Numerical Analysis
An Inquiry and Problem Based Approach

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

Content Last Updated: June 8, 2017

Contents

0 To the Student and the Instructor **5**
- 0.1 An Inquiry Based Approach . . . 5
- 0.2 Online Texts and Other Resources . . . 6
- 0.3 To the Instructor . . . 6

1 Introductory Topics **7**
- 1.1 Base 2 and Binary Arithmetic . . . 7
- 1.2 Floating Point Arithmetic . . . 9
- 1.3 The Taylor Series . . . 12
- 1.4 Exercises . . . 14

2 Numerical Algebra – Approximating Roots **17**
- 2.1 The Bisection Method . . . 18
- 2.2 The Regula Falsi Method . . . 19
- 2.3 Newton's Method . . . 20
- 2.4 Quasi-Newton Methods . . . 21
- 2.5 Exercises . . . 23

3 Numerical Calculus **26**
- 3.1 Numerical Differentiation . . . 26
- 3.2 Numerical Integration . . . 30
- 3.3 Exercises . . . 33

4 Numerical Linear Algebra **37**
- 4.1 Matrix Operations . . . 37
- 4.2 Solving Systems of Linear Equations . . . 38
  - 4.2.1 Upper and Lower Triangular Systems . . . 38
  - 4.2.2 The LU Factorization . . . 40
- 4.3 The Least Squares Problems . . . 43
- 4.4 The QR Factorization . . . 45
- 4.5 The Eigenvalue-Eigenvector Problem . . . 48
- 4.6 The Singular Value Decomposition . . . 50
- 4.7 Low Rank Approximations of Matrices . . . 52
- 4.8 The Google Page Rank Algorithm . . . 53
- 4.9 Principal Component Analysis (Incomplete) . . . 57

4.10 Exercises . . . 57

5 Numerical Ordinary Differential Equations **62**
5.1 Euler, Runge-Kutta, and Friends . . . 62
5.2 Implicit Methods and Shooting Methods . . . 66
5.3 Exercises . . . 67

6 Numerical Partial Differential Equations **71**
6.1 Review of Multivariable Calculus . . . 71
6.2 Introduction to PDEs . . . 72
6.3 Boundary Conditions . . . 75
6.4 1D and 2D Heat Equation . . . 76
6.5 1D and 2D Wave Equation . . . 78
6.6 Traveling Waves . . . 79
6.7 1D and 2D Steady State PDEs . . . 79
6.8 Exercises . . . 81

A Writing and Projects **83**
A.1 The Paper . . . 83
A.2 Figures and Tables . . . 83
A.3 Writing Style . . . 84
A.4 Tips For Writing Clear Mathematics . . . 84
A.4.1 Audience . . . 84
A.4.2 How To Make Mathematics Readable – 10 Things To Do . . . 85
A.4.3 Some Writing Tips . . . 85
A.4.4 Mathematical Vocabulary . . . 88
A.5 Sensitivity Analysis . . . 90
A.6 Example of Sensitivity Analysis: . . . 91

B Scientific Computing with MATLAB **93**
B.1 Looping . . . 93
B.1.1 The For Loop . . . 93
B.1.2 The While Loop . . . 93
B.2 Conditional Statements . . . 93
B.2.1 If Statements . . . 93
B.2.2 Case-Switch Statements . . . 93
B.3 Functions . . . 93
B.4 Plotting . . . 93
B.5 Animations . . . 93
B.6 Additional Resources . . . 93

C LaTeX **94**
C.1 Equation Environments and Cross Referencing . . . 94
C.2 Tables, Tabular, Figures, Shortcuts, and Other Environments . . . 97
C.2.1 Tables and Tabular Environments . . . 97

C.2.2 Excel To LaTeX 98
C.2.3 Figures 98
C.2.4 New Commands: Shortcuts are AWESOME! 98
C.3 Graphics in LaTeX 101
C.3.1 The Tikz and PGFPlots Packages 101
C.4 Bibliography Management 104
C.4.1 Embedded Bibliography 105
C.4.2 Bibliography Database: BibTeX 105

Chapter 0

To the Student and the Instructor

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 Dr. 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 (and are hence purposefully invisible to the student).

0.1 An Inquiry Based Approach

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 though.
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 linear algebra and differential equations I can point you to a few. 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!

Chapter 1

Introductory Topics

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.

In this chapter we will discuss some of the basic underlying ideas behind the scenes in Numerical Analysis. 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. 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. Why have the computer gods failed you? More importantly, what happened on the computer and why did it give you a different answer? Most importantly, what is the cautionary tale hiding behind the scenes with this problem? ▲

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

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

In other words, the location of the digit (as read from the right-hand side of the number starting at 0) is the power on the base, 10. Similarly

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

Now let's switch to binary. In a binary number system the base is 2 so the only allowable digits are 0 and 1. Similar to the base-10 system, 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

$$101101_2 = 32 + 0 + 8 + 4 + 0 + 1 = 45.$$

Problem 1.4. Convert to following numbers from base 10 to base 2 or visa versa.

- Write 12_{10} in binary
- What is 100101_2 in base 10?

▲

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 1.5. 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 1.6. 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 1.7. 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 1.8. Use `=if( )` 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 1.9. 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 1.10. Convert the base 10 fraction 1/10 into binary. How does your solution relate to 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 1.11. 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

$$x = -1.\underline{\hspace{2cm}} \times 2^{\underline{\hspace{0.5cm}}}$$

(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?

▲

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 $0 \le m < 1$ For example:

- 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 $E=-4$.

Definition 1.12. For a number $x = (-1)^s \times m \times 2^E$ 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.

Definition 1.13 (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 1.14. 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 1.15. 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 1.16. What are the largest and smallest numbers that can be stored in single and double precicision? ▲

Problem 1.17. What is machine epsilon for single and and double precision? ▲

Problem 1.18. A typical computer number:

0	E=4	10011001100000000000000

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

Problem 1.19. 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 intersting read for the curious student.

1.3 The Taylor Series

Consider the function $f(x) = e^x$. Euler's number, e, is irrational and potentially difficult for a computer to work with 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).

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 1.20. (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 $k \le n$.

▲

Problem 1.21. Repeat Problem 1.20 with the function $f(x) = \sin(x)$. ▲

Problem 1.22. Repeat Problem 1.20 with the function $f(x) = \cos(x)$. ▲

Now let's formally state the definition of a Taylor Series.

Definition 1.23 (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 $x = a$.

Problem 1.24. 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 $a = 0$. ▲

The great thing about the Taylor Series is that it allow 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 certain number of terms.

Theorem 1.25 (Taylor's Theorem)**.** Let $f, f', f'', \ldots, f^{(n)}$ be continuous on *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 1.26 (Taylor's Theorem at $a = 0$)**.** Let $f, f', f'', \ldots, f^{(n)}$ be continuous on *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 1.27. 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

$$R_3(x) = \frac{f^{(3)}(\xi)}{4!}(x-0)^3 = \frac{e^\xi}{3!}x^3.$$

Therefore $R_3(x) \le \frac{e^1}{3!} \cdot 1^3 = \frac{e}{6} \approx 0.45$. 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 $0.218 < 0.45$.

Taylor's theorem gives a bound on the amount of error that you can make when using a truncated Taylor series.

