Elementary Numerical Methods
and computing with Python

Steven Pav[1] Mark McClure[2]

April 14, 2016

[1]Portions Copyright © 2004-2006 Steven E. Pav. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

[2]Copyright © 2016 Mark McClure. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Contents

Preface

This preliminary version 0.2.1 of an open numerical methods text has been assembled by Mark McClure for use in UNCA's Math/CS 441 course in Numerical Analysis. It draws largely on Stephen Pav's Numerical Methods Course Notes. Most of chapters 4 through 8 of this version, as well as parts of chapters 2 and 3, are drawn directly from that work. The little that remains is my own, though, I hope to expand it as I teach the subject.

History

0.1 2005 (Pav's complete original)

0.2 January 11, 2016 - First new draft
Chapters 1, 2, and 3 are largely new, though section 2.1.4 on "Taylor series" and section 3.4 on "Loss of significance" are mostly Pav's work.

0.2.1 January 16, 2016
Incremental changes to sections

- 2.1.1 Geometric Series
- 2.1.2 The integral test
- 2.1.3 Alternating series

0.2.2 January 31, 2016
Addition of sections

- 3.3.3 Distribution of computer numbers
- 4.2 Functional iteration

Modifications to section 4.1 The bisection method, including Python code.
Addition of several exercises to chapters 3 and 4.
Fixed many typos and some hyperlinks

License

As Pav's work is licensed under GNU FDL V2, so is this. A full copy of the license may be found at the end of the text.

Chapter 1

Introduction

The (over?)emphasis of the traditional calculus curriculum on algebraic manipulation often leaves the impression that most interesting problems in mathematics can be solved in closed form. Yet, this is far from true! In fact, *most* problems in applied mathematics can only be solved *approximately* and there are *very many* quite elementary problems that can only be solved approximately as well.

1.1 Examples

Let's illustrate the idea of the opening paragraph with a few simple examples:

- $x^p - x - 1 = 0$ for various, postive integers p.
- $\cos(x) = 0$
- $\cos(x) = x$

These are all examples of equations and a solution is simply some value of x that makes the equation true. Note that techniques for solving equations of all types is of tremendous importance in applied mathematics since, typically, equations arise in the process of mathematical modelling and their solutions have interpretation in the context of the applied problem. In this text, we focus on the mathematical portion of the problem from a numerical perspective.

Example 1.1. *Solve the quadratic* $x^2 - x - 1 = 0$.

This is a very simple problem that will be useful in demonstrating the difference between a *closed form* or *exact solution* form versus an *approximate solution*. It's very easy to solve this equation using the quadratic formula. There is one positive solution, namely

$$x = \frac{1 \pm \sqrt{5}}{2}.$$

This is is the exact or closed form solution. It's not too hard to get a sense of this number. The square root of 5 is a bit bigger than 2 so we expect x to be a bit bigger than $\frac{3}{2} = 1.5$. More precisely, we find

$$x \approx 1.61803.$$

The distinction between these two types of solutions is very important for us. It is this second type of solution that we are mainly interested in here.

Example 1.2. *Solve the cubic* $x^3 - x - 1 = 0$.

This is quite a bit harder but it can solved in closed form using a technique called Cardano's method. There is one positive root which is exactly

$$x = \frac{1}{3}\sqrt[3]{\frac{27}{2} - \frac{3\sqrt{69}}{2}} + \frac{\sqrt[3]{\frac{1}{2}\left(9 + \sqrt{69}\right)}}{3^{2/3}}.$$

Well, this number is *much* harder to get a handle on. Do you get a sense of how big it is just by looking? It probably makes sense to look at a graph, which clearly shows a root between 1 and 1.5. Using the techniques we learn in this text, we'll be able to show that the solution is more precisely $x \approx 1.32472$.

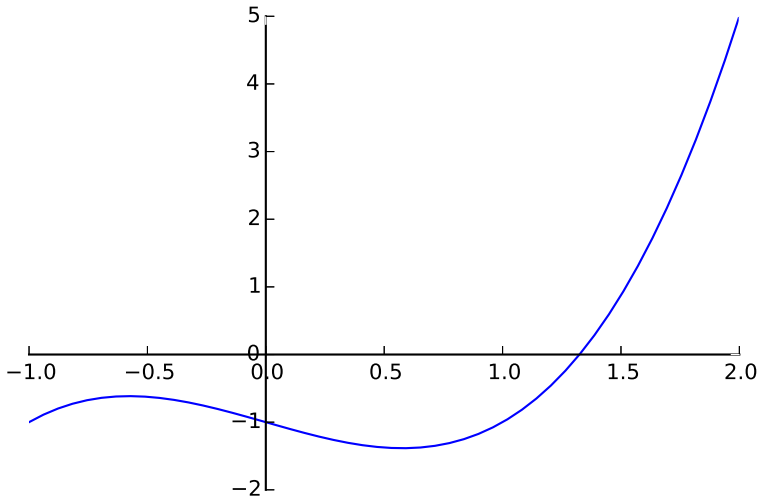


Figure 1.1: The graph of $f(x) = x^3 - x - 1$

The reader is invited to use a computer algebra system to find the exact forms of the solutions of $x^4 - x - 1 = 0$ and $x^5 - x - 1 = 0$. Te solution to the first is so complicated that it is virtually useless. The second *cannot*

be expressed in closed form; this is the content of Abel's famous impossibility theorem.

Example 1.3. *Solve the trigonometric equation* $\cos(x) = 0$.

Now we're back on easy street! The solutions are exactly the odd multiples of $\pi/2$:

$$x_n = \frac{2n+1}{2}\pi.$$

Again, this is expressed in closed form. If we are interested in a decimal approximation to the first positive root, we have:

$$x \approx 1.5708.$$

Example 1.4. *Solve the trigonometric equation* $\cos(x) = x$.

Well, this seems quite a bit harder! In fact, it can be proved that there is no closed form solution. It's very easy to see a solution between 0.5 and 1 by looking at a graph, however. Again, using the techniques in this text, we'll be able to find the more precise approximations $x \approx 0.739085$.

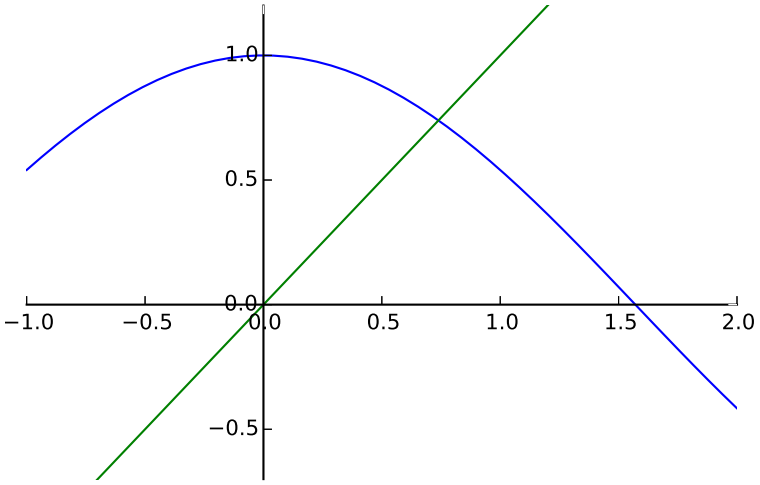


Figure 1.2: The graph of the cosine together with $y = x$

1.2 Iteration

Example 1.4 can be solved using a technique called *functional iteration*, a topic we will explore further in chapter 4. The idea is simple: Given a function f, start

with an initial seed, namely any real number x_0. Let $x_1 = f(x_0)$, $x_2 = f(x_1)$, and generally $x_n = f(x_{n-1})$. This generates a sequence which, if convergent, should converge to a fixed point of f.

This idea is very easy to implement with Python:

```
from numpy import cos, abs
x1 = 1
x2 = cos(1)
cnt = 0
while abs(x1-x2) > 10**(-8) and cnt < 100:
    x1 = x2
    x2 = cos(x2)
    cnt = cnt + 1
(x2,cnt)
# Out: (0.7390851366465718 3, 45)
```

The point is that `cos(x2)` must be very close to `x2` by this point.

```
cos(x2)
# Out: 0.7390851309037207 4
```

Thus, we've *approximately* found a fixed point.

In addition to illustrating some nice mathematics, this example illustrates the fact that we'll be playing with Python in this text. There is a companion *Crash course in Python for beginning numerical analysts* in which you can learn how to use Matplotlib to create graphs like figure 1.3.

1.3 Topics

In numerical analysis, we tackle the classic problems of applied mathematics from a numerical perspective. These include

- Solving equations, or finding roots of functions
- Fitting functions to data exactly or approximately or otherwise modelling real data with numerical procedures
- Differentiating functions based on data
- Computing definite integrals.

Potential further topics include

- Optimizing functions by finding extrema
- Solving ordinary differential equations given initial conditions
- Doing all the above in higher dimensions
- Solving problems in linear algebra given (potentially) very large matrices.

