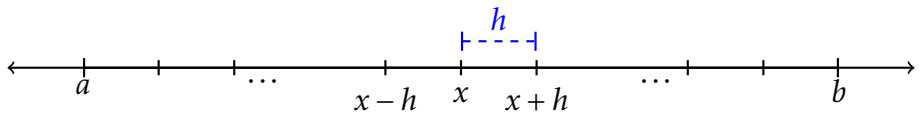


Figure 3.1. A partition of an interval on the real line.

If we recall that the definition of the first derivative of a function is

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{3.1}$$

our first approximation for the first derivative is naturally

$$\frac{df}{dx} \approx \frac{f(x+h)-f(x)}{h}. \tag{3.2}$$

In (3.2) we have simply removed the limit and instead approximated the derivative as the slope. It should be clear that this approximation is only good if h is *small*. The linear combination that we spoke about before is

$$\frac{df}{dx} \approx \frac{1}{h}f(x+h) - \frac{1}{h}f(x).$$

While this is the simplest and most obvious approximation for the first derivative there is a much more elegant technique, using Taylor series, for arriving at this approximation. Furthermore, the Taylor series technique suggests an infinite family of other techniques.

Problem 3.1. From Taylor's Theorem we know that for an infinitely differentiable function $f(x)$,

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a)^1 + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \frac{f^{(4)}(a)}{4!}(x-a)^4 + \cdots.$$

What do we get if we replace x with $x+h$ and a with x? In other words, in Figure 3.1 we want to center the Taylor series at x and evaluate the resulting series at the point $x+h$. ▲

Problem 3.2. Solve the result from the previous problem for $f'(x)$ to create an approximation for $f'(x)$ using $f(x+h)$, $f(x)$, and some higher order terms. How can we use Taylor's Theorem to quantify the error of this approximation? ▲

Definition 3.3 (Order of a Numerical Derivative)**.** The **order** of a numerical derivative is the power of the step size in the remainder term. For example, a first order method will have "h^1" in the remainder term. A second order method will have "h^2" in the remainder term.

Theorem 3.4 (First Order Approximation of the First Derivative)**.** In problem 3.2 we derived a first order approximation of the first derivative:

$$f'(x) = \frac{f(x+h)-f(x)}{h} + \mathcal{O}(h).$$

In this formula, $h = \Delta x$ is the step size.

In the previous definition, "$\mathcal{O}(h)$" (read: big-O of h) states that the method is first order. This means that the maximum error that you're making with this method is on the order of the size of the step. Not surprisingly, if we let h get arbitrarily small then the error in this method gets arbitrarily small. More formally we have the following definition.

Definition 3.5 (Big $\mathcal{O}$ Notation)**.** We say that the error in a differentiation method is "big O of h", $E = \mathcal{O}(h)$, if and only if there is a positive constant M such that

$$|Error| \leq M|h|.$$

This is equivalent to saying that a differentiation method is first order.

Problem 3.6. Explain what the phrase

"The approximation of $f'(x)$ in Theorem 3.4 is $\mathcal{O}(h)$"

into your own words. ▲

Problem 3.7. Write MATLAB code that takes a function and a domain $(xmin, xmax)$ and returns a numerical approximation to the derivative on the interval $(xmin, xmax)$. You should ALWAYS start by writing pseudo-code as comments in your MATLAB file. Your function should accept an anonymous function handle, the bounds on the domain, and the number of interior points used for approximation within the domain. Your function should output the x values and y values associated with the derivative.

`function [new_x,dfdx]=FirstDerivFirstOrder(f,xmin,xmax,num_interior_pts)` ▲

The only two ways to really check a numerical derivative is to plot the numerical approximation and to do the derivative by hand and to plot the error. Be warned, however, that the numerical derivative that we have built from Theorem 3.4 should have one less value than the original list of x and y values. Think about why this must be true. Also double check your code from the previous problem and make sure that you can plot `new_x` vs `dfdx` without having to change their size.

Problem 3.8. Write a MATLAB script that find a first order approximation for the first derivative of $f(x) = \sin(x) - x\sin(x)$ on the interval $x \in (0, 15)$. Your script should output two plots (side-by-side).

1. The left-hand plot should show the function in blue and the first derivative as a red dashed curve. Sample code for this problem is

```
f = @(x) sin(x) - x*sin(x);
a=0; b=15;
num_interior_pts = 1000; % this should be LARGE
x = linspace(a,b,num_interior_pts);
[new_x,dfdx] = FirstDerivFirstOrder(f,a,b,num_interior_pts);
subplot(1,2,1)
plot(x, f(x) , 'b' , new_x , dfdx , 'r--')
```

2. The right-hand plot should show the absolute error between the exact derivative and the numerical derivative.

```
df = @(x) ... % write code for the exact derivative
subplot(1,2,2)
plot(new_x, abs( df(new_x) - dfdx ) , 'k--')
```

Discuss how you can see the fact that this is a first order method. You may want to put one (or both) of the plots on a log-log scale or a semi-log scale ... I'll let you figure out which one makes the most sense. ▲

Problem 3.9. Consider again the Taylor series for an infinitely differentiable function $f(x)$:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a)^1 + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \frac{f^{(4)}(a)}{4!}(x-a)^4 + \cdots$$

This time, replace x with $x-h$ and a with x and simplify. Once you have the Taylor series centered at x and evaluated at $x-h$ form the linear combination

$$f(x+h) - f(x-h)$$

using your result from 3.1 and solve for $f'(x)$. Your result should be a second-order accurate approximation for the first derivative of f. Simplify your approximation formula and verify that it is indeed second order. ▲

Theorem 3.10 (Second Order Approximation of the First Derivative)**.**

$$f'(x) = \underline{\qquad\qquad\qquad\qquad} + \mathcal{O}(h^2)$$

Problem 3.11. Write a MATLAB function that takes a function and a domain and returns a second order numerical approximation to the first derivative on the interval. You should ALWAYS start by writing pseudo-code as comments in your MATLAB file. Your function should accept an anonymous function handle, the bounds on the domain, and the number of interior points used for approximation within the domain. Your function should output the x values and y values associated with the derivative.

```
function [new_x,dfdx]=FirstDerivSecondOrder(f,xmin,xmax,num_interior_pts)
```

▲

Problem 3.12. Add the Taylor series for $f(x+h)$ and $f(x-h)$ to arrive at an approximation of the second derivative. What is the order of the error? ▲

Problem 3.13. Write a MATLAB function that takes a function and a domain and returns a second order numerical approximation to the second derivative on the interval. You should ALWAYS start by writing pseudo-code as comments in your MATLAB file. Your function should accept an anonymous function handle, the bounds on the domain, and the number of interior points used for approximation within the domain. Your function should output the x values and y values associated with the derivative.

```
function [new_x,ddfdxx]=SecondDerivSecondOrder(f,xmin,xmax,num_interior_pts)
```

▲

Derivative	Formula	Error	Name
1^{st}	$f'(x) \approx \frac{f(x+h)-f(x)}{h}$	$\mathcal{O}(h)$	Forward Difference
1^{st}	$f'(x) \approx \frac{f(x)-f(x-h)}{h}$	$\mathcal{O}(h)$	Backward Difference
1^{st}	$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$	$\mathcal{O}(h^2)$	Centered Difference
2^{nd}	$f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$	$\mathcal{O}(h^2)$	Centered Difference

Table 3.1. First and second derivatives.

Problem 3.14. Test your second derivative code on the function $f(x) = \sin(x) - x\sin(x)$. Create an error plot showing the accuracy of the method. ▲

Table 3.1 summarizes the formulas that we have for derivatives thus far. The exercises at the end of this chapter contain several more derivative approximations. We will return to this idea when we study numerical differential equations in Chapter 5.

3.2 Numerical Integration

Next we will build methods for approximating integrals. Recall from Calculus that the definition of the Riemann integral is

$$\int_a^b f(x)dx = \lim_{\Delta x \to 0} \sum_{j=1}^{N} f(x_j)\Delta x \tag{3.3}$$

where N is the number of subintervals on the interval $[a, b]$ and Δx is the width of the interval. As with differentiation, we can remove the limit and have a decent approximation of the integral

$$\int_a^b f(x)dx \approx \sum_{j=1}^{N} f(x_j)\Delta x.$$

You are likely familiar with this approximation of the integral from Calculus. The value of x_j can be chosen anywhere within the subinterval and three common choices are to use the left endpoint, the midpoint, and the right endpoint. We see a depiction of this in Figure 3.2.

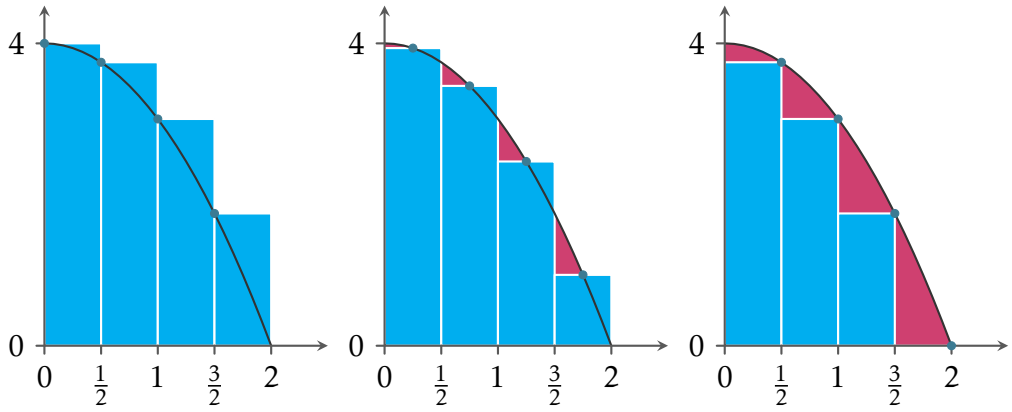


Figure 3.2. Riemann sum to approximate an integral with left, midpoint, and right rectangles.

Clearly, the more rectangles we choose the closer the sum of the areas of the rectangles will get to the integral.

Problem 3.15. Write MATLAB code approximate an integral withe Riemann sums. You should ALWAYS start by writing pseudo-code as comments in your MATLAB file. Your MATLAB function should accept an anonymous function handle, a lower bound, an upper bound, the number of subintervals, and an optional input that allows the user to designate whether they want left, right, or midpoint rectangles.

```
function Area=MyRiemannSum(f, a, b, num_subintervals, type)
```

Test your code on several functions for which you know the integral. ▲

Problem 3.16. Create a plot with the width of the subintervals on the horizontal axis and the absolute error between your (left) Riemann sum calculation and the exact integral for a known definite integral. Your plot should be on a log-log scale. Based on your plot, what is the approximate order of the error in the Riemann sum approximation? ▲

Problem 3.17. We want to approximate $\int_a^b f(x)dx$. One of the simplest ways is to approximate the area under the function with a trapezoid. Recall from basic geometry that area of a trapezoid is $A = \frac{1}{2}(b_1 + b_2)h$. In terms of the integration problem we can do the following:

1. First partition $[a, b]$ into the set $\{x_0 = a, x_1, x_2, \ldots, x_{n-1}, x_n = b\}$.
2. On each part of the partition approximate the area with a trapezoid:

$$A_j = \frac{1}{2}\Big[f(x_j) + f(x_{j-1})\Big]\Big(x_j - x_{j-1}\Big)$$

3. Approximate the integral as

$$\int_a^b f(x)dx = \sum_{j=1}^{n} A_j$$

Draw a picture depicting how the trapezoidal rule works. ▲

The trapezoidal rule does a decent job approximating integrals, but ultimately you are using linear functions to approximate $f(x)$ and the accuracy may suffer if the step size is too large or the function too non-linear. You likely notice that the trapezoidal rule will give an exact answer if you were to integrate a linear or constant function. A potentially better approach would be to get an integral that evaluates quadratic functions exactly. In order to do this we need to evaluate the function at three points (not two like the trapezoidal rule). Let's integrate a function $f(x)$ on the interval $[a, b]$ by using the three points $(a, f(a))$, $(m, f(m))$, and $(b, f(b))$ where $m = \frac{a+b}{2}$ is the midpoint of the two boundary points. We want to find constants $A1$, $A2$, and $A3$ such that the integral

$$\int_a^b f(x)dx = A_1 f(a) + A_2 f\left(\frac{a+b}{2}\right) + A_3 f(b)$$

is exact for all constant, linear, and quadratic functions. This would guarantee that we have an exact method for all polynomials of order 2 or less but should serve as a decent approximation if the function is not quadratic.

To find the constants A_1, A_2, and A_3 we can write the following system of three equations

$$\int_a^b 1\,dx = b-a = A_1 + A_2 + A_3$$
$$\int_a^b x\,dx = \frac{b^2-a^2}{2} = A_1 a + A_2\left(\frac{a+b}{2}\right) + A_3 b$$
$$\int_a^b x^2\,dx = \frac{b^3-a^3}{3} = A_1 a^2 + A_2\left(\frac{a+b}{2}\right)^2 + A_3 b^2.$$

Solving the linear system gives

$$A_1 = \frac{b-a}{6}, \quad A_2 = \frac{4(b-a)}{6}, \quad \text{and} \quad A_3 = \frac{b-a}{6}.$$

At this point we can see that the integral can be approximated as

$$\int_a^b f(x)dx \approx \left(\frac{b-a}{6}\right)\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

and the technique will give an exact answer for any polynomial of order 2 or below. To improve upon this idea we now examine the problem of partitioning the interval $[a,b]$ into small pieces and running this process on each piece.

Problem 3.18. Now we put the process explained above into a form that can be coded to approximate integrals. We call this method Simpson's Rule after Thomas Simpson (1710-1761) who, by the way, was a basket weaver in his day job so he could pay the bills and keep doing math.

1. First parition $[a,b]$ into the set $\{x_0 = a, x_1, x_2, \ldots, x_{n-1}, x_n = b\}$.
2. On each part of the partition approximate the area with a parabola:
$$A_j = \frac{1}{6}\left[f(x_j) + 4f\left(\frac{x_j + x_{j-1}}{2}\right) + f(x_{j-1})\right]\left(x_j - x_{j-1}\right)$$
3. Approximate the integral as
$$\int_a^b f(x)dx = \sum_{j=1}^{n} A_j$$

Draw a picture depicting how Simpson's rule works. ▲

Problem 3.19. Write MATLAB functions that implement both the trapezoidal rule and Simpson's rule. You should ALWAYS start by writing pseudo-code as comments in your MATLAB file. Keep in mind that MATLAB deals with vectors and iteration in very nice ways. You shouldn't need a `for` loop in your function.

Test both MATLAB functions on known integrals and approximate the order of the error based on the mesh size. ▲

Thus far we have three numerical approximations for definite integrals: Riemann sums (with rectangles), the trapezoidal rule, and Simpsons's rule. There are MANY other approximations for integrals and we leave the further research to the curious reader.

Theorem 3.20 (Numerical Integration Techniques)**.** Let $f(x)$ be a continuous function on the interval $[a,b]$. The integral $\int_a^b f(x)dx$ can be approximated with any of the following.

$$\text{Riemann Sum: } \int_a^b f(x)dx \approx \sum_{j=1}^{N} f(x_j)\Delta x$$

Error for Riemann Sums: $\mathcal{O}(\Delta x)$

$$\text{Trapezoidal Rule: } \int_a^b f(x)dx \approx \frac{1}{2}\sum_{j=1}^{N}\Big(f(x_j)+f(x_{j-1})\Big)\Delta x$$

Error for Trapezoidal Rule: $\mathcal{O}(\Delta x^2)$

$$\text{Simpson's Rule: } \int_a^b f(x)dx \approx \frac{1}{6}\sum_{j=1}^{N}\left(f(x_j)+4f\left(\frac{x_j+x_{j-1}}{2}\right)+f(x_{j-1})\right)\Delta x$$

Error for Simpson's Rule: $\mathcal{O}(\Delta x^4)$

where $\Delta x = x_j - x_{j-1}$ and N is the number of subintervals.

Problem 3.21. Theorem 3.20 simply states the error rates for our three primary integration schemes. For this problem you need to empirically verify these error rates. Use the integration problem and exact answer

$$\int_0^{\pi/4} e^{3x}\sin(2x)dx = \frac{3}{13}e^{3\pi/4}+\frac{2}{13}$$

and write code that produces a log-log error plot with Δx on the horizontal axis and the absolute error on the vertical axis. Fully explain how the error rates show themselves in your plot. ▲

Example 3.22. In Figure 3.3 you'll see the error for our three primary integration techniques on the problem

$$\int_0^1 \sin(2x)dx$$

which has an exact answer of

$$\int_0^1 \sin(2x)dx = \frac{1}{2}-\frac{\cos(2)}{2}.$$

You'll see that Simpson's rule gains 4 orders of accuracy for every order of magnitude that the Δx is decreased. Similarly you'll see that Trapezoidal method gains 2 orders of magnitude of accuracy for every order of magnitude that Δx is decreased, and Riemann sums only gain 1 order of magnitude. Plots such as this can reveal the performance of a numerical method and give you a sense of how it is going to behave on a problem where you don't know the exact answer.

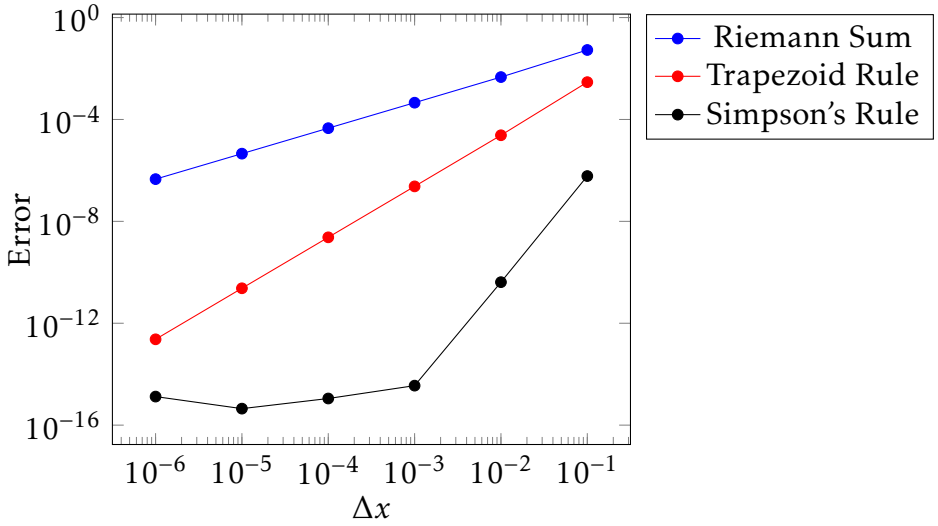
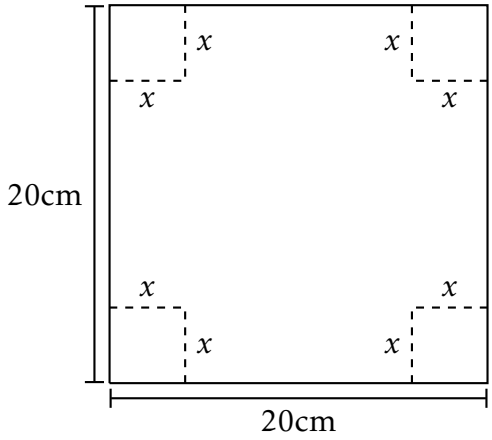


Figure 3.3. Errors with integration techniques.

3.3 Numerical Optimization

You likely recall that one of the major applications of Calculus was to solve optimization problems – find the value of x which makes some function as big or as small as possible. The process itself can sometimes be rather challenging since due to the modeling aspect of the problems but all due to the fact that the differentiation might be quite cumbersome. In this section we will revisit those problems from Calculus, but our goal will be to build a numerical method for the Calculus step in hopes to avoid the messy algebra and differentiation.

Problem 3.23 (The Classic Cardboard Box Problem)**.** A piece of cardboard measuring 20cm by 20cm is to be cut so that it can be folded into a box without a lid. Find the size of the cut, x, that maximizes the volume of the box.



(a) Write a function for the volume of the box resulting from a cut of size x. What is the domain of your function?

(b) We know that we want to maximize this function so go through the full Calculus exercise to find the maximum:

- take the derivative
- set it to zero
- find the critical points
- test the critical points and the boundaries of the domain using the extreme value theorem to determine the x that gives the maximum.

▲

3.3.1 Single Variable Numerical Optimization

Now we'll propose a few numerical techniques that can approximate the solution to these types of problems. The basic ideas are simple!

Problem 3.24. If you were blind folded and standing on a hill could you find the top of the hill? (assume no trees and no cliffs . . . this isn't supposed to be dangerous) How would you do it? Explain your technique clearly. ▲

Problem 3.25. If you were blind folded and standing on a crater on the moon could you find the lowest point? How would you do it? Remember that you can hop as far as you like . . . because gravity . . . but sometimes that's not a great thing because you could hop too far. ▲

The intuition of numerical optimization schemes is typically to visualize the function that you're trying to minimize or maximize and think about either climbing the hill to the top (maximization) or descending the hill to the bottom (minimization).

Problem 3.26. Let's turn your intuition into an algorithm. If $f(x)$ is the function that you are trying to maximize then turn your algorithm from Problem 3.24 into a step-by-step algorithm which could be coded. Then try out your code on the function

$$f(x) = e^{-x^2} + \sin(x^2)$$

to see if your algorithm can find the local maximum near $x \approx 1.14$. ▲

Some of the most common algorithms are listed below. Read through them and see which one(s) you ended up recreating? The intuition for these algorithms is pretty darn simple – travel uphill if you want to maximize – travel downhill if you want to minimize.

Algorithm 3.27 (Derivative Free Optimization)**.** Let $f(x)$ be the objective function which you are seeking to maximize (or minimize).

- Pick a starting point, x_0, and find the value of your objective function at this point, $f(x_0)$.
- Pick a small step size (say, $\Delta x \approx 0.01$).
- Calculate the objective function one step to the left and one step to the right from your starting point. Which ever point is larger (if you're seeking a maximum) is the point that you keep for your next step.
- Iterate (decide on a good stopping rule)

Problem 3.28. Write code to implement the 1D derivative free optimization algorithm and use it to solve Problem 3.23. Compare your answer to the analytic solution. ▲

Algorithm 3.29 (Gradient Ascent/Descent)**.** Let $f(x)$ be the objective function which you are seeking to maximize (or minimize).

- Find the derivative of your objective function, $f'(x)$.
- Pick a starting point, x_0.
- Pick a small control parameter, α, called the "learning rate"
- Use the iteration $x_{n+1} = x_n + \alpha f'(x_n)$ if you're maximizing. Use the iteration $x_{n+1} = x_n - \alpha f'(x_n)$ if you're minimizing.
- Iterate (decide on a good stopping rule)

Problem 3.30. Write code to implement the 1D gradient descent algorithm and use it to solve Problem 3.23. Compare your answer to the analytic solution. ▲

Algorithm 3.31 (Monte Carlo Search)**.** Let $f(x)$ be the objective function which you are seeking to maximize (or minimize).

- Pick many (perhaps several thousand!) different x values.
- Find the value of the objective function at every one of these points (Hint: use lists, not loops)
- Keep the x value that has the largest (or smallest if you're minimizing) value of the objective function.
- Iterate many times and compare the function value in each iteration to the previous best function value

Problem 3.32. Write code to implement the 1D monte carlo search algorithm and use it to solve Problem 3.23. Compare your answer to the analytic solution. ▲

Algorithm 3.33 (Optimization via Numerical Root Finding)**.** Let $f(x)$ be the objective function which you are seeking to maximize (or minimize).

- Find the derivative of your objective function.
- Set the derivative to zero and use a numerical root finding method (such as bisection or Newton) to find the critical point.
- Use the extreme value theorem to determine if the critical point or one of the endpoints is the maximum (or minimum).

Problem 3.34. Write code to implement the 1D numerical root finding optimization algorithm and use it to solve Problem 3.23. Compare your answer to the analytic solution. ▲

Problem 3.35. In this problem we will compare an contrast the four methods proposed in the previous problem.

(a) What are the advantages to each of the methods proposed?

(b) What are the disadvantages to each of the methods proposed?

(c) Which method, do you suppose, will be faster in general? Why?

(d) Which method, do you suppose, will be slower in general? Why?

▲

Problem 3.36. The Gradient Ascent/Descent algorithm is the most geometrically interesting of the four that we have proposed. The others are pretty brute force algorithms. What is the Gradient Ascent/Descent algorithm doing geometrically? Draw a picture and be prepared to explain to your peers. ▲

Problem 3.37. (This problem is modified from [6])
A pig weighs 200 pounds and gains weight at a rate proportional to its current weight. Today the growth rate if 5 pounds per day. The pig costs 45 cents per day to keep due mostly to the price of food. The market price for pigs if 65 cents per pound but is falling at a rate of 1 cent per day. When should the pig be sold and how much profit do you make on the pig when you sell it? Write this situation as a single variable mathematical model and solve the problem analytically (by hand). Then solve the problem with all four methods outlined thus far in this section. ▲

Problem 3.38. (This problem is modified from [6])
Reconsider the pig problem 3.37 but now suppose that the weight of the pig after t days is

$$w = \frac{800}{1 + 3e^{-t/30}} \text{ pounds.}$$

When should the pig be sold and how much profit do you make on the pig when you sell it? Write this situation as a single variable mathematical model. You should notice that the algebra and calculus for solving this problem is no longer really a desirable way to go. Use an appropriate numerical technique to solve this problem. ▲

Problem 3.39. Numerical optimization is often seen as quite challenging since the algorithms that we have introduced here could all get "stuck" at local extrema. To illustrate this see the function shown in Figure 3.4. How will derivative free optimization methods have trouble finding the red point starting at the black point with this function? How will gradient descent/ascent methods have trouble? Why? ▲

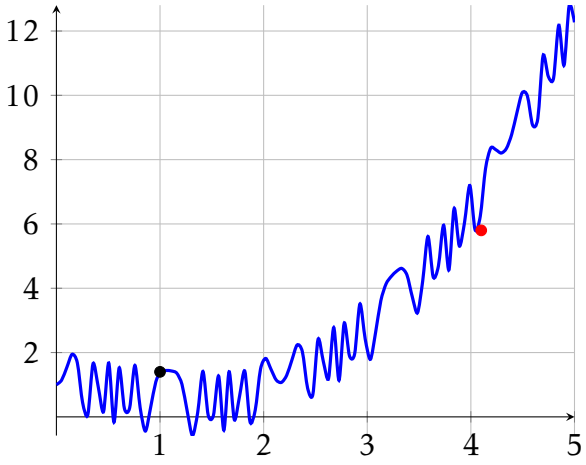


Figure 3.4. A challenging numerical optimization problem. If we start at the black point then how will any of our algorithms find the local minimum at the red point?

3.3.2 Multivariable Numerical Optimization

Now let's look at multivariable optimization. The analytic process for finding optimal solutions is essentially the same as for single variable.

- Write a function that models a scenario in multiple variables,
- find the gradient vector (presuming that the function is differentiable),
- set the gradient vector equal to the zero vector and solve for the critical point(s), and
- interpret your answer in the context of the problem.

The trouble with unconstrained multivariable optimization is that finding the critical points is now equivalent to solving a system of nonlinear equations; a task that is likely impossible even with a computer algebra system. (In Section 4.8 we will build a multivariable version of Newton's method if you want to look ahead.)

Let's see if you can extend your intuition from single variable to multivariable.

Problem 3.40. The derivative free optimization method discussed in the single variable optimization section just said that you should pick two points and pick the one that takes you furthest uphill.

(a) Why is it insufficient to choose just two points if we are dealing with a function of two variables? Hint: think about contour line.

(b) For a function of two variables, how many points should you use to compare and determine the direction of "uphill"?

(c) Extend your answer from part (b) to n dimensions. How many points should we compare if we are in n dimensions and need to determine which direction is "uphill"?

(d) Back in the case of a two-variable function, you should have decided that three points was best. Explain an algorithm for moving one point at a time so that your three points eventually converge to a nearby local maximum. It may be helpful to make a surface plot or a contour plot of a well-known function just as a visual.

▲

Problem 3.41. Now let's tackle the gradient ascent/descent algorithm. You should recall that the gradient vector points in the direction of maximum change. How can you use this fact to modify the gradient ascent/descent algorithm given previously? Clearly write your algorithm so that a classmate could turn it into code. ▲

Problem 3.42. How does the Monte Carlo algorithm extend to a two-variable optimization problem? Clearly write your algorithm. ▲

Problem 3.43. Try out the three algorithms that you just wrote on the function $f(x,y) = \sin(x)\cos(y) + 0.1x^2$ which has many local extrema and no global maximum. ▲

The derivative free, gradient ascent/descent, and monte carlo techniques still have good analogues in higher dimensions. We just need to be a bit careful since in higher dimensions there is much more room to move. Below we'll give the full description of the gradient ascent/descent algorithm. We don't give the full description of the derivative free or Monte Carlo algorithms since there are many ways to implement them. The interested reader should see a course in mathematical optimization or machine learning.

Algorithm 3.44 (Gradient Descent)**.** We want to solve the problem

$$\text{minimize } f(x_1, x_2, \ldots, x_n) \text{ subject to } (x_1, x_2, \ldots, x_n) \in S.$$

1. Choose an arbitrary starting point $\mathbf{x}_0 = (x_1, x_2, \ldots, x_n) \in S$.
2. We are going to define a difference equation that gives successive guesses for the optimal value:
$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla f(\mathbf{x}_n).$$
The difference equation says to follow the negative gradient a certain distance from your present point (why are we doing this). Note that the value of α is up to you so experiment with a few values (you should probably take $\alpha \leq 1$ ... why?).
3. Repeat the iterative process in step 2 until two successive points are *close enough* to each other.

Note: If you are looking to maximize your objective function then in the Monte-Carlo search you should examine if z is greater than your current largest value. For gradient descent you should actually do a gradient *ascent* and follow the positive gradient instead of the negative gradient.

Problem 3.45. Write generic code to implement the Gradient Descent algorithm in as many dimensions as necessary and test your code on a multivariable function where you know the location and value of a local minimum. ▲

Problem 3.46. The functions like $f(x,y) = \sin(x)\cos(y)$ have many local extreme values which makes optimization challenging. Impelement your Gradient Descent code on this function to find the local minimum $(0,0)$. Start somewhere near $(0,0)$ and show by way of example that your gradient descent code may not converge to this particular local minimum. Why is this important? ▲

3.4 Numerical Curve Fitting

In this section we'll change our focus a bit to look at a different question from algebra ... and reveal a hidden numerical optimization problem. Instead of finding the roots of an algebraic equation, what if we have a few data points and a reasonable guess for the type of function fitting the points, how would we determine the actual function?

Problem 3.47. Consider the function $f(x)$ that goes exactly through the points $(0,1)$, $(1,4)$, and $(2,13)$.

(a) Find a function that goes through these points exactly. Be able to defend your work.

(b) Is your function unique? That is to say, is there another function out there that also goes exactly through these points?

▲

Problem 3.48. Now let's make a minor tweak to the previous problem. Let's say that we have the data points $(0,1.07)$, $(1,3.9)$, $(2,14.8)$, and $(3,26.8)$. Notice that these points are *close* to the points we had in the previous problem, but all of the y values have a little noise in them and we have added a fourth point. If we suspect that a function $f(x)$ that *best* fits this data is quadratic then $f(x) = ax^2 + bx + c$ for some constants a, b, and c.

(a) Plot the four points along with the function $f(x)$ for arbitrarily chosen values of a, b, and c.

(b) Work with your partner(s) to systematically change a, b, and c so that you get a good visual match to the data.

(c) Now let's be a bit more systematic about things. Let's say that you have a pretty good guess that $b \approx 2$ and $c \approx 0.7$. We need to get a good estimate for a.

(i) Pick an arbitrary starting value for a. For each of the four points find the error between the predicted y value and the actual y value.

(ii) Square all four of your errors and add them up. (Stop and think: why are we squaring the errors before we sum them ... have a discussion here)

(iii) Now change your value of a to several different values and record the sum of the square errors for each of your values of a. It may be worth while to use a spreadsheet to keep track of your work here.

(iv) Make a plot with the value of a on the horizontal axis and the value of the sum of the square errors on the vertical axis. Use your plot to defend the optimal choice for a.

▲

Problem 3.49. We're going to revisit part (c) of the previous problem. Write a loop that tries many values of a in very small increments and calculates the sum of the squared errors. Have your loop stop when you think that it has reached the best value of a. ▲

Technique 3.50 (Nonlinear Least Squares Curve Fitting)**.** Let

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

be a set of n ordered pairs in $\mathbb{R}^2$. If we guess that a function $f(x)$ is a best choice to fit the data and if $f(x)$ depends on parameters $a_1, a_2, \ldots$ then

1. Pick initial values for the parameters $a_1, a_2, \ldots$ so that the function $f(x)$ *looks like* it is close to the data (this is strictly a visual step ... take care that it may take some playing around to guess the initial values of the parameters)

2. Calculate the square error between the data point and the prediction from the function $f(x)$

   $$\text{error for the point } x_i\text{: } e_i = (y_i - f(x_i))^2.$$

   Note that squaring the error has the advantages of removing the sign, accentuating errors larger than 1, and decreasing errors that are less than 1.

3. As a measure of the total error between the function and the data, sum the squared errors

   $$\text{sum of square errors } = \sum_{i=1}^{n} (y_i - f(x_i))^2.$$

   (Take note that if there were a continuum of points instead of a discrete set then we would integrate the square errors instead of taking a sum.)

4. Change the parameters $a_1, a_2, \ldots$ so as to minimize the sum of the square errors.

In Technique 3.50 the last step is a bit vague. That was purposeful since there are many techniques that could be used to minimize the sum of the square errors. In this particular section of this text we'll look at two of them: (1) Solver in MS Excel and (2) `fminsearch` in MATLAB. Before we get into how those techniques work let's look at an example of how the minimization step works graphically.

Let's return to the data set

$$S = \{(0, 1.07), (1, 3.9), (2, 14.8), (3, 26.8)\}$$

under the assumption that the curve we want to use is quadratic: $f(x) = ax^2 + bx + c$.* In Figure 3.5 we see the points in red along with several guesses for the values of the parameters. In these figures we are just changing one of the parameters while keeping the other two fixed for easier illustration. In reality we should note that all three of the parameters need to be changed to achieve the desired minimization.

In Figure 3.5 we saw that for $b = 2$ and $c = 0.7$ then the quadratic function $f(x) \approx 2.5x^2+2x+0.7$ is a pretty good approximation of the data $S = \{(0, 1.07), (1, 3.9), (2, 14.8), (3, 26.8)\}$.

*Take note that since we have four points then we could find an exact interpolating fit if we used a cubic function instead, but that isn't the point. In fact, using higher order polynomials has the disadvantage of potentially fitting the curve to measurement noise.