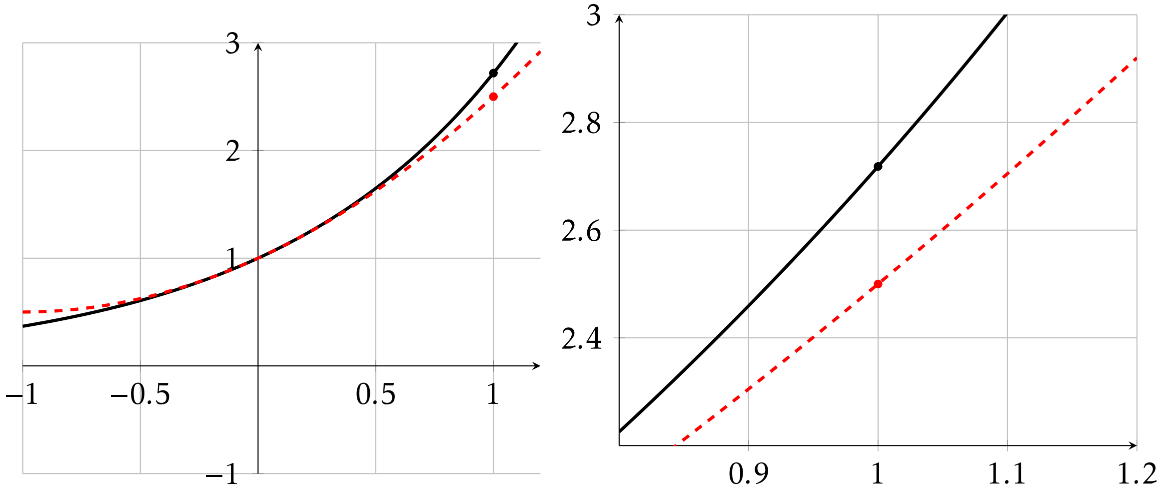

Figure 1.1. The function $f(x) = e^x$ and a second order Taylor approximation. The solid black curve is $f(x) = e^x$ and the dashed red curve is the Taylor approximation. The right-hand plot shows a zoomed in view near the point $x = 1$.

Problem 1.28. 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 1.29. 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? ▲

1.4 Exercises

Problem 1.30. 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 Chapter 2 of the text if you are stuck. ▲

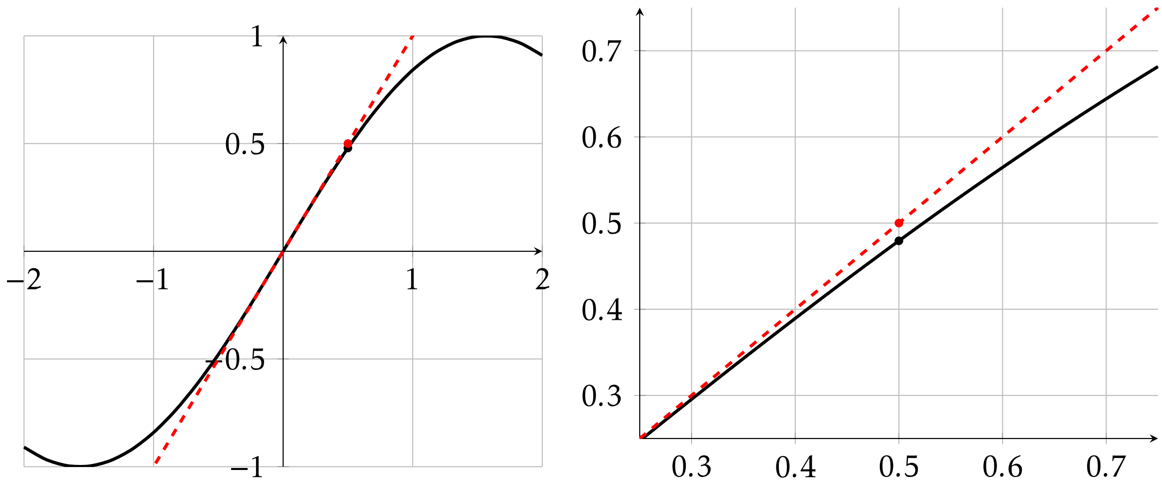

Figure 1.2. The function $f(x) = \sin(x)$ and a second order Taylor approximation. The solid black curve is $f(x) = \sin(x)$ and the dashed red curve is the Taylor approximation. The right-hand plot shows a zoomed in view near the point $x = 0.5$.

Homework Exercise 1.31. 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 Chapter 2 of the text if you are stuck.

Problem 1.32. 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.

▲

Problem 1.33. 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 1.34. (a) Write the Taylor series centered at $a = 0$ for the function $f(x) = \frac{1}{1+x}$.

(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.

▲

Chapter 2

Numerical Algebra – Approximating Roots

In this chapter we want to solve 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 $\sin(x) + 9 = x^2 - \tan(x)$ then this is the same as solving $(\sin(x) + 9) - (x^2 - \tan(x)) = 0$ (please don't try to solve this one by hand!). Hence, we can write $f(x) = \ell(x) - r(x)$ and observe that every algebraic equation can be written as

$$\text{if } f(x) = 0 \quad \text{find } x.$$

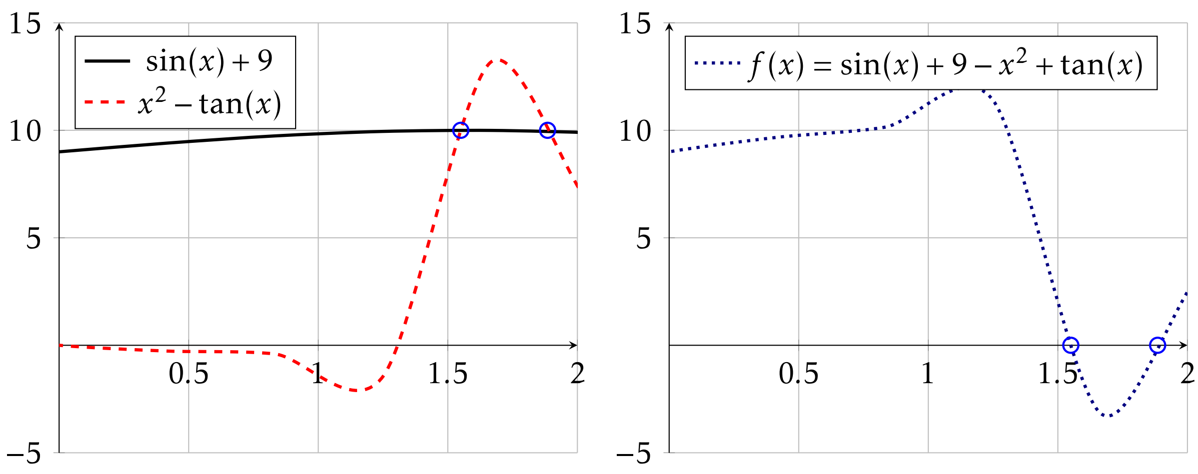

Figure 2.1. The left-hand plot shows two nonlinear functions, $\ell(x) = \sin(x) + 9$ and $r(x) = x^2 - \tan(x)$, with their intersection points marked. The right-hand plot shows the equivalent problem formed by solving $\ell(x) - r(x) = 0$.