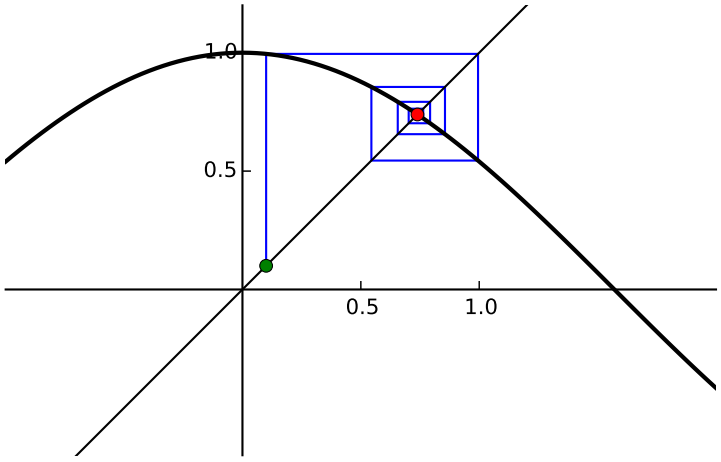


Figure 1.3: Cobweb plot for $f(x) = \cos(x)$

To do any of this effectively, we must consider computer issues such as

- Representation of numbers on the computer
- Computational error
- Implementations of our algorithms.

Chapter 2

Some mathematical preliminaries

2.1 Series

Series of numbers and functions are typically encountered in a second semester calculus class are central to much of what we do in numerical analysis. A general series has the form

$$\sum_{n=m}^{\infty} a_n.$$

Each a_n is a number and the sum of the series is defined to be

$$\lim_{N\to\infty} \sum_{n=m}^{N} a_n.$$

The starting point m is often 0 or 1, but may be any integer.

There are two central questions associated with any numerical series.

- Does the series converge?
- If so, what is the sum?

Often, we can answer the first question, which is not too hard, but not the second. Approximation of the sum of a convergent series, whose sum we do not know exactly, is a basic question in numerical computing, in fact.

2.1.1 Geometric series

A geometric series has the basic form

$$\sum_{n=0}^{\infty} r^n.$$

In calculus, we learn that the N^{th} partial sum of this series can be written

$$\sum_{n=0}^{N} r^n = \frac{1 - r^{N+1}}{1 - r} \to \frac{1}{1 - r},$$

as $N \to \infty$. This yields the formula

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1 - r},$$

provided that $|r| < 1$. A simple factorization yields the somewhat more general formula

$$\sum_{n=m}^{\infty} ar^n = ar^m \sum_{n=0}^{\infty} ar^n = \frac{r^m}{1 - r}.$$

Among other things, this can be used to resolve Zeno's famous paradox:

$$\sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n = \frac{1/2}{1 - 1/2} = 1. \tag{2.1}$$

Decimal and binary expansions

In numerical analysis we are very interested in representations of real numbers. One of the most fundamental such representations is that of a decimal expansion. It should be clear that a decimal expansion is really just a sum. When we write $\pi \approx 3.14159$, we mean that

$$\pi \approx 3 + \frac{1}{10} + \frac{4}{10^2} + \frac{1}{10^3} + \frac{5}{10^4} + \frac{9}{10^5}.$$

When the digits form an eventually repeating pattern, the sum can be computed exactly using the geometric series formula. For example, we can prove the very fun fact that $0.\overline{9} = 1$ via the computation

$$0.\overline{9} = \sum_{n=1}^{\infty} \frac{9}{10^n} = \frac{9(1/10)}{1 - 1/10} = \frac{9/10}{9/10} = 1.$$

As a slightly tricker example, note that

$$1.23\overline{345} = 1 + \frac{2}{10} + \frac{3}{10^2} + \frac{1}{100} \sum_{n=1}^{\infty} \frac{345}{1000^n} = \frac{20537}{16650}.$$

Computers are digital and most floating point processors work in base 2. Thus, it's worth thinking about binary expansions. Such an expansion has the form

$$(d_n)(d_{n-1})(d_{n-2}) \cdots (d_1)(d_0)_{\dot{2}}(d_{-1})(d_{-2}) \cdots = \sum_{k=-\infty}^{n} \frac{d_k}{2^k}.$$

where each digit d_k is either zero or one. For example

$$101_{\dot{2}}011 = 2^2 + 1 + \frac{1}{2^2} + \frac{1}{2^3} = \frac{43}{8}.$$

Repeating binary expansions can be computed using the geometric series formula, just as repeating decimal expansions can. The analog to the decimal fact that $0.\overline{9} = 1$ is $0_{\dot{2}}\overline{1} = 1$. The sum we must evaluate to show this is exactly Zeno's sum from equation 2.1.

We now consider the question of representing nice decimal numbers in binary. The fact is that terminating decimal numbers are not necessarily terminating in binary. This leads to some confusing behavior, as we'll see in chapter 3 on computer arithmetic.

The simplest such example is the decimal 0.1, which is an exact representation of the rational number 1/10. The binary expansion, however, is nonterminating:

$$0_{\dot{2}}0001\overline{1001} = \frac{1}{2^4}\left(1 + \sum_{n=1}^{\infty} \frac{9}{2^{4k}}\right).$$

The geometric series formula easily shows that this sums to 1/10. The 4 in the exponent of 2^{4k} arises since we've got a repeating block of length 4 and the 9 arises because 1001 is its binary expansion. I chose to start the repeating block after a 1 because, as we'll see in chapter 3, that's standard for (normalized) floating point numbers.

Base 16, or hexadecimal, can be useful as well, mainly because it provides a convenient shorthand for binary. These expansions have the form

$$(d_n)(d_{n-1})(d_{n-2})\cdots(d_1)(d_0)_{\dot{16}}(d_{-1})(d_{-2})\cdots = \sum_{k=-\infty}^{n} \frac{d_k}{2^k}.$$

where each digit d_k is an integer between 0 and 15. Often, the digits 10 through 15 are abbreviated by the characters a through f. Note that each hexadecimal digit has a binary representation using four digits.

binary	0000	0001	$\cdots$	1100	1101	1110	1111
hexadecimal	0	1	$\cdots$	c	d	e	f
decimal	0	1	$\cdots$	12	13	14	15

This makes it easy to go back and forth by simply make these replacements.

2.1.2 The integral test

The integral test is typically stated in a Calculus test as *just* a test for convergence.

Theorem 2.1 (Integral test for convergence)**.** *Suppose that f is a positive, real-valued, continuous, decreasing function on $[1, \infty)$ and let $a_n = f(n)$ for*

each natural number n. Then

$$\sum_{n=1}^{\infty} a_n \text{ converges iff } \int_1^{\infty} f(x)\,dx \text{ converges.}$$

From the viewpoint of numerical analysis, this can be made much more precise.

Theorem 2.2 (Integral test)**.** *Suppose again that f is a positive, real-valued, continuous, decreasing function on $[1, \infty)$ and let $a_n = f(n)$ for each natural number n. Suppose also that*

$$\int_1^{\infty} f(x)\,dx$$

converges. Then,

$$\int_N^{\infty} f(x)dx \leq \sum_{n=N}^{\infty} a_n \leq f(N) + \int_N^{\infty} f(x)dx.$$

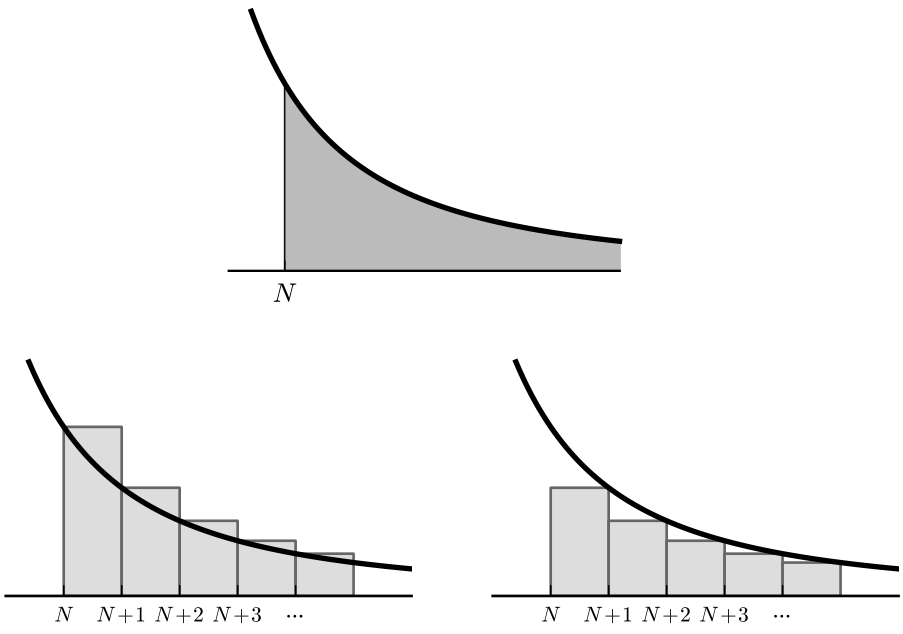


Figure 2.1: Comparing an integral to a sum

Proof. The integral test follows from a simple geometric interpretation of the quantities involved. Assuming the sum is convergent, the series