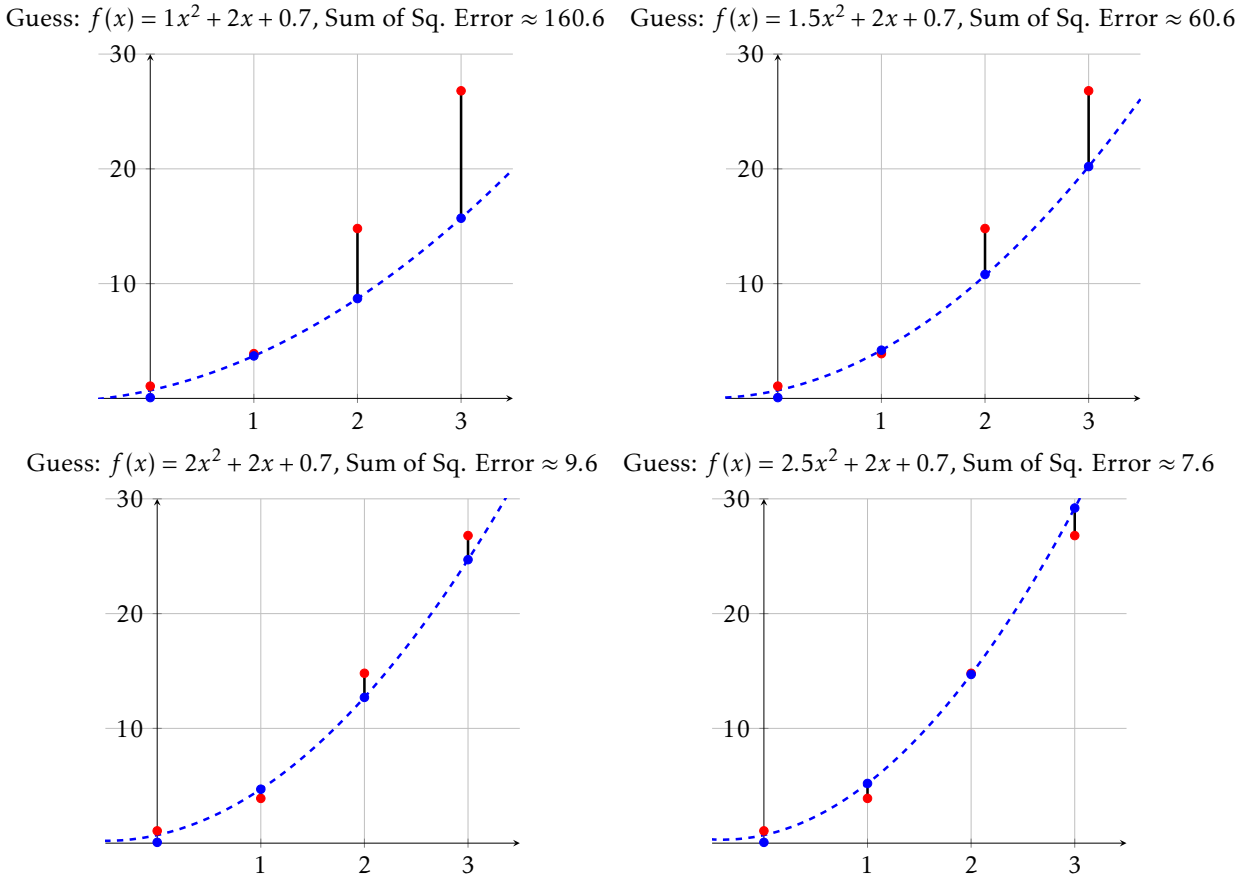


Figure 3.5. Demonstration of the curve fitting technique. The vertical black bars between the points show the point-wise error in the estimate. In these four pictures we are only modifying the value of a while leaving $b = 2$ and $c = 0.7$ fixed for simplicity. In reality we need to change all three parameters in the minimization process - hence the actual minimization takes place in three dimensions.

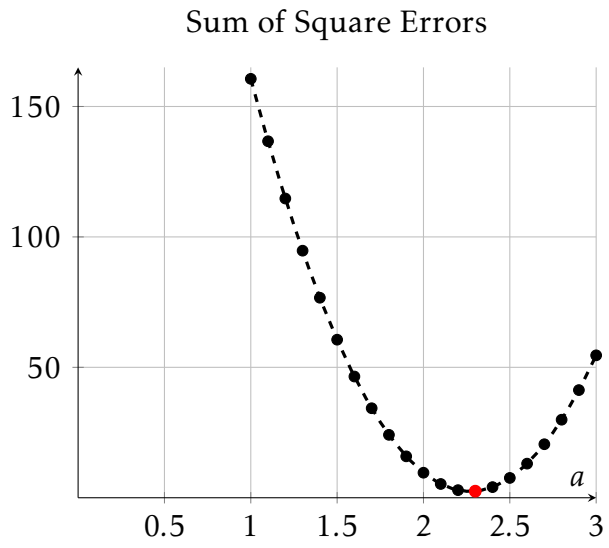


Figure 3.6. The sum of the squared errors for the function $f(x) = ax^2 + bx + c$ for the data set $S = \{(0, 1.07), (1, 3.9), (2, 14.8), (3, 26.8)\}$. For $b = 2$ and $c = 0.7$ the minimum occurs on this curve at approximately $a \approx 2.3$ with the sum of the square errors of approximately 2.51. (To build this plot we took increments of 0.1 between different guesses for a from 1 to 3 – a very brute-force way of approximating a minimum.

If, however, we take finer increments between our guesses for a we can get a refined estimate of the value of a that minimizes the sum of the squared errors. In Figure 3.6 we see the values of the sum of the squared errors for many estimates of a. In 3.6 we see that $a \approx 2.3$ gives a slightly better value of the sum of the square errors and is (visually) near the minimum of the error curve.

Problem 3.51. For each of the plots in Figure 3.7 give a functional form that *might* be a good model for the data. Be sure to choose the most general form of your guess (for example, if you choose "quadratic" then your functional guess is $f(x) = ax^2 + bx + c$). Once you have a guess of the function type create a plot showing your data along with your guess for a reasonable set of parameters. Your instructor will make the data available to you. ▲

It should be clear to the reader at this point that curve fitting is actually an optimization problem. For every collection of parameters in your chosen function there is a unique sum of square errors. The goal, of course, is to pick parameters that minimize the sum of square errors. It is beyond the scope of this book to write a gradient descent or derivative free optimization routine to do this optimization. Instead we will lean on several pre-built tools that can do this optimization for us: MS Excel's Solver and MATLAB's `fminsearch` to name two of them. If you are curious about how the optimization occurs see a course in mathematical optimization or machine learning.

Problem 3.52. In Problem 3.51 you made hypotheses about the types of functions that fit the data shown in 3.7. Use Excel's Solver function to minimize the sum of the square residuals for between your parameterized function and the data. ▲

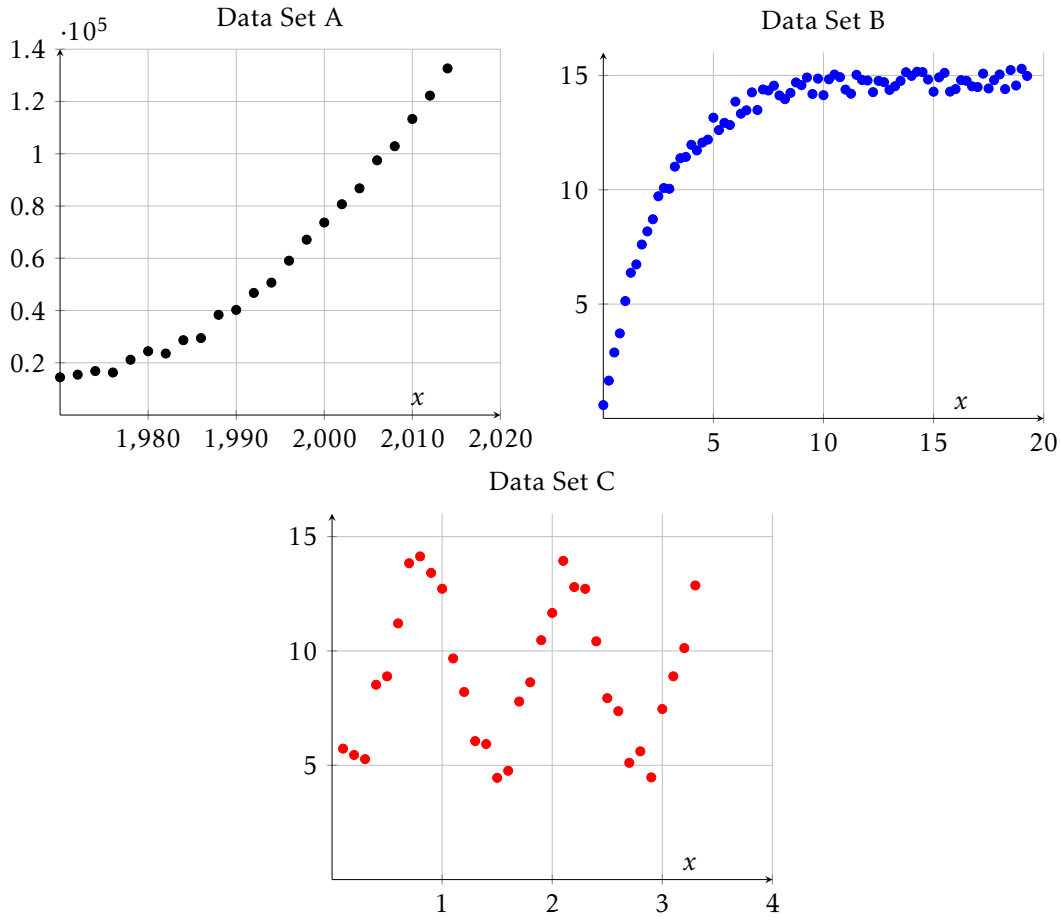


Figure 3.7. Several data plots for Problem 3.51.

Problem 3.53. Repeat the previous problem with MATLAB's `fminsearch` function. ▲

Problem 3.54. With your partner come up with an algebraic function governed by several parameters. Add some random noise to your function and then save the data (without any formulas). Give your data to a different group in the class and have them reverse engineer your function from your data. The second group should report the function, the parameters, and the sum of the square errors. ▲

3.5 Exercises

Problem 3.55. For each of the following numerical differentiation formulas (1) prove that the formula is true and (2) find the order of the method. To prove that each of the formulas is true you will need to write the Taylor series for all of the terms in the numerator on the right and then simplify to solve for the necessary derivative. The highest power of the remainder should reveal the order of the method. It would be very wise to redo problems 3.1 - 3.2, 3.9, and 3.12 before beginning this problem.

(a) $f'(x) \approx \dfrac{\frac{1}{12}f(x-2h) - \frac{2}{3}f(x-h) + \frac{2}{3}f(x+h) - \frac{1}{12}f(x+2h)}{h}$

(b) $f'(x) \approx \dfrac{-\frac{3}{2}f(x) + 2f(x+h) - \frac{1}{2}f(x+2h)}{h}$

(c) $f''(x) \approx \dfrac{-\frac{1}{12}f(x-2h) + \frac{4}{3}f(x-h) - \frac{5}{2}f(x) + \frac{4}{3}f(x+h) - \frac{1}{12}f(x+2h)}{h^2}$

(d) $f'''(x) \approx \dfrac{-\frac{1}{2}f(x-2h) + f(x-h) - f(x+h) + \frac{1}{2}f(x+2h)}{h^3}$

▲

Problem 3.56. Write a MATLAB function that accepts a list of (x,y) ordered pairs from an Excel spreadsheet and returns a list of (x,y) ordered pairs for a first order approximation of the first derivative of the underlying function.

```
function [new_x,dydx] = FirstDerivFromData(ExcelFileName,Xrange,Yrange)
```

In the function all of the inputs are strings. For example

```
FirstDerivativeFromData('MyFunctionData.xlsx','A2:A101','B2:B101')
```

Create a test Excel file and a test script that have graphical output showing that your MATLAB function is finding the correct derivative. ▲

Problem 3.57. Write a MATLAB function that accepts a list of (x,y) ordered pairs from an Excel spreadsheet and returns a list of (x,y) ordered pairs for a second order approximation of the second derivative of the underlying function.

```
function [new_x,dydx] = SecondDerivFromData(ExcelFileName,Xrange,Yrange)
```

In the function all of the inputs are strings. For example

```
SecondDerivativeFromData('MyFunctionData.xlsx','A2:A101','B2:B101')
```

Create a test Excel file and a test script that have graphical output showing that your MATLAB function is finding the correct derivative. ▲

Problem 3.58. Write a MATLAB function that implements the trapezoidal rule on a list of (x,y) order pairs representing the integrand function. The list of ordered pairs should be read from an Excel file.

```
function Area = TrapezoidFromData(ExcelFilename, Xlist, Ylist)
```

In the function all of the inputs are strings. For example

```
Area = TrapezoidFromData('MyFunctionData.xlsx','A2:A101','B2:B101')
```

Create a test Excel file and a test script showing that your MATLAB function is finding the correct integral. ▲

Problem 3.59. Use numerical integration to answer the question in each of the following scenarios

(a) We measure the rate at which water is flowing out of a reservoir (in gallons per second) several times over the course of one hour. Estimate the total amount of water which left the reservoir during that hour.

time (min)	0	7	19	25	38	47	55
flow rate (gal/sec)	316	309	296	298	305	314	322

(b) The department of transportation finds that the rate at which cars cross a bridge can be approximated by the fucntion

$$f(t) = \frac{22.8\text{cars/min}}{3.5 + 7(t - 1.25)^4},$$

where $t = 0$ at 4pm, and is measured in hours. Estimate the total number of cars that cross the bridge betwee 4 and 6pm. Make sure that your estimate has an error less than 5% and provide sufficient mathematical evidence of your error estimate.

▲

Problem 3.60. Consider the integrals

$$\int_{-2}^{2} e^{-x^2/2} dx \quad \text{and} \quad \int_{0}^{1} \cos(x^2) dx.$$

Neither of these integrals have closed-form solutions so a numerical method is necessary. Create a loglog plot that shows the errors for the integrals with different values of h (log of h on the x-axis and log of the absolute error on the y-axis). Write a complete interpretation of the loglog plot. To get the *exact* answer for these plots use the MATLAB `quad` command. (What we're really doing here is comparing our algorithms to MATLAB's built in algorithm). ▲

Problem 3.61. Go to data.gov or the World Health Organization Data Repository and find data sets for the following tasks.

(a) Find a data set where the variables naturally lead to a meaningful derivative. Use your code from Problem 3.56 to evaluate and plot the derivative. If your data appears to be subject to significant noise then use the Excel curve fitting tools first to smooth the data; then do the derivative. Write a few sentences explaning what the derivative means in the context of the data.

(b) Find a data set where the variables naturally lead to a meaningfun definite integral. Use your code from Problem 3.58 to evaluate the definite integral. If your data appears to be subject to significant noise then use the Excel curve fitting tools first to smooth the data; then do the integral. Write a few sentences explaning what the integral means in the context of the data.

In both of these tasks be very cautious of the units on the data sets and the units of your answer. ▲

Problem 3.62. Go to the USGS water data repository:
https://maps.waterdata.usgs.gov/mapper/index.html.
Here you'll find a map with information about water resources around the country.

- Zoom in to a dam of your choice (make sure that it is a dam).
- Click on the map tag then click "Access Data"
- From the dropdown menu at the top select either "Daily Data" or "Current / Historical Data". If these options don't appear then choose a different dam.
- Change the dates so you have the past year's worth of information.
- Select "Tab-separated" under "Output format" and press Go. Be sure that the data you got has a flow rate (ft^3/sec).
- At this point you should have access to the entire data set. Copy it into a `csv` file and save it to your computer.

For the data that you just downloaded you have three tasks: (1) plot the data in a reasonable way giving appropriate units, (2) find the total amount of water that has been discharged from the dam during the past calendar year, and (3) report any margin of error in your calculation based on the numerical method that you used in part (2). ▲

Problem 3.63. Integrate each of the functions over the interval $[-1, 2]$ and verify mathematically that your numerical integral is correct to 10 decimal places. Then provide a plot of the function along with its numerical first derivative.

(a) $f(x) = \dfrac{x}{1+x^4}$

(b) $g(x) = (x_1)^3(x-2)^2$

(c) $h(x) = \sin\left(x^2\right)$

▲

Problem 3.64. Although Simpsons rule was derived from parabolas, prove that it integrates all cubic polynomials exactly. ▲

Problem 3.65. The plots in Figure 3.8 show noisy data from measurements of elementary single variable functions.

(a) Make a hypothesis about which function would best model the data. Be sure to choose the most general (parameterized) form of your function.

(b) Use appropriate tools to find the parameters for the function that best fits the data. Report you sum of square residuals for each function.

Your instructor will provide you with the raw data. Note that in the pictures the points are connected with line segments simply for visual effect. The functions that you propose must be continuous functions. ▲

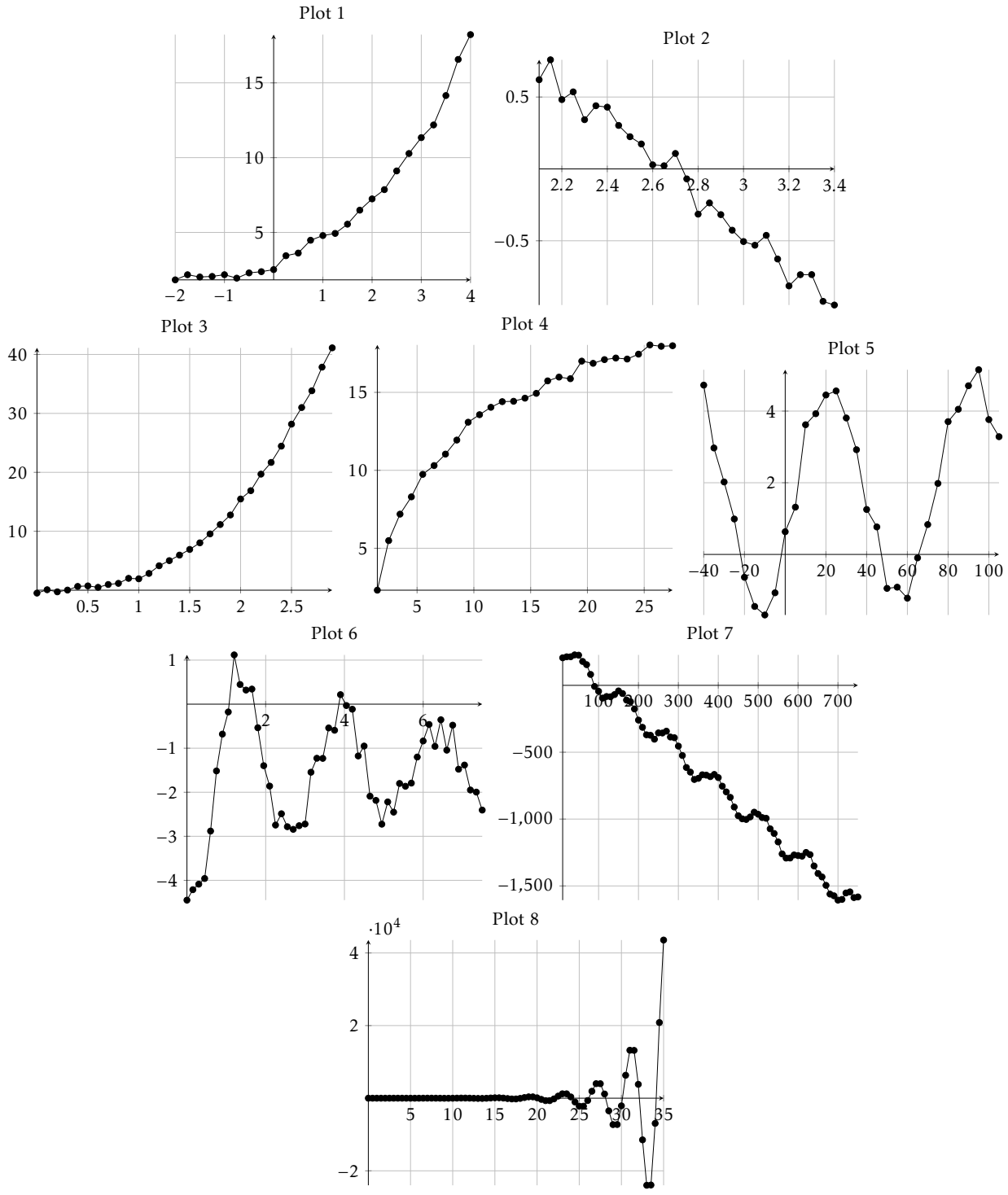


Figure 3.8. Plots for Problem 3.65.

Chapter 4

Numerical Linear Algebra

> "Linear algebra is the central subject of mathematics. You cannot learn too much linear algebra."
> –Benedict Gross, Harvard University

The preceding quote says it all – linear algebra is the most important of all of the mathematical tools that you can learn and build. The theorems, proofs, conjectures, and big ideas in almost every other mathematical field find their roots in linear algebra. Our goal in this chapter is to explore numerical algorithms for the primary questions of linear algebra: solving systems of equations, approximating solutions to over-determined and under-determined systems of equations, the eigenvalue-eigenvector problem, and the singular value problem. Take careful note, that in our current digital age numerical linear algebra and its fast algorithms are behind the scenes for wide varieties of computing applications.

4.1 Matrix Operations

We start this chapter with the basics: the dot product and matrix multiplication. MATLAB is designed to do these tasks in very efficient ways but it is a good coding exercise to build your own dot product and matrix multiplication routines. You'll find in numerical linear algebra that the indexing and the housekeeping in the codes is the hardest part, so why don't we start "easy".

Problem 4.1. Recall that the dot product of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ is

$$\mathbf{u} \cdot \mathbf{v} = \sum_{j=1}^{n} u_j v_j.$$

Write a MATLAB function that accepts two vectors and returns the dot product.

```
function DotProduct = MyDotProduct(u, v)
```

It would be wise to put an error check in your code to make sure that the vectors are the same size. You should be able to write this code without any loops. ▲

Problem 4.2. Write a test script for the dot product of two random $n \times 1$ vectors. Your script should output the absolute error between your dot product code and MATLAB's `dot` command. ▲

Problem 4.3. Recall that if $A \in \mathbb{R}^{n\times p}$ and $B \in \mathbb{R}^{p\times m}$ then the product AB is defined as

$$(AB)_{ij} = \sum_{k=1}^{p} A_{ik}B_{kj}.$$

A moments reflection reveals that each entry in the matrix product is actually a dot product,

$$(AB)_{ij} = (\text{Row } i \text{ of matrix } A) \cdot (\text{Column } j \text{ of matirx } B).$$

Write a MATLAB function that accepts two matrices and returns the matrix product.

```
function MatrixProduct = MatrixMultiply(A, B)
```

You should be able to write this code with only two loops; one for i and one for j. The rest of the work should be done using dot products. ▲

Problem 4.4. Write a test script for the matrix product of two random matrices of appropriate sizes. Your script should output the normed error between your matrix product code and MATLAB's matrix multiplication. ▲

If you're having trouble seeing that matrix multiplication is just a bunch of dot products then let's examine a fairly simple example. You are welcome to use this example to test your code in the previous problem, but be sure that your code is robust enough to accept matrices of any size.

Example 4.5. Find the product of matrices A and B using dot products.

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \qquad B = \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

Solution:
The product AB will clearly be a 3×3 matrix since $A \in \mathbb{R}^{3\times 2}$ and $B \in \mathbb{R}^{2\times 3}$. To build the matrix product we'll first write matrix A as a matrix filled with row vectors and matrix B as a matrix filled with column vectors. Let $\mathbf{a}_1 = \begin{pmatrix} 1 & 2 \end{pmatrix}$, $\mathbf{a}_2 = \begin{pmatrix} 3 & 4 \end{pmatrix}$, and $\mathbf{a}_3 = \begin{pmatrix} 5 & 6 \end{pmatrix}$ so that we can write A as

$$A = \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix}.$$

Similarly, write $\mathbf{b}_1 = \begin{pmatrix} 7 \\ 10 \end{pmatrix}$, $\mathbf{b}_2 = \begin{pmatrix} 8 \\ 11 \end{pmatrix}$, and $\mathbf{b}_3 = \begin{pmatrix} 9 \\ 12 \end{pmatrix}$ so that we can write B as

$$B = \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 \end{pmatrix}.$$

Now to build the matrix multiplication we see that AB is given as

$$AB = \begin{pmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \mathbf{a}_1 \cdot \mathbf{b}_2 & \mathbf{a}_1 \cdot \mathbf{b}_3 \\ \mathbf{a}_2 \cdot \mathbf{b}_1 & \mathbf{a}_2 \cdot \mathbf{b}_2 & \mathbf{a}_2 \cdot \mathbf{b}_3 \\ \mathbf{a}_3 \cdot \mathbf{b}_1 & \mathbf{a}_3 \cdot \mathbf{b}_2 & \mathbf{a}_3 \cdot \mathbf{b}_3 \end{pmatrix}$$

Therefore,

$$AB = \begin{pmatrix} 27 & 30 & 33 \\ 61 & 68 & 75 \\ 95 & 106 & 117 \end{pmatrix}$$

The following MATLAB code gives the matrix product between two matrices, but when you build your own code you need to have sufficient catches that avoid inappropriately sized matrices.

```
for i = 1:size(A,1) % row index
    for j = 1:size(B,2) % column index
        AB(i,j) = dot( A(i,:) , B(:,j) );
    end
end
```

Problem 4.6. In matrix arithmetic there are two types of multiplication: matrix-matrix multiplication and scalar multiplication. Modify your MATLAB code written in problem 4.3 so that if one of the two matrices is entered as a scalar (a 1×1 matrix) then your `MatrixMultiply` code gives the correct result. You should be able to do this with no loops. ▲

4.2 Solving Systems of Linear Equations

One of the many classic problems of linear algebra is to solve the linear system $A\mathbf{x} = \mathbf{b}$. In this chapter we will talk about efficient ways to have the computer solve these systems. You likely recall row reduction (AKA Gaussian Elimination or RREF) from previous linear algebra courses, but the algorithm that you used is actually slow an cumbersome for computer implementation. Even so, let's blow the dust off of what you recall with a small practice problem.

Problem 4.7. Solve the following problem by hand using Gaussian Elimination (row reduction).

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$$

Hint: Start by augmenting the coefficient matrix and the right-hand column vector to build the augmented system

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 0 & 2 \end{array}\right).$$

Then perform row operations to get to the reduced row echelon form

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

▲

4.2.1 Upper and Lower Triangular Systems

Row reduction works well on dense (or nearly dense) matrices but if the coefficient matrix has special structure then we can avoid row reduction in lieu of faster algorithms. Furthermore, the process that you know as Gaussian Elimination is really just a collection of sneaky matrix operations. In the following two problems you will devise algorithms for triangular matrices. After we know how to work with triangular matrices we'll build a general tool for doing Gaussian Elimination that is easily implemented in a computer.

Problem 4.8. Outline a fast algorithm (without formal row reduction) for solving the lower triangular system

$$\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}.$$

As a convention we will always write our lower triangular matrices with ones on the main diagonal. Your outline should be a list of explicit steps to solve the system. The most natural algorithm that most people devise here is called *forward substitution*. ▲

Technique 4.9 (Forward Substitution: `LSolve`)**.** The following code solves the problem $L\mathbf{y} = \mathbf{b}$ using forward substitution. The matrix L is assumed the be lower triangular with ones on the main diagonal.

```
function y = LSolve(L , b)
n = length(b);
y = zeros(n,1);
for i = 1:n
    y(i) = b(i);
    for j = 1 : (i-1)
        y(i) = y(i) - L(i,j) * y(j);
    end
end
```

Problem 4.10. Consider the lower triangular system

$$\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}.$$

Work the code from Technique 4.9 by hand to solve the system. Keep track of all of the indices as you work through the code. ▲

Problem 4.11. Copy the code from Technique 4.9 into a MATLAB function but in your code write a comment on every line stating what it is doing. Write a test script that creates a lower triangular matrix of the correct form and a right-hand side $\mathbf{b}$ and solve for $\mathbf{y}$. Your code needs to work on systems of arbitrarily large size. ▲

Now that we have a method for solving lower triangular systems, let's build a similar method for solving upper triangular systems. The merging of lower and upper triangular systems will play an important role in solving systems of equations.

Problem 4.12. Outline a fast algorithm (without formal row reduction) for solving the upper triangular system

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & -9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -4 \\ 3 \end{pmatrix}$$

The most natural algorithm that most people devise here is called *backward substitution*. Notice that in our upper triangular matrix we do not have a diagonal containing all ones. ▲

Technique 4.13 (Backward Substitution: `USolve`)**.** The following code solves the problem $U\mathbf{x} = \mathbf{y}$ using backward substitution. The matrix U is assumed the be upper triangular. You'll notice that most of this code is incomplete. It is your job to complete this code.

```
function x = USolve(U , y)
n = length(y);
x = zeros(n,1);
for i = ? : ? : ?         % what should you be looping over?
    x(i) = y(i) / ???;    % what should you be dividing by?
    for j = ? : ? : ?     % what should you be looping over?
        x(i) = x(i) - U(i,j) * x(j) / ???;  % what should you be dividing by?
    end
end
```

Problem 4.14. Consider the upper triangular system

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & -9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -4 \\ 3 \end{pmatrix}$$

Work the code from Technique 4.13 by hand to solve the system. Keep track of all of the indices as you work through the code. You may want to work this problem in conjunction with the previous two problems to unpack all of the parts of the *backward substitution* algorithm. ▲

Problem 4.15. Copy the code from Technique 4.13 into a MATLAB function but in your code write a comment on every line stating what it is doing. Write a test script that creates an upper triangular matrix of the correct form and a right-hand side **y** and solve for **x**. Your code needs to work on systems of arbitrarily large size. ▲

4.2.2 The LU Factorization

In the next few problems we will solve the system of equations

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$$

using upper and lower triangular matrices. We have already solved this problem with your Gaussian Elimination algorithm – now let's improve upon that algorithm and reveal some amazing underlying structure.

Throughout the following several problems, let A and b be defined as

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}.$$

Problem 4.16. Open a new MATLAB script and enter the matrix A. Observe what happens when we do the following in sequence:

- left multiply A by the matrix

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

  This gives the matrix L_1A. (Look at the resulting matrix. What happened here?)

- left multiply L_1A by the matrix

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -7 & 0 & 1 \end{pmatrix}$$

  This gives the matrix L_2L_1A. (Look at the resulting matrix. What happened here?)

- left multiply L_2L_1A by the matrix

$$L_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}$$

  (Look at the resulting matrix. What happened here?)

▲

Problem 4.17. Make a conjecture: If you wanted to multiply row j of an $n \times n$ matrix by c and add it to row k, that is the same as multiplying by what lower triangular matrix? ▲

Problem 4.18. After the process from the previous problem you should notice that you now have an upper triangular matrix. Hence, in general, we have done this:

$$L_3L_2L_1A = U,$$

so if you solve for A we see that A can be written as

$$A = L_1^{-1}L_2^{-1}L_3^{-1}U.$$

Therefore, we could rewrite the problem $A\mathbf{x} = \mathbf{b}$ as the problem $LU\mathbf{x} = \mathbf{b}$ where L is which matrix? ▲

Problem 4.19. In the previous problem you likely found that $L = L_1^{-1}L_2^{-1}L_3^{-1}$. Use MATLAB to find L_j^{-1} for each j and discuss general observations about how to find inverses of lower triangular matrices. ▲

Problem 4.20. Now for the punch line: If we want to solve $A\mathbf{x} = \mathbf{b}$ then if we can write it as $LU\mathbf{x} = \mathbf{b}$ we can

1. Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ using forward substitution. Then,
2. solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ using backward substitution.

For our running example, write down L and U and solve the system. ▲

Problem 4.21. Try the process again on the 3×3 system of equations

$$\begin{pmatrix} 3 & 6 & 8 \\ 2 & 7 & -1 \\ 5 & 2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -13 \\ 4 \\ 1 \end{pmatrix}$$

That is: Find matrices L and U such that $A\mathbf{x} = \mathbf{b}$ can be written as $LU\mathbf{x} = \mathbf{b}$. Then do two triangular solves to determine $\mathbf{x}$.
Notice that this time there isn't a "1" in the top left corner to begin with. Be careful. ▲

Technique 4.22 (LU Factorization)**.** The following MATLAB function takes a square matrix A and outputs the matrices L and U such that $A = LU$. Partial code is given below. Complete the code.

```
function [L,U] = MyLU(A)
n = size(A,1);       % finds the size of the matrix
if size(A,1) ~= size(A,2)   % what does this do?
    fprintf('Error: The matrix A is not square\n')
    break
end
L = eye(n,n);        % initialize L as an identity matrix (why?)
U = A;               % initialize U as A (why?)
for j = 1 : (n-1)    % loop over the columns
    for i = (j+1) : n   % loop over the rows
        mult = A(i,j) / A(j,j); % what does this line do?
        A(i, j+1:n) = A(i, j+1:n) - mult*A(j, j+1:n);
        U(i, j+1:n) = ???   % what part of A should you be putting in U here?
        L(i,j) = ???        % what should go in the lower triangular matrix?
        U(i,j) = 0;         % zero out the bottom portion of U (why?)
    end
end
```

Theorem 4.23 (LU Factorization Algorithm)**.** Let A be a square matrix in $\mathbb{R}^{n\times n}$ and let $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. To solve the problem $A\mathbf{x} = \mathbf{b}$,

1. Factor A into lower and upper triangular matrices $A = LU$.
   `[L, U] = MyLU(A);`
2. The system can now be written as $LU\mathbf{x} = \mathbf{b}$. Substitute $U\mathbf{x} = \mathbf{y}$ and solve the system $L\mathbf{y} = \mathbf{b}$ with forward substitution.
   `y = LSolve(L, b);`
3. Finally, solve the system $U\mathbf{x} = \mathbf{y}$ with backward substitution.

```
    x = USolve(U, y);
```

Problem 4.24. Test your `MyLU`, `LSolve`, and `USolve` functions on a linear system for which you know the answer. Then test your problem on a system that you don't know the solution to. Discuss where your code will fail. Use the following partial code to test your functions.

```
A = [...; ...; ... ];
b=  [...; ...; ... ];
[L,U] = MyLU(A);
y = Lsolve(L,b);
x = Usolve(U,y)
MATLABExactAnswer = A\b
MyError = norm(x-MATLABExactAnswer)
```

▲

Problem 4.25. For this problem we are going to run a numerical experiment to see how the process of solving the equation $A\mathbf{x} = \mathbf{b}$ using the LU factorization performs.
Create a loop that does the following

- Build a random matrix of size $n \times n$. You can do this with the code:
  `A=rand(n,n)`
- Build a random vector in $\mathbb{R}^n$. You can do this with the code:
  `b = rand(n,1)`
- Find MATLAB's exact answer to the problem $A\mathbf{x} = \mathbf{b}$ using the backslash:
  `Xexact = A \ b`
- Write code that uses your three LU functions (`MyLU, Lsolve, Usolve`) to find a solution to the equation $A\mathbf{x} = \mathbf{b}$.
- Find the error between your answer and the exact answer using the code:
  `error = norm(x - Xexact)`
- Make a plot that shows how the error behaves as the size of the problem changes. You should run this for matrices of larger and larger size but be warned that the loop will run for quite a long time if you go above 300×300 matrices. Just be patient.