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 numberical techniques for solving the equation will apply.

2.1 The Bisection Method

Theorem 2.1 (The Intermediate Value Theorem)**.** If f is a continuous function on the closed interval $[a,b]$ and y_* lies between $f(a)$ and $f(b)$, then there is a point $x \in [a,b]$ where $f(x) = y_*$.

Problem 2.2. Draw a picture of what the intermediate value theorem says graphically. ▲

Problem 2.3. If $y_* = 0$ the intermediate value theorem gives us important information about solving equations. What does it tell us? ▲

The following (partial) algorithm, known as the Bisection Method, uses the Intermediate Value Theorem to systematically approximate solutions to the algebraic equation $f(x) = 0$.

Algorithm 2.4 (The Bisection 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 (why is this important?).
2. Compute the midpoint $m = (a+b)/2$ 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 with $m = (a+b)/2$. Which one do you replace and why?
4. Repeat steps 2 and 3
5. Stop when $f(m)$ is *close enough* to zero.

Problem 2.5. Draw a picture illustrating what the Bisection Method does to approximate solutions to the algebraic equation $f(x) = 0$. ▲

Problem 2.6. Write a MATLAB function for the Bisection Method, and 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 2.7. 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 2.8. 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 2.9. 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 $[a, b] = [0, 2]$? ▲

2.2 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 2.10 (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.

$$x = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$$

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.

6. Stop when $f(c)$ is *close enough* to zero.

Problem 2.11. Draw a picture of what the Regula Falsi method does to approximate a root. ▲

Problem 2.12. 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 2.13. Write a MATLAB function to implement 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)
```
▲

2.3 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 2.14 (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.
$$y - \underline{\qquad} = \underline{\qquad} \cdot (x - \underline{\qquad})$$
4. Find the x intercept of the equation of the tangent line and call this new point x_1.
$$x_1 = \underline{\qquad\qquad\qquad\qquad\qquad}$$
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.
$$x_{new} = \underline{\qquad\qquad\qquad\qquad\qquad}$$
6. Iterate step 5 until $f(x_n)$ is *close* to zero.

Problem 2.15. Draw a picture of what Newton's method does graphically. ▲

Problem 2.16. 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. ▲

Problem 2.17. Write a MATLAB function for Newton's method. Your function needs to accept an anonymous function handle, the derivative of the anonymous function hand, 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 2.18. 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 roots 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 2.19. 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. ▲

2.4 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 2.20 (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. (Note: the points x_0 and x_1 should be close to each other. 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 2.21. Draw several pictures showing what the Secant method does pictorially. ▲

Problem 2.22. Write MATLAB code for solving algebraic equations of the form $f(x) = 0$ with the Secant method. Also write a test script that clearly shows that your code is working. ▲

Problem 2.23. 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 2.18. ▲

Algorithm 2.24 (Steffensen's Method)**.** To solve $f(x) = 0$:

1. Determine if there is a root *near* an arbitrary starting point x_0.
2. In Steffensen's method we approximate the derivative with

$$f'(x_n) \approx \frac{f(x_n + f(x_x)) - f(x_n)}{f(x_n)}$$

3. Perform the Newton-type iteration

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

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

$$x_{n+1} = x_n - \frac{f(x_n)^2}{f(x_n+f(x_n))-f(x_n)}.$$

Problem 2.25. Write MATLAB code for solving algebraic equations of the form $f(x) = 0$ with the Steffensen's method. Also write a test script that clearly shows that your code is working. ▲

Problem 2.26. Choose a non-trivial algebraic equation for which you know the solution and write a script to empirically determine the convergence rate of Steffensen's method. You may want to look back at 2.18. ▲

2.5 Exercises

Problem 2.27. 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 $3025385 = 2640$.

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 2.28. 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 2.29. 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 2.30. 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 differential equation $v_y'(t) = -g$ 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 $v_y(t) = v_0 \sin(\theta)$.

(b) Solve the differential equation $s_y'(t) = v_y(t)$ for the position function $s_y(t)$. Assume that $s_y(0) = 0$.

(c) Solve $s_y(t) = 0$ for t 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 $R(v_0, \theta)$.

(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

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

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 2.31. Write MATLAB code that compares the convergence rates for the bisection method, the regula-falsi method, Newton's method, the secant method, and Steffensen's method. Find examples and conditions where each method "wins" by having the faster convergence rate. For example, find different starting points that favor one method over the others, find functions that favor one method over the others, etc. ▲

Problem 2.32. The Newton's method that we derived in this chapter is only applicable to functions $f : \mathbb{R} \to \mathbb{R}$ (functions mapping a real number to a real number). What about vector-valued functions? In particular, we would like to have an analogous method for finding roots of a function F where $F : \mathbb{R}^n \to \mathbb{R}^n$.

Let $\mathbf{x}$ be a vector in $\mathbb{R}^n$, let

$$F(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{pmatrix}$$

be a vector valued function, and let J be the Jacobian matrix

$$J(\mathbf{x}) = \begin{pmatrix} \partial f_1/\partial x_1(\mathbf{x}) & \partial f_1/\partial x_2(\mathbf{x}) & \cdots & \partial f_1/\partial x_n(\mathbf{x}) \\ \partial f_2/\partial x_1(\mathbf{x}) & \partial f_2/\partial x_2(\mathbf{x}) & \cdots & \partial f_2/\partial x_n(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1(\mathbf{x}) & \partial f_n/\partial x_2(\mathbf{x}) & \cdots & \partial f_n/\partial x_n(\mathbf{x}) \end{pmatrix}$$

By analogy, the multi-dimensional Newton's method is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - F(\mathbf{x}_n)J^{-1}(\mathbf{x}_n)$$

where J^{-1} is the inverse of the Jacobian matrix.

(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 $F(\mathbf{x}) = \mathbf{0}$.

(b) Test your code 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 $f_1(x,y) = 1 + x^2 - y^2 + e^x \cos(y)$ and $f_2(x,y) = 2xy + e^x \sin(y)$.

▲

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.

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.

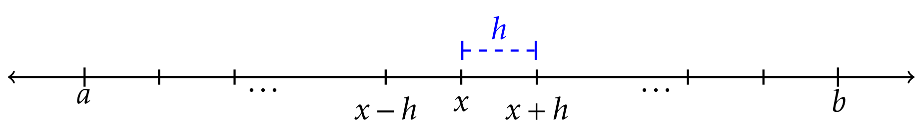

Figure 3.1. A partition of an interval on the real line.

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 reaminder 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

$$|Error| \le M|h|.$$

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. Writet MATLAB code that takes a function and a domain $(xmin, xmax)$ and returns a numerical approximation to the derivative on the interval $(xmin, xmax)$. 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) ... % 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. ▲

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)**.**

$$f'(x) = \underline{\hspace{6cm}} + \mathcal{O}(h^2)$$

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. 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. 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)
```

▲

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.

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

Table 3.1. First and second derivatives.

3.2 Numerical Integration

Next we will build methods for approximating integrals. Recall 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.