$$\sum_{n=N}^{\infty} a_n$$

can be interpreted as the combined areas of the rectangles shown in the bottom left of figure 2.1. The integral

$$\int_N^{\infty} f(x)\,dx$$

can be interpreted as the area under the curve $y = f(x)$ and over the interval $[N, \infty)$, as shown in the top of figure 2.1. As the rectangles lie wholly above the curve, we have the first inequality, namely

$$\int_N^{\infty} f(x)\,dx \leq \sum_{n=N}^{\infty} a_n.$$

Next, comparing the integral to the sum

$$\sum_{n=N+1}^{\infty} a_n$$

as shown in the bottom right of figure 2.1, we see that

$$\sum_{n=N+1}^{\infty} a_n \leq \int_N^{\infty} f(x)\,dx.$$

Since $a_N = f(N)$, we can add a_N to the left and $f(N)$ to the right to obtain

$$\sum_{n=N}^{\infty} a_n \leq f(N) + \int_N^{\infty} f(x)dx,$$

as desired. □

The integral test provides a tool to solve an important canonical problem in numerical analysis. Suppose we'd like to approximate some value, say x_0. We have a sequence x_n, which we know converges to x_0. Thus, one reasonable approximation is x_N for some large value of N. But now, suppose we are given some particular error tolerance, say ε, and we wish to ensure that our approximation is within ε of the actual result. In symbols, we want

$$|x_N - x_0| < \varepsilon.$$

How large does N have to be? We don't want N to be ridiculously large, because we want to be efficient, but we definitely want it to be large enough.

In our particular problem, the value we'd like to approximate x_0 is an infinite sum. The easy approximation is a partial sum. In symbols,

$$\sum_{n=0}^{\infty} a_n \approx \sum_{n=0}^{N} a_n.$$

How large does N have to be to ensure that our approximation is as close as we'd like? Well,

$$\sum_{n=0}^{\infty} a_n - \sum_{n=0}^{N-1} a_n = \sum_{n=N}^{\infty} a_n.$$

Thus, the integral test theorem 2.2 gives us upper and lower bounds on exactly this quantity.

Example 2.3. *Estimate*

$$\sum_{n=1}^{\infty} \frac{1}{n^3}$$

so that your result is within 0.0001 *of the actual value.*

Solution: We first compute

$$\int_N^{\infty} \frac{1}{x^3} dx = \frac{1}{2N^2}.$$

For $N > 2$ we have

$$\frac{1}{N^3} + \frac{1}{2N^2} \leq \frac{1}{N^2}.$$

Thus, if we choose N to be an integer such that $1/N^2 < 0.0001$, then our error will be even smaller. Any N larger than 100 will certainly work. Thus, we have

$$\sum_{n=1}^{\infty} \frac{1}{n^3} \approx \sum_{n=1}^{100} \frac{1}{n^3} \approx 1.202007$$

Note that this sum can be easily computed in Python:

```
sum([1/n**3 for n in range(1,101)])
# Out: 1.2020074006596781
```

2.1.3 Alternating Series

Another theorem for proving convergence, with an error estimate, is the alternating series test:

Theorem 2.4 (Alternating Series Test)**.** *If* $a_0 \geq a_1 \geq a_2 \geq a_3 \geq \ldots \geq 0$, *and* $\lim_{k\to\infty} a_k = 0$ *then the series*

$$\sum_{k=0}^{\infty} (-1)^k a_k = a_0 - a_1 + a_2 - \cdots$$

converges. Moreover, if you let S be the value of the sum, and S_n be the partial sum $\sum_{k=0}^{n}(-1)^k a_k$, then

$$|S_n - S| \leq a_{n+1}.$$

This theorem can help us figure out how many terms to take in an approximation.

Example 2.5. *Use the alternating series test to determine how many terms are needed to obtain a good approximation (say, within $\frac{1}{2}\times 10^{-9}$) to the alternating harmonic series:*

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

Solution: The alternating series test tells us that

$$|S_n - n| \leq a_{n+1} = \frac{1}{n+1}.$$

Thus it suffices to take $\frac{1}{n+1} \leq \frac{1}{2}\times 10^{-9}$. That is, we need to take 2 billion terms in our approximation; this is very slow convergence!

2.1.4 Taylor's Theorem

Recall from calculus the Taylor's series for a function, $f(x)$, expanded about some number, c, is written as

$$f(x) \sim a_0 + a_1(x-c) + a_2(x-c)^2 + \ldots.$$

Here the symbol $\sim$ is used to denote a "formal series," meaning that convergence is not guaranteed in general. The constants a_i are related to the function f and its derivatives evaluated at c. When $c = 0$, this is a MacLaurin series.

For example we have the following Taylor's series (with $c = 0$):

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \tag{2.2}$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots \tag{2.3}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots \tag{2.4}$$

Theorem 2.6 (Taylor's Theorem)**.** *If $f(x)$ has derivatives of order $0, 1, 2, \ldots, n+1$ on the closed interval $[a, b]$, then for any x and c in this interval*

$$f(x) = \sum_{k=0}^{n}\frac{f^{(k)}(c)\ (x-c)^k}{k!} + \frac{f^{(n+1)}(\xi)\ (x-c)^{n+1}}{(n+1)!},$$

where ξ is some number between x and c, and $f^k(x)$ is the k^{th} derivative of f at x.

We will use this theorem again and again in this class. The main usage is to approximate a function by the first few terms of its Taylor's series expansion; the theorem then tells us the approximation is "as good" as the final term, also known as the *error term*. That is, we can make the following manipulation:

$$\begin{aligned} f(x) &= \sum_{k=0}^{n} \frac{f^{(k)}(c)\ (x-c)^k}{k!} + \frac{f^{(n+1)}(\xi)\ (x-c)^{n+1}}{(n+1)!} \\ f(x) - \sum_{k=0}^{n} \frac{f^{(k)}(c)\ (x-c)^k}{k!} &= \frac{f^{(n+1)}(\xi)\ (x-c)^{n+1}}{(n+1)!} \\ \left| f(x) - \sum_{k=0}^{n} \frac{f^{(k)}(c)\ (x-c)^k}{k!} \right| &= \frac{\left|f^{(n+1)}(\xi)\right| |x-c|^{n+1}}{(n+1)!}. \end{aligned}$$

On the left hand side is the difference between $f(x)$ and its approximation by Taylor's series. We will then use our knowledge about $f^{(n+1)}(\xi)$ on the interval $[a,b]$ to find some constant M such that

$$\left| f(x) - \sum_{k=0}^{n} \frac{f^{(k)}(c)\ (x-c)^k}{k!} \right| = \frac{\left|f^{(n+1)}(\xi)\right| |x-c|^{n+1}}{(n+1)!} \leq M |x-c|^{n+1}.$$

Example Problem 2.7. *Find an approximation for $f(x) = \sin x$, expanded about $c = 0$, using $n = 3$.* Solution: *Solving for $f^{(k)}$ is fairly easy for this function. We find that*

$$\begin{aligned} f(x) = \sin x &= \sin(0) + \frac{\cos(0)\, x}{1!} + \frac{-\sin(0)\, x^2}{2!} + \frac{-\cos(0)\, x^3}{3!} + \frac{\sin(\xi)\, x^4}{4!} \\ &= x - \frac{x^3}{6} + \frac{\sin(\xi)\, x^4}{24}, \end{aligned}$$

so

$$\left| \sin x - \left(x - \frac{x^3}{6} \right) \right| = \left| \frac{\sin(\xi)\, x^4}{24} \right| \leq \frac{x^4}{24},$$

because $|\sin(\xi)| \leq 1$. ⊣

Example Problem 2.8. *Apply Taylor's Theorem for the case $n = 1$.* Solution: *Taylor's Theorem for $n = 1$ states: Given a function, $f(x)$ with a continuous derivative on $[a,b]$, then*

$$f(x) = f(c) + f'(\xi)(x-c)$$

for some ξ between x, c when x, c are in $[a,b]$.
This is the Mean Value Theorem. As a one-liner, the MVT says that at some time during a trip, your velocity is the same as your average velocity for the trip. ⊣

Example Problem 2.9. *Apply Taylor's Theorem to expand* $f(x) = x^3 - 21x^2 + 17$ *around* $c = 1$. Solution: *Simple calculus gives us*

$$\begin{aligned} f^{(0)}(x) &= x^3 - 21x^2 + 17, \\ f^{(1)}(x) &= 3x^2 - 42x, \\ f^{(2)}(x) &= 6x - 42, \\ f^{(3)}(x) &= 6, \\ f^{(k)}(x) &= 0. \end{aligned}$$

with the last holding for $k > 3$. *Evaluating these at* $c = 1$ *gives*

$$f(x) = -3 + -39(x-1) + \frac{-36\,(x-1)^2}{2} + \frac{6\,(x-1)^3}{6}.$$

Note there is no error term, since the higher order derivatives are identically zero. By carrying out simple algebra, you will find that the above expansion is, in fact, the function $f(x)$. ⊣

There is an alternative form of Taylor's Theorem, in this case substituting $x + h$ for x, and x for c in the more general version. This gives

Theorem 2.10 (Taylor's Theorem, Alternative Form)**.** *If* $f(x)$ *has derivatives of order* $0, 1, \ldots, n+1$ *on the closed interval* $[a, b]$, *then for any* x *in this interval and any* h *such that* $x + h$ *is in this interval,*

$$f(x+h) = \sum_{k=0}^{n} \frac{f^{(k)}(x)\ (h)^k}{k!} + \frac{f^{(n+1)}(\xi)\ (h)^{n+1}}{(n+1)!},$$

where ξ *is some number between* x *and* $x + h$.