▲

Problem 4.26. Write a summary of how the `LU` factorization solve a square system of equations. ▲

4.3 The Least Squares Problems

Problem 4.27. Grab the data sets from the Google Sheet HERE. Copy the data into MS Excel for the following exercise. We're going to develop a way to match the data sets using linear algebra. Before doing the linear algebra versions of this problem though we'll use Excel to match the data sets. Our goal is to make a guess at the type of polynomial that models the function (given in this case) and to minimize the error between our guess and the data. Follow these steps in Excel to find best fitting curves. We'll start with the linear data.

1. In column C set up a function for your guess of the model based on the x data in column A. You will need to set up cells for the polynomial parameters (for the linear case these are the slope and the y-intercept).
2. Fill in your guesses based on initial approximations for the slope and y-intercept.
3. Use column D to calculate the residual value for each data point.

$$\text{Residual} = \text{Actual } y \text{ Value} - \text{Approximate } y \text{ Value}$$

4. Use column E to calculate the square of the residual for each data point.
5. Our goal is the minimize the sum of the squares of the residuals.

$$\min\left(\sum_{j=1}^{n}\left(y_j - \hat{y}_j\right)^2\right)$$

For this we can use the `Excel Solver` to minimize the sum of the square residuals.

Now repeat the process for the quadratic data. ▲

At this point we need to discuss the quality of the fits that we are building. You are likely familiar with the R and R^2 values from statistics, but let's just recap here.

Definition 4.28 (Correlation Coefficient, R)**.** The **correlation coefficient**, R, is a measure of the quality of a linear fit.

- If $R = +1$ then the data represent a perfect linear fit with a positive slope.
- If $R = -1$ then the data represent a perfect linear fit with a negative slope.
- If $R = 0$ then there is no linear relationship between the two variables.

Definition 4.29 (Coefficient of Determination, R^2)**.** The **coefficient of determination**, R^2, is the proportion of variance in the dependent (y) variable that is explained by the independent (x) variable.

- If $R^2 = 1$ then 100% of the variance in y is explained by x and the fit is perfect.

- If $R^2 = 0$ then 0% of the variance in y is explain by x and there is no correlation between the two variables.

In the problems that we've studied thus far we have calculated the sum of the squares of the residuals as a measure of how well our function fits the data. The trouble with the sum of the squares of the residuals is that it is context dependent. Hence, there is no way to tell at the outset what a *good* sum of squares of residuals is.

Theorem 4.30. To calculate the R^2 value we can use the following equations where y_j is the j^{th} data point, $\bar{y}$ is the mean y-value in the data, and $\hat{y}_j$ is the j^{th} predicted output from our model.

$$\text{Total Sum of Squares: } TSS = \sum_j \left(y_j - \bar{y}\right)^2 \tag{4.1}$$

$$\text{Residual Sum of Squares: } RSS = \sum_j \left(y_j - \hat{y}_j\right)^2 \tag{4.2}$$

$$\text{Coeff. of Determination: } R^2 = 1 - \frac{RSS}{TSS} \tag{4.3}$$

Problem 4.31. Compute the R^2 value for each of the data sets from Problem 4.27. ▲

Problem 4.32. Use equation (4.3) to explain why $R^2 = 1$ implies a perfect fit. ▲

There are MANY statistics packages out there that will do least squares regression for us. MATLAB has the `polyfit` command, R has the `lm` command, and Excel has the data analysis toolpack. Behind the scenes in all of these is actually linear algebra. It is informative pedagogically to do the least squares problems using Excel a few times (as we did in Problem 4.27), but in reality there is some beautifully simple linear algebra behind the scenes.

Problem 4.33. Now we'll use Linear Algebra to complete the same types of problems. Set up a MATLAB script the reads the linear data from the Excel sheet. We are assuming that the data are linear so for each x-value x_j we assume that the equation

$$\hat{y}_j = \beta_0 + \beta_1 x_j \approx y_j.$$

The values of β_0 and β_1 are the intercept and slope that best predict the y values from the data.

The equation $\beta_0 + \beta_1 x_j \approx y_j$ gives rise to a system of linear equations

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \approx \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}$$

when we consider all n of the data points. We'll call the left-hand matrix A and the right-hand vector $\mathbf{y}$.

1. For each data set get A and $\mathbf{y}$ into MATLAB so we can use them.

2. This is not a square system. In fact, it is *overdetermined*. Consider that the column space of A ($Col(A)$) is a subspace of $\mathbb{R}^n$ and that $\mathbf{y} \in \mathbb{R}^n$ is likely not in the column space of A. If we project $\mathbf{y}$ onto $Col(A)$ we should be able to find the best approximation of $\mathbf{y}$ that lies in $Col(A)$.

   Projections onto a subspace can be achieved with matrix multiplication, but which matrix shall we multiply by ...

   $$(??)A\begin{pmatrix}\beta_0\\ \beta_1\end{pmatrix} = (??)\mathbf{y}.$$

   Once you are satisfied that the right-hand side is a projection of $\mathbf{y}$ onto $Col(A)$ you have formed the **normal equations**. If you've done everthing right then you also now have a square system (2×2 in the case of the linear data).

3. Now that this is a square matrix you can solve for β_0 and β_1 using your LU code. (You can also use `MATLAB`'s *backslash* (\) command to do the linear solve)

4. Now that you have β_0 and β_1 you can form the linear approximation. Plot the approximation along with your data. Also write code to plot the residuals (subplot would be great here).

5. Repeat the process for the quadratic. For the quadratic data you are assuming that

   $$\beta_0 + \beta_1 x_j + \beta_2 x_j^2 \approx y_j.$$

▲

Definition 4.34 (The Normal Equations)**.** Let $A \in \mathbb{R}^{n\times m}$, $\beta \in \mathbb{R}^m$, and $\mathbf{y} \in \mathbb{R}^n$ where $n \gg m$. The system of equations

$$A\beta = \mathbf{y}$$

is over determine since there are more equations than unknowns. Multiply both sides of the equation by A^T yields the **normal equations**

$$A^T A\beta = A^T\mathbf{y}.$$

We note here that $A^T A \in \mathbb{R}^{m\times m}$ is square and much smaller than A. The right-hand side of the normal equations is the projection of $\mathbf{y}$ onto the columns space of A.

You should take careful note of something here. The process that we just formulated is called *linear regression* even though we were fitting a quadratic function to data. This may, at first, seem like an unfortunate or inappropriate naming convention, but stop and think more carefully about what we did. ...Good. Now that you've though about it, I'll give you my take.

For the quadratic data fit we are trying to create the function $\hat{y}_j = \beta_0 + \beta_1 x_j + \beta_2 x_j^2$. Notice that this function is really just a multiple regression that is, indeed, linear in the coefficients β_0, β_1, and β_2. The term *linear regression* does not have anything to do fitting lines to data – a common misconception. Instead, it pertains to the relationship between the coefficients. Since the coefficients of a polynomial function are linearly related to each other, we can use linear regression to fit polynomial models for any order polynomial.

The reader should be further warned that polynomial regression comes with some down sides. If the order of the polynomial grows the matrix $A^T A$ arising from the normal equations formulation gets closer and closer to being singular. That is to say that as the order of the polynomial increases it becomes less and less desirable to solve the problem through matrix inversion (or Gaussian Elimination).

Problem 4.35. Create a data set from a third or fourth order polynomial. Then introduce some random noise onto your data set. Write a loop in MATLAB that fits polynomials of increasing order to the data set (starting with linear and increasing by 1 each time). Create two plots. In the first plot put the order of the polynomial on the horizontal axis and the R^2 value on the vertical axis. On the second plot put the order of the polynomial on the horizontal axis and the ratio $|\lambda_{max}|/|\lambda_{min}|$ on the vertical axis, where λ_{max} and λ_{min} are the maximum and minimum eigenvalues respectively. You'll probably want to use a `semilogy` scale for the second plot.
Note: Recall that the determinant of a matrix is the product of the eigenvalues. Therefore, if the determinant is getting close to zero then the ratio given above goes to infinity. To get the eigenvalues use MATLAB's `eig` command. ▲

Problem 4.36. Explain what the previous problem tells you about fitting polynomials to data. ▲

4.4 The QR Factorization

Our goal in this section is to improve the efficiency of solving the least squares problems via the normal equations. Recall that we want to find $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$ but where $\mathbf{b}$ is not in the column space of A. This necessitated the use of the normal equations $A^TA\mathbf{x} = A^T\mathbf{b}$ to project $\mathbf{b}$ onto the column space of A. This would be FAR more efficient if the columns of A were orthogonal (perpendicular) and normalized (unit vectors). Hence, our goal is to take A and factor it into $A = QR$ where the columns of Q are orthonormal (orthogonal and normalized) and R is an upper triangular matrix.

Problem 4.37. If $Q \in \mathbb{R}^{m\times n}$ is an orthonormal matrix then what are the products Q^TQ and QQ^T? ▲

Problem 4.38. If $A = QR$ where Q is an orthonormal matrix and R is upper triangular then how would we go about solving the equation $A\mathbf{x} = \mathbf{b}$ and where would the QR factorization help us? ▲

Problem 4.39. First we'll do a problem by hand:
Consider the matrix $A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$. We want to factor A into $A = QR$ where the columns of Q are orthonormal and R is upper triangular. Here is the algorithm (run every step by hand!). For notation purposes, $\mathbf{a}_j$ will be the j^{th} column of A.

1. Define $\mathbf{q}_1 = \dfrac{\mathbf{a}_1}{\|\mathbf{a}_1\|}$. This will be the first column of Q.

2. Define $\mathbf{q}_2 = \mathbf{a}_2 - (\mathbf{a}_2 \cdot \mathbf{q}_1)\mathbf{q}_1$. Once you've done this calculation normalize your result so $\mathbf{q}_2 = \dfrac{\mathbf{q}_2}{\|\mathbf{q}_2\|}$. This is the second column of Q.
   Explain the geometry of this step (DRAW A PICTURE!)

3. Define $\mathbf{q}_3 = \mathbf{a}_3 - (\mathbf{a}_3 \cdot \mathbf{q}_1)\mathbf{q}_1 - (\mathbf{a}_3 \cdot \mathbf{q}_2)\mathbf{q}_2$ and then redefine $\mathbf{q}_3 = \dfrac{\mathbf{q}_3}{\|\mathbf{q}_3\|}$. This is now the third column of Q.

4. The matrix R is formed as follows:

$$R = \begin{pmatrix} \mathbf{a}_1 \cdot \mathbf{q}_1 & \mathbf{a}_2 \cdot \mathbf{q}_1 & \mathbf{a}_3 \cdot \mathbf{q}_1 \\ 0 & \mathbf{a}_2 \cdot \mathbf{q}_2 & \mathbf{a}_3 \cdot \mathbf{q}_2 \\ 0 & 0 & \mathbf{a}_3 \cdot \mathbf{q}_3 \end{pmatrix}$$

5. Write down Q and observe that the process is just begging for several loops on a computer to implement this on bigger matrices.

6. Use MATLAB to check that $A = QR$.

7. Finally let's look at the utility of the result:

   (a) Since Q is an orthonormal matrix Q^TQ is the identity matrix! (Explain why.)

   (b) If we want to solve $A\mathbf{x} = \mathbf{b}$ and we can write $A = QR$ then

   $$A\mathbf{x} = \mathbf{b} \quad \Longrightarrow \quad QR\mathbf{x} = \mathbf{b} \quad \Longrightarrow \quad R\mathbf{x} = Q^T\mathbf{b}$$

   (c) Since R is upper triangular we can use the `Usolve` code we have from our work with the LU-factorization to solve for $\mathbf{x}$. THIS IS REALLY FAST!!

   (d) Compare this to the number of computer operations needed so solve the normal equations $A^TA\mathbf{x} = A^T\mathbf{b}$ with an LU-solver.

▲

Problem 4.40. Write MATLAB code that takes a matrix A (not necessarily square) and outputs both the Q matrix and the R matrix. Pseducode for this function is as follows:

- Set up the function call:
  ```
  function [Q,R] = MyQR(A)
  ```
- Set up `zeros` matrices for both Q and R. Remember that A may not be square so assume that the columns are in $\mathbb{R}^m$ and the rows are in $\mathbb{R}^n$. Hence Q will be $m \times n$ and R will be $n \times n$.
- Start a loop that counts across the columns:
  ```
  for j=1:n
  ```
  - define a temporary variable: $\hat{\mathbf{q}} = \mathbf{a}_j$
  - start a loop that will do all of the subtractions and build some of the R's:
    ```
    for i=1:j-1
    ```
    * build one of the R's: $R_{ij} = \mathbf{a}_j \cdot \mathbf{q}_i$
    * Do one of the subtractions: $\hat{\mathbf{q}} = \hat{\mathbf{q}} - R_{ij}\mathbf{q}_i$
  - end the loop for `i`
  - normalize the j^{th} column in Q: $\mathbf{q}_j = \hat{\mathbf{q}}/\|\hat{\mathbf{q}}\|$
  - build the R's on the diagonal: $R_{jj} = \mathbf{a}_j \cdot \mathbf{q}_j$
- end the loop for `j`

▲

Problem 4.41. Write a script that tests your `MyQR` function on randomly generated $m \times n$ matrices with randomly generated right-hand sides. Compare the time that it takes to solve the least squares problems using QR to the time necessary to solve with LU via the normal equations. When is LU more efficient? When is QR more efficient? ▲

4.5 Interpolation

The least squares problem that we studied in the previous sections seeks to find a best fitting function that is *closest* (in the 2-norm sense) to a set of data. What if, instead, we want to match the data points exactly with a function. This is the realm of interpolation. Take note that there are many many forms of interpolation that are tailored to specific problems. In this brief section we cover only a few of the simplest forms of interpolation involving only polynomial functions. The problem that we'll focus on can be phrased as: Given a set of $n+1$ data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, find a polynomial of degree at most n that exactly fits these points.

4.5.1 Vandemonde Interpolation

We have technically already seen Vandermonde interpolation. In Section 4.3 we built a system of equations to solve the least squares problem. In the least squares problem we had more data points than unknown parameters resulting in an over-determined system (look back to Section 4.3 to remind yourself). If, however, we choose a polynomial model that has the same number of unknown parameters as data points then the resulting system if not over-determined.

For example, let's say that we have the data set

$$S = \{(0,1), (1,2), (2,5), (3,10)\}$$

and we want to fit a polynomial then we can use a cubic function (which has 4 parameters) to match the data perfectly. Indeed, if we choose $p(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$ then the resulting system of equations is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 5 \\ 10 \end{pmatrix}.$$

Notice that the system of equations is square, and solving using any method discussed in this chapter results in $\beta_0 = 1$, $\beta_1 = 0$, $\beta_2 = 1$, and $\beta_3 = 0$. Hence, the interpolating function is $p(x) = 1 + 0x + 1x^2 + 0x^3 = 1 + x^2$, and we know that $p(x)$ matches this data set perfectly as seen in Figure 4.1.

Problem 4.42. Write a MATLAB function that accepts a list of ordered pairs (where each x value is unique) and builds a Vandermonde interpolation polynomial. Test your function on the simple example listed above and then on several larger problems. It may be simplest to initially test on functions that we know. ▲

Problem 4.43. Build a Vandermonde interpolation polynomial to interpolate the function $f(x) = \cos(2\pi x)$ with 10 points that are linearly spaced on the interval $x \in [0, 2]$. ▲

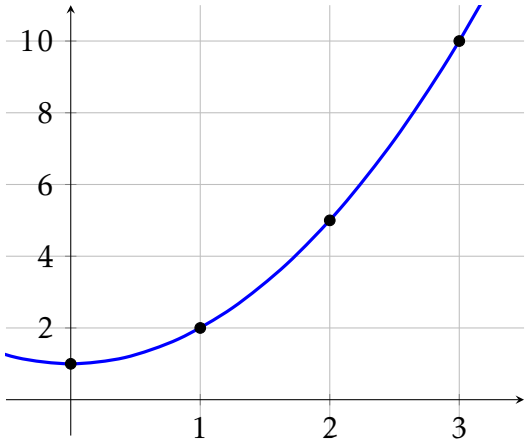


Figure 4.1. A simple Vandermonde interpolation for the data set $S = \{(0,1),(1,2),(2,5),(3,10)\}$ resulting in the interpolating function $p(x) = 1 + x^2$.

Definition 4.44 (The Vandermonde Matrix)**.** Let $S = \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ be a list of ordered pairs where the x values are all unique. Using Vandermonde interpolation we arrive at the system of equations

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}. \tag{4.4}$$

The matrix on the left-hand side of (4.4) is called the **Vandermonde Matrix**.

Problem 4.45. Vandermonde matrix is relatively easy to conceptualize and code, but there is an inherent problem. Use your code from Problem 4.42 to create a plot on a `semilogy` scale. The horizontal axis of the plot is the order of the interpolating polynomial and the vertical axis is the ratio $|\lambda_{max}|/|\lambda_{min}|$ where λ_{max} and λ_{min} are the maximum and minimum eigenvalues of the Vandermonde matrix respectively. What does this plot tell you about Vandermonde interpolation for high-order polynomials? ▲

4.5.2 Lagrange Interpolation

Lagrange interpolation is a rather clever interpolation scheme where we build up the polynomial from simpler polynomials. For interpolation we want to build a polynomial $p(x)$ such that $p(x_j) = y_j$. If we can find a polynomial $\phi_j(x)$ that

$$\phi_j(x) = \begin{cases} 0, & \text{if } x = x_i \text{ and } i \neq j \\ 1, & \text{if } x = x_j \end{cases}$$

then for Lagrange interpolation we build $p(x)$ as a linear combination of the ϕ_j functions.

Problem 4.46. Consider the data set $S = \{(0,1), (1,2), (2,5), (3,10)\}$.

(a) Based on the descriptions of the $p(x)$ and $\phi_j(x)$ functions, why would $p(x)$ be defined as

$$p(x) = 1\phi_0(x) + 2\phi_1(x) + 5\phi_2(x) + 10\phi_3(x)?$$

(b) Verify that $\phi_0(x)$ can be defined as

$$\phi_0(x) = \frac{(x-1)(x-2)(x-3)}{(0-1)(0-2)(0-3)}.$$

(c) Verify that $\phi_1(x)$ can be defined as

$$\phi_1(x) = \frac{(x-0)(x-2)(x-3)}{(1-0)(1-2)(1-3)}.$$

(d) Define $\phi_2(x)$ and $\phi_3(x)$ in a similar way.

(e) Build the linear combination from part (a) and create a plot showing that this polynomial indeed interpolates the points in the set S.

▲

Technique 4.47 (Lagrange Interpolation)**.** To build an interpolating polynomial $p(x)$ for the set of points $\{(x_0,y_0), (x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$ we first build the polynomials $\phi_j(x)$ for each $j = 0,1,2,\ldots,n$ and then construct the polynomial $p(x)$ as

$$p(x) = \sum_{j=0}^{n} y_j \phi_j(x).$$

The $\phi_j(x)$ functions are defined as

$$\phi_j(x) = \prod_{i \neq j} \frac{x - x_i}{x_j - x_i}.$$

Example 4.48. Build a Lagrange interpolation polynomial for the set of points

$$S = \{(1,5), (2,9), (3,11)\}.$$

Solution

We first build the three ϕ_j functions.

$$\phi_0(x) = \frac{(x-2)(x-3)}{(1-2)(1-3)}$$
$$\phi_1(x) = \frac{(x-1)(x-3)}{(2-1)(2-3)}$$
$$\phi_2(x) = \frac{(x-1)(x-2)}{(3-1)(3-2)}.$$

Take careful note that the ϕ functions are built in a very particular way. Indeed, $\phi_0(1) = 1$, $\phi_0(2) = 0$, and $\phi_0(3) = 0$. Also, $\phi_1(1) = 0$, $\phi_1(2) = 1$), and $\phi_1(3) = 0$. Finally, note that $\phi_2(1) = 0$, $\phi_2(1) = 0$ and $\phi_2(3) = 1$. Thus, the polynomial $p(x)$ can be built as

$$p(x) = 5\phi_0(x) + 9\phi_1(x) + 11\phi(2(x) = 5\frac{(x-2)(x-3)}{(1-2)(1-3)} + \frac{(x-1)(x-3)}{(2-1)(2-3)} + \frac{(x-1)(x-2)}{(3-1)(3-2)}.$$

The remainder of the simplification is left to the reader.

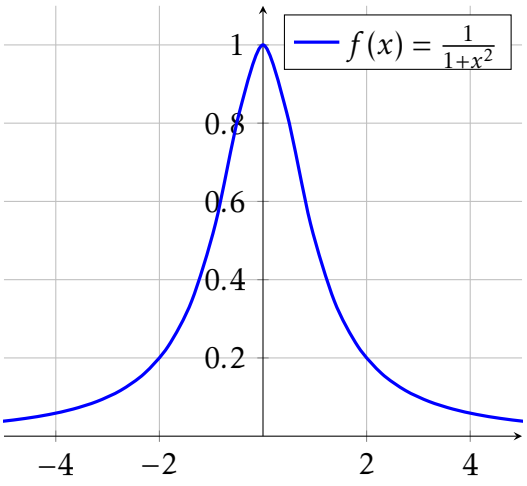
Problem 4.49. Write a MATLAB function that accepts a list of list of ordered pairs (where each x value is unique) and builds a Lagrange interpolation polynomial. Test your function on the examples that we've presented in this section. ▲

4.5.3 Interpolation at Chebyshev Points

Problem 4.50. Using either Vandermonde or Lagrange interpolation build a polynomial that interpolates the function

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

for $x \in [-5, 5]$ with polynomials of order $n = 2, 3, \ldots$ and linearly spaced interpolation points. What do you notice about the quality of the interpolating polynomial near the endpoints?



▲

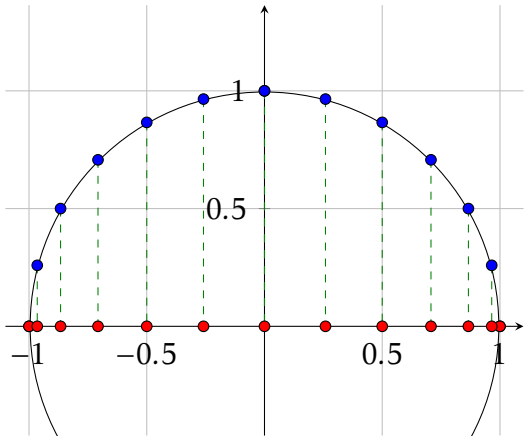


Figure 4.2. Chebyshev interpolation nodes for the interval $[-1,1]$. In this case each node is separated by $\pi/8$ radians giving 13 interpolations points including the endpoints.

As you should have noticed the quality of the interpolation gets rather terrible near the endpoints when you use linearly spaced points for the interpolation. A fix to this was first proposed by the Russian mathematician Pafnuty Chebyshev (1821-1894). The idea is as follows:

- Draw a semicircle above the closed interval on which you are interpolating (shown in black in Figure 4.2).
- Pick n equally spaced points along the semicircle (i.e. same arc length between each point). (shown in blue in Figure 4.2)
- Project the points on the semicircle down to the interval. Use these projected points for the interpolation. (shown in red in Figure 4.2)

It should be clear that since we are projecting down to the x-axis from a circle then all we need are the cosine values from the circle. Hence we can form the Chebyshev interpolation points from the formula

$$x_j = \cos\left(\frac{\pi j}{n}\right), \quad \text{for} \quad j = 0, 1, \ldots, n \tag{4.5}$$

on the interval $[-1,1]$.

To transform the Chebyshev points from the interval $[-1,1]$ (found with (4.5)) to the interval $[a,b]$ we can apply a linear function which maps -1 to a and 1 to b:

$$x_j \leftarrow \left(\frac{b-a}{2}\right)\left(x_j + 1\right) + a$$

where the "x_j" on the left is on the interval $[a,b]$ and the "x_j" on the right is on the interval $[-1,1]$.

Problem 4.51. Consider the function $f(x) = \frac{1}{1+x^2}$ just as we did for the first problem in this subsection. Write MATLAB code that overlays an interpolation with linearly spaced points an interpolation with Chebyshev nodes. Give plots for polynomial of order $n = 2, 3, 4, \ldots$. Be sure to show the original function on your plots as well. ▲

Problem 4.52. Demonstrate that the Chebyshev interpolation nodes will improve the stability of the Vandermonde matrix over using linearly spaced nodes. ▲

4.6 The Eigenvalue-Eigenvector Problem

Recall that the eigenvectors, $\mathbf{x}$, and the eigenvalues, λ of a square matrix satisfy the equation $A\mathbf{x} = \lambda\mathbf{x}$. Geometrically, the eign-problem is the task of finding the special vectors $\mathbf{x}$ such that multiplication by the matrix A only produces a scalar multiple of $\mathbf{x}$. Thinking about matrix multiplication, this is rather peculiar since matrix-vector multiplication usually results in a scaling and a rotation of the vector. Therefore, in some sense the eigenvectors are the only special vectors which avoid geometric rotation under matrix multiplication. For a graphical exploration of this idea see: https://www.geogebra.org/m/JP2XZpzV.

Recall that to solve the eigen-problem for a square matrix A we complete the following steps:

1. First rearrange the definition of the eigenvalue-eigenvector pair to
$$(A\mathbf{x} - \lambda\mathbf{x}) = \mathbf{0}.$$
2. Next, factor the $\mathbf{x}$ on the right to get
$$(A - \lambda I)\mathbf{x} = \mathbf{0}.$$
3. Now observe that since $\mathbf{x} \neq 0$ the matrix $A - \lambda I$ must NOT have an inverse. Therefore,
$$\det(A - \lambda I) = 0.$$
4. Solve the equation $\det(A - \lambda I) = 0$ for all of the values of λ.
5. For each λ, find a solution to the equation $(A - \lambda I)\mathbf{x} = \mathbf{0}$. Note that there will be infinitely many solutions so you will need to make wise choices for the free variables.

Problem 4.53. Find the eigenvalues and eigenvectors of $A = \begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$ by hand. ▲

Problem 4.54. In the matrix $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ one of the eigenvalues is $\lambda_1 = 0$.

1. What does that tell us about the matrix A?
2. What is the eigenvector $\mathbf{v}_1$ associated with $\lambda_1 = 0$?
3. What is the null space of the matrix A?

▲

Problem 4.55. Find matrices P and D such that $A = \begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$ can be written as $A = PDP^{-1}$ where P is a dense 2×2 matrix and D is a diagonal matrix. Once you have this factorization of A, use it to determine A^{10}. ▲

Problem 4.56. Let A be an $n \times n$ matrix with n distinct eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ and let $\mathbf{x} \in \mathbb{R}^n$ be a vector such that $\mathbf{x} = \sum_{j=1}^{n} c_j \mathbf{v}_j$. Find expressions for $A\mathbf{x}$, $A^2\mathbf{x}$, $A^3\mathbf{x}$, ... ▲

Problem 4.57. In this problem we first describes the mathematical idea for the **power method** for computing the largest eigenvalue / eigenvector pair. Then we write an algorithm for find the largest eigen-pair numerically.

1. Assume that A has n linearly independent eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ and choose $\mathbf{x} = \sum_{j=1}^{n} c_j \mathbf{v}_j$. From the previous problem,

$$A^k \mathbf{x} = \_\_\_\_\_$$

2. Factor the right-hand side so that

$$A^k \mathbf{x} = \lambda_1^k \left(c_1 \mathbf{v}_1 + c_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k \mathbf{v}_2 + c_3 \left(\frac{\lambda_3}{\lambda_1} \right)^k \mathbf{v}_3 + \cdots + c_n \left(\frac{\lambda_n}{\lambda_1} \right)^k \mathbf{v}_n \right)$$

3. If $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n$ then what happens to each of the $(\lambda_j/\lambda_1)^k$ terms as $k \to \infty$? Using this answer, what is $\lim_{k\to\infty} A^k \mathbf{x}$?

▲

Technique 4.58 (The Power Method Algorithm)**.** This algorithm will quickly find the eigenvalue of largest absolute value for a square matrix $A \in \mathbb{R}^{n\times n}$ as well as the associated (normalized) eigenvector. We are assuming that there are n linearly independent eigenvectors of A.

Step #1: Given a nonzero vector $\mathbf{x}$, set $\mathbf{v}^{(1)} = \mathbf{x}/\|\mathbf{x}\|$. (Here the superscript indiates the iteration number)

Step #2: For $k = 2, 3, \ldots$

Step #2a: Compute $\tilde{\mathbf{v}}^{(k)} = A\mathbf{v}^{(k-1)}$ (this gives a non-normalized version of the next estimate of the dominant eigenvector.)

Step #2b: Set $\lambda^{(k)} = \left\langle \tilde{\mathbf{v}}^{(k)}, \mathbf{v}^{(k-1)} \right\rangle$. (this gives an approximation of the eigenvalue since if $\mathbf{v}^{(k-1)}$ was the actual eigenvector we would have $\lambda = \left\langle A\mathbf{v}^{(k-1)}, \mathbf{v}^{(k-1)} \right\rangle$)

Step #2c: Normalize $\tilde{\mathbf{v}}^{(k)}$ by computing $\mathbf{v}^{(k)} = \tilde{\mathbf{v}}^{(k)}/\|\tilde{\mathbf{v}}^{(k)}\|$. (This guarantees that you will be sending a unit vector into the next iteration of the loop)

Problem 4.59. Write a MATLAB function to implement the power method for finding the eigenvalue of largest absolute value and the associated eigenvector. Test it on a matrix where you know the eigenvalue of interest. ▲

4.7 The Singular Value Decomposition

Our overarching goal of this section is to discuss an analogue to the eigenvalue-eigenvector problem for non-square matrices. That is, we would like to take a matrix A that is $m \times n$ and find vectors are values that behave similarly to how eigenvectors and eigenvalues behave for square matrices. The key to this discussion is the matrix $A^T A$, so let's start there.

Problem 4.60. Let A be an $m \times n$ matrix. What is the size of $A^T A$? Prove that $A^T A$ must be a symmetric matrix (a matrix B is symmetric if $B_{ij} = B_{ji}$). Finally, what bearing does the next Theorem have on the matrix $A^T A$? ▲

Theorem 4.61. An $n \times n$ matrix A has n orthogonal eigenvectors if and only if A is a symmetric matrix.

Definition 4.62. The **singular values** of an $m \times n$ matrix A are the square roots of the eigenvalues of $A^T A$. They are typically denoted as $\sigma_1, \sigma_2, \ldots, \sigma_n$ where $\sigma_j = \sqrt{\lambda_j}$ and λ_j is an eigenvalue of $A^T A$.

Definition 4.63. The **singular value decomposition** of an $m \times n$ matrix A with rank r is a factorization of A into the product of three matrices, U, Σ, and V, such that

$$A = U\Sigma V^T.$$

In the singular value decomposition, U ($m \times m$) and V ($n \times n$) have orthogonal columns and Σ ($m \times n$) is a block diagonal matrix

$$\Sigma = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}$$

where D is an $r \times r$ diagonal matrix containing the r singular values of A in rank order (largest to smallest).

To build the singular value decomposition:

1. Form $A^T A$ and find the eigenvalues and eigenvectors (guaranteed to exist by Theorem 4.61).
2. Form Σ
3. The columns of V are the eigenvectors of $A^T A$.

4. The columns of U are the normalized vectors obtained by

$$\mathbf{u}_1 = \frac{1}{\sigma_1} A\mathbf{v}_1, \mathbf{u}_2 = \frac{1}{\sigma_2} A\mathbf{v}_2, \ldots, \mathbf{u}_m = \frac{1}{\sigma_m} A\mathbf{v}_m$$

Problem 4.64. Use MATLAB to find the singular value decomposition of

$$A = \begin{pmatrix} 4 & 11 & 14 \\ 8 & 7 & -2 \end{pmatrix}$$

Some practical MATLAB tips follow:

1. Define A
2. Define the sizes: `m=size(A,1); n=size(A,2)`
3. Find the rank of A: `r = rank(A);`
4. Define the matrices `Sigma` and `U` to be zero matrices with the right size.
5. Have MATLAB calculate the eigenvectors and eigenvalues of A^TA:
   `[vectors,values]=eig(A'A,'vector');`
   The `'vector'` command spits out the eigenvalues as a vector instead of a diagonal matrix. This will be helpful in the next step.
6. Have MATLAB sort the eigenvalues and strip any negative *approximate zero* eigenvalues that arise from numerical approximation of zero.
   ```
   values = abs(values);
   [values,indices] = sort(values,'descend')
   ```
7. Sort the columns of V using the indices coming out of the sort command:
   `V = vectors(:,indices);`
8. Build the singular values from the eigenvalues of A^TA (remember the square root!):
   `singularvalues=...`
9. Build non-zero diagonal entries of the Σ matrix with a loop. Also build a temporary matrix B the same size as Σ but with the diagonal entries $1/\sigma_j$. We'll need B in the next step.

```
1 B=zeros(size(Sigma));
2 for j=1:r}
3 Sigma(j,j) = ...
4 B(j,j) = ...
5 end
```

10. Observe that since V has orthonormal columns we can write $AV = U\Sigma$. Now, Σ is not square, but we know that it has diagonal entries only so we have a *pseudo-inverse* B^T already built. Hence, $U = AVB^T$. Build U.

11. Check that $A = U\Sigma V^T$

▲

Problem 4.65. Create a MATLAB function that accepts a matrix A and outputs the three matrices for the singular value decomposition. Test your function on a large random rectangular matrix.

```
A = rand(500,300);
[U,S,V] = MySVD(A);
error = norm(A - U*S*V')
```

▲

4.8 Multi-Dimensional Newton's Method

Now that we know some linear algebra let's return to the Newton's Method root finding technique from earlier in the book. This time we will consider root finding problems where we are not just solving the equation $f(x) = 0$ as we did Chapter 2. Instead consider the function F that takes a vector of variables in and outputs a vector. An example of such a function is

$$F(x,y) = \begin{pmatrix} x\sin(y) \\ \cos(x) + \sin(y^2) \end{pmatrix}.$$

It should be clear that making a picture of this type of function is a frivolous endeavor! In the case of the previous example, there are two inputs and two outputs so the "picture" would have to be four dimensional. Even so, we can still ask the question:

For what values of x and y does the function F give the zero vector?

That is, what if we have F defined as

$$F(x,y) = \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

and want to solve the system of equations

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0. \end{aligned}$$

In the present problem this amounts to solving the nonlinear system of equations

$$\begin{aligned} x\sin(y) &= 0 \\ \cos(x) + \sin(y^2) &= 0. \end{aligned}$$

In this case it should be clear that we are implicitly defining $f(x,y) = x\sin(y)$ and $g(x,y) = \cos(x) + \sin(y^2)$. A moment's reflection (or perhaps some deep meditation) should reveal that $(\pm\pi/2, 0)$ are two solutions to the system, and given the trig functions it stands to reason that $(\pi/2 + \pi k, \pi j)$ will be a solution for all integer values of k and j.

Problem 4.66. To build a numerical solver for a nonlinear system of equations, let's just recall Newton's Method in one dimension and then mimic that for systems of higher dimensions. We'll stick to two dimensions in this problem for relative simplicity.