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. 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? ▲

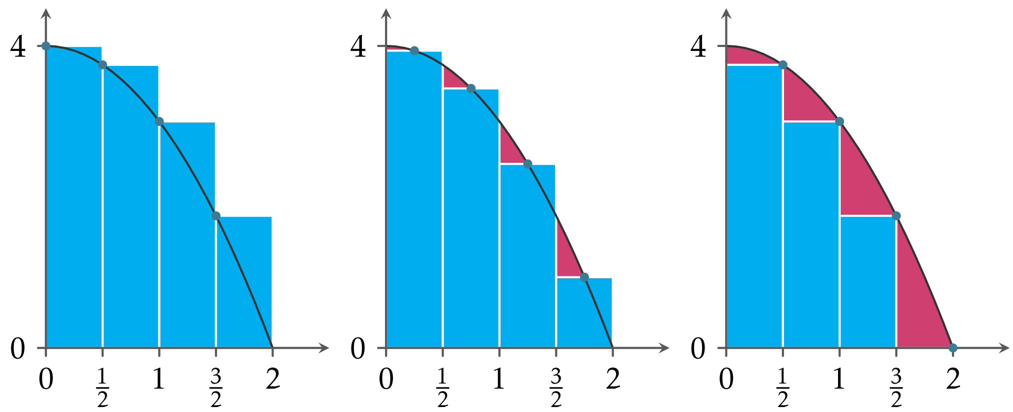

Figure 3.2. Riemann sum to approximate an integral with left, midpoint, and right rectangles.

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

3. Approximate the integral as
$$\int_a^b f(x)dx = \sum_{j=1}^{n} A_j$$

Draw a picture depicting how 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

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

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 $\{x_0 = a, x_1, x_2, \dots, x_{n-1}, x_n = b\}$.

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 the trapezoidal rule works. ▲

Problem 3.19. Write MATLAB functions that implement both the trapezoidal rule and Simpson's rule. 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.

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

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

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

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

3.3 Exercises

Problem 3.21. Coding Challenge: The four adjacent digits in the 1000-digit number that have the greatest product are $9 \times 9 \times 8 \times 9 = 5832$.

73167176531330624919225119674426574742355349194934
96983520312774506326239578318016984801869478851843
85861560789112949495459501737958331952853208805511
12540698747158523863050715693290963295227443043557
66896648950445244523161731856403098711121722383113
62229893423380308135336276614282806444486645238749
30358907296290491560440772390713810515859307960866
70172427121883998797908792274921901699720888093776
65727333001053367881220235421809751254540594752243
52584907711670556013604839586446706324415722155397
53697817977846174064955149290862569321978468622482
83972241375657056057490261407972968652414535100474

82166370484403199890008895243450658541227588666881
16427171479924442928230863465674813919123162824586
17866458359124566529476545682848912883142607690042
24219022671055626321111109370544217506941658960408
07198403850962455444362981230987879927244284909188
84580156166097919133875499200524063689912560717606
05886116467109405077541002256983155200055935729725
71636269561882670428252483600823257530420752963450

Write code to find the thirteen adjacent digits in the 1000-digit number that have the greatest product. What is the value of this product? ▲

Problem 3.22. Coding Challenge: 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 3.23. 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.

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

(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 3.24. 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] = FirstDeriveFromData(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 3.25. 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] = SecondDeriveFromData(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 3.26. 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 3.27. 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 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 algorithm). ▲

Problem 3.28. 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 3.24 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 3.26 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 3.29. 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). ▲

Chapter 4

Numerical Linear Algebra

In this chapter we explore 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. Along the way we will encounter several novel ways to factor matrices into products of other matrices with more desirable properties.

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. ▲

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 spend a few days talking about efficient ways to have the computer solve these systems. You likely recall row reduction (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 4.5. Solve the following problem by hand using Gaussian Elimination (row reduction):

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

▲

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. In the following two problems you will devise algorithms for triangular matrices.

Problem 4.6. 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 4.7 (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 4.8. 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 4.7 by hand to solve the system. Keep track of all of the indices as you work through the code. ▲

Problem 4.9. Copy the code from Technique 4.7 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. ▲

Problem 4.10. 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*. ▲

Technique 4.11 (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 4.12. 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 4.11 by hand to solve the system. Keep track of all of the indices as you work through the code. ▲

Problem 4.13. Copy the code from Technique 4.11 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

Problem 4.14. Open a new MATLAB script and enter the left-hand matrix of the previous problem into 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 $L_1 A$.
- left multiply $L_1 A$ by the matrix
$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -7 & 0 & 1 \end{pmatrix}$$
This gives the matrix $L_2 L_1 A$.
- left multiply $L_2 L_1 A$ by the matrix
$$L_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}$$

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 4.15. 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:

$$L_3 L_2 L_1 A = U,$$

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 4.16. 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 4.17. Now for the punch line: If we can write $A\mathbf{x} = \mathbf{b}$ 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 4.18. 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 $LU\mathbf{x} = \mathbf{b}$. The 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 4.19 (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 4.20 (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 4.21. 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 4.22. 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 300300 matrices. Just be patient.

▲

4.3 The Least Squares Problems

Problem 4.23. Download the Excel data set from Moodle. 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.

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

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 and cubic data. ▲

Problem 4.24. 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

$$c_0 + c_1 x_j \approx y_j.$$

This gives rise to a system of linear equations

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \approx \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}$$

We'll call the left-hand matrix A and the right-hand vector $\mathbf{y}$.

1. 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 ...

   $$(??)A\begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = (??)\mathbf{y}.$$

   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 this case).

3. Now that this is a square matrix you can solve for c_0 and c_1 using your LU code. (You can also use MATLAB's *backslash* (\) command to do the linear solve)

4. Now that you have c_0 and c_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 and cubic data. For the quadratic data you are assuming that

   $$c_0 + c_1 x_j + c_2 x_j^2 \approx y_j,$$

   and for the cubic data you are assuming that

   $$c_0 + c_1 x_j + c_2 x_j^2 + c_3 y_j^3 \approx y_j.$$

▲

Definition 4.25 (The Normal Equations)**.** Let $A \in \mathbb{R}^{n \times m}$, $\mathbf{c} \in \mathbb{R}^m$, and $\mathbf{y} \in \mathbb{R}^n$ where $n \gg m$. The system of equations

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

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

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

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.

Problem 4.26. In the same Excel document you will find a set of data representing a nonlinear function. Use Excel to find the best fit curve by making a *guess* about the form of the function. Since this is a nonlinear relationship our method of using the normal equations will not work. ▲

Problem 4.27. The method of using the **normal equations** to solve a least squares regression problem also works for multiple regression. In this example go to the second tab of the Excel document to find the Hollywood Box Office Receipts (measured in millions). We are seeking a model of the form

$$X_1 \approx \beta_1 + \beta_2 X_2 + \beta_3 X_3.$$

That is, we seek to predict first year box office receipts (X_1) from production cost data (X_2) and promotional cost data (X_3).

- Read the data into MATLAB
- Find the matrix A to form the overdetermined system of equations

$$A \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = \mathbf{x}_1$$

- Form the normal equations and solve for β_1, β_2, and β_3.
- Star Wars: The Force Awakens had a production cost of \$32M and a promotional cost of only \$17M. What does our model predict about the first year box office receipts?