We generally apply this form of the theorem with $h \to 0$. This leads to a discussion on the matter of *Orders of Convergence*. The following definition will suffice for this class

Definition 2.11. *We say that a function* $f(h)$ *is in the class* $\mathcal{O}\left(h^k\right)$ *(pronounced "big-Oh of* h^k*") if there is some constant* C *such that*

$$|f(h)| \leq C\,|h|^k$$

for all h *"sufficiently small,"* i.e., *smaller than some* h^* *in absolute value.*

For a function $f \in \mathcal{O}\left(h^k\right)$ *we sometimes write* $f = \mathcal{O}\left(h^k\right)$. *We sometimes also write* $\mathcal{O}\left(h^k\right)$, *meaning some function which is a member of this class.*

Roughly speaking, through use of the "Big-O" function we can write an expression without "sweating the small stuff." This can give us an intuitive understanding of how an approximation works, without losing too many of the details.

Example 2.12. *Consider the Taylor expansion of* $\ln x$*:*

$$\ln(x+h) \quad = \quad \ln x + \frac{(1/x)\ h}{1} + \frac{\left(-1/x^2\right)\ h^2}{2} + \frac{\left(2/\xi^3\right)\ h^3}{6}$$

Letting $x = 1$*, we have*

$$\ln(1+h) \quad = \quad h - \frac{h^2}{2} + \frac{1}{3\xi^3}h^3.$$

Using the fact that ξ *is between* 1 *and* $1+h$*, as long as* h *is relatively small (say smaller than* $\frac{1}{2}$*), the term* $\frac{1}{3\xi^3}$ *can be bounded by a constant, and thus*

$$\ln(1+h) \quad = \quad h - \frac{h^2}{2} + \mathcal{O}\left(h^3\right).$$

Thus we say that $h - \frac{h^2}{2}$ *is a* $\mathcal{O}\left(h^3\right)$ *approximation to* $\ln(1+h)$*. For example*

$$\ln(1+0.01) \approx 0.009950331 \approx 0.00995 = 0.01 - \frac{0.01^2}{2}.$$

Exercises

(2.1) Find the binary and hexadecimal expansions of 123.

(2.2) Use the geometric series formula to express the following numbers as fractions:

(a) $2.34\overline{98}$

(b) $101_{\dot{2}}10\overline{011}$

(2.3) Approximate the following series to an accuracy of 0.001 and *justify* the accuracy of your approximation.

(a) $\displaystyle\sum_{n=1}^{\infty} \frac{1}{n^4}$

(b) $\displaystyle\sum_{n=1}^{\infty} \frac{\arctan(n)}{n^2}$

(c) $\displaystyle\sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{2n+1}$

(2.4) Suppose $f \in \mathcal{O}\left(h^k\right)$. Show that $f \in \mathcal{O}\left(h^m\right)$ for any m with $0 < m < k$. (*Hint:* Take $h^* < 1$.) Note this may appear counterintuitive, unless you remember that $\mathcal{O}\left(h^k\right)$ is a *better* approximation than $\mathcal{O}\left(h^m\right)$ when $m < k$.

(2.5) Suppose $f \in \mathcal{O}\left(h^k\right)$, and $g \in \mathcal{O}\left(h^m\right)$. Show that $fg \in \mathcal{O}\left(h^{k+m}\right)$.

(2.6) Suppose $f \in \mathcal{O}\left(h^k\right)$, and $g \in \mathcal{O}\left(h^m\right)$, with $m < k$. Show that $f + g \in \mathcal{O}\left(h^m\right)$.

(2.7) Prove that $f(h) = h^3$ is *not* in $\mathcal{O}\left(h^4\right)$ (*Hint:* Proof by contradiction.)

(2.8) Prove that $\sin(h)$ is in $\mathcal{O}(h)$.

(2.9) Find a $\mathcal{O}\left(h^4\right)$ approximation to $\ln(1+h)$. Compare the approximate value to the actual when $h = 0.1$. How does this approximation compare to the $\mathcal{O}\left(h^3\right)$ approximate from Example 2.12 for $h = 0.1$?

(2.10) Suppose that $f \in \mathcal{O}\left(h^k\right)$. Can you show that $f' \in \mathcal{O}\left(h^{k-1}\right)$?

(2.11) Calculate $\sin(1)$ to within 8 decimal places by using the Taylor's expansion.

(2.12) Calculate $\cos(\pi/2+0.001)$ to within 8 decimal places by using the Taylor's expansion.

Chapter 3

Computer arithmetic

3.1 Strange arithmetic

Computer arithmetic is a bit strange. It's *not* associative, for example:

```
(0.1+0.2)+0.3 == 0.1+(0.2+0.3)
# Out: False
```

Well, that's discouraging! It might help to look at the difference between the two:

```
((0.1+0.2)+0.3) - (0.1+(0.2+0.3))
# Out: 1.1102230246251565e-16
```

This can cause serious issues and create bugs, if you don't know how to work with it. What do you expect the following code to produce?

```
x = 1
while x != 0:
    x = x - 0.1
    print(x)
```

Note: I certainly didn't execute this in a notebook. The terminal would be better, though, here's a safer version.

```
x = 1
cnt = 0
while x != 0 and cnt < 20:
    x = x - 0.1
    cnt = cnt + 1
    print(x)

# Out:
# 0.9
# 0.8
```

```
# 0.7000000000000001
# 0.6000000000000001
# 0.5000000000000001
# 0.40000000000000013
# 0.30000000000000016
# 0.20000000000000015
# 0.10000000000000014
# 1.387778780781457e-16
# -0.09999999999999987
# -0.19999999999999987
# -0.2999999999999999
# -0.3999999999999999
# -0.4999999999999999
# -0.5999999999999999
# -0.6999999999999998
# -0.7999999999999998
# -0.8999999999999998
# -0.9999999999999998
```

Now we can see that `x` just missed zero a little bit. As we will learn, these behaviors are due to *round off error* - one type of error that is crucial to understand in numerical computing.

3.2 Error

Error is a precise, technical term in numerical analysis that is *not* synonymous with mistake. Typically, we are interested the approximation of some quantity, say π, using finitely many digits, say 3.14. The *error* associated with this approximation is simply the difference between the two. Generally, this *must* be nonzero. An important question then is, how big is the absolute value of the error? If you know a more precise decimal expansion of π, then it's easy to see that the error in this little example is less than 0.002. Much of the theoretical portion of numerical analysis involves estimates of error that arises in a computation.

There are two main types of error:

- Truncation error
- Roundoff error

Both arise from the fact that the computer has only a finite memory and can store only finitely many numbers to approximate the infinitely many real numbers. Truncation error arises, more specifically, by approximating an infinite expansion with a finite one. If we write $\pi \approx 3.14$, this is truncation error. More generally, truncation error arises by approximating an infinite sum with a finite sum. Keep in mind that $3.14 = 3 + 1/10 + 4/100$ is an example of a sum, as are

all decimal expansions. Another example is given by

$$e^x \approx \sum_{n=0}^{3} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}.$$

Roundoff error is a little different. Our computer can only represent finitely many numbers. If we perform some computations with those numbers, we generally don't generate another one. Technically, we say that the set of computer numbers is not closed under the arithmetic operations. Thus, we must represent the result of the computation by one of the numbers we have - ideally, the closest one. Even if the original inputs were exactly correct, a computation involving them generally won't be.

Suppose, for example, that the numbers at are disposal are the numbers less than or equal to 10 in absolute value with a precision of 1/10:

$$-10, -9.9, -9.8, \ldots, 9.8, 9.9, 10.$$

Now suppose we want to compute $(1.4 + 5.7)/4$. The exact value is 1.775, but that's not one of our computer numbers. Thus, we represent the result as 1.8. That's roundoff error. Note that if we compute $(1.4 + 5.7)/2$, the exact result is 3.55; should we round to 3.5 or 3.6? There are a number of strategies for breaking the tie but the one that is most relevant here is "round to even". Thus, the computed result would be 3.6.