(a) In Newton's Method we first found the derivative of our function. In a nonlinear system such as this one, talking about "the derivative" is a bit nonsense since there are many first derivatives. Instead we will define the Jacobian matrix $J(x,y)$ as a matrix of the first partial derivatives of the functions f and g.

$$J(x,y) = \begin{pmatrix} f_x & f_y \\ g_x & g_y \end{pmatrix}.$$

In the present example (fill in the rest of the blanks),

$$J(x,y) = \begin{pmatrix} \sin(y) & \_\_\_\_\_ \\ \_\_\_\_\_ & \_\_\_\_\_ \end{pmatrix}.$$

(b) Now let's do some Calculus and algebra. Your job in this part of this problem is to follow all of the algebraic steps.

(i) In one-dimensional Newton's Method we then write the equation of a tangent line at a point $(x_0, f(x_0))$

$$f(x) - f(x_0) \approx f'(x_0)(x - x_0)$$

to give a local approximation to the function. We'll do the exact same thing here, but in place of "x" we need to have a vector and in place of the derivative we need to have the Jacobian

$$F(x,y) - F(x_0,y_0) \approx J(x_0,y_0)\left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}\right).$$

(ii) In one-dimensional Newton's Method we then set $f(x)$ to zero since we were ultimately trying to solve the equation $f(x) = 0$. Hence we got the equation

$$0 - f(x_0) \approx f'(x_0)(x - x_0)$$

and then rearranged to solve for x. This gave us

$$x \approx x_0 - \frac{f(x_0)}{f'(x_0)}.$$

In the multi-dimensional case we have the same goal. If we set $F(x,y)$ to the zero vector and solve for the vector $\begin{pmatrix} x \\ y \end{pmatrix}$ then we get

$$\begin{pmatrix} x \\ y \end{pmatrix} \approx \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} - [J(x_0,y_0)]^{-1} F(x_0,y_0).$$

Take very careful note here that we didn't divide by the Jacobian ... it is a matrix after all!!

(iii) The final step in one-dimensional Newton's Method was to turn the approximation of x into an iterative process by replacing x with x_{n+1} and replacing x_0 with x_n resulting in the iterative form of Newton's Method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

We can do the exact same thing in the two-dimensional version of Newton's Method to arrive at

$$\boxed{\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - J^{-1}(x_n,y_n)F(x_n,y_n).}$$

Writing this in full matrix-vector form we get

$$\boxed{\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - \begin{pmatrix} f_x & f_y \\ g_x & g_y \end{pmatrix}^{-1} \begin{pmatrix} f(x_n,y_n) \\ g(x_n,y_n) \end{pmatrix}.}$$

(c) Write down the Newton iteration formula for the system

$$\begin{aligned} x\sin(y) &= 0 \\ \cos(x) + \sin(y^2) &= 0. \end{aligned}$$

Do not actually compute the matrix inverse of the Jacobian.

(d) The inverse of the Jacobian needs to be dealt with carefully. We typically don't calculate inverses directly in numerical analysis, but since we have some other tools to do the work we can think of it as follows:

- We need the vector $\mathbf{b} = J^{-1}(x_n, y_n)F(x_n, y_n)$.
- The vector $\mathbf{b}$ is the same as the solution to the equation $J(x_n, y_n)\mathbf{b} = F(x_n, y_n)$ at each iteration of Newton's Method.
- Therefore we can so a relatively fast linear solve (using any technique from this chapter) to find $\mathbf{b}$.
- The Newton iteration becomes

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - \mathbf{b}.$$

Write code to solve the present nonlinear system of equations. Implement some sort of linear solver within your code and be able to defend your technique. Be prepared to share your answer and your technique with your peers. Try to pick a starting point so that you find the solution $(\pi/2, \pi)$ on your first attempt at solving this problem. Then play with the starting point to verify that you can get the other solutions.

▲

Problem 4.67. Test your code from the previous problem on the system of nonlinear equations

$$\begin{aligned} 1 + x^2 - y^2 + e^x\cos(y) &= 0 \\ 2xy + e^x\sin(y) &= 0. \end{aligned}$$

Note here that $f(x, y) = 1 + x^2 - y^2 + e^x\cos(y)$ and $g(x, y) = 2xy + e^x\sin(y)$. ▲

Problem 4.68. Let's generalize the process a bit so we can numerically approximate solutions to systems of nonlinear algebraic equations in any number of dimensions. The Newton's method that we derived in Chapter 2 is only applicable to functions $f : \mathbb{R} \to \mathbb{R}$ (functions mapping a real number to a real number). In the previous problem we build a method for solving the equation $F(x, y) = (0, 0)$ where $F : \mathbb{R}^2 \to \mathbb{R}^2$. What about vector-valued functions in n dimensions? In particular, we would like to have an analogous method for finding roots of a function F where $F : \mathbb{R}^k \to \mathbb{R}^k$.

Let $\mathbf{x}$ be a vector in $\mathbb{R}^k$, let

$$F(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_k(\mathbf{x}) \end{pmatrix}$$

be a vector valued function, and let J be the Jacobian matrix

$$J(\mathbf{x}) = \begin{pmatrix} \partial f_1/\partial x_1(\mathbf{x}) & \partial f_1/\partial x_2(\mathbf{x}) & \cdots & \partial f_1/\partial x_k(\mathbf{x}) \\ \partial f_2/\partial x_1(\mathbf{x}) & \partial f_2/\partial x_2(\mathbf{x}) & \cdots & \partial f_2/\partial x_k(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_k/\partial x_1(\mathbf{x}) & \partial f_k/\partial x_2(\mathbf{x}) & \cdots & \partial f_k/\partial x_k(\mathbf{x}) \end{pmatrix}$$

By analogy, the multi-dimensional Newton's method is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J^{-1}(\mathbf{x}_n)F(\mathbf{x}_n)$$

where $J^{-1}(\mathbf{x}_n)$ is the inverse of the Jacobian matrix evaluated at the point $\mathbf{x}_n$.

(a) Write MATLAB code that accepts any number of functions and an initial vector guess and returns an approximation to the root for the problem $F(\mathbf{x}) = \mathbf{0}$.

(c) Use Newton's method to find an approximate solution to the system of equations

$$\begin{aligned} x^2 + y^2 + z^2 &= 100 \\ xyz &= 1 \\ x - y - \sin(z) &= 0 \end{aligned}$$

▲

Technique 4.69 (Multi-Dimensional Newton's Method)**.** The iteration for Newton's Method in multiple dimensions is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J^{-1}(\mathbf{x}_n)F(\mathbf{x}_n)$$

where $\mathbf{x}_n \in \mathbb{R}^k$, J is the k-dimensional Jacobian matrix, and $F : \mathbb{R}^k \to \mathbb{R}^k$. Take note that numerically you do not want to directly compute the inverse of the Jacobian. Instead you solve the sub-problem

$$J(\mathbf{x}_n)\mathbf{b} = F(\mathbf{x}_n)$$

for the vector $\mathbf{b}$ using any technique from linear algebra (e.g. an LU or QR decomposition) and iterate the equation

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{b}.$$

Problem 4.70. When will the multi-dimensional version of Newton's Method fail? Compare and contrast this with what you found about the one-dimensional version of Newton's Method in Chapter 2. Extend your discussion to talk about the eigenvalues of the Jacobian matrix for a nonlinear system. ▲

One place that solving nonlinear systems arises naturally is when we need to find equilibrium points for systems of differential equations. Remember that to find the equilibrium points for a first order differential equation we set the derivative term to zero and solve the resulting equation.

Problem 4.71. Find the equilibrium point(s) for the system of differential equations

$$x' = \alpha x - \beta xy$$
$$y' = \delta y + \gamma xy$$

where $\alpha = 1, \beta = 0.05, \gamma = 0.01$ and $\delta = 1$. You will play more with this differential equation system later. ▲

4.9 Exercises

Problem 4.72. Write code to solve the following systems of equations via both LU and QR decompositions.

(a)

$$\begin{aligned} x + 2y + 3z &= 4 \\ 2x + 4y + 3z &= 5 \\ x + y &= 4 \end{aligned}$$

(b)

$$\begin{aligned} 2y + 3z &= 4 \\ 2x + 3z &= 5 \\ y &= 4 \end{aligned}$$

(b)

$$\begin{aligned} 2y + 3z &= 4 \\ 2x + 4y + 3z &= 5 \\ x + y &= 4 \end{aligned}$$

▲

Problem 4.73. Find a least squares solution to the equation $A\mathbf{x} = \mathbf{b}$ in two different ways with

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 4 & -2 & 6 \\ 4 & 7 & 8 \\ 3 & 7 & 19 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 5 \\ 2 \\ -2 \\ 8 \end{pmatrix}.$$

▲

Problem 4.74. Now that you have QR and LU code we're going to use both of them! The problem is as follows:
We are going to find the polynomial of degree 4 that best fits the function

$$y = \cos(4t) + 0.1\varepsilon(t)$$

at 50 equally spaced points t between 0 and 1. Here we are using $\varepsilon(t)$ as a function that outputs normally distributed random white noise. In MATLAB you will build y as

```
y = cos(4*t) + 0.1*randn(size(t));
```

Build the t vector and the y vector (these are your data). We need to set up the least squares problems $A\mathbf{x} = \mathbf{b}$ by setting up the matrix A as we did in the other least squares curve fitting problems and by setting up the $\mathbf{b}$ vector using the y data you just built.

(a) Solve the normal equations $A^TA\mathbf{x} = A^T\mathbf{b}$ using your LU code.

(b) Solve the system $A\mathbf{x} = \mathbf{b}$ by first transforming A to $A = QR$ and then solving $R\mathbf{x} = Q^T\mathbf{b}$.

(c) Use MATLAB to find the sum of the square errors between the polynomial approximation and the function $f(t) = \cos(4t)$ for both the QR and the LU approaches.

(d) Build MATLAB code that does parts (a) - (c) several hundred times and complies results comparing which method gives the better approximation (smaller sum of square error).

▲

Problem 4.75. In this exercise we will use numerical linear algebra to do some handwriting recognition on the classical data set `mnist`. The `mnist` data set contains a training set of 60,000 numbers and a test set of 10,000 numbers. Each digit in the database was placed in a 28 by 29 grayscale image such that the center of mass of its pixels is at the center of the picture. While our primary goal for this problem is to use numerical linear algebra, our secondary goal is to get some experience with the logic of machine learning. In machine learning problems we often follow the following logic:

- First use a set of data for which we know the *answer* (in this case the *answer* is the numerical value of the digit that was written). We call this the training data set in the sense that we train the numerical method with the correct answers in mind.
- Next we compare data with hidden answers to our training set and build a method for using the training set to make a prediction for the answer. This is called the testing phase of a machine learning algorithm. In the testing phase we know the *answer* but keep it hidden from the algorithm. We let our algorithm predict the *answer* and then compare to the hidden truth. At the end of this step we can give a percent effectiveness for our algorithm.
- In the final step of a machine learning process we give the numerical algorithm data where we don't know the *answer* and use the algorithm to predict for us.

Let's be more specific. Let's say that we want to determine if a handwritten digit is the number "0". From the training set we average all of the zeros together to get a best estimate of what a "0" looks like. Then we build a mathematical technique for doing the comparison between our new digit and the averaged training 0. If the comparison technique tells us that our new digit is *close enough* then we call that new digit a zero. We can do this for all of the `test` 0's and determine the percent effectiveness for our comparison technique.

Your Tasks:

(a) Start by going to www.cs.nyu.edu/~roweis/data.html to download the data set. Download the file titled `mnist_all.mat`. This file is a MATLAB file that needs to be read into your working session of MATLAB.

(b) To read the `mnist_all.mat` data into MATLAB
`load mnist_all.mat`
Then type `whos` to see the variables containing training digits (`train0, ..., train9`) and test digits (`test0, ..., test9`).

(c) To visualize the first image in the matrix `train0` use

```
digit = train0(1,:); % read the first row all columns out of train0
digitImage = reshape(digit,28,28); % turn into a 28x28 matrix
image(rot90(flipup(digitImage),-1))
colormap(gray(256))
axis square tight off
```

(d) Create a 10 by 784 matrix T whose i^{th} row contains the average pixel values over all of the training images of the number $i-1$. For instance, the first row of T can be formed by typing
`T(1,:) = mean(train0);`
Visualize these average digits using the `subplot` command creating a 2×5 matrix of plots with the average 0 in the upper left and the average 9 in the lower right. Check yourself by making sure that your image is identical to Figure 4.3.

(e) We are going to try two methods for handwriting recognition. There are 10,000 test numbers that are not in the training set and we want to try two different ways of determining which digit is in the test image. Your job is to implement both of these methods. You need to test all 10,000 test images and gather statistics on how often the method identifies the test image. Report your answers by stating the proportion of correct identifications for each of the 10 numerals.

Method #1 (Min Norm): Compare pixels in the test digit to each row of the training matrix T and determine which row most closely resembles the test digit. Pseudo code for this method is:

- Let D be the first test digit in `test0` using
  `D = double(test0(1,:));`
- For each row $i = 1, 2, \cdots, 10$ compute
  `norm(T(i,:) - D)`
  and determine which value of i this is smallest
- D is probably the digit $i-1$.
- repeat for all of the test digits in `test0`, `test1`, ...

Be sure to write code to test all of the 10,000 test digits. There are elegant ways to code this but you can also complete this task with a bunch of copy and paste.

Method #2 (Min Projections): Project the test image vector D onto the mean training image $T(i,:)$ and find the error in the projection. In this method we seek to minimize the size of the error. Recall that the projection of D onto $T(i,:)$ is

$$\frac{D \cdot T(i,:)}{T(i,:) \cdot T(i,:)}$$

and the error in the projection is

$$\left\| D - \left(\frac{D \cdot T(i,:)}{T(i,:) \cdot T(i,:)} \right) T(i,:) \right\|$$

Figure 4.3. The 10 images are the averages of all of the training images for each digit. They represent what a *typical* digit should look like for each of the 10 digits.

▲

Problem 4.76. Find the largest eigenvalue of the matrix A WITHOUT using the built in "`eig`" or "`eigs`" commands in MATLAB.

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 0 & 1 & 2 \\ 3 & 4 & 5 & 6 \end{pmatrix}$$

▲

Problem 4.77. Find a least square cubic function that best fits the following data. Solve this problem with Excel and with MATLAB using the normal equations.

x	y
0	1.0220
0.0500	1.0174
0.1000	1.0428
0.1500	1.0690
0.2000	1.0505
0.2500	1.0631
0.3000	1.0458
0.3500	1.0513
0.4000	1.0199
0.4500	1.0180
0.5000	1.0156
0.5500	0.9817
0.6000	0.9652
0.6500	0.9429
0.7000	0.9393
0.7500	0.9266
0.8000	0.8959
0.8500	0.9014
0.9000	0.8990
0.9500	0.9038
1.0000	0.8989

▲

Theorem 4.78 (Eigen-Structure of Symmetric Matrices)**.** If A is a symmetric matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ then $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. Furthermore, the eigenvectors will be orthogonal to each other.

Problem 4.79. For symmetric matrices we can build an extension to the Power Method in order to find the second most dominant eigen-pair for a matrix A. Theorem 4.78 suggests the following method for finding the second dominant eigen-pair for a symmetric matrix. This method is called the **deflation method**.

- Use the power method to find the dominant eigenvalue and eigenvector.
- Start with a random unit vector of the correct shape.
- Multiplying your vector by A will *pull it toward* the dominant eigenvector. After you multiply, project your vector onto the dominant eigenvector and find the projection error.
- Use the projection error as the new approximation for the eigenvector.

Note that the deflation method is really exactly the same as the power method with the exception that we orthogonalize at every step. Hence, when you write your code expect to only change a few lines from your Power method.

Write a MATLAB function `MyPower2` to find the second largest eigenvalue and eigenvector pair by putting the deflation method into practice. Test your code on a symmetric matrix A and compare against MATLAB's `eig` command. Your code needs to work on symmetric matrices of arbitrary size and you need to write test code that clearly shows the error between your calculated eigenvalue and MATLAB's eigenvalue as well as your calculated eigenvector and MATLAB's eigenvector.
To guarantee that you start with a symmetric matrix you can use the following code.

```
N = 40; % size of the matrix ... make this large-ish
A = rand(N,N);
A = A'*A; % this will be a random symmetric NxN matrix.
```

▲

Problem 4.80. Find the equilibrium point(s) for the system of differential equations

$$\begin{aligned} x' &= -0.1xy - x \\ y' &= -x + 0.9y \\ z' &= \cos(y) - xz \end{aligned}$$

if they exist. ▲

Problem 4.81. (This problem is modified from [6])
A manufacturer of lawn furniture makes two types of lawn chairs, one with a wood frame and one with a tubular aluminum frame. The wood-frame model costs 418 per unit to manufacture, and the aluminum-frame model costs \$10 per unit. The company operates in a market where the number of units that can be sold depends on the price. It is estimated that in order to sell x units per day of the wood-frame model and y units per day of the aluminum-frame model, the selling price cannot exceed

$$10 + \frac{31}{\sqrt{x}} + \frac{1.3}{y^{0.2}} \text{ dollars per unit}$$

for wood-frame chairs, and

$$5 + \frac{15}{y^{0.4}} + \frac{0.8}{x^{0.08}} \text{ dollars per unit}$$

for the aluminum chairs. We want to find the optimal production levels. Write this situation as a multi-variable mathematical model, use a computer algebra system (or by-hand computation) to find the gradient vector, and then use the multi-variable Newton's method to find the critical points. Classify the critical points as either local maximums or local minimums. ▲

Chapter 5

Numerical Ordinary Differential Equations

> "The mathematical discipline of differential equations furnishes the explanation of all those elementary manifestations of nature which involve time."
> –Sophus Lie

In this chapter we will solve first order ordinary differential equations of the form

$$y'(t) = f(t, y(t))$$

with initial condition $y(t_0) = y_0$ for $t \geq t_0$. These are known as "ordinary" differenatial equations since they contain only "ordinary" derivatives; not partial derivatives. Given that we are solving the problem with given intial information these are also called intial value problems.

5.1 Euler, Runge-Kutta, and Friends

The notion of approximating solutions to differential equations is simple: make a discrete approximation to the derivative and step forward through time as a difference equation. The fun part is making the approximation to the derivative(s). There are many methods for approximating derivatives, and that is exactly where we'll start.

Technique 5.1 (Euler's Method)**.** You're probably already familiar with Euler's method for approximating the solution to a differential equation. We want to approximate a solution to $y'(t) = f(t, y(t))$. Recall from Problem 3.2 that

$$y'(t) = \frac{y(t+h) - y(t)}{h} + \mathcal{O}(h)$$

so the differential equation $y'(t) = f(t, y(t))$ becomes

$$\frac{y(t+h) - y(t)}{h} \approx f(t, y(t)).$$

Rewriting as a difference equation, letting $y_{n+1} = y(t_n + h)$ and $y_n = y(t_n)$, we get

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{5.1}$$

A way to think about Euler's method is that at a given point, the slope is approximated by the value of the right-hand side of the differential equation and then we step forward h units in time following that slope. Figure 5.1 shows a depiction of the idea. Notice in the figure that in regions of high curvature Euler's method will overshoot the exact solution to the differential equation. However, taking $h \to 0$ theoretically gives the exact solution at the tradeoff of needing infinite computational resources.

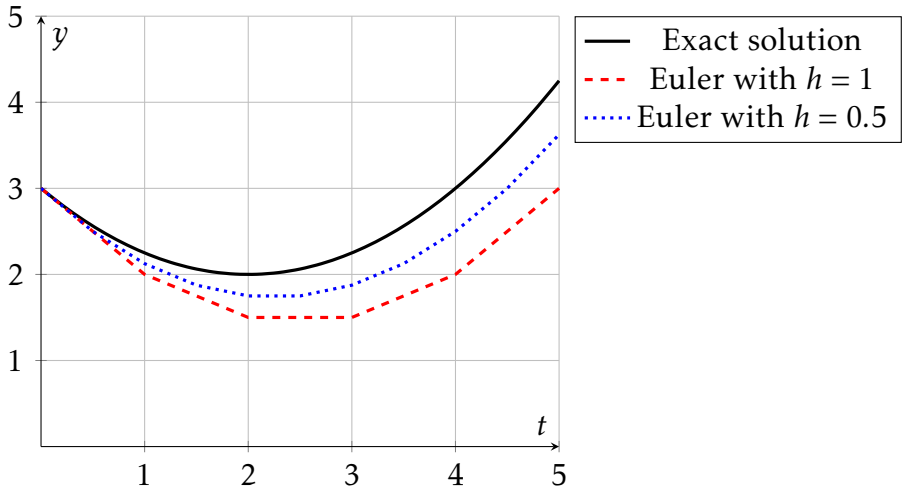


Figure 5.1. A depiction of Euler's method with step size $h = 1$ (red) and $h = 0.5$ (blue).

Problem 5.2. Write code to implement Euler's method for initial value problems. Your MATLAB function should accept as input: $f(t, y)$, `tmin`, `tmax`, the number of grid points (the value of $h = \Delta t$ should be calculated within your code), and an intial condition. The output should be vectors for t and y.

```
function [t,y] = MyEuler1D(f,tmin,tmax,num_pts,IC)
```

Test your code on a first order differential equation where you know the answer and then test your code on the differential equation

$$y' = -\frac{1}{3}y + \sin(t) \quad \text{where} \quad y(0) = 1.$$

▲

Problem 5.3. Write code that implements a 2D version of Euler's method that will solve a system of two differential equations in two dependent variables. Test your code on the

following problem by showing a time evolution plot (time on x and populations on y) as well as a phase plot (x on the x and y on the y with time understood implicitly):

The Lotka-Volterra Predator-Prey Model:
Let $x(t)$ denote the number of rabbits (prey) and $y(t)$ denote the number of foxes (predator) at time t. The relationship between the species can be modeled by the classic 1920's Lotka-Volterra Model:

$$\begin{cases} x' &= \alpha x - \beta xy \\ y' &= -\delta y + \gamma xy \end{cases}$$

where α, β, γ, and δ are positive constants. For this problems take $\alpha \approx 1$, $\beta \approx 0.05$, $\gamma \approx 0.01$, and $\delta \approx 1$. Be sure to explain the meaning of each of the parameters and each of the components of the model. ▲

Technique 5.4 (The Midpoint Method)**.** Now we begin the journey of creating better solvers than Euler's method. The midpoint method is defined by first taking a half step with Euler's method to approximate a solution at time $t_{n+1/2} \equiv (t_n + t_{n+1})/2$ and then taking a full step using the value of f at $t_{n+1/2}$ and the approximate $y_{n+1/2}$.

$$\begin{aligned} y_{n+1/2} &= y_n + \frac{h}{2} f(t_n, y_n) \\ y_{n+1} &= y_n + h f(t_{n+1/2}, y_{n+1/2}) \end{aligned}$$

Note: Indexing by 1/2 in a computer is nonsense. Instead, we implement the midpoint method with:

$$\begin{aligned} y_{temp} &= y_n + \frac{h}{2} f(t_n, y_n) \\ y_{n+1} &= y_n + h f\left(\frac{t_n + t_{n+1}}{2}, y_{temp}\right) \end{aligned}$$

Problem 5.5. Write MATLAB code to implement the midpoint method

```
function [t,y]=MyMidpointMethod(f,tmin,tmax,num_pts,IC)
```

▲

Problem 5.6. Test your midpoint method code against your Euler1D code on the same single variable ODE as before. You will likely see very little difference on a very small step size (equivalently, a large number of points), but for a smaller number of points there will be a remarkable difference. ▲

We have studied two methods thus far: Euler's method and the Midpoint method. In Figure 5.2 we see a graphical depiction of how each method works on the differential equation $y' = y$ with $\Delta t = 1$ and $y(0) = 1$. The exact solution at $t = 1$ is $y(1) = e^1 \approx 2.718$ and is shown in red in each figure. The methods can be summarized as

Euler's Method	Midpoint Method
1. Get the slope at time t_n	1. Get the slope at time t_n
2. Follow the slope for time Δt	2. Follow the slope for time $\Delta t/2$
	3. Get the slope at the point $t_n + \Delta t/2$
	4. Follow the new slope from time t_n for time Δt

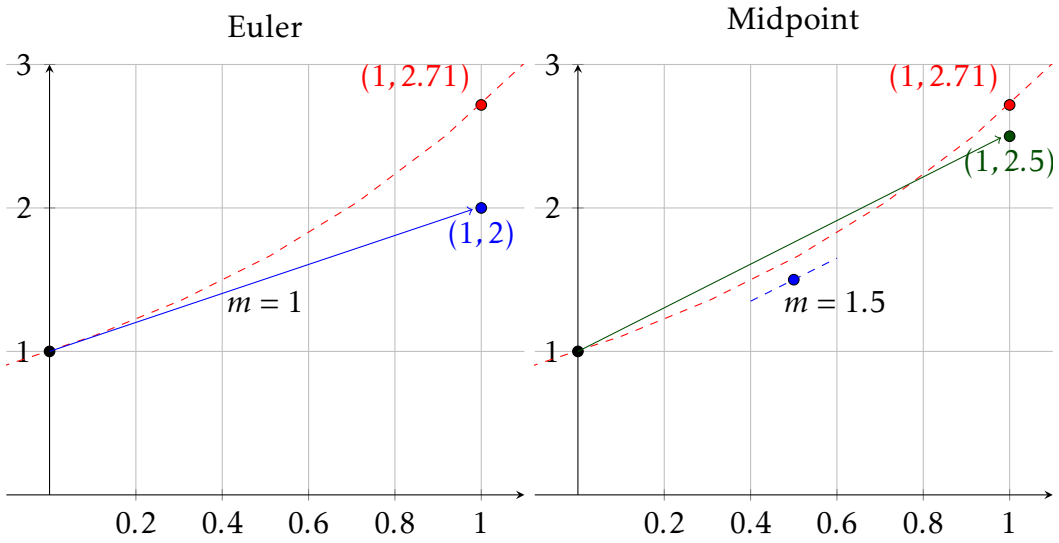


Figure 5.2. Graphical depictions of two numerical methods: Euler (left) and Midpoint (right). Here we use the simple differential equation $y' = y$ with $y(0) = 0.5$ and $\Delta t = 1$. The exact solution is shown in red.

Technique 5.7 (The Runge-Kutta 4 (RK4) Method)**.** Another method for approximating the solution to a first order initial value problem is to take several approximations and average them in a smart way. The Runge-Kutta 4 method is one (of many) such methods. In this method, each k_j is an approximation of the slope and we combine them in as a weighted average in the end.

$$
\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \\
k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \\
k_4 &= f(t_n + h, y_n + hk_3) \\
y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
$$

Before we write code to implement the RK4 method we will examine it graphically just as we did with Euler's method and the Midpoint method in Figure 5.2. For simplicity

we will examine the differential equation $y' = y$ with initial condition $y(0) = 1$ and $\Delta t = 1$. In Figure 5.3 the red dashed line is the exact solution $y(t) = e^t$. In this example, $k_1 = 1$, $k_2 = 1.5$, $k_3 = 1.75$, and $k_4 = 2.75$. Hence the final slope propogating forward with $\Delta t = 1$ is

$$\frac{1}{6}(1 + 2(1.5) + 2(1.75) + 2.75) = 1.708.$$

Propogating this forward from the point $(0, 1)$ gives the new point $(1, 2.708)$. Knowing that $e \approx 2.718$ we see a very high level of accuracy even with a really large time step!

Runge-Kutta 4 Method
1. k_1 is the slope evaluated at time t_n Project this slope half a step forward from time t_n to the point y_1
2. k_2 is the slope evaluated at y_1 Project the slope k_2 half a step forward from time t_n to the point y_2
3. k_3 is the slope evaluated at the point y_2 Project the slope k_3 a full step foward from time t_n to the point y_3
4. k_4 is the slope evaluated at the point y_3
5. Project forward with slope $\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ from time t_n

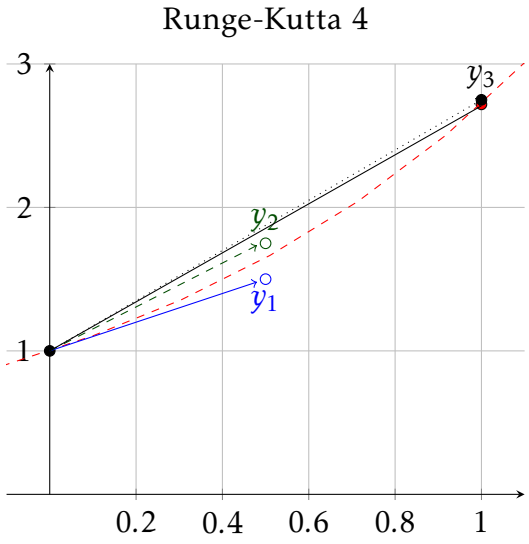


Figure 5.3. Graphical depiction of the RK4 method. The red dashed curve gives the exact solution to the differential equation $y' = y$ with initial condition $y(0) = 1$. The solid black line gives the final projection.

Problem 5.8. Write a MATLAB function that implements the Runge-Kutta 4 method in one dimension.

```
function [t,y]=MyRk4(f,tmin,tmax,num_pts,IC)
```

Test the problem on a known differential equation. ▲

Problem 5.9. Modify your Runge-Kutta 4 code to work for two dependent variables. I'll get you started:
We want to solve

$$\begin{cases} x' &= f(t,x,y) \\ y' &= g(t,x,y) \end{cases}$$

and to do so we extend the Runge Kutta method as

$$\begin{aligned} k_1 &= f(t_n, x_n, y_n) \\ q_1 &= g(t_n, x_n, y_n) \\ k_2 &= f(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_1, y_n + \frac{h}{2}q_1) \\ q_2 &= g(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_1, y_n + \frac{h}{2}q_1) \\ k_3 &= \dots \\ q_3 &= \dots \\ k_4 &= \dots \\ q_4 &= \dots \\ x_{n+1} &= x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ y_{n+1} &= y_n + \frac{h}{6}(q_1 + 2q_2 + 2q_3 + q_4) \end{aligned}$$

Test your code on the predator prey model in problem 5.3. ▲

Problem 5.10. Solving systems of ordinary differential equations would become challenging if we were to continue coding in the same way as in the previous problem – modifying your code to account for the number of differential equations. Write a MATLAB function that accepts any number of right-hand sides from a system of differential equations and then leverages the fact that MATLAB works very well with vectors to create Euler and Runge-Kutta solutions to these systems. Devise several systems to test your code (including 1D and 2D).

One particular nonlinear system of differential equations that you can test on is

$$\begin{cases} y_1' &= -0.1y_2y_1 - y_1 & y_1(0) = 2 \\ y_2' &= -y_1 + 0.9y_2 & y_2(0) = 1 \\ y_3' &= \sin(t) + \cos(y_2) & y_3(0) = -2 \end{cases}.$$

Note that we would probably never try to solve this problem by hand so a numerical method is warranted.

A test script to check your General Euler and General RK code is as follows. You can use my notation here to possibly reverse engineer the construction of your general ODE solver codes.

```
numpts = 1000;
```

```
f = @(t,y) [-0.1*y(2)*y(1)-y(1) ;
    -y(1)+0.9*y(2) ;
    sin(t) + cos(y(2))];

tmax = 5.5;

[t,yeuler]=MyGeneralEuler(f,0,tmax,numpts,[2,1,-2]);
plot(t,yeuler(1,:),'b--',t,yeuler(2,:),'r--',t,yeuler(3,:),'k--')
hold on

[t,yrk]=MyGeneralRK4(f,0,tmax,numpts,[2,1,-2]);
plot(t,yrk(1,:),'k-.',t,yrk(2,:),'m-.',t,yrk(3,:),'c--')
grid on
legend('y_1''(t)' , 'y_2''(t)' , 'y_3''(t)')
```

▲

5.2 Implicit Methods and Shooting Methods

Problem 5.11. The major trouble with the RK (and midpoint) methods is that they take many more function evaluations than Euler's method. We can improve upon Euler's method in the following way:
We want to solve $y' = f(t, y)$ so:

1. Approximate the derivative by looking forward in time(!)

$$\frac{y_{n+1} - y_n}{h} \approx f(t_{n+1}, y_{n+1})$$

2. Rearrange to get the difference equation

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}).$$

3. Notice that we will have t_{n+1} but we do not have y_{n+1}. The major trouble is that y_{n+1} shows up on both sides of the equation. Can you think of a way to solve for it? ... you have code that does this step!!!

4. This method is called the **backward Euler** method and is known as an **implicit method** since you need to solve a nonlinear equation at each step. The advantage (usually) is that you can take far fewer steps with reasonably little loss of accuracy.

▲

Problem 5.12. Write MATLAB code to implement the backward Euler's method for a 1D initial value problem.

```
function [t,y] = MyBackwardEuler(f, tmin, tmax, num_pts, IC)
```

▲

Problem 5.13. Write a MATLAB script that outputs a log-log plot with the step size on the horizontal axis and the error in the numerical method on the vertical axis. Plot the errors for Euler, Midpoint, Runge Kutta, and Backward Euler measured against a differential equation with a known analytic solution. Use this plot to conjecture the convergence rates of the four methods. ▲

Problem 5.14. We wish to solve the boundary valued problem $x'' + 4x = \sin(t)$ with initial condition $x(0) = 1$ and boundary condition $x(1) = 2$. Notice that you do not have the initial position and initial velocity as you normally would with a second order differential equation. Devise a method for finding a numerical solution to this problem.
Hint: First write the problem as a system of first order differential equations. Then think about how your bisection method code might help you. ▲

5.3 ODE Modeling Exercises

In this section we give several problems which model real-world scenarios with ordinary differential equations. For every one of these problems you will need to use a numerical method to solve the differential equation(s).

Problem 5.15. In this model there are two characters, Romeo and Juliet, whose affection is quantified on the scale from −5 to 5 described below:

−5	Hysterical Hatred
−2.5	Disgust
0	Indifference
2.5	Sweet Affection
5	Ecstatic Love

The characters struggle with frustrated love due to the lack of reciprocity of their feelings. Mathematically,

- Romeo: "My feelings for Juliet decrease in proportion to her love for me."
- Juliet: "My love for Romeo grows in proportion to his love for me."
- Juliet's emotional swings lead to many sleepless nights, which consequently dampens her emotions.

This give rise to

$$\begin{cases} \frac{dx}{dt} &= -\alpha y \\ \frac{dy}{dt} &= \beta x - \gamma y^2 \end{cases}$$

where $x(t)$ is Romeo's love for Juliet and $y(t)$ is Juliet's love for Romeo at time t.

Your tasks:

1. First implement this 2D system with $x(0) = 2$, $y(0) = 0$, $\alpha = 0.2$, $\beta = 0.8$, and $\gamma = 0.1$ for $t \in [0, 60]$. What is the fate of this pair's love under these assumptions?
2. Write MATLAB code that approximates the parameter γ that will result in Juliet having a feeling of indifference at $t = 30$. Your code should not need human supervision: you should be able to tell it that you're looking for *indifference* at $t = 30$ and turn it loose to find an approximation for γ. Assume throughout this problem that $\alpha = 0.2$, $\beta = 0.8$, $x(0) = 2$, and $y(0) = 0$. Write a description for how your code works in your homework document and also submit your MATLAB file (along with any support files) demonstrating how it works.
   Hint: One way to do this problem is very similar to a bisection method for root finding.
   - Shoot two solutions with two different parameters.
   - Compare their results at the desired time against the result of shooting with the average value of the parameter.

- Use the logic of the bisection method to make a new estimate of the parameter.