▲

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 4.28. 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 4.29. Write MATLAB code that takes a matrix A (not necessarily square) and outputs the 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 4.30. Write a script that tests your `MyQR` function on randomly generated mn 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? ▲

Problem 4.31. 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 distrubuted 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).

▲

4.5 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 4.32. Find the eigenvalues and eigenvectors of $A = \begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$ by hand. ▲

Problem 4.33. 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 $\lambda_1 = 0$?

▲

Problem 4.34. 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 4.35. 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 4.36. 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,

$$A^k \mathbf{x} = \underline{\hspace{6cm}}$$

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 $\lim_{k\to\infty} A^k \mathbf{x}$?

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 4.37. 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. ▲

4.6 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 4.38. 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 4.39. An $n \times n$ matrix A has n orthogonal eigenvectors if and only if A is a symmetric matrix.

Definition 4.40. The **singular values** of an $m \times n$ matrix A are the square roots 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 4.41. 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 4.39).

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 4.42. 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 $A^T A$:
   `[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 $A^T A$ (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.
   ```
   B=zeros(size(Sigma));
   for j=1:r
   Sigma(j,j) = ...
   B(j,j) = ...
   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 4.43. 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')
```

▲

4.7 Low Rank Approximations of Matrices

Problem 4.44. 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.

▲

4.8 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 4.1 (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 4.1, node 3 has three outlinks (to nodes 1, 2, and 5) and 1 inlink (from node 1).

We will first define some notation in the PageRank algorithm:

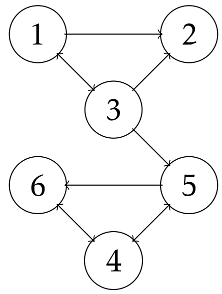

Figure 4.1. Sample graph of a web with six pages.

- $|P_i|$ 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 4.1 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 4.1. 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 4.45. If A is an $n\times n$ matrix with n linearly independent eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \dots, \mathbf{v}_n$ and associated eigenvalues $\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_n$ 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 $c_1, c_2, c_3, \dots, c_n$ 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 4.46. A **probability vector** is a vector with entries on the interval $[0, 1]$ that add up to 1.

Definition 4.47. A **stochastic matrix** is a square matrix whose columns are probability vectors.

Theorem 4.48. 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: $0 \le \lambda_n < 1$.

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 4.49. Finish writing the hyperlink matrix H from Figure 4.1. ▲

Problem 4.50. Write MATLAB code to implement the iterative process defined previously. Make a plot that shows how the rank evolves over the iterations. ▲

Problem 4.51. 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 4.52. 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

$$\mathbf{x}_{equilib} = \lim_{k \to \infty} A^k \mathbf{x}_0 = \underline{\hspace{3cm}}.$$

Proof. First note that A is an $n \times n$ stochastic matrix so from Theorem 4.48 we know that there are n linearly independent eigenvectors. We can then substitute the eigenvalues from Theorem 4.48 in Theorem 4.45. Noting that if $0 < \lambda_j < 1$ we have $\lim_{k \to \infty} \lambda_j^k = 0$ the result follows immediately. □

Problem 4.53. Discuss how Theorem 4.52 greatly simplifies the PageRank iterative process described previously. In other words: there is no reason to iterate at all. Instead, just find ____________. ▲

Problem 4.54.

Now use the previous two problems to find the resulting PageRank vector from the web in Figure 4.1? Be sure to rank the pages in order of importance. Compare your answer to the one that you got in problem 2. ▲

Problem 4.55. Consider the web in Figure 4.2.

(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 4.52 and the dominant eigenvector.

(c) Rank the pages in order of importance.

▲

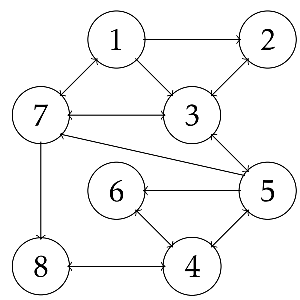

Figure 4.2. Graph of a web with eight pages.

4.9 Principal Component Analysis (Incomplete)

4.10 Exercises

Problem 4.56. 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 taht the center of mass of its pixels is at the center of the picture.

(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.

(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 $i-1$.

Be sure to write code to test all of the 10,000 test digits (this will likely take 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

$$\left\| D - \left(\frac{D \cdot T(i,:)}{T(i,:) \cdot T(i,:)} \right) T(i,:) \right\|$$

▲

Problem 4.57. 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 4.58. 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 4.59. 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

▲

Problem 4.60. Coding Challenge: 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 ▲

Theorem 4.61 (Eigen-Structure of Symmetric Matrices)**.** If A is a symmetric matrix with eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$ then $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. Furthermore, the eigenvectors will be orthogonal to each other.