3.3 Computer numbers

3.3.1 Types of numbers

It's worth understanding how numbers are represented on a computer. Of course, there are a few different types of numbers that can be represented:

- Integers - typically, represented via a dedicated type like `int`.
- Rational or other exact numbers - typically, represented in software.
- Floating point numbers - typically, represented via a dedicated type like `float`.
- Arbitrary precision numbers - somewhat similar to floating point but represented in software to have greater allowable precision.

Numerical analysis is traditionally concerned with floating point arithmetic or, to a lesser extent, arbitrary precision numbers. Rational and other exact numbers lie more in the domain of symbolic computation.

3.3.2 Floating point numbers

Floating point numbers are represented in terms of a base β, a precision p, and an exponent e. If $\beta = 10$ and $p = 2$, then the number 0.034 can be represented as 3.4×10^{-2}. With this representation, its exponent is $e = -2$. With this specification, floating point numbers are not unique, as this same number can be represented as 34×10^{-3} or 0.34×10^{-1}. Thus, we typically take a floating point number to have the specific form

$$\pm (d_0 + d_1\beta^{-1} + \cdots + d_{p-1}\beta^{-(p-1)})\beta^e, \tag{3.1}$$

where each d_i is an integer in $[0, \beta)$ and $d_0 \neq 0$. Such a representation is said to be a normalized floating point number. Here are a few examples where $\beta = 10$, $p = 5$, and e is in the range $-10 \leq e \leq 10$.

- $123450000 = 1.2345 \times 10^8$
- $0.00321 = 3.2100 \times 10^{-3}$
- $1.23 = 1.2300 \times 10^0$

Again, in this system, we have five digits of precision. The number 123450001.2 would be truncated to yield the same number as the first example in our list above.

In these examples, we are using base $\beta = 10$ to ease into the material due to our familiarity with it. Computers work more naturally in binary (or base 2), however, and most are hardwired this way. Thus, in equation 3.1, we have $\beta = 2$ and each digit d_i is zero or one. Thus, here are some easily representable numbers with, again, $p = 5$ and $-5 \leq e \leq 5$:

- $8 = 1.0000 \times 2^3 = 1000_{\dot{2}}0$
- $\frac{1}{4} = 1.0000 \times 2^{-2} = 0_{\dot{2}}01$
- $\frac{5}{16} = 1.0100 \times 2^{-2} = 0_{\dot{2}}01$
- $42 = 1.0101 \times 2^5 = 101010_{\dot{2}}$

Note: I have it on good authority that Douglas Adams used to sign his books 101010.

Now, your computer reserves a specific number of bits to represent a floating point number. The IEEE standard for double precision floating point numbers specifies that 64 bits be used to represent the number. Specifically, it requires

- 1 bit for the sign s.
- 11 bits for the exponent e giving a range of $2^{11} = 2048$ choices for the exponent, which is assumed to be between -1023 and 1024.
- 53 bits for the significant or mantissa $c = 1.d_1d_2 \cdots d_{52}$. Since the leading digit must be a 1, we only need to store 52 digits.

Given those choices for s, e, and c, our number is $(-1)^s \times c \times \beta^e$.

It's interesting to note that numbers that are easily represented in one base are not necessarily representable with finite precision in another. In base 10, for example, $\frac{1}{10} = 0.1$ is exact. However, the geometric series formula shows that

$$\frac{1}{10} = \frac{1}{2}\sum_{n=1}^{\infty} \frac{3}{2^{4n}} = 0_{\dot{2}}0\overline{0011}.$$

In particular, the finite decimal expansion 0.1 has no finite binary expansion.

3.3.3 Distribution of computer numbers

It turns out that normalized floating point numbers are not distributed uniformly. For simplicity, let's explore the binary floating point numbers with precision 3 and maximum exponent size 2. If we fix the exponent to zero, for the moment, there are 8 of these, namely the numbers of the form

$$1 + \frac{d_1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3}$$

where each d_i is zero or one. In decimal, these are

$$1.0,\ 1.125,\ 1.25,\ 1.375,\ 1.5,\ 1.625,\ 1.75,\ \text{and } 1.875$$

To generate the rest of the positive numbers in this system, we take these 8 numbers and multiply by 2^n for $n = -2, -1, 0, 1, 2$. That gives us 40 positive numbers. Of course, we also add on the negative numbers for a total of 80 representable numbers in this system. These are plotted in figure 3.1 with the eight numbers with with exponent zero shown in red. Note that the spacing generally increases as we move away from zero. Note, also, the hole near zero. That's because the smallest, postive, representable number (in this system) is 2^{-2}. The next smallest is $(1 + 1/8) \times 2^{-2}$ and the difference between these is quite a bit smaller than the smallest postive.

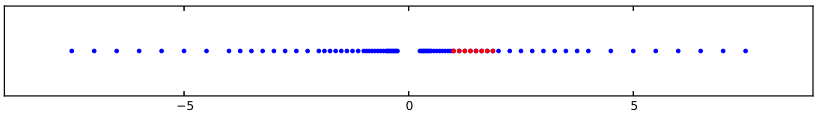


Figure 3.1: The distribution of floating point numbers

The distribution of computer numbers has important consequences when it comes to rounding. Suppose, for example, that we want to estimate the harmonic number

$$\sum_{k=1}^{n} \frac{1}{k}$$

for $n = 1000$. It turns out that the order matters. We can try it with Python:

```
terms = [1/k for k in range(1,1001)]
sum1 = sum(terms)
terms.reverse()
sum2 = sum(terms)
sum1-sum2
# Out: 2.6645352591003757e-15
```

The estimates might appear to be close but their difference is 10 times larger than machine epsilon. Which is better?

When we add two machine numbers a and b, the result $\text{fl}(a+b)$ generally looks like

$$\text{fl}(a+b) = (a+b)(1+\varepsilon).$$

We might think of ε as a psuedo-random number between zero and machine epsilon. The point is, due to the distribution of machine numbers, the potential size of the error is proportional to the actual value of the sum.

Now, if we add three numbers together, we get

$$\text{fl}(\text{fl}(a+b)+c) = ((a+b)(1+\varepsilon)+c)(1+\varepsilon) = (a+b+c)+(2a+2b+c)\varepsilon+O(\varepsilon^2).$$

Note that the numbers that are added first contribute twice to the error. Thus, we have less error, if we add the smaller numbers first. The problem is compounded if we add four numbers:

$$\text{fl}(\text{fl}(\text{fl}(a+b)+c)+d) = (a+b+c+d)+(3a+3b+2c+d)\varepsilon+O(\varepsilon^2).$$

3.3.4 Exploring numbers with Python

Python provides some nice tools to explore numbers and their expansions. `sys.float_info` stores information on how floating point numbers are stored on your machine. Here's how it works on my Mac.

```
import sys
sys.float_info
# Out:
# sys.float_info(
#         max=1.7976931348623157e+308,
#         max_exp=1024, max_10_exp=308,
#         min=2.2250738585072014e-308,
#         min_exp=-1021, min_10_exp=-307,
#         dig=15, mant_dig=53,
#         epsilon=2.220446049250313e-16,
#         radix=2, rounds=1)
```

As Python is built on top of C, the definitive guide to understanding these parameters is the C standard, specifically section 5.2.4.2.2. There we find that a number is represented as

$$x = (-1)^s \beta^e \sum_{k=1}^{p} d_k \beta^{-k}.$$

To get the largest possible value, for example, we take each digit to be $d_k = 1$ and the exponent to be $e = 1024$. We can compute the largest number in Python:

```
s = sum([2**(-k) for k in range(1,54)])
for i in range(1024):
    s = 2*s
s
# Out: 1.7976931348623157e+308
```

This agrees exactly with `sys.float_info.max`.

Another useful tool is `float.hex`, which displays the hexadecimal expansion of a number.

```
float.hex(0.1)
# Out: 0x1.999999999999ap-4
```

This indicates that the internal representation of `0.1` is

$$2^{-4}\left(1+\sum_{k=1}^{12}\frac{9}{16^k}+\frac{11}{16^{13}}\right)\approx 0.10000000000000002.$$

We can use this information to fully understand the discrepancy between $(0.1+0.2)+0.3$ and $0.1+(0.2+0.3)$. Not surprisingly, the hexadecimal representation of 0.2 is `0x1.999999999999ap-4`. The hexadecimal representation of 0.3 is `0x1.3333333333333p-2`, which to 17 digits is 0.29999999999999999 or exactly 0.3 when rounded. The sum of these three numbers yields 0.6000000000000001, which is slightly larger than the desired value of 0.6. By contrast, the computer's computation of $0.2+0.3$ yields *exactly* 0.5, which is not surprising since 0.5 easily representable in binary. Then, $0.1+0.5$ yields exactly the computer representation of 0.6 after rounding.