▲

Problem 5.16 (Orbiting Bodies Problem)**.** In this problem we'll look at the orbit of a celestial body around the sun. The body could be a satellite, comet, plant, or any other object whose mass is negligible compared to the mass of the sun. We assume that the motion takes place in a two dimensional plane so we can describe the path of the orbit with two coordinates, x and y with the point $(0,0)$ being used as the reference point for the sun. According to Newton's law of universal gravitation the system of differential equations that describes the motion is

$$x''(t) = \frac{-x}{\left(\sqrt{x^2+y^2}\right)^3} \quad \text{and} \quad y''(t) = \frac{-y}{\left(\sqrt{x^2+y^2}\right)^3}.$$

(a) Make a change of variables to turn the system of second order differential equations into a system of first order differential equations. Explain the meaning of all four of the resulting variables.

(b) Solve the system of equations from part (a) using an appropriate solver. Start with $x(0) = 4$, $y(0) = 0$, the initial x velocity as 0, and the initial y velocity as 0.5. Create several plots showing how the dynamics of the system change for various values of the initial y velocity in the interval $(0, 0.5]$.

▲

Problem 5.17 (Pursuit and Evasion Problem)**.** In this problem we consider the pursuit and evasion problem where $E(t)$ is the vector for an evader (e.g. a rabbit or a bank robber) and $P(t)$ is the vector for a pursuer (e.g. a fox chasing the rabbit or the police chasing the bank robber)

$$E(t) = \begin{pmatrix} x_e(t) \\ y_e(t) \end{pmatrix} \quad \text{and} \quad P(t) = \begin{pmatrix} x_p(t) \\ y_p(t) \end{pmatrix}.$$

Let's presume the following:

Assumption 1: the evader has a predetermined path (known only to him/her),

Assumption 2: the pursuer heads directly toward the evader at all times, and

Assumption 3: the pursuer's speed is directly proportional to the evader's speed.

From the third assumption we have

$$||P'(t)|| = k||E'(t)|| \tag{5.2}$$

and from the second assumption we have

$$\frac{P'(t)}{||P'(t)||} = \frac{E(t) - P(t)}{||E(t) - P(t)||}. \tag{5.3}$$

Solving for $P'(t)$ and using 5.2 the differential equation that we need to solve becomes

$$P'(t) = k\|E'(t)\|\frac{E(t)-P(t)}{\|E(t)-P(t)\|}. \tag{5.4}$$

Your Tasks:

(a) Explain assumption #2 mathematically.

(b) Explain assumption #3 physically. Why is this assumption necessary mathematically?

(c) Write code to find the path of the pursuer if the evader has the parameterized path

$$E(t) = \begin{pmatrix} 0 \\ 5t \end{pmatrix} \quad \text{for} \quad t \geq 0$$

and the pursuer initially starts at the point $P(0) = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$. Write your code so that it stops when the pursuer is within 0.1 units of the evader. Run your code for several values of k.

(d) Modify your code from part (c) to find the path of the pursuer if the evader has the parameterized path

$$E(t) = \begin{pmatrix} 5+\cos(2\pi t)+2\sin(4\pi t) \\ 4+3\cos(3\pi t) \end{pmatrix} \quad \text{for} \quad t \geq 0$$

and the pursuer initially starts at the point $P(0) = \begin{pmatrix} 0 \\ 50 \end{pmatrix}$. Write your code so that it stops when the pursuer is within 0.1 units of the evader. Run your code for several values of k.

(e) Create your own smooth path for the evader that is *challenging* for the pursuer to catch. Write your code so that it stops when the pursuer is within 0.1 units of the evader. Run your code for several values of k.

(f) (Challenge) If you extend this problem to three spatial dimensions you can have the pursuer and the evader moving on a multivariable surface (i.e. hilly terrain). Implement a path along an appropriate surface but be sure that the velocities of both parties are appropriately related to the gradient of the surface.

Note: It may be easiest to build this code from scratch instead of using one of our pre-written codes. ▲

Problem 5.18 (Whales and Krill Problem)**.** One of the favorite foods of the blue whale is krill. Blue whales are baleen whales and feed almost exclusively on krill. These tiny shrimp-like creatures are devoured in massive amounts to provide the principal food source for the huge whales. In the absence of predators, in uncrowded conditions, the

krill population density grows at a rate of 25% per year. The presence of 500 tons/acre of krill increases the blue whale population growth rate by 2% per year, and the presence of 150,000 blue whales decreases krill growth rate by 10% per year. The population of blue whales decreases at a rate of 5% per year in the absence of krill.

These assumptions yield a pair of differential equations (a Lotka-Volterra model) that describe the population of the blue whales (B) and the krill population density (K) over time given by

$$\frac{dB}{dt} = -0.05B + \left(\frac{0.02}{500}\right)BK$$
$$\frac{dK}{dt} = 0.25K - \left(\frac{0.10}{150000}\right)BK.$$

(a) What are the units of $\frac{dB}{dt}$ and $\frac{dK}{dt}$?

(b) Explain what each of the four terms on the right-hand sides of the differential equations mean in the context of the problem. Include a reason for why each term is positive or negative.

(c) Find a numerical solution to the differential equation model using $B(0) = 75,000$ whales and $K(0) = 150$ tons per acre.

(d) Whaling is a huge concern in the oceans world wide. Implement a *harvesting* term into the whale differential equation, defend your mathematical choices and provide a thorough exploration of any parameters that are introduced.

▲

Problem 5.19 (Drone Path Problem)**.** You just received a new long-range helicopter drone for your birthday! After a little practice, you try a long-range test of it by having it carry a small package to your home. A friend volunteers to take it 5 miles east of your home with the goal of flying directly back to your home. So you program and guide the drone to always head directly toward home at a speed of 6 miles per hour. However, a wind is blowing from the south at a steady 4 miles per hour. The drone, though, always attempts to head directly home. We will assume the drone always flies at the same height. What is the drones flight path? Does it get the package to your home? What happens if the speeds are different? What if the initial distance is different? How much time does the drone's battery have to last to get home?
▲

Problem 5.20 (The Combustion Problem)**.** Let T be the temperature of a combustible material (e.g. oily rags, dry hay, etc.). The conservation of energy equation states that

$$\rho c_p \frac{dT}{dt} = A_1 e^{-B/(T-T_0)} - h(T - T_a)$$

where

- T is temperature in Kelvin,

- T_0 is a reference temperature above which the fuel starts oxidizing,
- T_a is the ambient temperature of the surrounding air,
- ρ is the density of the fuel,
- c_p is the specific heat of the fuel source,
- h is a measure of the power per volume per degree Kelvin,
- A_1 is a measure of the power per unit volume, and
- B is a rate constant measured in degrees Kelvin.

If we divide both sides of the differential equation by ρc_p we arrive at the first order non-homogeneous differential equation

$$\frac{dT}{dt} = Ae^{-B/(T-T_0)} - C(T - T_a).$$

Assume that A, B, and C are all positive coefficients.
Your Tasks:

(a) Why must $T > T_0$ in order for the equation to make sense physically?

(b) Let's suppose that $T_a = 300°K$ and that T_0 is also at the ambient temperature. Let $A = 20$, $B = 600$, and $C = 0.01$. Analyze the differential equation graphically plotting the phase portrait, identifying equilibrium points, and discussing stability of each point.

(c) Discuss the physical interpretation of each equilibrium point.

(d) Suppose we don't know what A, B, and Ca re, but we do know from experiments that the three fixed points are $T_1 = T_a = 300°K$, $T_2 = 670°K$, and $T_3 = 1200°K$. What can you say about the coefficients A, B, and C?

(e) Write a script that solves the problem numerically for several different initial conditions.

▲

Problem 5.21 (HIV Problem)**.** In a normal, HIV-free body, uninfected T-cells are introduced into the system at a constant rate λ. The T-cells in the system have a finite life span, and hence a proportion μ of the T-cells die. In an HIV infected body, there is a rate k at which the T-cells become infected. This rate depends both on the presence of T-cells and free HIV-1 particles to infect the T-cells. Putting this together, we get the following equation for the healthy T-cells:

$$\frac{dT}{dt} = \lambda - \mu T - kTV$$

where

- $T(t)$ is the number of uninfected and activated T-cells at time t
- $L(t)$ is the number of latently infected T-cells at time t
- $I(t)$ is the number of actively infected T-cells at time t
- $V(t)$ is the number of free HIV-1 particles at time t.

Once a T-cell becomes infected, a proportion p of them will become latently infected, while the remainder immediately be actively infected. A latently infected T-cell can later become actively infected, and this happens at a rate α. Latently infected cells die at the same rate μ as uninfected T-cells. Actively infected T-cells will be assumed to die at a different rate a. We can now write the equations for the infected T-cells:

$$\frac{dL}{dt} = kpTV - \mu L - \alpha L$$
$$\frac{dI}{dt} = k(1-p)TV + \alpha L - aI$$

Finally, free HIV-1 particles are manufactured inside actively infected T-cells at a rate c. The particles die at a rate γ. The particles are also used in the process of infecting healthy T-cells at the rate k. This gives us the following equation for HIV-1 particles:

$$\frac{dV}{dt} = cI - \gamma V - kTV.$$

Numerically solve system of differential equations and plot the time evolution of all four components. Write a thorough mathematical and biological description of the evolution of the system. The table below gives values for the initial conditions and parameters.

Parameters	Value
$T(0)$	200 T-cells / mm^3
$L(0)$	0 T-cells / mm^3
$I(0)$	0 T-cells / mm^3
$V(0)$	4×10^{-7} HIV-1 cells / mm^3
λ	0.272 / mm^3 day
μ	1.36×10^{-3} / day
k	2.7×10^{-4} mm^3/day
p	0.1
α	3.6×10^{-2} / day
a	0.33 / day
c	100 / day
γ	2 / day

▲

Problem 5.22 (Trebuchet Problem)**.** A trebuchet catapult throws a cow vertically into the air. The differential equation describing its acceleration is

$$\frac{d^2y}{dt^2} = -g - c\frac{dy}{dt}\left|\frac{dy}{dt}\right|$$

where $g \approx 9.8$ m/s^2 and $c \approx 0.02$ m^{-1} for a typical cow. If the cow is launched at an initial upward velocity of 30 m/s, how high will it go, and when will it crash back into the ground? Hint: Change this second order differential equation into a system of first order differential equations. ▲

Problem 5.23 (Classical SIR Model)**.** When a virus is introduced into a small homogeneously mixed population the people in the population can be split into three categories: susceptible to the virus (S), infected with the virus (I), and recovered from the virus (R). Assume the following:

- a susceptible person becomes infected at a rate proportional to the product of the number of susceptible people and the number of infected people,
- the recovery rate is constant,
- the recovered people are immune to re-infection,
- the virus is not fatal so the total population stays fixed.

For this problem we will assume that there are $N = 1000$ people in the population with $I(0) = 1$ person initially infected. Your Tasks:

(a) Write a differential equation for the susceptible population assuming that the infection rate is proportional to the product of the sizes of the susceptible and infected populations

$$\frac{dS}{dt} = \_\_\_\_\_.$$

(let the proportionality constant be α)

(b) Write a differential equation for the infected population knowing that susceptible people are becoming infected and that infected people are recovering at a constant rate

$$\frac{dI}{dt} = \_\_\_\_\_.$$

(let the proportionality constant be β)

(c) Since the total population is fixed in size and only contains the three categories we know that $S + I + R = N$ and $\frac{dN}{dt} = 0$. Hence, the differential equations that you wrote in parts (a) and (b) are sufficient for modeling the three populations. Write numerical code that generates a plot of the three populations over time. Fully explore the parameters α and β and provide several plots that show the different dynamics of the problem.

▲

Problem 5.24 (H1N1 Problem)**.** The H1N1 virus, also known as the "bird flu", is a particularly virulent bug but thankfully is also very predicable. Once a person is infected they are infectious for 9 days. Assume that a closed population of $N = 1500$ people (like

a small college campus) starts with exactly 1 infected person and hence the remainder of the population is considered susceptible to the virus. Furthermore, once a person is recovered they have an immunity that typically lasts longer than the outbreak. Mathematically we can model an H1N1 outbreak of this kind using 11 compartments: susceptible people (S), 9 groups of infected people (I_j for $j = 1, 2, \cdots, 9$), and recovered people (R). Write and numerically solve a system of 11 differential equations modeling the H1N1 outbreak assuming that susceptible people become infected at a rate proportional to the product of the number of susceptible people and the total number of infected people. You may assume that the initial infected person is on the first day of their infection and determine and unknown parameters using the fact that 1 week after the infection starts there are 10 total people infected. ▲

Problem 5.25 (Pain Management)**.** When a patient undergoing surgery is asked about their pain the doctors often ask patients to rate their pain on a subjective 0 to 10 scale with 0 meaning no pain and 10 meaning excruciating pain. After surgery the unmitigated pain level in a typical patient will be quite high and as such doctors typically treat with narcotics. A mathematical model (inspired by THIS article and THIS paper) of a patient's subjective pain level as treated pharmaceutically by three drugs is given as:

$$\begin{aligned}
\frac{dP}{dt} &= -\left(k_0 + k_1 D_1 + k_2 D_2 + k_3 D_3\right) P + k_0 u \\
\frac{dD_1}{dt} &= -k_{D_1} D_1 + \sum_{j=1}^{N_1} \delta(t - \tau_{1,j}) \\
\frac{dD_2}{dt} &= -k_{D_2} D_2 + \sum_{j=1}^{N_2} \delta(t - \tau_{2,j}) \\
\frac{dD_3}{dt} &= -k_{D_3} D_3 + \sum_{j=1}^{N_3} \delta(t - \tau_{3,j})
\end{aligned}$$

where

- P is a patient's subjective pain level on a 0 to 10 scale,
- D_i is the amount of the i^{th} drug in the patient's bloodstream,
  - D_1 is a long-acting opioid
  - D_2 is a short-acting opioid
  - D_3 is a non-opioid
- k_0 is the relaxation rate to baseline pain without drugs,
- k_i is the impact of the i^{th} drug on the relaxation rate,
- u is the patient's baseline (unmitigated) pain,

- k_{D_i} is the elimination rate of the i^{th} drug from the bloodstream,
- N_i is the total number of the i^{th} drug doses taken, and
- $\tau_{i,j}$ are the time times the patient takes the i^{th} drug.

Implement this model with parameters $u = 8.01$, $k_0 = \ln(2)/2$, $k_1 = 0.319$, $k_2 = 0.184$, $k_3 = 0.201$, $k_{D_1} = \ln(0.5)/(-10)$, $k_{D_2} = \ln(0.5)/(-4)$, and $k_{D_3} = \ln(0.5)/(-4)$. Take the initial pain level to be $P(0) = 3$ with no drugs on board. Assume that the patient begins dosing the long-acting opioid at hour 2 and takes 1 dose periodically every 24 hours. Assume that the patient begins dosing the short-acting opioid at hour 0 and takes 1 dose periodically every 12 hours. Finally assume that the patient takes 1 dose of the non-opioid drug every 48 hours starts at hour 24. Of particular interest are how the pain level evolves over the first week out of surgery and how the drug concentrations evolve over this time.
Other questions:

- What does this medication schedule do the patient's pain level?
- What happens to the patient's pain level if he/she forgets the non-opioid drug?
- What happens to the patient's pain level if he/she has a bad reaction to opioids and only takes the non-opioid drug?
- What happens to the dynamics of the system if the patient's pain starts at 9/10?
- In reality, the unmitigated pain u will decrease in time. Propose a differential equation model for the unmitigated pain that will have a stable equilibrium at 3 and has a value of 5 on day 5. Add this fifth differential equation to the pain model and examine what happens to the patient's pain over the first week. In this model, what happens after the first week if the narcotics are ceased?

▲

5.4 Additional ODE Exercises

Problem 5.26. Test the Euler, Midpoint, and Runge Kutta methods on the differential equation

$$y' = \lambda(y - \cos(t)) - \sin(t) \quad \text{with} \quad y(0) = 1.5.$$

Find the exact solution by hand using the method of undetermined coefficients and note that your exact solution will involve the parameter λ. Produce log-log plots for the error between your numerical solution and the exact solution for $\lambda = -1$, $\lambda = -10$, $\lambda = -10^2$, $\ldots$, $\lambda = -10^6$. In other words, create 7 plots (one for each λ) showing how each of the 3 methods performs for that value of λ at different values for Δt. ▲

Problem 5.27. Write code to solve the boundary valued differential equation

$$y'' = \cos(t)y' + \sin(t)y \quad \text{with} \quad y(0) = 0 \quad \text{and} \quad y(1) = 1.$$

▲

Chapter 6

Numerical Partial Differential Equations

Partial differential equations (PDEs) are differential equations involving the partial derivatives of an unknown multivariable function. In many cases we are interested in ultimately solving PDEs in terms of our usual three spatial dimensions along with an extra dimension for time. Since PDEs require a strong background in the notions of vector calculus we'll start there.

6.1 Quick Review – Main Ideas from Vector Calculus

Let's start with some basic review of multivariable calculus.

Problem 6.1. With your partner answer each of the following questions. The main ideas in this problem *should* be review from multivariable calculus. If you and your partner are stuck then ask another group.

(a) What is a partial derivative (explain geometrically)

(b) What is the gradient of a function? What does it tell us physically or geometrically? If $u(x,y) = x^2 + \sin(xy)$ then what is ∇u?

(c) What is the divergence of a vector-valued function? What does it tell us physically or geometrically? If $F(x,y) = \left\langle \sin(xy), x^2 + y^2 \right\rangle$ then what is $\nabla \cdot F$?

(d) If u is a function of x, y, and z then what is $\nabla \cdot \nabla u$?

(e) What is the divergence theorem? (ok . . . go ahead and use the internet for this one) Be able to explain what you find.

▲

Now that you've realized that you don't recall most of your multivariable calculus, let's simply recap.

Definition 6.2 (Definitions from Multivariable Calculus)**.** The following are a few of the primary definitions and theorems from multivariable calculus.

- The **del** or **grad** operator, ∇, is a vector operator

$$\nabla = \left\langle \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right\rangle.$$

- The **gradient** of a multivariable function $u(x,y,z)$ is the vector

$$\nabla u = \left\langle \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z} \right\rangle.$$

- The **divergence** of a vector valued function $F = \langle F_1, F_2, F_3 \rangle$ is the scalar

$$\nabla \cdot F = \frac{\partial F_1}{\partial x} + \frac{\partial F_2}{\partial y} + \frac{\partial F_3}{\partial z}.$$

- The **curl** of a vector valued function $F = \langle F_1, F_2, F_3 \rangle$ is the vector

$$\nabla \times F = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_1 & F_2 & F_3 \end{vmatrix} = \left\langle \frac{\partial F_3}{\partial y} - \frac{\partial F_2}{\partial z}, -\left(\frac{\partial F_3}{\partial x} - \frac{\partial F_1}{\partial z} \right), \frac{\partial F_2}{\partial x} - \frac{\partial F_1}{\partial y} \right\rangle.$$

- The **Laplacian** of a multivariable function $u(x,y,z)$ is the scalar

$$\nabla \cdot \nabla u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}.$$

- The **divergence theorem** states that the flux of a vector field out of closed body is the same as the integral of the divergence of the vector field within the body.

$$\iint F \cdot n dA = \iiint \nabla \cdot F dV.$$

6.2 An Intuitive Introduction to some Common PDEs

To build intuition for partial differential equations we'll first start with your intuition with ordinary differential equations. Let's consider the really simple ODE $y' = y$. We can verbalize this problems with the phrase *the rate of change of y is equal to the current value of y*, and we can use this phrase, along with our understanding of linear approximation from Calculus, to build intuition about how the value of y propogates in time. If we start with a value of $y(0) = 0.5$ then after 1 unit of time passes we have a reasonable guess for the value of y based on our understanding of slope:

$$y(1) \approx 0.5 + (1)(0.5) = 1.$$

Similarly we can propogate forward 1 unit of time again to get

$$y(2) \approx 1 + 1(1) = 2,$$

and we can continue this to get

Time	0	1	2	3	4	$\cdots$
y	0.5	1	2	4	8	$\cdots$

We also know that this isn't quite correct since the solution to the differntial equation is $y(t) = 0.5e^t$. In Figure 6.1 we can see, however, that our intuitive guesses get us somewhat close to the basic behavior of the differential equation but clearly miss the finer details associated with the exact curvature.

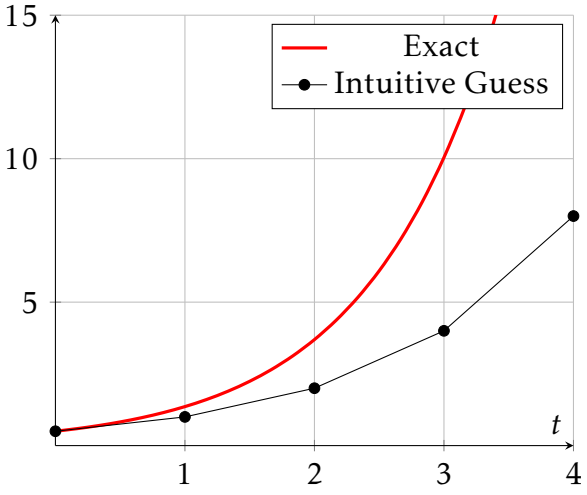


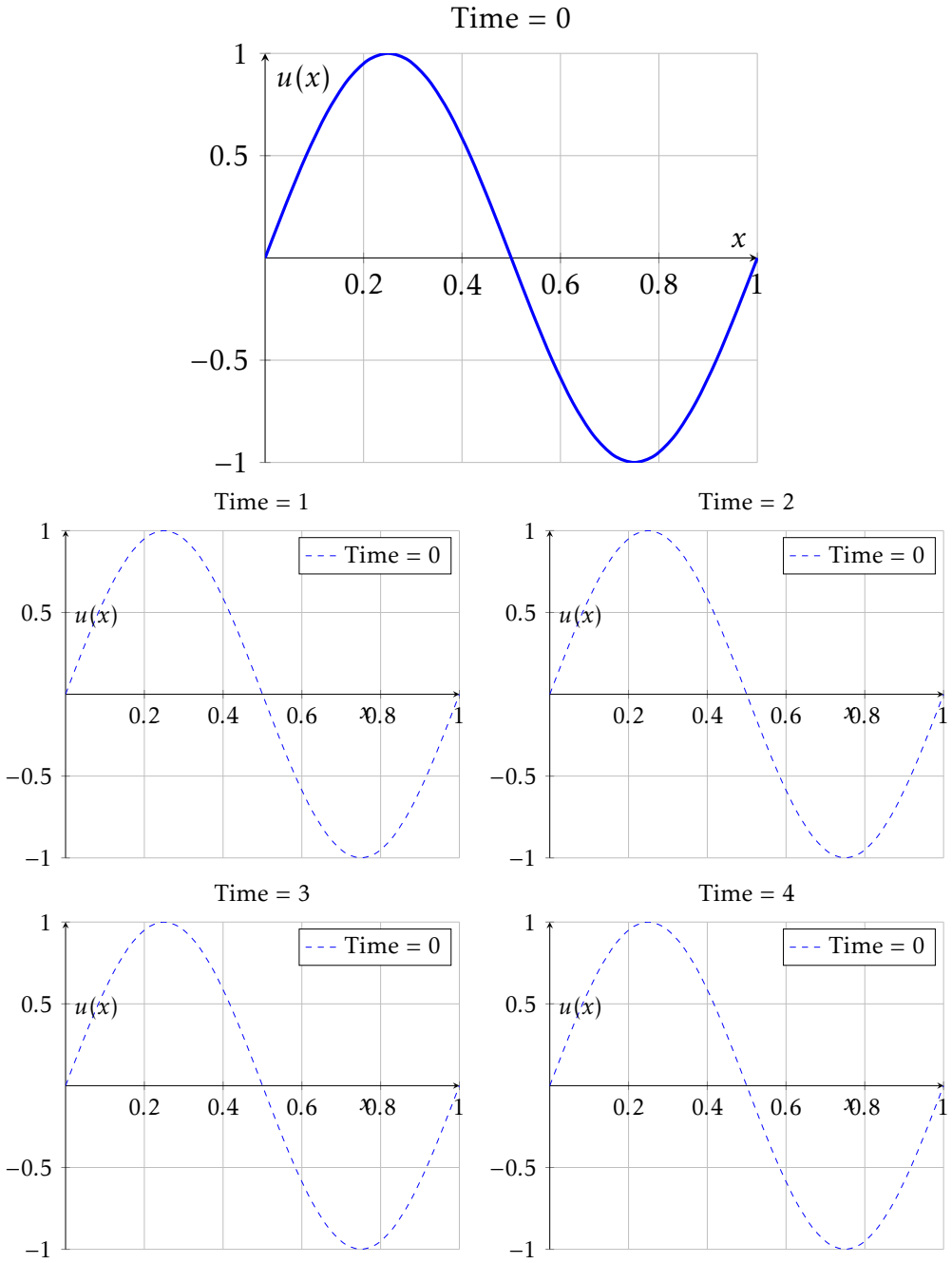
Figure 6.1. The analytic solution to $y' = y$ and the points given by calculus intuition.

We're going to use this same idea to build intuition for some of the most basic partial differential equations. In these partial differential equations we have two variables: t = time and x = a single spatial dimension.

Problem 6.3. Let $u(t,x)$ be the concentration of a quantity at time t and spatial location x. The quantity might be something like heat or chemical concentration. We will let $x \in [0,1]$ and we assume that at time $t = 0$ we have $u(0,x) = \sin(2\pi x)$ as shown in the plot below. Use the phrase

> *the time rate of change of concentration is equal to the concavity of the concentration function*

to give several plots showing how the concentration evolves in time. The initial condition function $u(0,x)$ is shown in each plot for reference. Think of this as creating four frames in an animation.



▲

Problem 6.4. Which of the following differential equations corresponds to the phrase *"the time rate of change of concentration is equal to the concavity of the concentration function"*?

$$\frac{\partial u}{\partial t} = -\frac{\partial u}{\partial x}, \qquad \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad \text{or} \quad \frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

Explain your reasoning clearly. ▲

Problem 6.5. Based on your answer to Problem 6.3 you should now have an intuitive sense for the behavior of the solution to the partial differential equation that you identified in Problem 6.4. This equation is called the **Heat Equation** or the **Diffusion Equation**. Give context to the physical process that is being described by this equation. ▲

Problem 6.6. How would your answer to Problem 6.3 change if we were to modify the heat equation to

$$\frac{\partial u}{\partial t} = 0.5\frac{\partial^2 u}{\partial x^2}?$$

What about

$$\frac{\partial u}{\partial t} = 2\frac{\partial^2 u}{\partial x^2}?$$

▲

Problem 6.7. The 1D Heat Equation is given as

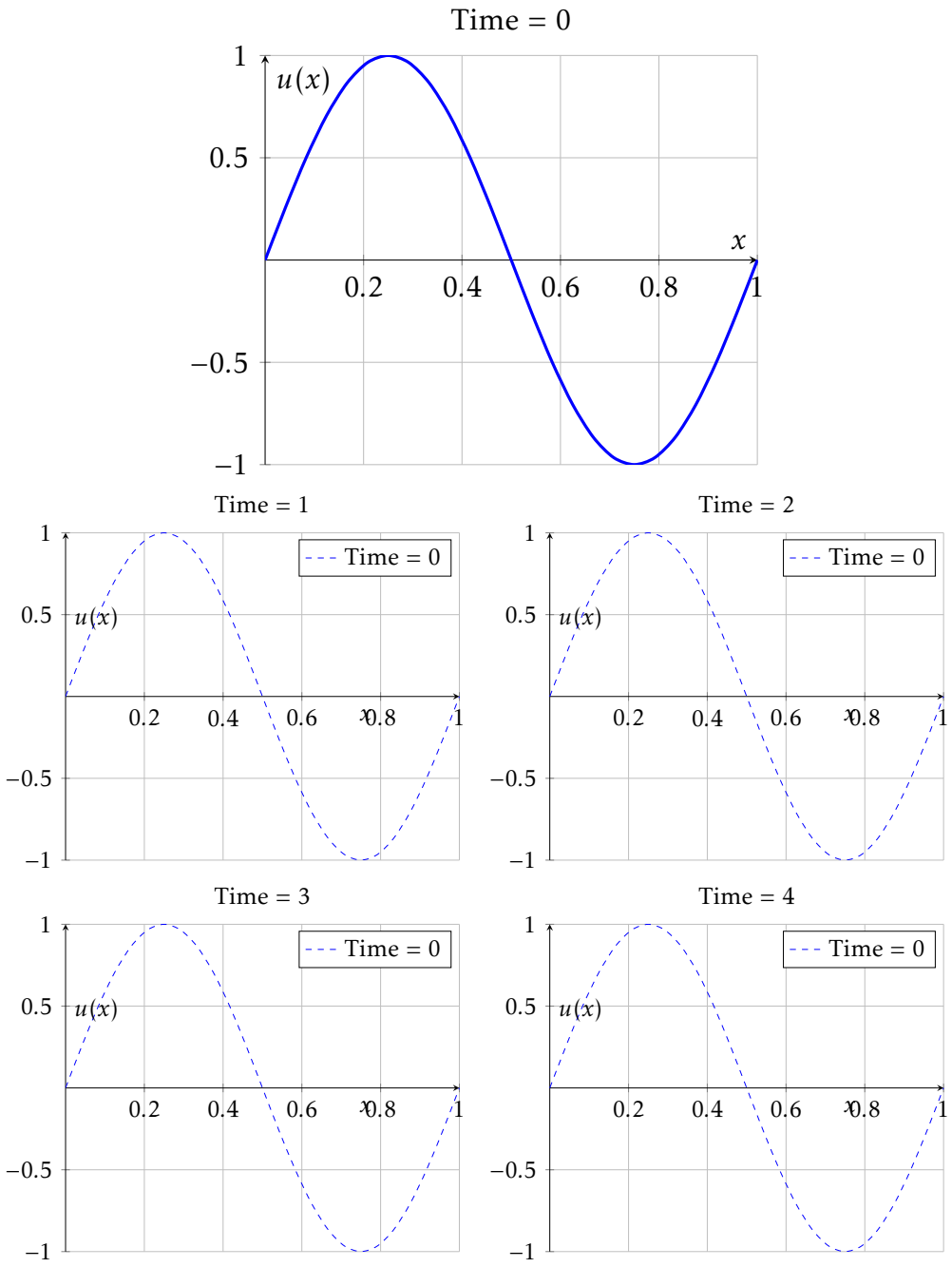
$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}.$$

What does the parameter D control in terms of the physics of the problem? ▲

Problem 6.8. Let $u(t,x)$ be the position of a string or cable at time t and spatial location x. We will let $x \in [0,1]$ and we assume that at time $t = 0$ we have $u(0,x) = \sin(2\pi x)$ as shown in the plot below. Use the phrase

the acceleration of each point on the string is equal to the concavity of the string

to give several plots showing how the concentration evolves in time. The initial condition function $u(0,x)$ is shown in each plot for reference. Think of this as creating four frames in an animation.



▲

Problem 6.9. Which of the following differential equations corresponds to the phrase *"the acceleration of each point on the string is equal to the concavity of the string"*?

$$\frac{\partial u}{\partial t} = -\frac{\partial u}{\partial x}, \qquad \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad \text{or} \quad \frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

Explain your reasoning clearly. ▲

Problem 6.10. Based on your answer to Problem 6.8 you should now have an intuitive sense for the behavior of the solution to the partial differential equation that you iden-

tified in Problem 6.9. This equation is called the **Wave Equation**. Give context to the physical process that is being described by this equation. ▲

Problem 6.11. How would your answer to Problem 6.8 change if we were to modify the wave equation to

$$\frac{\partial^2 u}{\partial t^2} = 0.5\frac{\partial^2 u}{\partial x^2}?$$

What about

$$\frac{\partial^2 u}{\partial t^2} = 2\frac{\partial^2 u}{\partial x^2}?$$

▲

Problem 6.12. The 1D Wave Equation is given as

$$\frac{\partial^2 u}{\partial t^2} = \alpha^2\frac{\partial^2 u}{\partial x^2}.$$

What does the parameter α^2 control in terms of the physics of the problem? ▲

6.3 Analytic Solutions to Linear PDEs

If I were to claim that $y = f(t) = 7e^{3t}$ is a solution to the ordinary differential equation $\frac{dy}{dt} = 3y$ with $y(0) = 7$ then you could easily check that my claim were true by checking the initial condition

$$f(0) = 7e^{3\cdot 0} = 7 \quad \checkmark$$

and checking that the function satisfies the differential equation

$$y' = 3 \cdot 7e^{3t} = 3y \quad \checkmark.$$

Let's do the same for some partial differential equations.

Example 6.13. Consider the diffusion equation $u_t = ku_{xx}$. This is known as the "heat" or "diffusion" equation as we saw in the previous section, and the function u can be thought of as the temperature of some heated object.
Question: Is the function $u(x,t) = x^2 - t^3$ a valid solution for this differential equation? If so, what is the value of the constant k?
Solution:
We will check this by taking a few derivatives.

$$\frac{\partial u}{\partial t} = -3t^2$$
$$\frac{\partial^2 u}{\partial x^2} = 2.$$

As we can see, there is no constant value of k that makes the equation $-3t^2 = 2k$ valid. Therefore, the function $u(x,t) = x^2 - t^2$ is not a solution to the heat equation.

Problem 6.14. Consider the diffusion equation $T_t = kT_{xx}$ where $T(x,t)$ is the temperature of a long thin metal rod at time t (in seconds) and spatial location x (in meters).

(a) What are the units of the constant k?

(b) For each of the following functions, test whether it is an analytical solution to this PDE by taking the first derivative with respect to time, the second derivative with respect to position, and substituting them into this equation to see if we get an identity. If $k = 3$, which of these functions is a solution? Be able to defend your answer.

(1) $T(x,t) = 4x^3 + 6t^2$
(2) $T(x,t) = 7x + 5$
(3) $T(x,t) = 8x^2 t$
(4) $T(x,t) = e^{3t+x}$
(5) $T(x,t) = 6e^{3t+x} + 5x - 2$

(6) $T(x,t) = e^{-3t} + \sin(x)$

(7) $T(x,t) = e^{3t}\sin(x)$

(8) $T(x,t) = e^{-3t}\sin(x)$

(9) $T(x,t) = 5e^{-3t}\sin(x) + 6x + 7$

(10) $T(x,t) = -4e^{-3t}\sin(x) + 3t + 2$

(11) $T(x,t) = e^{-2t}\sin(3x)$

(12) $T(x,t) = e^{-12t}\cos(3x)$

(13) $T(x,t) = e^{-12t}\cos(3x) + 4x^2 + 8$

(14) $T(x,t) = e^{-75t}\cos(5x)$

(15) $T(x,t) = 9e^{-75t}\cos(5x) + 2x + 7$

▲

Problem 6.15. Consider the diffusion equation $T_t = kT_{xx}$, and suppose that $k = 4$. For each of the following functions, find the value of the parameter a that will make the function solve the PDE, by taking derivatives and substituting them into the equation.