Problem 4.62. 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 4.61 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.
Hint: 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 symmetric.
```

▲

Chapter 5

Numerical Ordinary Differential Equations

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.

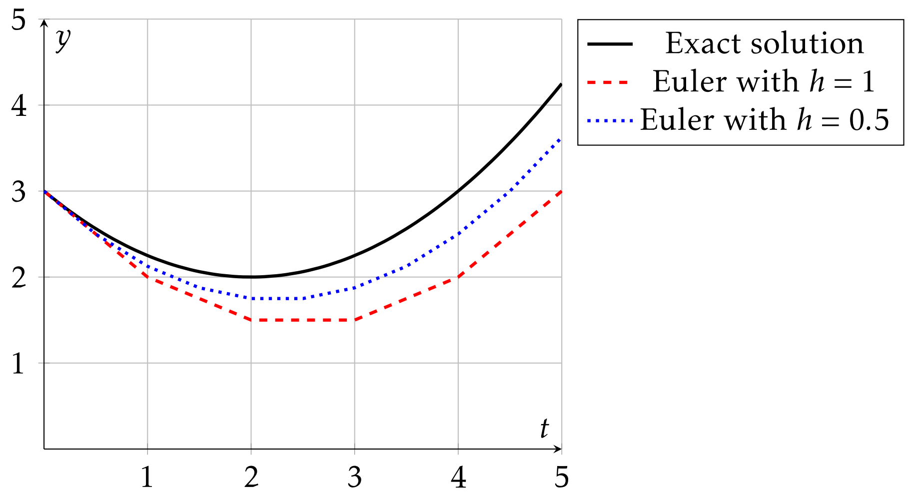

Figure 5.1. A depiction of Euler's method with step size $h = 1$ (red) and $h = 0.5$ (blue).

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 (prey) at time t. The relationship between the species can be modeled by the classic 1920's Lotka-Volterra Model:

$$\begin{cases} x' &= \alpha x - \beta x y \\ y' &= -\delta y + \gamma x y \end{cases}$$

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. ▲

Problem 5.7. The Runge-Kutta 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.

$$\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\
k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\
k_4 &= f(t_n + h, y_n + hk_3) \\
y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

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 5.8. 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 5.9. Solving systems of ordinary differential equations would become challenging if we were to continue coding in the same way as in the previous problem. 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). ▲

5.2 Implicit Methods and Shooting Methods

Problem 5.10. 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 5.11. 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 5.12. 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 5.13. 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 feelinfs 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 5.14. 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 Exercises

Problem 5.15. 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 5.16. 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

$$P'(t) = k\|E'(t)\| \frac{E(t) - P(t)}{\|E(t) - P(t)\|}. \tag{5.4}$$

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

$$E(t) = \begin{pmatrix} 0 \\ 5t \end{pmatrix}$$

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

$$E(t) = \begin{pmatrix} 5 + \cos(2\pi t) + 2\sin(4\pi t) \\ 4 + 3\cos(3\pi t) \end{pmatrix}$$

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.

Note: It may be easiest to build this code from scratch instead of using one of our prewritten codes. ▲

Problem 5.17. 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

$$\begin{aligned} \frac{dB}{dt} &= -0.05B + \left(\frac{0.02}{500}\right)BK \\ \frac{dK}{dt} &= 0.25K - \left(\frac{0.10}{150000}\right)BK. \end{aligned}$$

(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 5.18. 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?
Hint: You should model this with a system of first order nonlinear differential equations in the spatial positions x and y. ▲

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 Review of Multivariable Calculus

Let's start with some basic review of multivariable calculus.

Problem 6.1. With your partner answer each of the following questions without the aid of the internet.

(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 **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 **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 Introduction to PDEs

This section is meant to be an introduction to most of the primary differential equations of interest in basic mathematical physics.

We start with a brief derivation of a *general conservation law*. The result being a partial differential equation that can be used for 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

$$\frac{\partial}{\partial t} \iiint_{\Omega} u dV = -\iint_{\partial\Omega} \mathbf{q} \cdot n dA + \iiint_{\Omega} f dV. \tag{6.1}$$

This is a global balance law in the sense that it holds for all volumes Ω. The 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 get

$$\frac{\partial}{\partial t}\iiint_{\Omega} u\,dV = -\iiint_{\Omega} \nabla\cdot\mathbf{q}\,dV + \iiint_{\Omega} f\,dV. \tag{6.2}$$

Gathering all of the terms on the right of (6.2), interchanging the integral and the derivative on the left (since the volume is not changing in time), and rewriting gives

$$\iiint_{\Omega}\left(\frac{\partial u}{\partial t} + \nabla\cdot\mathbf{q}\right)dV = \iiint_{\Omega} f\,dV \tag{6.3}$$

If we presume that this equation holds for all volumes Ω then the integrands must be equal and we get the local balance law

$$\frac{\partial u}{\partial t} + \nabla\cdot\mathbf{q} = f. \tag{6.4}$$

Simplification of the Local Balance Law

If equation (6.4) 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

$$\frac{\partial u}{\partial t} + \nabla\cdot\mathbf{q} = 0. \tag{6.5}$$

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

$$\mathbf{q} = -k\nabla C. \tag{6.6}$$

Combining (6.6) with (6.5) (and assuming that k is independent of space, time, and concentration) gives

$$\frac{\partial C}{\partial t} = k\nabla\cdot\nabla C. \tag{6.7}$$

In the presenence of external sources of mass, (6.7) is

$$\frac{\partial C}{\partial t} = k\nabla\cdot\nabla C + f(x). \tag{6.8}$$

Problem 6.3. What does (6.8) equation look like in terms of spatial derivatives on the right-hand side?

$$\frac{\partial C}{\partial t} = \underline{\hspace{6cm}} \quad \text{(1 Spatial Dimension)}$$
$$\frac{\partial C}{\partial t} = \underline{\hspace{6cm}} \quad \text{(2 Spatial Dimensions)}$$
$$\frac{\partial C}{\partial t} = \underline{\hspace{6cm}} \quad \text{(3 Spatial Dimensions)}$$

▲

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

$$q = -k\nabla T. \tag{6.9}$$

Making the same simplifications as in the mass balance equation we arrive at

$$\frac{\partial T}{\partial t} = k\nabla \cdot \nabla T. \tag{6.10}$$

In the presence of external sources of heat, (6.10) becomes

$$\frac{\partial T}{\partial t} = k\nabla \cdot \nabla T + f(x). \tag{6.11}$$

Problem 6.4. What does (6.11) equation look like in terms of spatial derivatives on the right-hand side?

$$\frac{\partial T}{\partial t} = \underline{\hspace{6cm}} \quad \text{(1 Spatial Dimension)}$$
$$\frac{\partial T}{\partial t} = \underline{\hspace{6cm}} \quad \text{(2 Spatial Dimensions)}$$
$$\frac{\partial T}{\partial t} = \underline{\hspace{6cm}} \quad \text{(3 Spatial Dimensions)}$$

▲

Laplace's Equation and Poisson's Equation

Equations (6.8) and (6.11) 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

$$\nabla \cdot \nabla u = 0. \tag{6.12}$$

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 (6.12) is sometimes denoted as $\nabla\cdot\nabla u = \nabla^2 u = \Delta u$, and in terms of the partial derivatives it is written as

$0 =$ __________________________ (1 Spatial Dimension)

$0 =$ __________________________ (2 Spatial Dimensions)

$0 =$ __________________________ (3 Spatial Dimensions)

If there is a time-independent external source the right-hand side of (6.12) will be non-zero and we arrive at Poisson's equation:

$$\nabla\cdot\nabla u = -f(x). \tag{6.13}$$

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 taking 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	$\dfrac{\partial u}{\partial t} = k\nabla\cdot\nabla u + f(x)$	Diffusion
Laplace's Equation	$k\nabla\cdot\nabla u = -f(x)$	Minimal Energy Surfaces
The Wave Equation	$\dfrac{\partial^2 u}{\partial t^2} = k\nabla\cdot\nabla u + f(x)$	Wave phenomena

6.3 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 6.5. 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.

6.4 1D and 2D Heat Equation

In this section we'll use a technique called *the finite difference method* to find numerical approximations to the heat equation

$$u_t = k\nabla \cdot \nabla u + f(x).$$

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 $u_t = k\left(u_{xx} + u_{yy}\right) + f(x,y)$. 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. To begin with we'll let $f(x) = 0$.

Problem 6.6. Now we would like to consider the time dependent heat equation

$$u_t = \kappa \nabla \cdot \nabla u$$

in 1 spatial dimension. Note that κ is the diffusivity (the rate of diffusion) so in terms of physical problems, if κ is small then the diffusion occurs slowly and if κ is large then the diffusion occurs quickly.

In 1 spatial dimension, the heat equation is simply

$$u_t = \kappa u_{xx}$$

and we can approximate the derivatives with an Euler-type approximation of the time and a central difference in space:

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = \kappa \left(\frac{U_{i+1}^n - 2U_i^n + U_{i+1}^n}{\Delta x^2} \right).$$

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

$$U_i^{n+1} = U_i^n + \frac{\kappa \Delta t}{\Delta x^2}\left(U_{i+1}^n - 2U_i^n + U_{i+1}^n \right). \tag{6.14}$$

Implement (6.14) in MATLAB to approximate the solution to the following problem:

Solve: $u_t = 0.5u_{xx}$ with $x \in (0,1)$, $u(0,x) = \sin(2\pi x)$, $u(t,0) = 0$, and $u(t,1) = 0$.

▲

Problem 6.7. You may have noticed in the previous problem that you will have terribly unstable solutions for certain choices of Δx and Δt. Set $\kappa = 1$ in the previous problem and experiment with choices for Δx and Δt to find where (6.14) gives a stable numerical solution to the heat equation. For each choice of Δx and Δt report the value of $\frac{\Delta t}{\Delta x^2}$. ▲

Problem 6.8. 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:

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = \frac{1}{2}\left[\kappa\left(\frac{U_{i+1}^n - 2U_i^n + U_{i+1}^n}{\Delta x^2}\right) + \kappa\left(\frac{U_{i+1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}}{\Delta x^2}\right)\right].$$

Letting $r = \kappa\Delta t/(2\Delta x^2)$ we can rearrange to get

$$-rU_{i-1}^{n+1} + (1+2r)U_i^{n+1} - rU_{i+1}^{n+1} = rU_{i-1}^n + (1-2r)U_i^n + rU_{i+1}^n.$$

This can now be viewed as a system of equation that can be solved for all U_j^{n+1} all at once with one linear solve. Write MATLAB code to solve the heat equation from the previous problems with the Crank-Nicolson method. ▲

Problem 6.9. In Figure 6.1 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

$$\text{Solve: } u_t = \kappa\nabla\cdot\nabla u \quad \text{in} \quad x \in \Omega$$

with

$$u(0,x,y) = 2000(x-0.5)(y-0.5)\exp\left(-\frac{(x-0.5)^2+(y-0.5)^2}{0.02}\right)$$

subject to the boundary conditions in the figure.

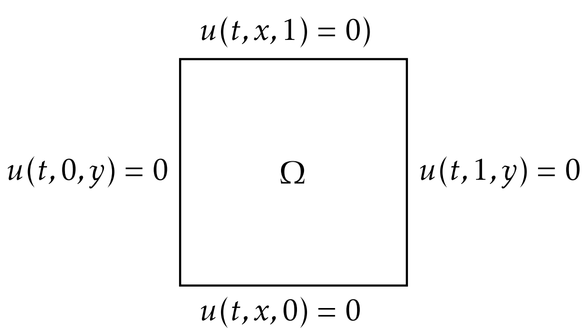

Figure 6.1. Dirichlet boundary conditions for a 2D Poisson equation.

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. ▲

6.5 1D and 2D Wave Equation

Problem 6.10. 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

$$u_{tt} = \alpha^2 u_{xx} \tag{6.15}$$

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

$$\begin{aligned} u_{tt}(t_{n+1}, x_j) &\approx \frac{U_j^{n+1} - 2U_j^n + U_j^{n-1}}{\Delta t^2} \\ u_{xx}(t_n, x_j) &\approx \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{\Delta x^2} \end{aligned}$$

▲

Problem 6.11. There is a natural stability question waiting to be asked about the discretization of the 1D wave equation. Ask and answer this question. ▲

Problem 6.12. 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

$$\text{Solve: } u_{tt} = \alpha^2 \nabla \cdot \nabla u \quad \text{in} \quad x \in \Omega \quad \text{with} \quad u(0,x,y) = \sin(2\pi(x-0.5))\sin(2\pi(y-0.5))$$

subject to the boundary conditions in the figure.

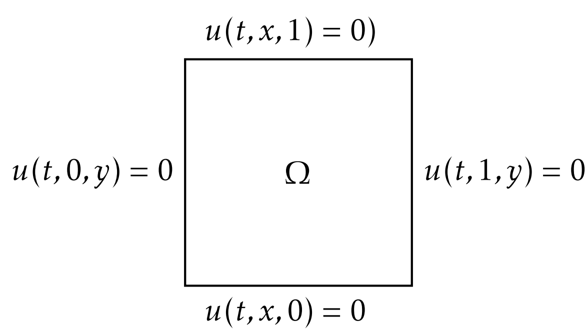

Figure 6.2. Dirichlet boundary conditions for a 2D Poisson equation.

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. ▲

6.6 Traveling Waves

Problem 6.13. A traveling wave can be modeled by the PDE

$$u_t + au_x = 0$$

where a 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 $u_t + au_x = 0$ then $u(t,x) = \eta(x-at)$ is the analytic solution to the PDE. You should check this using the chain rule. ▲

Problem 6.14. Three ways to fix the issues seen in the previous problem are the "Leapfrog" scheme, the "Lax-Friedrichs" scheme, and the "Lax-Wendroff" scheme:

$$\text{Leapfrog: } \frac{U_j^{n+1} - U_j^{n-1}}{2\Delta t} = -a\frac{U_{j+1}^n - U_{j-1}^n}{2\Delta x} \tag{6.16}$$

$$\text{Lax-Friedrichs: } \frac{U_j^{n+1} - \frac{1}{2}\left(U_{j+1}^n + U_{j-1}^n\right)}{\Delta t} = -a\frac{U_{j+1}^n - U_{j-1}^n}{2\Delta x} \tag{6.17}$$

$$\text{Lax-Wendroff: } U_j^{n+1} = U_j^n - \frac{a\Delta t}{2\Delta x}\left(U_{j+1}^n - U_{j-1}^n\right) + \left(\frac{a^2\Delta t^2}{2\Delta x^2}\right)\left(U_{j-1}^n - 2U_j^n + U_{j+1}^n.\right) \tag{6.18}$$

Implement all of these schemes and discuss stability and consistency. ▲

6.7 1D and 2D Steady State PDEs

Problem 6.15. 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 (6.12). Write this equation in terms of 1-dimensional spatial derivatives and solve for the temperature profile by hand. ▲

Problem 6.16. 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 6.17. 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 (6.13). 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 6.18. Generalize the previous problem with a MATLAB function that solves the 1D Poisson boundary valued equation:

$$\text{Solve: } u_{xx} = -f(x) \quad \text{on} \quad x \in (x_0, x_n) \quad \text{with} \quad u(x_0) = \alpha \quad \text{and} \quad u(x_n) = \beta.$$