3.4 Loss of Significance

Often, a loss of significance can be incurred if two nearly equal quantities are subtracted from one another. Thus if I were to direct my computer to subtract 0.177241 from 0.177589, the result would be $.348\times 10^{-3}$, and three significant digits have been lost. This loss is called *subtractive cancellation*, and can often be avoided by rewriting the expression. This will be made clearer by the examples below.

Errors can also occur when quantities of radically different magnitudes are summed. For example $0.1234+5.6789\times 10^{-20}$ might be rounded to 0.1234 by a system that keeps only 16 significant digits. This may lead to unexpected results.

The usual strategies for rewriting subtractive expressions are completing the square, factoring, or using the Taylor expansions, as the following examples illustrate.

Example Problem 3.1. *Consider the stability of* $\sqrt{x+1}-1$ *when* x *is near* 0. *Rewrite the expression to rid it of subtractive cancellation.* Solution: *Suppose that* $x = 1.2345678 \times 10^{-5}$. *Then* $\sqrt{x+1} \approx 1.000006173$. *If your computer (or calculator) can only keep 8 significant digits, this will be rounded to* 1.0000062. *When* 1 *is subtracted, the result is* 6.2×10^{-6}. *Thus* 6 *significant digits have been lost from the original.*

To fix this, we rationalize *the expression*

$$\sqrt{x+1}-1 \quad = \quad \sqrt{x+1}-1\frac{\sqrt{x+1}+1}{\sqrt{x+1}+1} = \frac{x+1-1}{\sqrt{x+1}+1} = \frac{x}{\sqrt{x+1}+1}.$$

This expression has no subtractions, and so is not subject to subtractive cancelling. When $x = 1.2345678 \times 10^{-5}$, *this expression evaluates approximately as*

$$\frac{1.2345678 \times 10^{-5}}{2.0000062} \approx 6.17281995 \times 10^{-6}$$

on a machine with 8 digits, there is no loss of precision. ⊣

Note that nearly all modern computers and calculators store intermediate results of calculations in higher precision formats. This minimizes, but does not eliminate, problems like those of the previous example problem.

Example Problem 3.2. *Write stable code to find the roots of the equation* $x^2 + bx + c = 0$. Solution: *The usual quadratic formula is*

$$x_\pm = \frac{-b \pm \sqrt{b^2-4c}}{2}$$

Supposing that $b \gg c > 0$, *the expression in the square root might be rounded to* b^2, *giving two roots* $x_+ = 0$, $x_- = -b$. *The latter root is nearly correct, while the former has no correct digits. To correct this problem, multiply the numerator and denominator of* x_+ *by* $-b-\sqrt{b^2-4c}$ *to get*

$$x_+ = \frac{2c}{-b-\sqrt{b^2-4c}}$$

Now if $b \gg c > 0$, *this expression gives root* $x_+ = -c/b$, *which is nearly correct. This leads to the pair:*

$$x_- = \frac{-b-\sqrt{b^2-4c}}{2}, \quad x_+ = \frac{2c}{-b-\sqrt{b^2-4c}}$$

Note that the two roots are nearly reciprocals, and if x_- *is computed,* x_+ *can easily be computed with little additional work.* ⊣

Example Problem 3.3. *Rewrite* $e^x - \cos x$ *to be stable when* x *is near* 0. Solution: *Look at the Taylor's Series expansion for these functions:*

$$\begin{aligned} e^x - \cos x &= \left[1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots\right] - \left[1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots\right] \\ &= x + x^2 + \frac{x^3}{3!} + \mathcal{O}\left(x^5\right) \end{aligned}$$

This expression has no subtractions, and so is not subject to subtractive cancelling. Note that we propose calculating $x + x^2 + x^3/6$ *as an* approximation *of* $e^x - \cos x$*, which we cannot calculate exactly anyway. Since we assume* x *is nearly zero, the approximate should be good. If* x *is* very *close to zero, we may only have to take the first one or two terms. If* x *is not so close to zero, we may need to take all three terms, or even more terms of the expansion; if* x *is far from zero we should use some other technique.* $\dashv$

EXERCISES

(3.1) Suppose $f \in \mathcal{O}\left(h^k\right)$. Show that $f \in \mathcal{O}\left(h^m\right)$ for any m with $0 < m < k$. (*Hint:* Take $h^* < 1$.) Note this may appear counterintuitive, unless you remember that $\mathcal{O}\left(h^k\right)$ is a *better* approximation than $\mathcal{O}\left(h^m\right)$ when $m < k$.

(3.2) Suppose $f \in \mathcal{O}\left(h^k\right)$, and $g \in \mathcal{O}\left(h^m\right)$. Show that $fg \in \mathcal{O}\left(h^{k+m}\right)$.

(3.3) Suppose $f \in \mathcal{O}\left(h^k\right)$, and $g \in \mathcal{O}\left(h^m\right)$, with $m < k$. Show that $f + g \in \mathcal{O}\left(h^m\right)$.

(3.4) Prove that $f(h) = -3h^5$ is in $\mathcal{O}\left(h^5\right)$.

(3.5) Prove that $f(h) = h^2 + 5h^{17}$ is in $\mathcal{O}\left(h^2\right)$.

(3.6) Prove that $f(h) = h^3$ is *not* in $\mathcal{O}\left(h^4\right)$ (*Hint:* Proof by contradiction.)

(3.7) Prove that $\sin(h)$ is in $\mathcal{O}\left(h\right)$.

(3.8) Find a $\mathcal{O}\left(h^3\right)$ approximation to $\sin h$.

(3.9) Find a $\mathcal{O}\left(h^4\right)$ approximation to $\ln(1+h)$. Compare the approximate value to the actual when $h = 0.1$. How does this approximation compare to the $\mathcal{O}\left(h^3\right)$ approximate from Example 2.12 for $h = 0.1$?

(3.10) Suppose that $f \in \mathcal{O}\left(h^k\right)$. Can you show that $f' \in \mathcal{O}\left(h^{k-1}\right)$?

(3.11) Rewrite $\sqrt{x+1} - \sqrt{1}$ to get rid of subtractive cancellation when $x \approx 0$.

(3.12) Rewrite $\sqrt{x+1} - \sqrt{x}$ to get rid of subtractive cancellation when x is very large.

(3.13) Use a Taylor's expansion to rid the expression $1 - \cos x$ of subtractive cancellation for x small. Use a $\mathcal{O}\left(x^5\right)$ approximate.

(3.14) Use a Taylor's expansion to rid the expression $1 - \cos^2 x$ of subtractive cancellation for x small. Use a $\mathcal{O}\left(x^6\right)$ approximate.

(3.15) Calculate $\cos(\pi/2+0.001)$ to within 8 decimal places by using the Taylor's expansion.

(3.16) Consider the error in $s_1 = (5 + 10) + 15$ versus $s_2 = 5 + (10 + 15)$ on a (not so good) machine with machine epsilon ε. Show that the error in s_1 is of the order 45ε while the error in s_2 is of the order 55ε.

Chapter 4

Finding Roots

We turn now to the problem of solving univariate equations. Here are a couple of examples to keep in mind:

- $x^5 - x - 1 = 0$
- $\cos(x) = x$

Neither of these problems have simple, closed form solutions. It's quite easy to see from a graph, however, that they each have a unique solution.

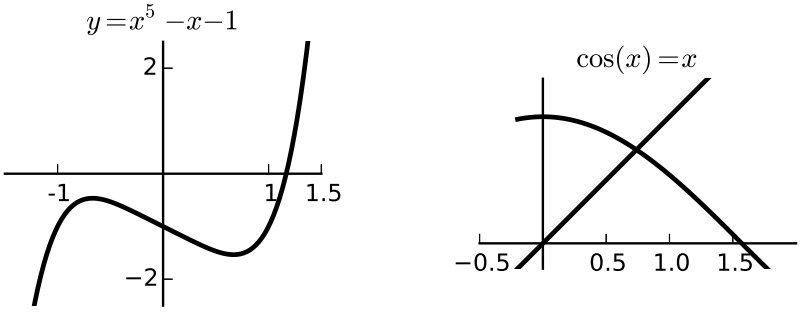


Figure 4.1: Two equations with clear solutions

Both of these are easily solved with the SciPy function `brentq`. For the first,

```
from scipy.optimize import brentq
def f(x): return x**5 - x - 1
brentq(f,0,2)
# Out: 1.1673039782614187
```

Note that `brentq` expects a function for its first argument and two numbers defining an interval containing a root of the function. Of course, any equation

LHS = RHS can be written in the form LHS − RHS = 0. Thus, we can also solve the second example.

```
import numpy as np
def f(x): return np.cos(x)-x
brentq(f,0,1)
# Out: 0.7390851332151559
```

In this chapter, we'll explore how such functions work.

4.1 Bisection

The simplest root finding method is that of bisection. The following theorem insures the success of the method

Theorem 4.1 (Intermediate Value Theorem)**.** *If $f(x)$ is continuous on $[a, b]$ then for any y such that y is between $f(a)$ and $f(b)$ there is a $c \in [a, b]$ such that $f(c) = y$.*

The IVT is best illustrated graphically. Note that continuity is really a requirement here–a single point of discontinuity could ruin your whole day, as the following example illustrates.