(a) $T(x,t) = 6e^{-8t}\sin(ax)$

(b) $T(x,t) = -5e^{38t}\cos(ax)$

(c) $T(x,t) = 3e^{at}\sin(5x)$

(d) $T(x,t) = 7e^{at}\cos(2x)$

(e) $T(x,t) = ae^{-36t}\cos(3x)$

(f) $T(x,t) = ae^{-4t}\cos(6x)$

▲

In your study of ordinary differential equations you likely ran into the dissatisfying realization that you could simply *guess* the answer by knowing a little bit of calculus. As much as it might make you uncomfortable, this is a totally legitimate technique: guess the form of the answer, put it into the differential equation, and see what has to happen in order for your guess to be valid.

Problem 6.16. Consider the diffusion equation

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2}.$$

(a) Reading from left-to-right, the partial differential equation says that the derivative of some function of t is related to that same function. If you had to guess the *type* of function, what would you guess and why?

(b) Reading from right-to-left, the partial differential equation says that the second derivative of some function of x is related to that same function. If you had to guess the *type* of function, what would you guess and why?

(c) Based on your guesses from parts (a) and (b), what type of function would think is a reasonable solution for the differential equation? Why?

▲

Problem 6.17. Find an analytical solution to the diffusion equation that satisfies the following:

(a) Our domain is $-10 \leq x \leq 10$ and $-20 \leq y \leq 20$, with both measured in meters.

(b) The thermal diffusivity is a constant k.

(c) The temperature around the edges is 20 degrees C.

(d) At the first instant of time, the temperature reaches a minimum at the center of 5 degrees C.

▲

Based on what you have explored in the previous few problems you should now have a reasonable guess for the functional forms of solutions to the diffusion equation are.

Theorem 6.18. If the function $u(x,t)$ solves the diffusion equation $u_t = ku_{xx}$ in one spatial dimension then $u(x,t)$ likely takes the form

$$u(x,t) = \_\_\_e^{(\_\_\_)} \sin(\_\_\_)$$

or

$$u(x,t) = \_\_\_e^{(\_\_\_)} \cos(\_\_\_).$$

Problem 6.19. Prove that your hypotheses in the previous theorem are indeed solutions to the diffusion equation $u_t = ku_{xx}$ for some constant k. ▲

Problem 6.20. As a sanity check, pick reasonable values for your constants and create an animation showing how your function evolves in time. Does your function show the behavior expected of the 1D diffusion equation?
Hint: Think of this as a cooling metal rod ... does your solution look plausible for this physical scenario? ▲

Theorem 6.21. If the function $u(x,y,t)$ solves the diffusion equation $u_t = k(u_{xx} + u_{yy})$ in two spatial dimensions then $u(x,y,t)$ likely takes the form

$$u(x,y,t) = \_\_\_\_\_$$

(Hint: There might be several answers)

Problem 6.22. Prove that your hypotheses in the previous theorem are indeed solutions to the 2D diffusion equation $u_t = k(u_{xx} + u_{yy})$ for some constant k. ▲

Problem 6.23. As a sanity check, pick reasonable values for your constants and create an animation showing how your function evolves in time. Does your function show the behavior expected of the 2D diffusion equation?
Hint: Think of this as a cooling metal plate ... does your solution look plausible for this physical scenario? ▲

Problem 6.24. Consider the wave equation

$$\frac{\partial^2 W}{\partial t^2} = c^2 \frac{\partial^2 W}{\partial x^2}.$$

(a) Reading from left-to-right, the partial differential equation says that the second derivative of some function of t is related to that same function. If you had to guess the *type* of function, what would you guess and why?

(b) Reading from right-to-left, the partial differential equation says that the second derivative of some function of x is related to that same function. If you had to guess the *type* of function, what would you guess and why?

(c) Based on your guesses from parts (a) and (b), what type of function would think is a reasonable solution for the differential equation? Why?

▲

Problem 6.25. Consider the wave equation $W_{tt} = c^2(W_{xx} + W_{yy})$ where $W(x,y,t)$ is the height (in centimeters) of a wave at time t in seconds and spatial location (x,y) (each in centimeters).

(a) What are the units of the constant c?

(b) For each of the following functions, test whether it is an analytical solution to this PDE by substituting the derivatives into the equation. If $c = 2$, which of these functions is a solution?

(1) $W(x,y,t) = 3x + 2y + 5t - 6$
(2) $W(x,y,t) = 3x^2 + 2y^2 + 5t^2 - 6$
(3) $W(x,y,t) = \sin(2x) + \cos(3y) + \sin(4t)$
(4) $W(x,y,t) = \sin(2x)\cos(3y)\sin(4t)$
(5) $W(x,y,t) = \sin(3x)\cos(4y)\sin(10t)$
(6) $W(x,y,t) = -6\sin(3x)\cos(4y)\sin(10t) + 2x - 3y + 9 - 12$
(7) $W(x,y,t) = \cos(7x)\cos(3y)\cos(12t)$

(8) $W(x,y,t) = \cos(5x)\sin(12y)\cos(26t)$

▲

Problem 6.26. Find an analytical solution to the wave equation that satisfies the following

(a) Our domain is $0 \leq x \leq 5$ and $-6 \leq y \leq 6$, with both measured in meters.

(b) The wave speed is a constant c.

(c) The function equals zero at the edges.

(d) The initial velocity is zero everywhere.

(e) At the first instant of time, the height reaches a maximum at the center of 2 cm.

▲

Theorem 6.27. If the function $u(x,t)$ solves the 1D wave equation $u_{tt} = c^2 u_{xx}$ then $u(x,t)$ likely has the functional form

$$u(x,t) = \_\_\_\_\_.$$

Problem 6.28. Prove your hypotheses from the previous theorem. ▲

Problem 6.29. As a sanity check, pick reasonable values for your constants and create an animation showing how your function evolves in time. Does your function show the behavior expected of a wave equation?
Hint: Think of this as a vibrating guitar string ... does your solution look plausible for this physical scenario? ▲

Theorem 6.30. If the function $u(x,y,t)$ solves the 2D wave equation $u_{tt} = c^2\left(u_{xx} + u_{yy}\right)$ then $u(x,y,t)$ likely has the functional form

$$u(x,y,t) = \_\_\_\_\_.$$

Problem 6.31. Prove your hypotheses from the previous theorem. ▲

Problem 6.32. As a sanity check, pick reasonable values for your constants and create an animation showing how your function evolves in time. Does your function show the behavior expected of a wave equation?
Hint: Think of this as a vibrating drum head ... does your solution look plausible for this physical scenario? ▲

6.4 Boundary Conditions

When we were solving ODEs we typically needed initial conditions to tell us where the solutions starts at time 0. Since PDEs require both spatial and temporal information we need to tell the differential equation how to behave both at time zero and on the boundaries of the domain.

Definition 6.33. Let's say that we want to solve the 1D heat equation $u_t = ku_{xx}$ on the domain $x \in [0,1]$.

- The initial condition is a function $\eta(x)$ where $u(0,x) = \eta(x)$. In other words, we are dictating the value of u at every point x at time $t = 0$.
- The boundary conditions are restrictions for how the solution behaves at $x = 0$ and $x = 1$ (for this problem).
  - If the value of the solution u at the boundary is either a fixed value or a fixed function of time then we call the boundary condition a **Dirichlet boundary condition.** For example, $u(t,0) = 1$ and $u(t,1) = 5$ are Dirichlet boundary conditions for this problem. They state that the value of the temperature is fixed at these points.
  - If the value of the solution u depends on the rate of change of u at the boundary then we call the boundary condition a **Neumann boundary condition.** For example, $\frac{\partial u}{\partial x}(t,0) = 0$ and $\frac{\partial u}{\partial x}(t,1) = 0$ are Neumann boundary conditions for this problem. They state that the flux of temperature is fixed at the boundaries.

Let's play with a couple problems that should help to build your intuition about boundary conditions in PDEs. Again, we will do this graphically instead of numerically.

Problem 6.34. Consider solving the heat equation $u_t = u_{xx}$ in 1 spatial dimension. It will be helpful to reconsider Problem 6.3 for this problem.

(a) In Problem 6.3 we didn't explicitly state the boundary conditions. What type of boundary conditions were they? How can you tell?

(b) What if we take the initial condition for the 1D heat equation to be $u(0,x) = \cos(2\pi x)$ and enforce the conditions $\left.\frac{\partial u}{\partial x}\right|_{x=0} = 0$ and $u(t,1) = 1$. What types of boundary conditions are these? Draw a collection of pictures showing the expected evolution of the heat equation with these boundary conditions.

▲

Problem 6.35. Consider solving the wave equation $u_{tt} = u_{xx}$ in 1 spatial dimension. It will be helpful to reconsider Problem 6.8 for this problem.

(a) In Problem 6.8 we didn't explicitly state the boundary conditions. What type of boundary conditions were they? How can you tell?

(b) What if we take the initial condition for the 1D wave equation to be $u(0,x) = \cos(2\pi x)$ and enforce the conditions $\left.\frac{\partial u}{\partial x}\right|_{x=0} = 0$ and $u(t,1) = 1$. What types of boundary conditions are these? Draw a collection of pictures showing the expected evolution of the heat equation with these boundary conditions.

▲

An important lesson when solving partial differential equations is that if you get the boundary conditions wrong then the solution to your problem is meaningless. The next two problems should help you to understand some of the basic scenarios that we might wish to solve with the heat and wave equation.

Problem 6.36. For each of the following situations propose meaningful boundary conditions for the 1D or 2D heat equation.

(a) A thin metal rod 1 meter long is heated to 100°C on the left end and is cooled to 0°C on the right end. We model the heat transport with the 1D heat equation $u_t = u_{xx}$. What are the appropriate boundary conditions?

(b) A thin metal rod 1 meter long is insulated on the left end so that the heat flux through that end is 0. The rod is held at a constant temperature of 50°C on the right end. We model the heat transport with the 1D heat equation $u_t = u_{xx}$. What are the appropriate boundary conditions?

(c) In a soil-science lab a column of packed soil is insulated on the sides and cooled to 20°C at the bottom. The top of the column is exposed to a heat lamp that cycles periodically between 15°C and 25°C and is supposed to mimic the heating and cooling that occurs during a day. We model the heat transport within the column with the 1D heat equation $u_t = u_{xx}$. Wat are the appropriate boundary conditions?

(d) A thin rectangular slab of concrete is being designed for a sidewalk. Imagine the slab as viewed from above. We expect the right-hand side to be heated to 50°C due to radiant heating from the road and the left-hand side to be cooled to approximately 20°C due to proximity to a grassy hillside. The top and bottom of the slab are insulated with a felt mat so that the flux of heat through both ends is zero. We model the heat transport with the 2D heat equation $u_t = u_{xx} + u_{yy}$. What are the appropriate boundary conditions?

▲

Problem 6.37. For each of the following situations propose meaningful boundary conditions for the 1D and 2D wave equation.

(a) A guitar string is held tight at both ends and plucked in the middle. We model the vibration of the guitar string with the 1D wave equation $u_{tt} = u_{xx}$. What are the appropriate boundary conditions?

(b) A rope is stretched between two people. The person on the left holds the rope tight and doesn't move. The person on the right wiggles the rope in a periodic fashion completing one full oscillation per second. We model the waves in the rope with the 1D wave equation $u_{tt} = u_{xx}$. What are the appropriate boundary conditions?

(c) A rubber membrane is stretched taught on a rectangular frame. The frame is held completely rigid while the membrane is stretched from equilibrium and then released. We model the vibrations in the membrane with the 2D wave equation $u_{tt} = u_{xx} + u_{yy}$. What are the appropriate boundary conditions?

▲

6.5 Numerical Solutions of The Heat Equation

In this section we'll use a technique called *the finite difference method* to find numerical approximations to the heat equation

$$u_t = D\nabla \cdot \nabla u + f(x).$$

Recall that this equation governs the diffusive process of heat diffusion.

In one spatial dimension the heat equation can be written as $u_t = ku_{xx} + f(x)$ and in two spatial dimensional it can be written as $u_t = D\left(u_{xx} + u_{yy}\right) + f(x,y)$. The function f is called a forcing term and in the case of thermal diffusion it is an external source of heat in the system. We'll let $f(x) = 0$ for the majority of this section for simplicity, but you can modify any of the code that you write in this section to include a forcing term.

6.5.1 1D Heat Equation

Problem 6.38. Now we would like to consider the time dependent heat equation

$$u_t = D\nabla \cdot \nabla u$$

in 1 spatial dimension. Note that D is the diffusivity (the rate of diffusion) so in terms of physical problems, if D is small then the diffusion occurs slowly and if D is large then the diffusion occurs quickly.

In 1 spatial dimension, the heat equation is simply

$$u_t = Du_{xx}$$

and we can approximate the derivatives with an Euler-type approximation of the time and a central difference in space:

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = D\left(\frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{\Delta x^2}\right).$$

Here we are taking $U_i^n \approx u(t_n, x_i)$ (superscripts represent time step and subscripts represent spatial steps). Rearranging we see that

$$U_i^{n+1} = U_i^n + \frac{D\Delta t}{\Delta x^2}\left(U_{i+1}^n - 2U_i^n + U_{i-1}^n\right). \tag{6.1}$$

Implement (6.1) in MATLAB to approximate the solution to the following problem:

$$\text{Solve: } u_t = 0.5u_{xx} \quad \text{with} \quad x \in (0,1),\, u(0,x) = \sin(2\pi x),\, u(t,0) = 0, \text{ and } u(t,1) = 0.$$

▲

Problem 6.39. You may have noticed in the previous problem that you will have terribly unstable solutions for certain choices of Δx and Δt. Set $D = 1$ in the previous problem and experiment with choices for Δx and Δt to find where (6.1) gives a stable numerical solution to the heat equation. For each choice of Δx and Δt report the value of $\frac{\Delta t}{\Delta x^2}$. ▲

Theorem 6.40 (Stability of Finite Differences for the Heat Equation)**.** Consider the 1D Heat Equation $u_t = Du_{xx}$. In the finite difference scheme for the 1D heat equation

$$U_i^{n+1} = U_i^n + \frac{D\Delta t}{\Delta x^2}\left(U_{i+1}^n - 2U_i^n + U_{i-1}^n\right)$$

the solution will be stable for

$$\frac{D\Delta t}{\Delta x^2} < \_\_\_\_\_$$

Proof. For a detailed proof of this fact we need to use a method called *Von Neumann Analysis*. See a detailed proof HERE. □

Problem 6.41. Modify your 1D heat equation code to solve the following problems. For each be sure to classify the type of boundary conditions given.

(a) Solve $u_t = 0.5u_{xx}$ with $x \in (0,1)$, $u(0,x) = x^2$, $u(t,0) = 0$ and $u(t,1) = 1$.

(b) Solve $u_t = 0.5u_{xx}$ with $x \in (0,1)$, $u(0,x) = x^2$, $u_x(t,0) = 0$ and $u(t,1) = 1$.

(c) Solve $u_t = 0.5u_{xx}$ with $x \in (0,1)$, $u(0,x) = \sin(2\pi x)$, $u(t,0) = 0$ and $u(t,1) = \sin(5\pi t)$.

(d) Solve $u_t = 0.5u_{xx} + x^2$ with $x \in (0,1)$, $u(0,x) = \sin(2\pi x)$, $u(t,0) = 0$ and $u(t,1) = 0$.

▲

6.5.2 Stabilized 1D Heat Equation – The Crank Nicolson Method

The next problem addresses the issue of stability in solving the heat equation with the finite difference method. There are MANY different techniques for dealing with stability issues in numerical partial differential equations, and surprisingly enough we can never completely beat these issues. That is to say that no matter how sophisticated of a method you use there will always be some region where the parameters of the problem give rise to instability.

Problem 6.42. The instabilities of the heat equation with and Euler-type time discretization and a central differencing scheme is maddening. Thankfully, we can avoid this issue almost entirely by considering an implicit scheme called the *Crank-Nicolson* method. In this method we approximate the temporal derivative with an Euler-type approximation, but we approximate the spatial derivative as the average of the central difference at the old time step and the central difference at the new time step. That is:

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} = \frac{1}{2}\left[D\left(\frac{U_{j+1}^n - 2U_j^n + U_{j+1}^n}{\Delta x^2}\right) + D\left(\frac{U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j+1}^{n+1}}{\Delta x^2}\right)\right].$$

Letting $r = D\Delta t/(2\Delta x^2)$ we can rearrange to get

$$-rU_{j-1}^{n+1} + (1+2r)U_j^{n+1} - rU_{j+1}^{n+1} = rU_{j-1}^n + (1-2r)U_j^n + rU_{j+1}^n.$$

This can now be viewed as a system of equations. Let's build this system carefully and then write MATLAB code to solve the heat equation from the previous problems with the Crank-Nicolson method. For this problem we will assume fixed Dirichlet boundary conditions on both the left- and right-hand sides of the domain.

(a) First let's write the equations for several values of j.

$$\begin{aligned}
(j=2): &\quad -rU_1^{n+1} + (1+2r)U_2^{n+1} - rU_3^{n+1} = rU_1^n + (1-2r)U_2^n + rU_3^n \\
(j=3): &\quad -rU_2^{n+1} + (1+2r)U_3^{n+1} - rU_4^{n+1} = rU_2^n + (1-2r)U_3^n + rU_4^n \\
(j=4): &\quad -rU_3^{n+1} + (1+2r)U_4^{n+1} - rU_5^{n+1} = rU_3^n + (1-2r)U_4^n + rU_5^n \\
&\qquad\qquad\qquad\qquad\qquad\qquad \vdots \quad\quad \vdots \\
(j=N-1): &\quad -rU_{N-2}^{n+1} + (1+2r)U_{N-1}^{n+1} - rU_N^{n+1} = rU_{N-2}^n + (1-2r)U_{N-1}^n + rU_N^n
\end{aligned}$$

where N is the number of spatial points.

(b) The first and last equations can be simplified since we have the Dirichlet boundary conditions. Therefore for $j = 2$ we can rearrange to move U_1 to the right-hand side since it is fixed for all time. Similarly for $j = N-1$ we can move U_N to the right-hand side since it is fixed for all time. Rewrite these two equtions.

(c) Verify that the left-hand side of the equations that we have built in parts (a) and (b) can be written as the following matrix-vector product:

$$\begin{pmatrix}
(1+2r) & -r & 0 & 0 & 0 & \cdots & 0 \\
-r & (1+2r) & -r & 0 & 0 & \cdots & 0 \\
0 & -r & (1+2r) & -r & 0 & \cdots & 0 \\
\vdots & & & \ddots & & & 0 \\
0 & \cdots & & & 0 & -r & (1+2r)
\end{pmatrix}
\begin{pmatrix}
U_2^{n+1} \\ U_3^{n+1} \\ U_4^{n+1} \\ \vdots \\ U_{N-1}^{n+1}
\end{pmatrix}$$

(d) Verify that the right-hand side of the equations that we built in parts (a) and (b) can be written as

$$\begin{pmatrix}
(1-2r) & r & 0 & 0 & 0 & \cdots & 0 \\
r & (1-2r) & r & 0 & 0 & \cdots & 0 \\
0 & r & (1-2r) & r & 0 & \cdots & 0 \\
\vdots & & & \ddots & & & 0 \\
0 & \cdots & & & 0 & r & (1-2r)
\end{pmatrix}
\begin{pmatrix}
U_2^n \\ U_3^n \\ U_4^n \\ \vdots \\ U_{N-1}^n
\end{pmatrix}
+
\begin{pmatrix}
2rU_1 \\ 0 \\ \vdots \\ 0 \\ 2rU_N
\end{pmatrix}$$

(e) Now for the wonderful part! The entire system of equations from part (a) can be written as

$$AU^{n+1} = BU^n + D.$$

What are the matrices A and B and what are the vectors U^{n+1}, U^n, and D?

(f) To solve for U^{n+1} at each time step we simply need to do a linear solve:

$$U^{n+1} = A^{-1}(BU^n + D).$$

Of course, we will never do a matrix inverse on a computer so in MATLAB this code looks like

```
Un = U(n,2:end-1)'; % note the transpose
RHS = B*Un + D;
temp = A \ RHS; % note the \ doing the work of the inverse
U(n+1, 2:end-1) = temp;
```

(g) Finally. Write code to solve the 1D Heat Equation implementing the Crank Nicolson method described in this problem. The setup of your code should be largely the same as for the regular heat equation. You will need to construct the matrices A and B as well as the vector D. Then your time stepping loop will contain the code from part (f) of this problem.

▲

6.5.3 2D Heat Equation

For the 2D heat equation we notice that the only new part of the PDE is the $\nabla \cdot \nabla u$ term in place of the 1 dimensional u_{xx} term. Recall that $\nabla \cdot \nabla u = u_{xx} + u_{yy}$ so we can use what we know about approximating second derivatives to approximate the right-hand side of the heat equation with

$$D\nabla \cdot \nabla u \approx D\left[\frac{U^n_{j+1,k} - 2U^n_{j,k} + U^n_{j-1,k}}{\Delta x^2} + \frac{U^n_{j,k+1} - 2U^n_{j,k} + U^n_{j,k-1}}{\Delta y^2}\right].$$

Assuming that $\Delta x = \Delta y$ and simplifying we can write the right-hand side of the 2D heat equation as

$$D\nabla \cdot \nabla u \approx \frac{D}{\Delta x^2}\left[U^n_{j+1,k} + U^n_{j,k+1} - 4U^n_{j,k} + U^n_{j-1,k} + U^n_{j,k-1}\right].$$

Notice that we had to invent a bit of notation in the process. The superscript, just as before, stands for the time step. The subscripts stand for the two spatial indices. More specifically, $U^n_{j,k} \approx u(t_n, x_j, y_k)$.

Problem 6.43. In Figure 6.2 you will see a schematic of the domain $\Omega = (0,1) \times (0,1)$ with homogeneous Dirichlet boundary conditions. Write code to solve the 2D Dirichlet problem

$$\text{Solve: } u_t = D\nabla \cdot \nabla u \quad \text{in} \quad x \in \Omega$$

with

$$u(0,x,y) = \sin(\pi x)\sin(\pi y)$$

subject to the boundary conditions in the figure.

For simplicity we suggest that you take $\Delta x = \Delta y$. You should also be very careful of the stability conditions for the heat equation. ▲

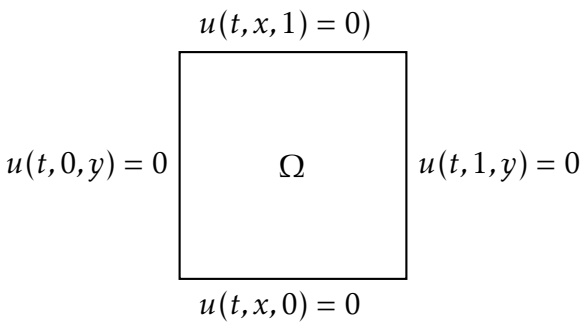


Figure 6.2. Dirichlet boundary conditions for a 2D Poisson equation.

Problem 6.44. Repeat the previous exercise with different boundary conditions (both Dirichlet and Neumann) and with different domains (rectangular instead of square). For a rectangular domain it will likely be necessary to have different values for Δx and for Δy. Be prepared to present your solutions to your classmates. ▲

6.6 Numerical Solutions of The Wave Equation

Problem 6.45. The problems that we've dealt with thus far all model natural diffusion processes: heat transport, molecular diffusion, etc. Another interesting physical phenomenon is that of wave propagation. In 1 spatial dimension the *wave equation* is

$$u_{tt} = \alpha^2 u_{xx} \tag{6.2}$$

where α is the stiffness of the wave. With Dirichlet boundary conditions we can think of this as the behaviour of a guitar string after it has been plucked.

Let's write MATLAB code to numerically solve this problem:
Consider $u_{tt} = 2u_{xx}$ in $x \in (0,1)$ with $u(0,x) = x(1-x)$, $u_t(0,x) = 0$, and $u(t,0) = u(t,1) = 0$ and $\alpha = 5$. We can discretize the derivatives as

$$u_{tt}(t_{n+1}, x_j) \approx \frac{U_j^{n+1} - 2U_j^n + U_j^{n-1}}{\Delta t^2}$$
$$u_{xx}(t_n, x_j) \approx \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{\Delta x^2}$$

▲

Problem 6.46. There is a natural stability question waiting to be asked about the discretization of the 1D wave equation. Ask and answer this question. ▲

Problem 6.47. In Figure 6.3 you will see a schematic of the domain $\Omega = (0,1) \times (0,1)$ with homogeneous Dirichlet boundary conditions. Write code to solve the 2D Dirichlet problem

$$\text{Solve: } u_{tt} = \alpha^2 \nabla \cdot \nabla u \quad \text{in} \quad x \in \Omega \quad \text{with} \quad u(0,x,y) = \sin(2\pi(x-0.5))\sin(2\pi(y-0.5))$$

subject to the boundary conditions in the figure.

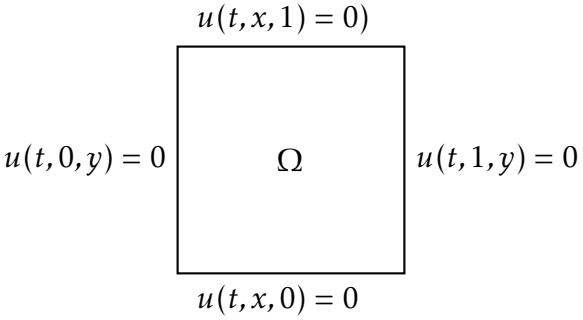


Figure 6.3. Dirichlet boundary conditions for a 2D Poisson equation.

For simplicity we suggest that you take $\Delta x = \Delta y$. You should also be very careful of the stability conditions for the heat equation. ▲

6.7 Traveling Waves

Problem 6.48. A traveling wave can be modeled by the PDE

$$u_t + v u_x = 0$$

where v is the speed of the wave propogation and $u(t,x)$ is the height of the wave. Write a numerical solve for the traveling wave problem on $x \in (0,\infty)$ with initial condition $u(0,x) = \exp\left(-\frac{(x-1)^2}{0.1}\right)$, boundary condition $u(t,0) = 0$, and $a = 1$.

Solve this problem with and Euler-type time step and

1. centered differences in space, and
2. upwind differences in space.

where the "wind" is from left to right. Plot both solutions on top of each other. What do you notice about the behavior of the solutions? Neither of these solutions should actually give a traveling wave, but that is what is expected out of the solution.

Note about the analytic solution: If $\eta(x)$ is the initial condition for $u_t + v u_x = 0$ then $u(t,x) = \eta(x-vt)$ is the analytic solution to the PDE. You should check this using the chain rule. ▲

Problem 6.49. Three ways to fix the issues seen in the previous problem are the "Leapfrog" scheme, the "Lax-Friedrichs" scheme, and the "Lax-Wendroff" scheme:

$$\text{Leapfrog: } \frac{U_j^{n+1} - U_j^{n-1}}{2\Delta t} = -v\frac{U_{j+1}^n - U_{j-1}^n}{2\Delta x} \tag{6.3}$$

$$\text{Lax-Friedrichs: } \frac{U_j^{n+1} - \frac{1}{2}\left(U_{j+1}^n + U_{j-1}^n\right)}{\Delta t} = -v\frac{U_{j+1}^n - U_{j-1}^n}{2\Delta x} \tag{6.4}$$

$$\text{Lax-Wendroff: } U_j^{n+1} = U_j^n - \frac{v\Delta t}{2\Delta x}\left(U_{j+1}^n - U_{j-1}^n\right) + \left(\frac{v^2\Delta t^2}{2\Delta x^2}\right)\left(U_{j-1}^n - 2U_j^n + U_{j+1}^n.\right) \tag{6.5}$$

Implement all of these schemes and discuss stability and consistency. ▲

6.8 The Laplace and Poisson Equations – Steady State PDEs

Problem 6.50. Consider a 1-dimensional rod that is infinitely thin and has unit length. For the sake of simplicity assume the following:

- the specific heat of the rod is exactly 1 for the entire length of the rod,
- the temperature of the left end is held fixed at $u(0) = 1$,
- the temperature of the right end is held fixed at $u(1) = 0$, and
- the temperature has reached a steady state.

(assume that the temperatures are *reference temperatures* instead of absolute temperatures).

Since there are no external sources of heat we model the steady-state heat profile with Laplace's equation (D.14). Write this equation in terms of 1-dimensional spatial derivatives and solve for the temperature profile by hand. ▲

Problem 6.51. Devise a way to approximate the temperature profile from the previous problem numerically. Recall that we already know how to build numerical second derivatives. Your method will eventually involve solving a system of linear equations. ▲

Problem 6.52. Now we will solve the steady state temperature profile problem assuming that there is an external source of heat. This means that we need to solve the 1D Poisson equation (D.15). Take $f(x) = 5\sin(2\pi x)$, $u(0) = 2$ and $u(1) = 0.5$.

$$\text{Solve:} \quad u_{xx} = -5\sin(2\pi x) \quad \text{on} \quad x \in (0,1) \quad \text{with} \quad u(0) = 2 \quad \text{and} \quad u(1) = 0.5$$

First do so by discretizing the domain with very few points so we can write the system of equations by hand. Write your code with the number of points as a parameter so you can later change it to several hundred points. ▲

Problem 6.53. Generalize the previous problem with a MATLAB function that solves the 1D Poisson boundary valued equation:

$$\text{Solve: } u_{xx} = -f(x) \quad \text{on} \quad x \in (x_0, x_n) \quad \text{with} \quad u(x_0) = \alpha \quad \text{and} \quad u(x_n) = \beta.$$

```
1 function [x,u] = Poisson1D(f , xmin , xmax , num_interior_pts , BCleft , BCright)
```

Test your code with a known function $f(x)$.

Note: when we are using fixed values for the boundary conditions these are called "Dirichlet boundary conditions." ▲

Problem 6.54. The previous problems only account for Dirichlet boundary conditions (fixed boundary conditions). We would now like to modify our Poisson solution to allow for a Neumann condition: where we know the derivative of u at one of the boundaries. The statement of the problem is as follows:

$$\text{Solve: } u_{xx} = -f(x) \quad \text{on} \quad x \in (x_0, x_n) \quad \text{with} \quad \left.\frac{du}{dx}\right|_{x_0} = \alpha \quad \text{and} \quad u(x_n) = \beta.$$

Write a function to solve this problem:

```
function [x,u] = Poisson1D_Neumann(f , xmin , xmax , num_interior_pts , NBC , DBC)
```

Write a MATLAB script to solve the Neumann problem with $f(x) = 2e^{-0.1x}$, $u'(0) = 0$, and $u(1) = 0$. ▲

Problem 6.55. Now let's ramp up our discussion of the Poisson equation to two spatial dimensions. In Figure 6.4 you will see a schematic of the domain $\Omega = (0,1) \times (0,1)$ with Dirichlet boundary conditions. With the help of your instructor, write code to solve the 2D Dirichlet problem

$$\text{Solve: } \nabla \cdot \nabla u = -f(x) \quad \text{in} \quad x \in \Omega \quad \text{with} \quad f(x,y) = 20\exp\left(-\frac{(x-0.5)^2 + (y-0.5)^2}{0.05}\right)$$

subject to the boundary conditions in the figure.

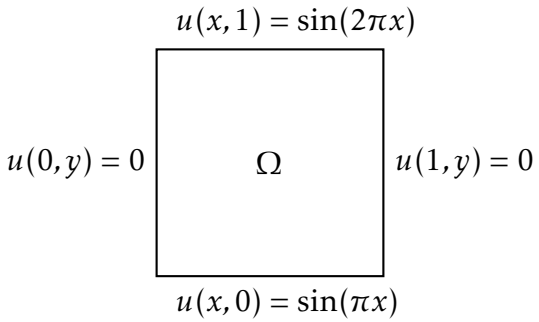


Figure 6.4. Dirichlet boundary conditions for a 2D Poisson equation.

▲

6.9 Exercises

Problem 6.56. For every one of the scenarios described in Problem 6.36, propose a sensible initial condition and solve the problem numerically. Notice that the diffusion coefficient, D, is set to 1 for all of these models. You are welcome to change D if you see that it is necessary. ▲

Problem 6.57. For every one of the scenarios described in Problem 6.37, propose a sensible initial condition and solve the problem numerically. Notice that the tension coefficient, α^2, is set to 1 for all of these models. You are welcome to change α^2 if you see that it is necessary. ▲

Problem 6.58. In this problem we will solve a more realistic 1D heat equation. We will allow the diffusivity to change spatially, so $D = D(x)$ and we want to solve

$$u_t = (D(x)u'(x))'$$

on $x \in (0,1)$ with Dirichlet boundary conditions $u(t,0) = u(t,1) = 0$ and initial condition $u(0,x) = \sin(2\pi x)$. In this problem we will take $D(x)$ to be the parabola $D(x) = x(1-x)$. We start by doing some calculus to rewrite the differential equation:

$$u_t = D(x)u''(x) + D'(x)u'(x).$$

Your jobs are:

(a) Describe what this choice of $D(x)$ might mean physically in the heat equation.

(b) Write an explicit scheme to solve this problem by using centered differences for the spatial derivatives and an Euler-type discretization for the temporal derivative. Write a clear and thorough explanation for how you are doing the discretization as well as a discussion for the errors that are being made with each discretization.

(c) Write a MATLAB script to find an approximate solution to this problem.

(d) Write a clear and thorough discussion about how your will choose Δx and Δt to give stable solutions to this equation.

(e) Graphically compare your solution to this problem with a heat equation where D is taken to be the constant average diffusivity found by calculating $D_{ave} = \int_0^1 D(x)dx$. How does the changing diffusivity change the shape of the solution?

▲

Problem 6.59 (The Diffusing Logo)**.** In a square domain create a function $u(0,x,y)$ that looks like your college logo. The simplest way to do this might be to take a photo of the logo, crop it to a square, and use the `imread` command to read in the image. Use this function as the initial condition for the heat equation on a square domain with homogeneous Dirichlet boundary conditions. Numerically solve the heat equation and show an animation for what happens to the logo as time evolves. ▲

Problem 6.60 (The Wiggling Logo)**.** Repeat the previous exercise but this time solve the wave equation with the logo as the initial condition. ▲

Problem 6.61. Let u be a function modeling a mobile population that in an environment where it has a growth rate of r% per year with a carrying capacity of K. If we were only worried about the size of the population we could solve the differential equation

$$\frac{du}{dt} = ru\left(1 - \frac{u}{K}\right),$$

but there is more to the story.

Hunters harvest h% of the population per year so we can append the differential equation with the harvesting term "$-hu$" to arrive at the ordinary differential equation

$$\frac{du}{dt} = ru\left(1 - \frac{u}{K}\right) - hu.$$

Since the population is mobile let's make a few assumptions about the environment that they're in and how the individuals move.

- Food is abundant in the entire environment.
- Individuals in the population like to spread out so that they don't interfere with each other's hunt for food.
- It is equally easy for the individuals to travel in any direction in the environment.

Clearly some of these assumptions are unreasonable for real populations and real environments, but let's go with it for now. Given the nature of these assumptions we assume that a diffusion term models the spread of the individuals in the population. Hence, the PDE model is

$$\frac{\partial u}{\partial t} = ru\left(1 - \frac{u}{K}\right) - hu + D\nabla \cdot \nabla u.$$

(a) Use any of your ODE codes to solve the ordinary differential equation with harvesting. Give a complete description of the parameter space.