```
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 6.19. 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 6.20. Now let's ramp up our discussion of the Poisson equation to two spatial dimensions. In Figure 6.3 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.

▲

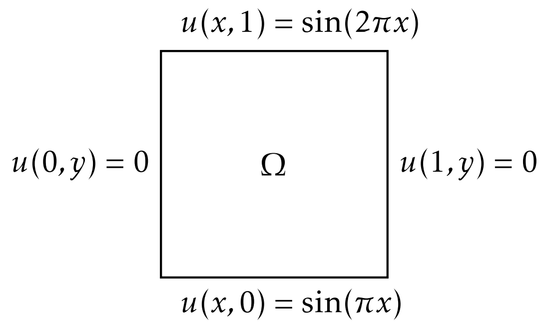

Figure 6.3. Dirichlet boundary conditions for a 2D Poisson equation.

6.8 Exercises

Problem 6.21. In this problem we will solve a more realistic 1D heat equation. We will allow the diffusivity to change spatially, so $\kappa = \kappa(x)$ and we want to solve

$$u_t = (\kappa(x)u'(x))'$$

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 $\kappa(x)$ to be the parabola $\kappa(x) = x(1-x)$. We start by doing some calculus to rewrite the differential equation:

$$u_t = \kappa(x)u''(x) + \kappa'(x)u'(x).$$

Your job is:

1. 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.