Example 4.2. *The function $f(x) = \frac{1}{x}$ is not continuous at 0. Thus if $0 \in [a, b]$, we* cannot *apply the IVT. In particular, if $0 \in [a, b]$ it happens to be the case that for* every *y between $f(a), f(b)$ there is no $c \in [a, b]$ such that $f(c) = y$.*

In particular, the IVT tells us that if $f(x)$ is continuous and we know a, b such that $f(a), f(b)$ have different sign, then there is some root in $[a, b]$. A decent estimate of the root is $c = \frac{a+b}{2}$. We can check whether $f(c) = 0$. If this does not hold then one and only one of the two following options holds:

1. $f(a), f(c)$ have different signs.
2. $f(c), f(b)$ have different signs.

We now choose to recursively apply bisection to either $[a, c]$ or $[c, b]$, respectively, depending on which of these two options hold.

4.1.1 Modifications

Unfortunately, it is impossible for a computer to test whether a given black box function is continuous. Thus malicious or incompetent users could cause a naïvely implemented bisection algorithm to fail. There are a number of easily conceivable problems:

1. The user might give f, a, b such that $f(a), f(b)$ have the same sign. In this case the function f might be legitimately continuous, and might have a root in the interval $[a, b]$. If, taking $c = \frac{a+b}{2}$, $f(a), f(b), f(c)$ all have the same sign, the algorithm would be at an impasse. We should perform a "sanity check" on the input to make sure $f(a), f(b)$ have different signs.

2. The user might give f, a, b such that f is not continuous on $[a, b]$, moreover has no root in the interval $[a, b]$. For a poorly implemented algorithm, this might lead to an infinite search on smaller and smaller intervals about some discontinuity of f. In fact, the algorithm might descend to intervals as small as machine precision, in which case the midpoint of the interval will, due to rounding, be the same as one of the endpoints, resulting in an infinite recursion.
3. The user might give f such that f has no root c that is representable in the computer's memory. Recall that we think of computers as storing numbers in the form $\pm r \times 10^k$; given a finite number of bits to represent a number, only a finite number of such numbers can be represented. It may legitimately be the case that none of them is a root to f. In this case, the behaviour of the algorithm may be like that of the previous case. A well implemented version of bisection should check the length of its input interval, and give up if the length is too small, compared to machine precision.

Another common error occurs in the testing of the signs of $f(a), f(b)$. A slick programmer might try to implement this test in the pseudocode:

```
if (f(a)f(b) > 0) then ...
```

Note however, that $|f(a)|, |f(b)|$ might be very small, and that $f(a)f(b)$ might be too small to be representable in the computer; this calculation would be rounded to zero, and unpredictable behaviour would ensue. A wiser choice is

```
if (sign(f(a)) * sign(f(b)) > 0) then ...
```

where the function $\text{sign}(x)$ returns $-1, 0, 1$ depending on whether x is negative, zero, or positive, respectively.

4.1.2 Convergence

We can see that each time `recursive_bisection`$(f, a, b, \ldots)$ is called that $|b - a|$ is half the length of the interval in the previous call. Formally call the first interval $[a_0, b_0]$, and the first midpoint c_0. Let the second interval be $[a_1, b_1]$, etc. Note that one of a_1, b_1 will be c_0, and the other will be either a_0 or b_0. We are claiming that

$$\begin{aligned} b_n - a_n &= \frac{b_{n-1} - a_{n-1}}{2} \\ &= \frac{b_0 - a_0}{2^n} \end{aligned}$$

Theorem 4.3 (Bisection Method Theorem)**.** *If $f(x)$ is a continuous function on $[a, b]$ such that $f(a)f(b) < 0$, then after n steps, the algorithm* `run_bisection` *will return c such that $|c - c'| \leq \frac{|b-a|}{2^n}$, where c' is some root of f.*

4.1.3 Implementation

Python code implementing the bisection method is shown in code listing 4.1.

Listing 4.1: The bisection method

```
import numpy as np
class SignError(Exception):
    def __str__(self):
        return 'function has same signs at the endpoints'
class IntervalError(Exception):
    def __str__(self):
        return 'need a<b'
def bisection(f,a,b,tol=10**(-15)):
    if b <= a:
        raise IntervalError
    y1 = f(a)
    if y1 == 0:
        return a
    y2 = f(b)
    if y2 == 0:
        return b
    if np.sign(y1)*np.sign(y2) > 0:
        raise SignError
    cnt = 0
    while abs(a-b)>tol and cnt < 100:
        c = (a+b)/2
        y3 = f(c)
        if y3 == 0:
            return c
        if np.sign(y1)*np.sign(y3) < 0:
            b = c
            y2 = y3
        elif np.sign(y2)*np.sign(y3) < 0:
            a = c
            y1 = y3
        cnt = cnt+1
    return a

# Usage:
def f(x): return x**5-x-1
bisection(f,0,2)
# Out: 1.167303978261418
```

4.2 Functional iteration

Back in section 1.2 we illustrated the fact that the equation $\cos(x) = x$ can be solved by simply iterating the cosine. We explore this more deeply in this section.

Definition 4.4 (Fixed point)**.** *Suppose $f : S \to \mathbb{R}$, where S is a subset of $\mathbb{R}$. If $f(x_0) = x_0$, then x_0 is called a **fixed point** of f.*

Using this language, the solution of $\cos(x) = x$ is simply a fixed point of the cosine and we can find this fixed point by simply iterating the cosine. As described in section 1.2, iteration is a fairly simple idea. Given a function f, we start with a real number (or initial seed) x_1 and define a sequence (x_n) recursively by $x_{n+1} = f(x_n)$. It turns out that we might expect this process to lead to a fixed point, if we iterate a particular type of function.

Definition 4.5 (Contraction)**.** *Suppose $f : \mathbb{S} \to \mathbb{R}$, where S is a subset of $\mathbb{R}$. If there is a number r such that $0 < r < 1$ and*

$$|f(x) - f(y)| \leq r|x - y|$$

*for all $x, y \in S$, then f is called a **contraction** on S and the number r is called a contraction ratio.*

Now suppose that f is a contraction on an interval I with contraction ratio r and that f has a fixed point $x_0 \in I$. Then, it's immediately apparent that

$$|f(x) - x_0| \leq r|x - x_0|$$

for all $x \in I$. But then

$$|f^2(x) - x_0| \leq r^2|x - x_0|,$$

etc. As a result, iteration of f from an arbitrary starting point in I will generate a sequence that tends geometrically to x_0.

But, when might we expect a function to be a contraction? One possible answer is provided by the mean value theorem. This was stated as a special case of Taylor's theorem back in example 2.8, but it is quite important in its own right.

Theorem 4.6 (Mean Value Theorem)**.** *Suppose that a function is continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b). Then, there is a number ξ in (a, b) such that*

$$f(b) - f(a) = f'(\xi)(b - a).$$

An immediate corollary is that, if f satisfies $|f'(x)| \leq r < 1$ for all x in (a, b), then f is a contraction on (a, b) with contraction ratio r. Furthermore, if f is continuously differentiable (meaning that f' exists and is continuous) with $|f'(x_0)| < r < 1$, then there will be an interval I about x_0 with $|f'(x_0)| \leq r$ on that interval. We thus have a wonderfully simple criterion to test when we might expect functional iteration to work - we need $|f'(x_0)| < 1$. This all leads to the following terminology:

Definition 4.7 (Attraction)**.** *Suppose that f is continuously differentiable on an open interval and that x_0 is a fixed point of f. We say that the fixed point is*

- ***Attractive*** *if* $|f'(x_0)| < 1$
- ***Super-attractive*** *if* $f'(x_0) = 0$
- ***Repulsive*** *if* $|f'(x_0)| > 1$
- ***Neutral*** *if* $|f'(x_0)| = 1$

In fairness, it should be stated that it can be quite hard to find a good starting point so that iteration will converge to a particular root. A deep investigation of that general question leads us to chaos theory. In practice, we typically simply use a graph to find a reasonable initial seed.

Example Problem 4.8. *Use functional iteration to find the unique solution of* $\sin(x/2) = x\cos(x/2)$ *in the interval* $0 < x < \pi$.