(b) Write code to solve the spatial+temporal PDE equation on the 2D domain $(x, y) \in [0,1]\times[0,1]$. Choose an appropriate initial condition and choose appropriate boundary conditions.

(c) The third assumption isn't necessary true for rough terrain. The true form of the spatial component of the differential equation is $\nabla \cdot (D(x,y)\nabla u)$ where $D(x,y)$ is a multivariable function dictating the ease of diffusion in different spatial locations. Propose a (non-negative) function $D(x,y)$ and repeat part (b) with this new diffusion term.

▲

Problem 6.62. Consider the time-independent partial differential equation $-\varepsilon u_{xx} + u_x = 1$ on the domain $x \in (0,1)$ with boundary conditions $u(0) = u(1) = 0$ and parameter ε with $0.001 < \varepsilon < 1$. Write code to solve this boundary valued problem and provide plots of your numerical solution for various values of ε. ▲

Problem 6.63. Suppose that we have a concrete slab that is 10 meters in length, with the left boundary held at a temperature of 75° and the right boundary held at a temperature of 90°. Assume that the thermal diffusivity of concrete is about $k = 10^{-5}$ m^2/s. Assume that the initial temperature of the slab is given by the function $T(x) = 75 + 1.5x - 20\sin(\pi x/10)$. In this case, the temperature can be analytically solved by the function $T(x,t) = 75 + 1.5x - 20\sin(\pi x/10)e^{-ct}$ for some value of c.

(a) Working by hand (no computers!) test this function by substituting it into the 1D heat equation and verifying that it is indeed a solution. In doing so you will be able to find the correct value of c.

(b) Write numerical code to solve this 1D heat equation. The output of your code should be an animation showing how the error between the numerical solution and the analytic solution evolve in time.

▲

Problem 6.64. Adobe houses, typically built in desert climates, are known for their great thermal efficiency. The heat equation

$$\frac{\partial T}{\partial t} = \frac{k}{c_p \rho} \nabla \cdot \nabla T,$$

where c_p is the specific heat of the adobe, ρ is the mass density of the adobe, and k is the thermal conductivity of the adobe, can be used to model the heat transfer through the adobe from the outside of the house to the inside. Clearly, the thicker the adobe walls the better, but there is a trade off to be considered:

- it would be prohibitively expensive to build walls so think that the inside temperature was (nearly) constant, and
- if the walls are too thin then the cost is low but the temperature inside has a large amount of variability.

Your Tasks:

(a) Pick a desert location in the southwestern US (New Mexico, Arizona, Nevada, or Southern California) and find some basic temperature data to model the outside temperature during typical summer and winter months.

(b) Do some research on the cost of building adobe walls and find approximations for the parameters in the heat equation.

(c) Use a numerical model to find the optimal thickness of an adobe wall. Be sure to fully describe your criteria for optimality, the initial and boundary conditions used, and any other simplifying assumptions needed for your model.

▲

Appendix A

Writing and Projects

This class is writing intensive and as such you will be writing several papers. This appendix is designed to give you helpful hints for the writing.

A.1 The Paper

Write your work in a formal paper that is typed and written at a college level using appropriate mathematical typesetting. This paper must be organized into sections, starting with a Summary or Abstract, followed by an Introduction, and ending with Conclusions and References. Each of these sections should begin with these headings in a large bold font (using the LaTeX `\section` and `\subsection` commands where appropriate). Within sections I would suggest using subheadings to further organize things and aid in clarity.

A.2 Figures and Tables

Figures and tables are a very important part of these projects. Never break tables or figures across pages. Each figure or table must fit completely onto one sheet of paper. If your table has too much information to fit onto one sheet, divide it into two separate tables. In addition to the figure, this sheet must contain the figure number, the figure title, and a brief caption; for example "Figure 2: A plot of heating oil price versus time from Model F1. We see that the effects of seasonal variation in price are dominated by random fluctuations." In the text, refer to the figure/table by its number. For example in the text you might say "As we see in Figure 2, in model F1 the effects of seasonal variation in price are dominated by random fluctuations." Every figure and table must be mentioned (by number) somewhere in the text of your paper. If you do not refer to it anywhere in the text, then you do not need it, and subsequently it will be ignored.

Think of figures and tables as containing the evidence that you are using to support the point you are trying to make with your paper. Always remember that the purpose of a figure or a table is to show a pattern, and when someone looks at the figure this pattern should be obvious. Figures should not be cluttered and confusing: They should make things very clear. Always label the horizontal and vertical axes of plots.

A.3 Writing Style

The real goal of mathematical writing is to take a complex and intricate subject and to explain it so simply and so plainly that the results are obvious for everyone. I want your paper to demonstrate that you not only did the right calculations, but that you understand what you did and why your methods worked.

Write this paper using the word 'we' instead of 'I.' For example: "First we calculate the sample mean." This 'we' refers to you and the reader as you guide the reader through the work that you've done. Also please avoid the word "prove" or "proof." Numerical methods usually deal with approximations, not absolutes, and in mathematics we reserve the word "prove" for things that are absolutely 100% certain. Often the word "test" can be used instead of "prove."

A.4 Tips For Writing Clear Mathematics

At this point you know just enough mathematics and LaTeX to be dangerous. It is time to clean up your act and teach you some of the formalities about writing mathematics. The following sections stem from a document that I give all of my upper level mathematics students.

Some tips for writing clear mathematics can be found here:
http://www.ohio.edu/people/mohlenka/goodproblems/goodstudent.html

A.4.1 Audience

The following suggestions will help you to submit properly written homework solutions, papers, projects, labs, and proofs. The goal of any writing is to clearly communicate ideas to another person. Remember that the other person may even be your future self. When you write for another person, you will need to include ideas that may be in your mind but omitted when you are writing a rough draft on scratch paper. If you keep your intended audience in mind, you will produce higher quality work. For a course in mathematics, the intended audience is usually your instructor, your classmates, or a student grader. This implies that your task is to show that you thoroughly understand your solution. Consequently, you should routinely include more details.

One rule of thumb must prevail throughout all mathematical writing:

> When you read a mathematical solution out loud it needs to make sense as grammatically correct English writing. This includes reading all of the symbols with the proper language.

Don't forget that mathematics is a language that is meant to be spoken and read just like works of literature!

A.4.2 How To Make Mathematics Readable – 10 Things To Do

1. When read aloud, the text and formulas must form complete English sentences. If you get lost, say aloud what you mean and write it down.
2. Every mathematical statement must be complete and meaningful. Avoid fragments.
3. If a statement is something you want to prove or something you assume temporarily, e.g., to discuss possible cases or to get a contradiction, say so clearly. Otherwise, anything you put down must be a true statement that follows from your up front assumptions.
4. Write what your plan is. It will also help you focus on what to do.
5. There must be sufficient detail to verify your argument. If you do not have the details, you have no way of knowing if what you wrote is correct or not. Keep the level of detail uniform.
6. If you are not sure, even slightly, about something, work out the details on the side with utmost honesty, going as deep as necessary. Decide later how much detail to include.
7. Do not write irrelevant things just to fill paper and show you know something.
8. Your argument should flow well. Make the reading easy. Logical and intuitive notation matters.
9. Keep in mind what the problem is and make sure you are not doing something else. Many problems are solved and proofs done simply by understanding what is what.
10. The state of mind when you are inventing a solution is completely different from the mode of work when you are writing the solution down and verifying it. Learn how to go back and forth between the two. The act of typing your solutions forces you to iterate over this process but remember that the process isn't done until you've proofread what you typed.

A.4.3 Some Writing Tips

Use sentences: The feature that best distinguishes between a properly written mathematical exposition and a piece of scratch paper is the use (or lack) of sentences. Properly written mathematics can be read in the same manner as properly written sentences in any other discipline. Sentences force a linear presentation of ideas. They provide the connections between the various mathematical expressions you use. This linearity will also keep you from handing in a page with randomly scattered computations with no connections. The sentences may contain both words and mathematical expressions. Keep in mind that the way your present your solution may be different than the way that you arrived at the solution. It is imperative that you work problems on scratch paper first before formally writing the solution.

The following extract illustrates these ideas.

> Let n be odd. Then Definition 3.10 indicates that there does not exist an integer, k, such that $n = 2k$. That is, n is not divisible by 2. The Quotient Remainder theorem asserts that n can be uniquely expressed in the form $n = 2q + r$, where r is an integer with $0 \leq r < 2$. Thus, $r \in \{0, 1\}$. Since n is not divisible by 2, the only admissible choice is $r = 1$. Thus, $n = 2q + 1$, with q an integer.

Read out loud: The sentences you write should read well out loud. This will help you to avoid some common mistakes. Avoid sentences like:

> Suppose the graph has n number of vertices.
> The piggy bank contains n amount of coins.

If you substitute an actual number for n (such as 4 or 6) and read these out loud they will sound wrong (because they are wrong). The variable n is already a numeric variable so it should be read just like an actual number. The correct versions are:

> Suppose the graph has n vertices.
> (Read this as: "Suppose the graph has en vertices".)
> The piggy bank contains n coins.

You should also avoid sentences like:

> From the previous computation $x = 5$ is true.

A better way to say this is:

> From the previous computation we see that $x = 5$.

When you read the equal sign as part of the sentence you realize that there is no reason to write "is true".

= is NOT a conjunction: The mathematical symbol = is an assertion that the expression on its left and the expression on its right are equal. Do not use it as a connection between steps in a series of calculations. Use words for this purpose. Here is an example that misuses the = symbol when solving the equation $3x = 6$:

$$\textbf{Incorrect!} \qquad 3x = \underbrace{6 = \frac{3x}{3}}_{\text{false!}} = \frac{6}{3} = x = 2$$

One proper way to write his is:
$3x = 6$. Dividing both sides by 3 leads to $\frac{3x}{3} = \frac{6}{3}$, which simplifies to $x = 2$.

" $\Longrightarrow$ " means "implies": The double arrow " $\Longrightarrow$ " means that the statement on the left logically implies the statement on the right. This symbol is often misused in place of the "=" sign.

Do not merge steps: Suppose you need to calculate the final price for a \$20 item with 7% sales tax. One strategy is to first calculate the tax, then add the \$20. Here is an incorrect way to write this.

$$\textbf{Incorrect!} \qquad 20 \cdot 0.07 \underbrace{=}_{false!} 1.4 + 20 \underbrace{=}_{false!} \$21.4.$$

The main problem (besides the magically-appearing dollar sign at the end) is that $20 \cdot 0.07 \neq 1.4 + 20$. The writer has taken the result of the multiplication (1.4) and merged directly into the addition step, creating a lie (since $1.4 \neq 21.4$). The calculations could be written as:

$$\$20 \cdot 0.07 = \$1.40 \text{so the total price is } \$1.40 + \$20 = \$21.40$$

Avoid ambiguity: When in doubt, repeat a noun rather using unspecific words like "it" or "the". For example, in the sentences

> Let G be a simple graph with $n \geq 2$ vertices that is not complete and let G be its complement. Then it must contain at least one edge.

there is some ambiguity about whether "it" refers to G or to the complement of G. The second sentence is better written as "Then G must contain at least one edge".

Use Proper Notation: There are many notational conventions in mathematics. You need to follow the accepted conventions when using notation. For example, A summation or integral symbol always needs something to act on. The expressions

$$\sum_{i=1}^{n} \qquad \int_a^b$$

by themselves are meaningless. The expressions

$$\sum_{i=1}^{n} a_n \qquad \int_a^b f(x)dx$$

have well-understood meanings.

An another example,

$$\underbrace{\lim_{h\to 0} = \frac{2x+h}{2}}_{\text{incorrect!}} = \frac{2x}{2} = x$$

is incorrect. It should be written

$$\lim_{h\to 0} \frac{2x+h}{2} = \frac{2x}{2} = x$$

Parenthesis are important: Parenthesis show the grouping of terms, and the omission of parenthesis can lead to much unneeded confusion. For example,

$$x^2 + 5 \cdot x - 3 \quad \text{is very different than} \quad \left(x^2 + 5\right) \cdot (x - 3).$$

This is very important in differentiation and summation notation:

$$\frac{d}{dx} \sin(x) + x^2 \quad \text{is not the same as} \quad \frac{d}{dx}\left(\sin(x) + x^2\right)$$

$$\sum_{k=1}^{n} 2k + 3 \quad \text{is not the same as} \quad \sum_{k=1}^{n} (2k + 3)$$

Label and reference equations: When you need to refer to an equation later it is common practice to label the equation with a number and then to refer to this equation by that number. This avoids ambiguity and gives the reader a better chance at understanding what you're writing. Furthermore, avoid using words like "below" and "above" since the reader doesn't really know where to look. One implication to this style of referencing is that you should never reference an equation before you define it.

Incorrect:

In the equation below we consider the domain $x \in (-1, 1)$

$$f(x) = \sum_{j=1}^{\infty} \frac{x^n}{n!}.$$

Correct:

Consider the summation

$$f(x) = \sum_{j=1}^{\infty} \frac{x^n}{n!}. \tag{A.1}$$

In (A.1) we are assuming the domain $x \in (-1, 1)$

"Timesing": The act of multiplication should not be called "timesing" as in "I can times 3 and 5 to get 15". The correct version of this sentence is "I can multiply 3 and 5 to get 15". The phrase "3 times 5 is 15", on the other hand, is correct and is likely the root of the confusion. The mathematical operation being performed is not called "timesing". It seems as if this is an unfortunate carry over from childhood when a child hears "3 times 5", sees "3×5", and then incorrectly associates the symbol "$\times$" with the word multiply in the statement "I can multiply 3 and 5 to get 15".

A.4.4 Mathematical Vocabulary

Function: The word function can be used to refer just to the name of a function, such as "The function $s(t)$ gives the position of the particle as a function of time." Or function can refer to both the function name and the rule that describes the function. For example, we could elaborate and say, "The function $s(t) = t23t$ gives the position of the particle as a function of time." Notice that both times the word function is used twice, where the second usage is describing the mathematical nature of the relationship between time and position. (Remember that if position can be described as a function of time, then the position can be uniquely determined from the time.)

Equation: To begin with, an equation must have an equal sign (=), but just having an equal sign isn't enough to deserve the name equation. Generally, an equation is something that will be used to solve for a particular variable, and/or it expresses a relationship between variables. So you might say, "We solved the equation $x + y = 5$ for x to find that $x = 5y$," or you might say "The relationship between the variables can be expressed with the following equation: $xy = 2z$."

Formula: A formula might in fact be an equation or even a function, but generally the word formula is used when you are going to substitute numbers for some or all of the variables. For example, we might say, "The formula for the area of a circle is $A = \pi r^2$. Since $r = 2$ in this case, we find $A = \pi 2^2 = 4\pi$." The bottom line: If you're going to use algebra to solve for a variable, call it an equation. If youre going to use it exactly as it is and just put in numbers for the variables, then call it a formula.

Definition: A definition might be any of the above, but it is specifically being used to define a new term. For example, the definition of the derivative of a function f at a point a is

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

Now this does give us a formula to use to compute the derivative, but we prefer to call this particular formula a definition to highlight the fact that this is what we have chosen the word derivative to mean.

Expression: The word expression is used when there isn't an equal sign. You probably won't need this word very often, but it is used like this: "The factorization of the expression x^2x6 is $(x3)(x + 2)$."

Solve/Evaluate: Equations are solved, whereas functions are evaluated. So you would say, "We solved the equation for x," but you would say "We evaluated the function at $x = 5$ and found the function value to be 26."

Add Subtract vs Plus Minus: The word subtract is used when discussing what needs to be done: "Subtract two from five to get three." Add is used similarly: "Add two and five to get seven." Minus is used when reading a mathematical equation or expression. For example, the equation $xy = 5$ would be read as "x minus y is equal to five". Plus is used similarly. So the equation $x + y = 5$ would be read as "x plus

y is equal to five". Some things we don't say are "We plus 2 and 5 to get 7" or "We minus x from both sides of the equation."

Number/Amount: The word number is used when referring to discrete items, such as "there were a large number of cougars", or "there are a large number of books on my shelf". The word amount is used when referring to something that might come in a pile, such as "that is a huge amount of sand!" or, "I only use a small amount of salt when I cook".

Many/Much: These words are used in much the same way as number and amount, with many in place of number and much in place of amount. For example, we might say, "There aren't as many cougars here as before", or "I don't use as much salt as you do."

Fewer/Less: These are the diminutive analogues of many and much. So, "There are fewer cougars here than before", or "You use less salt than I do."

A.5 Sensitivity Analysis

(This section is paraphrased partly from Dianne O'Leary's book *Computing in Science & Engineering* and partly from Mark Meerschaert's text *Mathematical Modeling*.)

In contrast to to classroom exercises, the real world rarely presents us with a problem in which the data is known with absolute certainty. Some parameters (such as π) we can define with certainty, and others (such as Planck's constant $\hbar$) we know to high precision, but most data is measured and therefore contains measurement error.

Thus, what we really solve isn't the problem we want, but some *nearby* problem, and in addition to reporting the computed solution we really need to report a bound on either

- the difference between the true solution and the computed solution, or
- the difference between the problem we solved and the problem we wanted to solve.

This need occurs throughout computational science. For example,

- If we compute the resonant frequencies of a model of a building, we want to know how these frequencies change if the load within the building changes.
- If we compute the stresses on a bridge, we want to know how sensitive these values are to changes that might occur as the bridge ages.
- If we develop a model for our data and fit the parameters using least squares, we want to know how much the parameters would change if the data were wiggled within the uncertainty limits.

One of the best ways to measure the sensitivity of a parameter k on an output x is to measure the ratio between the relative change in x to the relative change in k. That is, one measure of sensitivity is

$$S = \left| \left(\frac{\Delta x}{x} \right) \Big/ \left(\frac{\Delta k}{k} \right) \right|.$$

Simplifying a bit gives

$$S = \left|\left(\frac{\Delta x}{\Delta k}\right)\cdot\left(\frac{k}{x(k)}\right)\right|$$

where we are now explicitly stating that the output x is a function of k: $x = x(k)$. Taking $\Delta k = \delta$ as well as taking $\Delta x = x(x \pm \delta) - x(k)$ we can rewrite one more time to give

$$S = \left|\left(\frac{x(k \pm \delta) - x(k)}{\delta}\right)\cdot\left(\frac{k}{x(k)}\right)\right|.$$

Notice that we could take the change of x by increasing k by δ or by decreasing k by δ.

The value of δ is related to known or estimated information about how the parameter varies. It is likely that k is a value from some statistical distribution (like a normal distribution) with an approximately known or estimated standard deviation. The value of δ should be related to the standard deviation and some basic statistics can be used to choose the δ for your sensitivity analysis. Recall that if you sample a parameter from the normal distribution then

- roughly 68% of the sampled parameters will be within 1 standard deviation of the mean of the normal distribution, and
- roughly 95% of the sampled parameters will be within 1.96 standard deviations of the mean of the normal distribution.

This means that if k comes from a normal distribution then a very typical choice for δ is 1.96 times the value of the estimated standard deviation (up or down from k). If, on the other hand, the values of the parameter are uniformly distributed then δ can be chosen as the maximum estimated deviation from the mean of the distribution (up or down from k).

Generally:

- If the value of S is approximately 1 then the relative changes are approximately the same and the output is not very sensitive to changes in the parameter.
- If the value of S is less than 1 then the relative changes in the output are less than the changes in the parameter and the output is not sensitive to changes in the parameter.
- Finally, if the value of S is larger than 1 then the relative changes in the output are greater than the changes in the parameter and the output is considered sensitive to changes in the parameter.

A.6 Example of Sensitivity Analysis:

Let's do a more specific example. If we are analyzing the differential equation $P' = kP$ and we estimate that the growth rate is normally distributed with sample mean $k \approx 0.009$ and a standard deviation of $\sigma \approx 0.001$, then we can estimate the sensitivity of the time needed

for the population to double as a function of the growth rate. In this case, the *doubling time* is the output and the *growth rate* is the parameter of interest.

The analytic solution to the differential equation is $P(t) = P_0 e^{kt}$ and the population doubling can be found by solving $2P_0 = P_0 e^{kT_d}$ where T_d is the time to double the population. Using some basic algebra we see that the doubling time as a function of the growth rate is $T_d(k) = \frac{\ln(2)}{k}$. Therefore, to measure the sensitivity of the doubling time to changes in the parameter k we can take $\delta = 1.96 \times 0.001 = 0.00196$. To measure sensitivity in doubling time to an increase in the growth rate we see that S is given as follows:

$$\begin{aligned}
S &= \left|\left(\frac{T_d(k+\delta) - T_d(k)}{\delta}\right)\cdot\left(\frac{k}{T_d(k)}\right)\right| \\
&= \left|\left(\frac{\frac{\ln(2)}{k+\delta} - \frac{\ln(2)}{k}}{\delta}\right)\cdot\left(\frac{k}{\frac{\ln(2)}{k}}\right)\right| \\
&= \left|\left(\frac{k\ln(2) - (k+\delta)\ln(2)}{\delta k(k+\delta)}\right)\cdot\left(\frac{k^2}{\ln(2)}\right)\right| \\
&= \left|\frac{k}{k+\delta}\right| \\
&= \frac{0.009}{0.009 + 1.96 \times 0.001} = \frac{0.009}{0.01096} \approx 0.8212
\end{aligned}$$

To measure sensitivity in doubling time to a decrease in the growth rate we see that S is given as*

$$\begin{aligned}
S &= \left|\left(\frac{T_d(k-\delta) - T_d(k)}{\delta}\right)\cdot\left(\frac{k}{T_d(k)}\right)\right| \\
&= \left|\left(\frac{\frac{\ln(2)}{k-\delta} - \frac{\ln(2)}{k}}{\delta}\right)\cdot\left(\frac{k}{\frac{\ln(2)}{k}}\right)\right| \\
&= \left|\left(\frac{k\ln(2) - (k-\delta)\ln(2)}{\delta k(k-\delta)}\right)\cdot\left(\frac{k^2}{\ln(2)}\right)\right| \\
&= \left|\frac{k}{k-\delta}\right| \\
&\approx \frac{0.009}{0.009 - 1.96 \times 0.001} = \frac{0.009}{0.00704} \approx 1.2784.
\end{aligned}$$

In the present example, the doubling time is not considered to be very sensitive to increases in the growth rate but it is sensitive to decreases in the growth rate. Given that in this simple example we are dealing with an exponential decay function this should also be intuitively *obvious*.

*I'm showing you the algebra here, but it really isn't necessary to show this level of routine algebra in your papers. Only show the algebra and other calculations that are necessary for the reader to understand the work that you're doing.

Appendix B

MATLAB Basics

In this appendix we'll go through a few of the basics in MATLAB. This is by no means meant to be an all-encompassing resource for MATLAB programming. A few more thorough resources for MATLAB are listed here.

- https://www.mathworks.com/help/pdf_doc/matlab/matlab_prog.pdf
- https://www.mathworks.com/products/matlab/examples.html
- https://en.wikibooks.org/wiki/MATLAB_Programming
- http://gribblelab.org/scicomp/scicomp.pdf (this is a personal favorite)

In this appendix we'll give examples of some of the more common coding practices that the reader will run into while working through the exercises and problems in these notes.

B.1 Vectors and Matrices

Example B.1. Write the vectors $\mathbf{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ and $\mathbf{w} = \begin{pmatrix} 4 & 5 & 6 & 7 \end{pmatrix}$ using MATLAB.

Solution:

```
1       v = [1 ; 2 ; 3]
2       w = [4 , 5 , 6 , 7]
3       w = 4:7  % this is shorthand for writing a sequence as a row vector
```

Example B.2. Consider the matrices and vectors

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 5 & 7 \\ 9 & 1 & 3 \\ 5 & 7 & 11 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 4 & 3 & -1 \end{pmatrix}$$

- Calculate the product AB using regular matrix multiplication

```
A = [1 , 2 , 3;
     4 , 5 , 6;
     7 , 8 , 0]
B = [3 , 5 , 7;
     9 , 1 , 3;
     5 , 7 , 11]
Product = A*B
```

- Calculate the element-by-element multiplication of A and B

```
ElementWiseProduct = A .* B
```

- Calculate the inverse of A

```
Ainv = A^(-1)
Ainv = inverse(A) % alternative
```

- Calculate the transpose of B

```
Atranspose = transpose(A)
% or as an alternative:
Atranspose = A'  % actually the conjugate transpose but if A is real then ok
```

- Solve the system of equations $A\mathbf{x} = \mathbf{b}$

```
b = [4 ; 3 ; -1]
x = A \ b
```

Example B.3. Code for a matrix of zeros

```
Z = zeros(5,5) % 5 x 5 matrix of all zeros
```

Example B.4. Code for an identity matrix

```
Ident = eye(5,5) % 5 x 5 identity matrix
```

Example B.5. Code for random matrices.

- random matrix from a uniform distribution on $[0,1]$

```
R = rand(5,5) % random 5 x 5 matrix
```

- random matrix from the standard normal distribution

```
R = randn(5,5) % random 5 x 5 matrix
```

Example B.6. A linearly spaced sequence

```
List = linspace(0,10,100)
% a list of 100 equally spaced numbers from 0 to 10
```

B.2 Looping

A loop is used when a process needs to be repeated several times.

B.2.1 For Loops

A `for` loop is code that repeats across a pre-defined sequence.

Example B.7. Write a loop that produces the squares of the first 10 integers.

```
for j = 1:10
    j^2
end
```

The output of this code will be

```
1
4
9
16
25
36
49
64
81
100
```

Example B.8. Plot the functions $f(x) = \sin(kx)$ for $k = 1, 1.5, 2, 2.5, \ldots, 5$ on the domain $x \in [0, 2\pi]$.

```
x = linspace(0,2*pi,1000);
for k = 1:0.5:5
    plot(x , sin(k*x) )
    hold on
end
```

B.2.2 The While Loop

A `while loop` is a process that only repeats while a conditional statement is true. Be careful with `while loops` since it is possible to create a loop that runs forever.

Example B.9. Build the Fibonnaci sequence up until the last term is greater than 1000.

```
F(1) = 1; % first term
F(2) = 1; % second term
n = 3;
while F(end)<1000
    F(n) = F(n-1) + F(n-2);
    n=n+1;
end
```

Example B.10. An example of a while loop that runs forever.

```
a = 1;
```

```
while a>0
    a=a+1;
end
```

Example B.11. An example of a while loop that runs forever but with a failsafe step that stops the loop after 1000 steps.

```
a = 1;
counter=1;
while a>0
    a=a+1;
    if counter >= 1000
        break
    end
    counter = counter+1;
end
```

B.3 Conditional Statements

Conditional statements are used to check if something is true or false. The output of a conditional statement is a boolean value; true (1) or false (0).

B.3.1 If Statements

Example B.12. Loop over the integers up to 100 and output only the multiples of three.

```
for j = 1:100
    if mod(j,3) == 0
        j
    end
end
```

Example B.13. Check the signs of two function values and determine if they are opposite.

```
f = @(x) x^3*(x-3);
a = 2;
b = 4;
if f(a)*f(b) < 0
```

```
    fprintf('The function values are opposite sign\n')
elseif f(a)*f(b) >0
    fprintf('The function values are the same sign\n')
else
    fprintf('The function values are both zero\n')
end
```

B.3.2 Case-Switch Statements

Example B.14. Evaluate over several cases.

```
n = 3
switch n
    case 1 % if n == 1
        fprintf('n is 1\n')
    case 2 % if n == 2
        fprintf('n is 2\n')
    case 3 % if n == 3
        fprintf('n is 3\n')
end
```

B.4 Functions

A mathematical function has a single output for every input, and in some sense a computer function is the same: one single executed process for each collection of inputs.

Example B.15. Define the function $f(x) = \sin(x^2)$ so that it can accept any type of input (symbol, number, or list of numbers).

```
f = @(x) sin(x.^2) % defines the function
f(3) % evaluates the function at x=3
x=linspace(0,pi,100);
f(x) % evaluates f at 100 points equally spaced from 0 to pi
```

Example B.16. Write a computer function that accepts two numbers as inputs and outputs the sum plus the product of the two numbers.
First write a file with the following contents.

```
function MyOutput = MyFunctionName(a,b)
    MyOutput = a + b + a*b;
end
```

Be sure that the file name is the same as the function name.
Then you can call the function by name in a script or another function.

```
SumPlusProduct = MyFunctionName(3,4)
```

which will output the number 19.

Example B.17. Write a function with three inputs that outputs the sum of the three. The third input should be optional and the default should be set to 5.

```
function AwesomeOutput = SumOfThree(a,b,c)
    if nargin < 3
        c = 5;
    end
    AwesomeOutput = a+b+c;
end
```

You can call this function with

```
SumOfThree(17,23)
```

which will output $17+25+5=47$. Notice that the third input was left off and a 5 was used in its place.

B.5 Plotting

In numerical analysis we are typically plotting numerically computed lists of numbers so as such we will give a few examples of this type of plotting here. We will not, however, give examples of symbolic plotting.

The `plot` command in MATLAB accepts a list of x values followed by a list of y values then followed by color and symbol options.

```
plot(xlist , ylist , color options)
```

Example B.18. Plot the function $f(x) = \sin(x^2)$ on the interval $[0, 2\pi]$ with 1000 equally spaced points. Make the plot color blue.

```
x = linspace(0,2*pi,1000);
f = @(x) sin(x.^2);
plot(x , f(x) , 'b')
```

Alternatively

```
x = linspace(0,2*pi,1000);
y = sin(x.^2);
plot(x, y, 'b')
```

Example B.19. Make a 2×2 array of 4 plots of $f(x) = \sin(kx^2)$ for $k = 1, 2, 3, 4$.

```
x = linspace(0,2*pi,1000);
for k=1:4
    subplot(2,2,k)
    plot(x , sin(k*x.^2) , 'b')
end
```

Example B.20. Plot $f(x) = \sin(kx^2)$ for $k = 1, 2, \ldots, 10$ all on the same plot.

```
x = linspace(0,2*pi,1000);
for k=1:10
    plot(x, sin(k*x.^2))
    hold on % this holds the figure window open so you can write on top of it
end
```

Example B.21. Plot the function $f(x) = e^{-x}\sin(x)$ and put a mark at the local max at $x = \pi/4$.

```
x = linspace(0,2*pi,1000); % set up the domain
f = @(x) exp(-x) .* sin(x);
plot(x,f(x),'b',pi/4,f(pi/4),'ro')
```

B.6 Animations

Example B.22. Plot $f(x) = \sin(kx^2)$ for $k = 1$ to $k = 10$ by small increments with a short pause in between each step.

```
x = linspace(0,2*pi,1000);
for k=1:0.01:10 % 1 to 10 by 0.01
    plot(x, sin(k*x.^2))
    hold on % this holds the figure window open so you can write on top of it
    drawnow % draws the plot
```

```
6       % the last line gives the illusion of animation
7   end
```

Appendix C

LaTeX

In this appendix we give the basics of writing with LaTeX.

C.1 Equation Environments and Cross Referencing

When working with equations is it often times convenient and necessary to cross-reference the equations that you're talking about. A simple example is:

Example C.1. Recall the Pythagorean Theorem: If a and b are the legs of a right triangle and c is the hypotenuse, then

$$a^2 + b^2 = c^2. \tag{C.1}$$

Let $a = 3$ and $b = 4$ in equation (C.1). If that is the case then ...
The LaTeX code for this is

```
Recall the Pythagorean Theorem: If $a$ and $b$ are the legs of a
right triangle and $c$ is the hypotenuse, then
\begin{flalign}
    a^2 + b^2 = c^2
    \label{eqn:pythag}
\end{flalign}

Let $a=3$ and $b=4$ in equation \eqref{eqn:pythag}.
If that is the case then \dots
```

Note in Example C.1 that the equations and the equation reference are part of the sentence. In fact, these are always part of the grammatical structure of your writing.

Other numbered environments include `align`, `flalign`, `eqnarray`, `equation` and several others. The modern convention for LaTeX is to use `align` or `flalign` for all equations. If you want to use one of these environments without numbers then use the *. In otherwords `align*` will align in the same way without numbering the equations. If you only want a number on one line then you can use `\notag` at the beginning of that line.

To align equations use the "align" enviroment, which requires the amsmath package. Align supersedes eqnarray. The ampersands control the vertical alignment:

```
\begin{align}
\frac{\partial{x}}{\partial{s}}&=zx  & \text{with} && x(0)=1\\
\frac{\partial{y}}{\partial{s}}&=x^2y & \text{with} && y(0)=t\\
\frac{\partial{z}}{\partial{s}}&=xyz & \text{with} && z(0)=t^2
\end{align}
```

$$\frac{\partial x}{\partial s} = zx \qquad \text{with} \qquad x(0) = 1 \tag{C.2}$$

$$\frac{\partial y}{\partial s} = x^2 y \qquad \text{with} \qquad y(0) = t \tag{C.3}$$

$$\frac{\partial z}{\partial s} = xyz \qquad \text{with} \qquad z(0) = t^2 \tag{C.4}$$

A few more math-related typesetting examples are included below:

- Inline math with and without numbering

```
\[ \sum_{j=1}^\infty \frac{1}{j^2} = \frac{\pi^2}{6} \]
```

$$\sum_{j=1}^{\infty} \frac{1}{j^2} = \frac{\pi^2}{6}$$

```
\begin{flalign}
    \sum_{j=1}^\infty \frac{1}{j^2} = \frac{\pi^2}{6}
    \label{eqn:sample_equation}
\end{flalign}
```

$$\sum_{j=1}^{\infty} \frac{1}{j^2} = \frac{\pi^2}{6} \tag{C.5}$$

```
\begin{subequations}
    \begin{eqnarray}
        \sin \left( \frac{\pi}{6} \right) &=& \frac{\sqrt{3}}{2}
        \label{eqn:sine} \\
        \cos \left( \frac{\pi}{6} \right) &=& \frac{1}{2}
        \label{eqn:cosine}
    \end{eqnarray}
    \label{eqn:trig}
\end{subequations}
```

$$\sin\left(\frac{\pi}{6}\right) = \frac{\sqrt{3}}{2} \tag{C.6a}$$

$$\cos\left(\frac{\pi}{6}\right) = \frac{1}{2} \tag{C.6b}$$

This second example allows you to cross reference equations like (C.5) using (`\ref{eqn:sample_equation}`) or, more simply, (C.5) using `\eqref{eqn:sample_equation}`. The third set of equations allows for multiple types of references. Like:
The sine equation, (C.6a) (`\eqref{eqn:sine}`), and the cosine equation, (C.6b) (`\eqref{eqn:cosine}`), are grouped together via equation (C.6) (`\eqref{eqn:trig}`).

- Matrices

```
\[ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \]
```
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
\[ \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \]
```
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
\[ \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \]
```
$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$$

- Including basic graphics
Be sure that the graphics file is in the same directory as your TeX file. Your picture should be a *.eps or *.pdf file. If not then some TeX compilers will complain (plus, *.jpg usually looks horrible).

```
\begin{center}
    \includegraphics[width=0.9\columnwidth]{filename.eps}
\end{center}
```

OR

```
\begin{center}
    \includegraphics[height=3in]{filename.eps}
\end{center}
```

There are many options for `\includegraphics`, but these two work for many pictures. Sometimes, though, it is desired to trim an image that you've saved from elsewhere. The basic syntax for `trim` and `clip` is

```
\begin{center}
    \includegraphics[trim = 1cm 2cm 1cm 2cm, clip=true,
        width=0.9\columnwidth]{filename.eps}
\end{center}
```

The four measurements after the trim command are the amount to trim from the left, bottom, right, and top (in that order).