2. Write a MATLAB script to find an approximate solution to this problem.

3. Write a clear and thorough discussion about how your will choose Δx and Δt to give stable solutions to this equation.

4. Graphically compare your solution to this problem with a heat equation where κ is taken to be the average diffusivity: $\kappa = 0.5$. How does the changing diffusivity change the shape of the solution?

▲

Problem 6.22. 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.

▲

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 $0 \le r < 2$. 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.
Incorrect:

In the equation below we consider the domain $x \in (-1, 1)$

$$f(x) = \sum_{j=1}^{\infty} \frac{x^n}{n!}.$$

Correct:

In equation (A.1) we consider the domain $x \in (-1, 1)$

$$f(x) = \sum_{j=1}^{\infty} \frac{x^n}{n!}. \tag{A.1}$$

"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

Scientific Computing with MATLAB

B.1 Looping

B.1.1 The For Loop

B.1.2 The While Loop

B.2 Conditional Statements

B.2.1 If Statements

B.2.2 Case-Switch Statements

B.3 Functions

B.4 Plotting

B.5 Animations

B.6 Additional Resources

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}
```

$$\begin{aligned}
\frac{\partial x}{\partial s} &= zx & \text{with} && x(0)=1 && \text{(C.2)}\\
\frac{\partial y}{\partial s} &= x^2y & \text{with} && y(0)=t && \text{(C.3)}\\
\frac{\partial z}{\partial s} &= xyz & \text{with} && z(0)=t^2 && \text{(C.4)}
\end{aligned}$$

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.

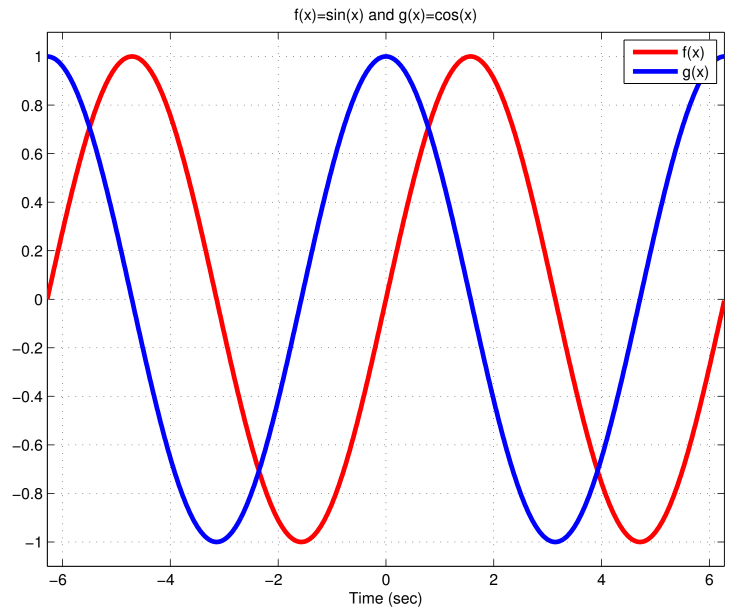

Figure C.1. Figure for Example C.4

- 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:

$$\mathbb{C} \hspace{1cm} \mathbb{N} \hspace{1cm} \mathbb{Q} \hspace{1cm} \mathbb{R} \hspace{1cm} \mathbb{Z}$$

- 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}
```

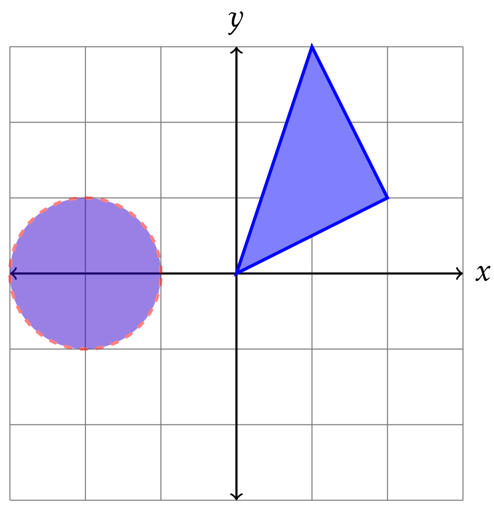

Figure C.2. A simple Tikz picture

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}
```

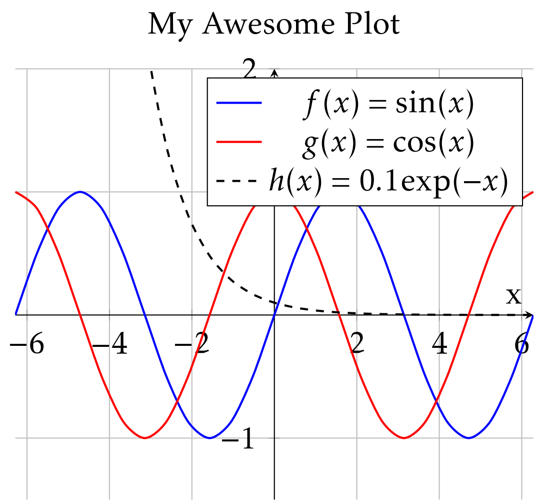

Figure C.3. A figure drawn with the `tikzpicture` and `axis` commands (leveraging the `pgfplots` package in the backgroud).

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,
```

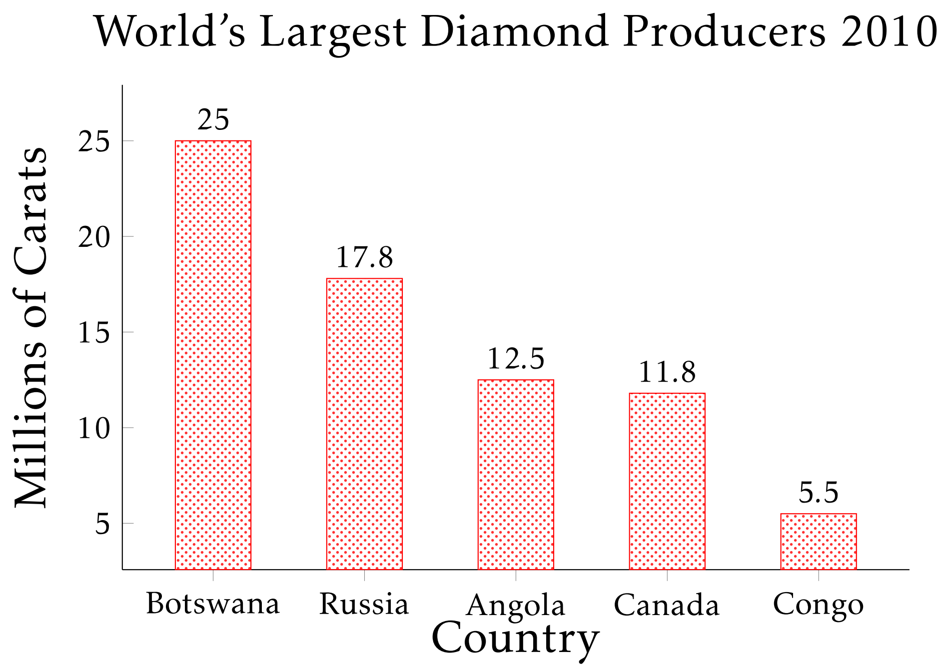

Figure C.4. Figure for Example C.7

```
            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.

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.