- Leaving white space:
  - horizontal Spacing: `\hspace{0.5in}`
  - vertical Spacing: `\vspace{2in}`

C.2 Tables, Tabular, Figures, Shortcuts, and Other Environments

C.2.1 Tables and Tabular Environments

Tables can be rather annoying in LaTeX, but it is important to get the basics down before moving on.

Example C.2. In this example we want the table to be place *here* [h*], the first column left justified, the middle column centered, and the last column right justified with vertical bars between each column.

```
\begin{center}
    \begin{tabular}[h*]{|l|c|r|}
        \hline
        Title 1 & Title 2 & Title 3 \\ \hline \hline
        Hello & Ni Hao & Bonjour \\ \hline
        good bye & zia jian & adieux \\ \hline
    \end{tabular}
\end{center}
```

Title 1	Title 2	Title 3
Hello	Ni Hao	Bonjour
good bye	zia jian	adieux

In Example C.2 we used the tabular environment. This builds the table. If you want to build a table where there is a caption and the environment *floats* to various parts of the page then you need to use the `table` command.

Example C.3. In this example we build the same table as in Example C.2 but this time we allow it to float and we want a caption. The code is:

```
\begin{table}
    \centering
        \begin{tabular}[h*]{|l|c|r|}
            \hline
            Title 1 & Title 2 & Title 3 \\ \hline \hline
```

Title 1	Title 2	Title 3
Hello	Ni Hao	Bonjour
good bye	zia jian	adieux

Table C.1. This is the amazing table of doom

```
            Hello & Ni Hao & Bonjour \\ \hline
            good bye & zia jian & adieux \\ \hline
        \end{tabular}
    \caption{This is the amazing table of doom}
    \label{tab:MyLabel}
\end{table}
```

C.2.2 Excel To LaTeX

One tool that is often overlooked is the `ExcelToLaTeX` macro for Excel. I'm leaving this one up to you. Google `ExcelToLaTeX` , download it, add it to the macros for your version of Excel, and have fun with it. This tool will allow you to convert Excel-based tables to LaTeX tables.

C.2.3 Figures

The `figure` environment in LaTeX is almost identical to that for `table`. For example:

Example C.4. This figure simply shows a MatLab plot of the sine and cosine functions together in all of their shared glory. The file type was eps, which is notoriously hard to handle on Windows machines and on Overleaf. Be sure to use the `epstopdf` package if you're using epsfile types.

```
\begin{figure}[ht!]
    \centering
    \includegraphics[width=0.7\columnwidth]{SampleFigure.eps}
    \caption{Figure for Example \ref{ex:C3:fig}
    \label{fig:C3:fig}
\end{figure}
```

C.2.4 New Commands: Shortcuts are AWESOME!

You can save yourself a vast amount of typing by defining new commands which meet your specific need. It is easy. The `newcommand` command goes in the preamble (before the `\begin{document}`). The examples that follow are a few handy ones that I've used in the past. The world is your oyster here, so make any shortcut for a LaTeX command that is cumbersome to type.

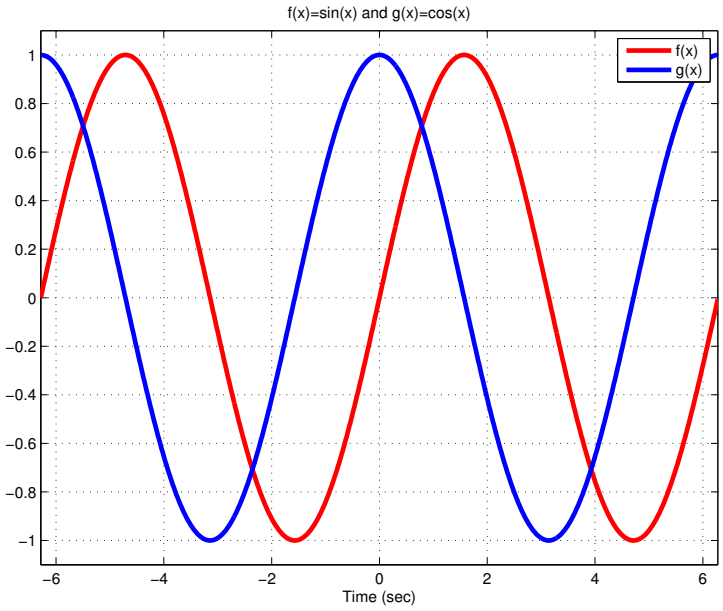


Figure C.1. Figure for Example C.4

- Derivatives

```
\newcommand{\dd}[2] {\frac{d #1}{d #2}}
\newcommand{\ddd}[2] {\frac{d^2 #1}{d #2^2}}

\dd{y}{x}
\ddd{y}{x}
```

The results are:

$$\frac{dy}{dx}$$
$$\frac{d^2y}{dx^2}$$

- Partial derivatives

```
\newcommand{\pd}[2] {\frac{\partial{#1}}{\partial{#2}}}
\newcommand{\pdd}[2] {\frac{\partial^2{#1}}{\partial{#2}^2}}
\newcommand{\pddm}[3] {\frac{\partial^2{#1}}{\partial{#2}\partial{#3}}}

\pd{y}{x}
\pdd{y}{x}
\pddm{y}{x}{z}
```

The results are:

$$\frac{\partial y}{\partial x}$$
$$\frac{\partial^2 y}{\partial x^2}$$
$$\frac{\partial^2 y}{\partial x \partial z}$$

- Some of the common number sets

```
\newcommand{\cc}{\mathbb{C}}
\newcommand{\rr}{\mathbb{R}}
\newcommand{\nn}{\mathbb{N}}
\newcommand{\qq}{\mathbb{Q}}
\newcommand{\zz}{\mathbb{Z}}

\cc \hspace{1cm} \nn \hspace{1cm} \qq \hspace{1cm} \rr \hspace{1cm} \zz
```

The result is:

$$\mathbb{C} \qquad \mathbb{N} \qquad \mathbb{Q} \qquad \mathbb{R} \qquad \mathbb{Z}$$

- Grouping symbols (parentheses, brackets, etc)

```
\newcommand{\lp}{\left(}
\newcommand{\rp}{\right)}
\newcommand{\lb}{\left[}
\newcommand{\rb}{\right]}

\lp\pdd{F}{x}\rp
compared to
(\pdd{F}{x})
```

The result is:

$$\left(\frac{\partial^2 F}{\partial x^2}\right)$$

compared to

$$(\frac{\partial^2 F}{\partial x^2})$$

- Common conjunctions

```
\newcommand{\andd}[1]{\quad\text{and}\quad}
\newcommand{\orr}[1]{\quad\text{or}\quad}
\newcommand{\forr}[1]{\quad\text{for}\quad}
```

```
\newcommand{\st}[1]{\quad\text{such that}\quad}
\newcommand{\conj}[1]{\quad\text{#1}\quad}

% Implies
\newcommand{\ra}{\quad\Rightarrow\quad}

A\ra B\st C\ne D
```

The result is:

$$A \quad \Rightarrow \quad B \quad \text{such that} \quad \neq D$$

C.3 Graphics in LaTeX

In this chapter we will focus on several tools that extend your knowledge of figures beyond just `includegraphics` and move you toward the domain of professional publications. The tools that we'll cover are:

1. The `tikz` package,
2. The `pgfplots` package,
3. Using GeoGebra to generate `tikz` code, and
4. Using MatLab to generate `tikz` code.

These tools take a lot of work, but the end result is well worth it.

There is nothing worse or more distracting than a poorly done figure.

There is more to these packages than we could possibly cover in a few days. It is imperative that you use the internet to its fullest extent with these packages. You can get yourself into a pickle with some of the internet-based examples, but starting with someone else's code for these packages is über helpful sometimes!

What I'll present here are simply a few examples to get you going.

C.3.1 The Tikz and PGFPlots Packages

The Tikz package is made for doing line drawings. The simplest mode of operation with Tikz is to do point-by-point drawings on a Cartesian grid.

Example C.5. Say we want to draw a coordinate plane with a few geometric shapes. Inside the `figure` environment we include a `tikzpicture` environment around the code for the picture. Be sure to end every Tikz line with a semicolon; Figure C.2 shows the results.

```
\begin{tikzpicture}
    \draw[color=gray] (-3,-3) grid (3,3);
```

```
    \draw[thick, <->] (-3,0) -- (3,0) node[anchor=west]{$x$};
    \draw[thick, <->] (0,-3) -- (0,3) node[anchor=south]{$y$};
    \draw[very thick, blue, fill=blue!50] (0,0) --
        (2,1) -- (1,3) -- cycle;
    \draw[very thick, dashed, color=red,
        fill=red!20!blue, opacity=0.5] (-2,0) circle(1cm);
\end{tikzpicture}
```

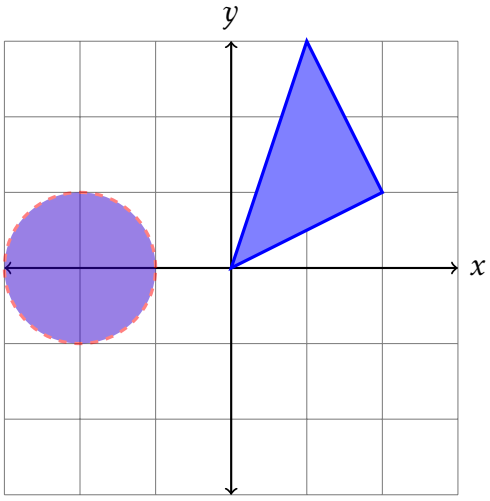


Figure C.2. A simple Tikz picture

For more examples about the Tikz package, see http://www.texample.net/tikz/examples/ ...texample is your new best friend.

You don't have to plot in MatLab, Excel, or any other tool when writing a technical document! Say this to yourself 100 times and be sure that you're sitting down.

Example C.6. This first example shows a simple way to plot functions.

```
\begin{tikzpicture}
    \begin{axis}[axis lines=center, xlabel={x},
            title={My Awesome Plot},
            domain=-2*pi:2*pi, ymin=-1.5, ymax=2, grid]
        \addplot[blue, thick, smooth] {sin(deg(x))};
        \addlegendentry{$f(x)=\sin(x)$};
        \addplot[red, thick, smooth] {cos(deg(x))};
        \addlegendentry{$g(x)=\cos(x)$};
        \addplot[black, dashed, thick, smooth] {0.1*exp(-x)};
        \addlegendentry{$h(x)=0.1\text{exp}(-x)$};
```

```
    \end{axis}
\end{tikzpicture}
```

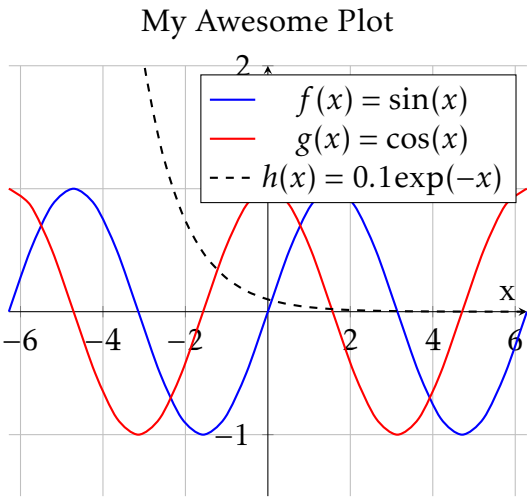


Figure C.3. A figure drawn with the `tikzpicture` and `axis` commands (leveraging the `pgfplots` package in the backgroud).

Next we'll follow with several more examples. Some of them are very advanced and some are beautifully simple.

Example C.7. Draw a bar chart for the following table of the world's largest producers of gem-quality diamonds in 2010. The solution is shown in Figure C.4.

Country	Millions of Carats
Botswana	25.0
Russia	17.8
Angola	12.5
Canada	11.8
Congo	5.5

Souce: USGS Mineral Commodity Summaries.

```
\usetikzlibrary{patterns}
\pgfplotsset{width=12cm,height=8cm}
\begin{tikzpicture}
    \begin{axis}[
            ybar,
            bar width=10mm,
            enlargelimits=0.15,
```

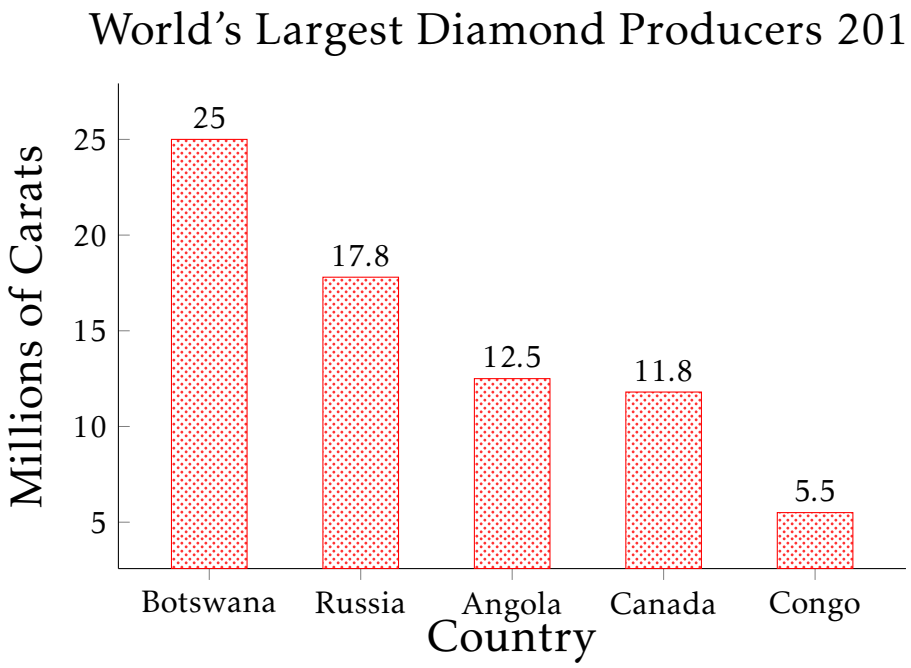


Figure C.4. Figure for Example C.7

```
            xlabel={\Large{Country}},
            ylabel={\Large{Millions of Carats}},
            title={\Large{World's Largest Diamond Producers 2010}},
            xtick=data,
            symbolic x coords={Botswana,Russia,Angola,Canada,Congo},
            nodes near coords,
            axis lines*=left
        ]
        \addplot [pattern=crosshatch dots,pattern color=red!80!white,
            draw=red] coordinates {(Botswana,25)
            (Russia,17.8) (Angola,12.5) (Canada,11.8) (Congo,5.5)};
    \end{axis}
\end{tikzpicture}
```

C.4 Bibliography Management

There are two primary ways to manage a bibliography file in LaTeX. In both ways you need to remember that (as usual) you have full control over everything! Two rules of thumb:

1. If you are using a short bibliography or if this paper stands alone then you probably want to use an embedded bibliography.

2. If you have a collection of references that will be used for several papers then you should consider using a BibTeX database.

Both types of bibliographies will save huge amounts of time and allow for very simple citation formats.

As usual, there is MUCH more to writing a good bibliography than what can possibly be listed here. A really good source is the wiki page for the latex bibliography: http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management.

C.4.1 Embedded Bibliography

If you're using an embedded bib for a stand-alone paper then just before the `\end{document}` you include all of the bibliography information. A simple example (with 1 paper) is included here:

```
\begin{thebibliography}{9}

\bibitem{lamport94}
  Leslie Lamport,
  \emph{\LaTeX: a document preparation system},
  Addison Wesley, Massachusetts,
  2nd edition,
  1994.

\end{thebibliography}
```

Use the `\cite{ }` command to cite items that are listed labeled inside the curly braces after `\bibitem`. For example, if we type `\cite{lamport94}` then we get a citation like this: [**?**].

C.4.2 Bibliography Database: BibTeX

BibTeX is a way for you to keep all of your bibliography materials in one place. The basic idea is as follows:

1. Start a file called `MyBib.bib` and follow the instructions from the link below to build your bibliography:
   http://ccm.ucdenver.edu/wiki/How_to_write_BibTeX_files

2. In your LaTeX file you can cite bib items with the `\cite{ }` command. As you cite works and compile you will build the bibliography automatically. You will need to compile MANY times to get all of the cross referencing and citations to appear.

3. Be sure that the `*.bib` file is in the same working directory as your LaTeX document (or at least give a path).

The primary utility of a bibtex file is that you can simply build it once when you're working on a large project and the citations will draw only the parts that are necessary for the current paper.

Appendix D

Optional Material

D.1 Low Rank Approximations of Matrices

Problem D.1. One particular use of the SVD is for data reduction. The word "reduction" here really means that we are going to make approximations of data using lower dimensions, and a very visually stunning way to do this is to do data reduction on images. The following code will read the file `TestImage.jpg` into MATLAB and convert it to a rectangular matrix of values. It is up to the reader to supply the necessary image.

```
A = imread('TestImage.jpg');
A = A(:,:,1);
A = im2double(A);
imshow(A)
```

Once the matrix is in MATLAB do the following. In this we assume that A is an $m \times n$ matrix.

1. Find the SVD of the image (remember the semicolons!!!!!!). This will take over a minute with our code so be patient. Once it is done check that your SVD code does a decent approximation of the original image.

```
[U,S,V] = MySVD(A);
error = norm(A - U*S*V')
```

2. Get the singular values out of Σ and find the largest P% of the singular values. Let's say that this is N values. Create four new matrices U_{new}, Σ_{new}, V_{new}, and A_{new} in the following way.
   (a) U_{new} is $m \times N$ and contains only the first N columns of U.
   (b) Σ_{new} is $N \times N$ and contains only the top P% of the singular values of A.
   (c) V_{new} is $n \times N$ and contains the first N columns of V.
   (d) A_{new} is $m \times n$ and is formed by $U_{new}\Sigma_{new}V_{new}^T$.
3. Show the newly data-reduced image with
   `imshow(Anew)`
4. The rank of the new image is equal to the number of singular values that you kept in step 2.
5. Experiment with several low rank approximations of an image starting with rank 1 and progress up to larger and larger ranks matrices. You'll find that the rank necessary to recover the full image is much lower than than the full original rank.

▲

D.2 The Google Page Rank Algorithm

In this section you will discover how the PageRank algorithm works to give the most relevant information as the top hit on a Google search.

Search engines compile large indexes of the dynamic information on the Internet so they are easily searched. This means that when you do a Google search, you are not actually searching the Internet; instead, you are searching the indexes at Google.

When you type a query into Google the following two steps take place:

1. Query Module: The query module at Google converts your natural language into a language that the search system can understand and consults the various indexes at Google in order to answer the query. This is done to find the list of relevant pages.

2. Ranking Module: The ranking module takes the set of relevant pages and ranks them. The outcome of the ranking is an ordered list of web pages such that the pages near the top of the list are most likely to be what you desire from your search. This ranking is the same as assigning a *popularity score* to each web site and then listing the relevant sites by this score.

This section focuses on the Linear Algebra behind the Ranking Module developed by the founders of Google: Sergey Brin and Larry Page. Their algorithm is called the *PageRank algorithm*, and you use it every single time you use Google's search engine.

In simple terms: *A webpage is important if it is pointed to by other important pages.*

The Internet can be viewed as a directed graph (look up this term here on Wikipedia) where the nodes are the web pages and the edges are the hyperlinks between the pages. The hyperlinks into a page are called *inlinks*, and the ones pointing out of a page are called *outlinks*. In essence, a hyperlink from my page to yours is my endorsement of your page. Thus, a page with more recommendations must be more important than a page with a few links. However, the status of the recommendation is also important.

Let us now translate this into mathematics. To help understand this we first consider the small web of six pages shown in Figure D.1 (a graph of the router level of the internet can be found here). The links between the pages are shown by arrows. An arrow pointing into a node is an *inlink* and an arrow pointing out of a node is an *outlink*. In Figure D.1, node 3 has three outlinks (to nodes 1, 2, and 5) and 1 inlink (from node 1).

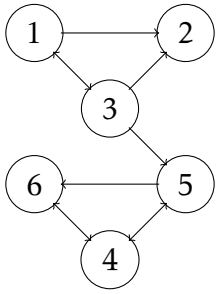


Figure D.1. Sample graph of a web with six pages.

We will first define some notation in the PageRank algorithm:

- $|P_i|$ is the number of outlinks from page P_i
- H is the *hyperlink* matrix defined as

$$H_{ij} = \begin{cases} \frac{1}{|P_j|}, & \text{if there is a link from node } j \text{ to node } i \\ 0, & \text{otherwise} \end{cases}$$

  where the "i" and "j" are the row and column indices respectively.
- $\mathbf{x}$ is a vector that contains all of the PageRanks for the individual pages.

The PageRank algorithm works as follows:

1. Initialize the page ranks to all be equal. This means that our initial assumption is that all pages are of equal rank. In the case of Figure D.1 we would take $\mathbf{x}_0$ to be

$$\mathbf{x}_0 = \begin{pmatrix} 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \end{pmatrix}.$$

2. Build the hyperlink matrix.
   As an example we'll consider node 3 in Figure D.1. There are three outlinks from node 3 (to nodes 1, 2, and 5). Hence $H_{13} = 1/3$, $H_{23} = 1/3$, and $H_{53} = 1/3$ and the partially complete hyperlink matrix is

$$H = \begin{pmatrix} - & - & 1/3 & - & - & - \\ - & - & 1/3 & - & - & - \\ - & - & 0 & - & - & - \\ - & - & 0 & - & - & - \\ - & - & 1/3 & - & - & - \\ - & - & 0 & - & - & - \end{pmatrix}$$

3. The difference equation $\mathbf{x}_{n+1} = H\mathbf{x}_n$ is used to iteratively refine the estimates of the page ranks. You can view the iterations as a person visiting a page and then following a link at random, then following a random link on the next page, and the next, and the next, etc. Hence we see that the iterations evolve exactly as expected for a difference equation.

Iteration	New Page Rank Estimation
0	$\mathbf{x}_0$
1	$\mathbf{x}_1 = H\mathbf{x}_0$
2	$\mathbf{x}_2 = H\mathbf{x}_1 = H^2\mathbf{x}_0$
3	$\mathbf{x}_3 = H\mathbf{x}_2 = H^3\mathbf{x}_0$
4	$\mathbf{x}_4 = H\mathbf{x}_3 = H^4\mathbf{x}_0$
$\vdots$	$\vdots$
k	$\mathbf{x}_k = H^k\mathbf{x}_0$

4. When a steady state is reached we sort the resulting vector $\mathbf{x}_k$ to give the page rank. The node (web page) with the highest rank will be the top search result, the second highest rank will be the second search result, and so on.

It doesn't take much to see that this process can be very time consuming. Think about your typical web search with hundreds of thousands of hits; that makes a square matrix H that has a size of hundreds of thousands of entries by hundreds of thousands of entries! The matrix multiplications alone would take many minutes (or possibly many hours) for every search! ... but Brin and Page were pretty smart dudes!!

We now state a few theorems and definitions that will help us simplify the iterative PageRank process.

Theorem D.2. If A is an $n \times n$ matrix with n linearly independent eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_n$ and associated eigenvalues $\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_n$ then for any initial vector $\mathbf{x} \in \mathbb{R}^n$ we can write $A^k\mathbf{x}$ as

$$A^k\mathbf{x} = c_1\lambda_1^k\mathbf{v}_1 + c_2\lambda_2^k\mathbf{v}_2 + c_3\lambda_3^k\mathbf{v}_3 + \cdots c_n\lambda_n^k\mathbf{v}_n$$

where $c_1, c_2, c_3, \ldots, c_n$ are the constants found by expressing $\mathbf{x}$ as a linear combination of the eigenvectors.
Note: We can assume that the eigenvalues are ordered such that $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n$.

Proof. (Prove the preceding theorem) □

Definition D.3. A **probability vector** is a vector with entries on the interval $[0, 1]$ that add up to 1.

Definition D.4. A **stochastic matrix** is a square matrix whose columns are probability vectors.

Theorem D.5. If A is a stochastic $n \times n$ matrix then A will have n linearly independent eigenvectors. Furthermore, the largest eigenvalue of a stochastic matrix will always be $\lambda_1 = 1$ and the smallest eigenvalue will always be nonnegative: $0 \leq \lambda_n < 1$.

Some of the following tasks will ask you to *prove* a statement or a theorem. This means to clearly write all of the logical and mathematical reasons why the statement is true. Your proof should be absolutely crystal clear to anyone with a similar mathematical background ... if you are in doubt then have a peer from a different group read your proof to you out loud.

Problem D.6. Finish writing the hyperlink matrix H from Figure D.1. ▲

Problem D.7. Write MATLAB code to implement the iterative process defined previously. Make a plot that shows how the rank evolves over the iterations. ▲

Problem D.8. What must be true about a collection of n pages such that an $n \times n$ hyperlink matrix H is a stochastic matrix. ▲

The statement of the next theorem is incomplete, but the proof is given to you. Fill in the blank in the statement of the theorem and provide a few sentences supporting your answer.

Theorem D.9. If A is an $n \times n$ stochastic matrix and $\mathbf{x}_0$ is some initial vector for the difference equation $\mathbf{x}_{n+1} = A\mathbf{x}_n$, then the steady state vector is

$$\mathbf{x}_{equilib} = \lim_{k \to \infty} A^k \mathbf{x}_0 = \_\_\_\_\_.$$

Proof. First note that A is an $n \times n$ stochastic matrix so from Theorem D.5 we know that there are n linearly independent eigenvectors. We can then substitute the eigenvalues from Theorem D.5 in Theorem D.2. Noting that if $0 < \lambda_j < 1$ we have $\lim_{k \to \infty} \lambda_j^k = 0$ the result follows immediately. □

Problem D.10. Discuss how Theorem D.9 greatly simplifies the PageRank iterative process described previously. In other words: there is no reason to iterate at all. Instead, just find _____. ▲

Problem D.11.

Now use the previous two problems to find the resulting PageRank vector from the web in Figure D.1? Be sure to rank the pages in order of importance. Compare your answer to the one that you got in problem 2. ▲

Problem D.12. Consider the web in Figure D.2.

(a) Write the H matrix and find the initial state $\mathbf{x}_0$,

(b) Find steady state PageRank vector using the two different methods described: one using the iterative difference equation and the other using Theorem D.9 and the dominant eigenvector.

(c) Rank the pages in order of importance.

▲

Problem D.13. One thing that we didn't consider in this version of the Google Page Rank algorithm is the random behavior of humans. One, admittedly slightly naive, modification that we can make to the present algorithm is to assume that the person surfing the web will randomly jump to any other page in the web at any time. For example, if someone is on page 1 in Figure D.2 then they could randomly jump to any page 2 - 8. They also have links to pages 2, 3, and 7. That is a total of 10 possible next steps for the web

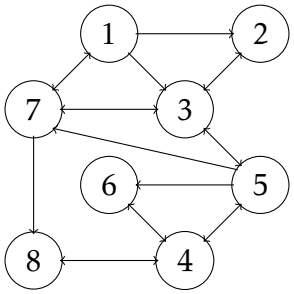


Figure D.2. Graph of a web with eight pages.

surfer. There is a 2/10 chance of heading to page 2. One of those is following the link from page 1 to page 2 and the other is a random jump to page 2 without following the link. Similarly, there is a 2/10 chance of heading to page 3, 2/10 chance of heading to page 7, and a 1/10 chance of randomly heading to any other page.

Implement this new algorithm, called the *random surfer algorithm*, on the web in Figure D.2. Compare your ranking to the non-random surfer results from the previous problem. ▲

D.3 Principal Component Analysis (Incomplete)

D.4 Building PDE's From Conservation Laws

In this section we'll give a more analytic introduction to most of the primary partial differential equations of interest in basic mathematical physics. We will make reference to Fick's Law for mass transport and Fourier's Law for thermal transport, so interested readers should dig deeper by examining the relevant Wikipedia pages or other sources.

Conservation laws pervade all of physics – conservation of energy, conservation of momentum, and conservation of mass. These laws are sometimes stated colloquially as *energy (or momentum or mass) can neither be created nor destroyed*, but this phrase is not super helpful mathematically. We start this section with a brief mathematical derivation of a *general conservation law* to further clarify what we mean mathematically. The resulting general conservation law will be a partial differential equation that can be used to mathematically express the physical laws of conservation of mass, momentum, or energy.

Let u be the quantity you are trying to conserve, $\mathbf{q}$ be the flux of that quantity, and f be any source of that quantity. For example, if we are to derive a conservation of energy equation, u might be energy, $\mathbf{q}$ might be temperature flux, and f might be a temperature source (or sink).

Derivation of General Balance Law

Let Ω be a fixed volume and denote the boundary of this volume by $\partial\Omega$. The rate at which u is changing in time throughout Ω needs to be balanced by the rate at which u leaves the volume plus any sources of u. Mathematically, this means that

$$\frac{\partial}{\partial t}\iiint_{\Omega} u\,dV = -\iint_{\partial\Omega} \mathbf{q}\cdot n\,dA + \iiint_{\Omega} f\,dV. \tag{D.1}$$

This is a global balance law in the sense that it holds for all volumes Ω. The mathematical troubles here are two fold: (1) there are many integrals, and (2) there are really two variables (u and q since $f = f(u,x,t)$) so the equation is not closed. In order to mitigate that fact we apply the divergence theorem to the first term on the right-hand side of (D.1) to get

$$\frac{\partial}{\partial t}\iiint_{\Omega} u\,dV = -\iiint_{\Omega} \nabla\cdot\mathbf{q}\,dV + \iiint_{\Omega} f\,dV. \tag{D.2}$$

Gathering all of the terms on the right of (D.2), interchanging the integral and the derivative on the left (since the volume is not changing in time), and rewriting gives

$$\iiint_{\Omega}\left(\frac{\partial u}{\partial t} + \nabla\cdot\mathbf{q}\right)dV = \iiint_{\Omega} f\,dV \tag{D.3}$$

If we presume that this equation holds for all volumes Ω then the integrands must be equal and we get the local balance law

$$\frac{\partial u}{\partial t} + \nabla\cdot\mathbf{q} = f. \tag{D.4}$$

Equation (D.4) is an expression of the balances of changes in time to changes in space of a conserved quantity such as mass, momentum, or energy. What remains is to make clear the meaning and functional form of the flux $\mathbf{q}$ and the source function f.

Simplification of the Local Balance Law

In equation (D.4) it is often assumed that the system is free of external sources. In this case we set f to zero and obtain the source-free balance law

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{q} = 0. \tag{D.5}$$

It is this form of balance law where many of the most interesting and important partial differential equations come from. In particular consider the following two cases: mass balance and energy balance.

Mass Balance

In mass balance we take u to either be the density of a substance (e.g. in the case of liquids) or the concentration of a substance in a mixture (e.g. in the case of gasses). If C is the mass concentration of a substance in a gas then the flux of that substance is given via Fick's Law as

$$\mathbf{q} = -k\nabla C. \tag{D.6}$$

Combining (D.6) with (D.5) (and assuming that k is independent of space, time, and concentration) gives

$$\frac{\partial C}{\partial t} = k\nabla \cdot \nabla C. \tag{D.7}$$

In the presence of external sources of mass, (D.7) is

$$\frac{\partial C}{\partial t} = k\nabla \cdot \nabla C + f(x). \tag{D.8}$$

Expanding the Laplacian operator on the right-hand side of (D.8) we get

$$\frac{\partial C}{\partial t} = k\left(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} + \frac{\partial^2 C}{\partial z^2}\right) + f(x) \tag{D.9}$$

where the reader should note that this can be easily simplified in 1 or 2 spatial dimensions.

Energy Balance

The energy balance equation is essentially the same as the mass balance equation. If u is temperature then the flux of temperature is given by Fourier's Law for heat conduction

$$\mathbf{q} = -k\nabla T. \tag{D.10}$$

Making the same simplifications as in the mass balance equation we arrive at

$$\frac{\partial T}{\partial t} = k\nabla \cdot \nabla T. \tag{D.11}$$

In the presence of external sources of heat, (D.11) becomes

$$\frac{\partial T}{\partial t} = k\nabla \cdot \nabla T + f(x). \tag{D.12}$$

Expanding the Laplacian operator on the right-hand side of (D.12) we get

$$\frac{\partial T}{\partial t} = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) + f(x) \tag{D.13}$$

where the reader should note that this can be easily simplified in 1 or 2 spatial dimensions.

Laplace's Equation and Poisson's Equation

Equations (D.8) and (D.12) are the same partial differential equation for two very important physical phenomenon; mass and heat transfer. In the case where time is allowed to run to infinity and no external sources of mass or energy are included these equations reach a steady state solution (no longer changing in time) and we arrive at Laplace's Equation

$$\nabla \cdot \nabla u = 0. \tag{D.14}$$

Laplace's equation is actually a statement of minimal energy as well as steady state heat or temperature. We can see this since entropy always drives systems from high energy to low energy, and if we have reached a steady state then we must have also reached a surface of minimal energy.

Equation (D.14) is sometimes denoted as $\nabla \cdot \nabla u = \nabla^2 u = \Delta u$, and in terms of the partial derivatives it is written as

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0.$$

If there is a time-independent external source the right-hand side of (D.14) will be non-zero and we arrive at Poisson's equation:

$$\nabla \cdot \nabla u = -f(x). \tag{D.15}$$

Note that the negative on the right-hand side comes from the fact that $\frac{\partial u}{\partial t} = k\nabla \cdot \nabla u + f(x)$ and $\frac{\partial u}{\partial t} \to 0$. Technically we are absorbing the constant k into f (that is "f" is really "f/k"). Also note that in many instances the value of k is not constant and cannot therefore be pulled out of the derivative without a use of the product rule.

Let's summarize:

Name of PDE	PDE	What the PDE Models
The Heat Equation	$\frac{\partial u}{\partial t} = k\nabla \cdot \nabla u + f(x)$	Diffusion
Laplace's Equation	$k\nabla \cdot \nabla u = -f(x)$	Minimal Energy Surfaces

Further discussion of the origins of the wave equation and other interesting PDE's is left to the reader.

Bibliography

[1] A. Greenbaum and T. Charier. *Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms* Princeton University Press. 2012.

[2] R. Burden, D, Faires, and A. Burden. *Numerical Analysis, 10ed* Cengage Learning. 2016.

[3] D. Kindaid and W. Cheney. *Numerical Analysis, 2ed.* Brooks/Cole Publishing, 1996.

[4] R. Haberman. *Applied Partial Differential Equations, 4ed*. Pearson Education Inc. Upper Saddle River, New Jersey, 2004

[5] D. Lay. *Linear Algebra 4ed.* Pearson Education Inc. Upper Saddle River, New Jersey, 2012.

[6] M. Meerschaert. *Mathematical Modeling 4ed.* Academic Press Publications, 2013.

[7] Holistic Numerical Methods http://nm.mathforcollege.com/
The Holistic Numerical Methods book is probably the most complete free reference that I've found on the web. This should be your source to look up deeper explanations of problems, algorithms, and code.

[8] Scientific Computing with MATLAB http://gribblelab.org/scicomp/scicomp.pdf

[9] Tea Time Numerical Analysis http://lqbrin.github.io/tea-time-numerical/