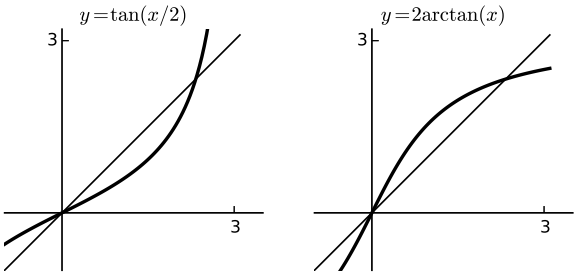


Figure 4.2: Graphs of function related to $\sin(x/2) = x\cos(x/2)$

Solution: The basic idea is to write the equation in the form $f(x) = x$ for some choice of f. We can then iterate f. One potential rewrite for this equation is to express it as $\tan(x/2) = x$. However, if we look at the graph of $y = \tan(x/2)$ shown in figure 4.2, we see that it crosses the line $y = x$ with a slope larger than 1. Another approach is to write it as $2\arctan(x) = x$; thus we can iterate $f(x) = 2\arctan(x)$. A look at the other graph in figure 4.2 indicates that this should probably work, so let's try.

```
import numpy as np
def f(x): return 2*np.arctan(x)
```

```
tol = 10**(-16)
cnt = 0
x1 = 2
x2 = f(x1)
while abs(x1-x2) > tol and cnt < 100:
    x1 = x2
    x2 = f(x2)
    cnt = cnt+1
x2
# Out: 2.3311223704144224
```

⊣

4.3 Newton's Method

Newton's method is an iterative technique to find a root of a function. Under certain mild assumptions on f with a root x_0, we can find a new function $N(x)$ with the property that x_0 is a super-attractive fixed point of N. As a result, we'll be able to find the root of f by iterating N.

The derivation of Newton's method is based on linearization. Assuming that x_k is close to a root of f, then we can generate new point x_{k+1} that is (hopefully) closer to the root by finding a root of the linear approximation to f at x_k:

$$L(x) = f(x_k) + f'(x_k)(x - x_k).$$

Setting $L(x) = 0$, solving for x and calling are new x_{k+1} this new solution, we find that

$$\boxed{x_{k+1} = x_k - f(x_k)/f'(x_k).} \tag{4.1}$$

Figure 4.3 illustrates this procedure and shows why we might expect for x_{k+1} to be closer to x_k.

4.3.1 Connection with functional iteration

As it turns out, Newton's method typically converges *very* quickly, though the proper choice of an initial seed can be tricky. Our understanding of functional iteration can shed light on this. From this perspective, Newton's method boils down to iteration of the function $N(x)$ defined by

$$N(x) = x - f(x)/f'(x).$$

Let us assume now that x_0 is a root of f. The fact that Newton's method might converge rapidly to x_0 boils down to two observations:

1. x_0 is a fixed point of N:
   This is a simple computation:

   $$N(x_0) = x_0 - f(x_0)/f'(x_0) = x_0 - 0 = x_0,$$

   since x_0 is a root of f.

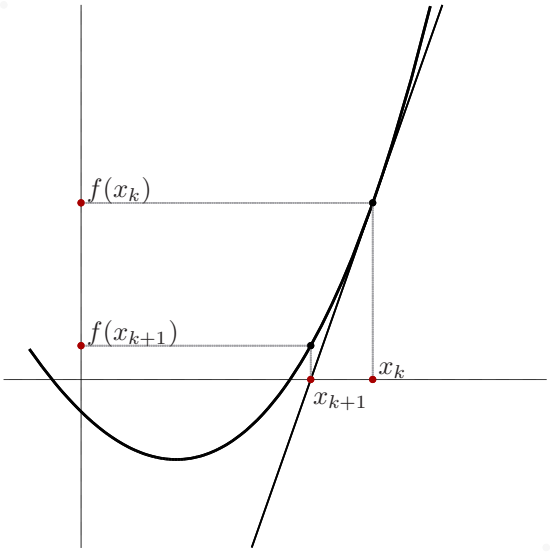


Figure 4.3: One iteration of Newton's method is shown for a quadratic function $f(x)$. The linearization of $f(x)$ at x_k is shown. It is clear that x_{k+1} is a root of the linearization. It happens to be the case that $|f(x_{k+1})|$ is smaller than $|f(x_k)|$, *i.e.*, x_{k+1} is a better guess than x_k.

2. If $f'(x_0) \neq 0$, then x_0 is super-attractive under iteration of N.
   Another computation:

$$\begin{aligned} N'(x_0) &= 1 - \frac{f'(x_0)^2 - f''(x_0)f(x_0)}{f'(x_0)^2} \\ &= 1 - \frac{f'(x_0)^2}{f'(x_0)^2} = 0. \end{aligned}$$

4.3.2 Implementation

Use of Newton's method requires that the function $f(x)$ be differentiable. Moreover, the derivative of the function must be known. This may preclude Newton's method from being used when $f(x)$ is a black box. As is the case for the bisection method, our algorithm cannot explicitly check for continuity of $f(x)$. Moreover, the success of Newton's method is dependent on the initial guess x_0. This was also the case with bisection, but for bisection there was an easy test of the initial interval–*i.e.*, test if $f(a_0)f(b_0) < 0$.

Our algorithm will test for goodness of the estimate by looking at $|f(x_k)|$. The algorithm will also test for near-zero derivative. Note that if it were the case that $f'(x_k) = 0$ then h would be ill defined.

Algorithm 1: Algorithm for finding root by Newton's Method.
Input: a function, its derivative, an initial guess, an iteration limit, and a tolerance
Output: a point for which the function has small value.
RUN_NEWTON$(f, f', x0, N, tol)$
(1) Let $x \leftarrow x0, n \leftarrow 0$.
(2) **while** $n \leq N$
(3) Let $fx \leftarrow f(x)$.
(4) **if** $|fx| < tol$
(5) **return** x.
(6) Let $fpx \leftarrow f'(x)$.
(7) **if** $|fpx| < tol$
(8) Warn "$f'(x)$ is small; giving up."
(9) **return** x.
(10) Let $x \leftarrow x - fx/fpx$.
(11) Let $n \leftarrow n + 1$.

4.3.3 Problems

As mentioned above, convergence is dependent on $f(x)$, and the initial estimate x_0. A number of conceivable problems might come up. We illustrate them here.

Example 4.9. *Consider Newton's method applied to the function $f(x) = x^j$ with $j > 1$, and with initial estimate $x_0 \neq 0$.*
Note that $f(x) = 0$ has the single root $x = 0$. Now note that

$$x_{k+1} = x_k - \frac{x_k^j}{j x_k^{j-1}} = \left(1 - \frac{1}{j}\right) x_k.$$

Since the equation has the single root $x = 0$, we find that x_k is converging to the root. However, it is converging at a rate slower than we expect from Newton's method: at each step we have a constant decrease of $1 - (1/j)$, which is a larger number (and thus worse decrease) when j is larger.

Example 4.10. *Consider Newton's method applied to the function $f(x) = \frac{\ln x}{x}$, with initial estimate $x_0 = 3$.*
Note that $f(x)$ is continuous on $\mathbb{R}^+$. It has a single root at $x = 1$. Our initial guess is not too far from this root. However, consider the derivative:

$$f'(x) = \frac{x\frac{1}{x} - \ln x}{x^2} = \frac{1 - \ln x}{x^2}$$

If $x > e^1$, then $1 - \ln x < 0$, and so $f'(x) < 0$. However, for $x > 1$, we know $f(x) > 0$. Thus taking

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} > x_k.$$

The estimates will "run away" from the root $x = 1$.

Example 4.11. *Consider Newton's method applied to the function* $f(x) = \sin(x)$ *for the initial estimate* $x_0 \neq 0$*, where* x_0 *has the odious property* $2x_0 = \tan x_0$.
You should verify that there are an infinite number of such x_0*. Consider the identity of* x_1 :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 - \frac{\sin(x_0)}{\cos(x_0)} = x_0 - \tan x_0 = x_0 - 2x_0 = -x_0.$$

Now consider x_2 :

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = -x_0 - \frac{\sin(-x_0)}{\cos(-x_0)} - x_0 + \frac{\sin(x_0)}{\cos(x_0)} = -x_0 + \tan x_0 = -x_0 + 2x_0 = x_0.$$

Thus Newton's method "cycles" between the two values $x_0, -x_0$.

Of course, Newton's method may find some iterate x_k for which $f'(x_k) = 0$, in which case, there is no well-defined x_{k+1}.

4.3.4 Convergence

When Newton's Method converges, it actually displays *quadratic convergence*. That is, if $e_k = x_k - r$, where r is the root that the x_k are converging to, that $|e_{k+1}| \leq C |e_k|^2$. If, for example, it were the case that $C = 1$, then we would double the accuracy of our root estimate with each iterate. That is, if e_0 were 0.001, we would expect e_1 to be on the order of 0.000001. The following theorem formalizes our claim:

Theorem 4.12 (Newton's Method Convergence)**.** *If* $f(x)$ *has two continuous derivatives, and* r *is a simple root of* $f(x)$*, then there is some* D *such that if* $|x_0 - r| < D$*, Newton's method will converge quadratically to* r.

The proof is based on arguments from real analysis, and is omitted; see Cheney & Kincaid for the proof [**?**]. Take note of the following, though:

1. The proof requires that r be a simple root, that is that $f'(r) \neq 0$. When this does not hold we may get only linear convergence, as in Example 4.9.
2. The key to the proof is using Taylor's theorem, and the definition of Newton's method to show that

$$e_{k+1} = \frac{-f''(\xi_k) e_k^2}{2f'(x_k)},$$

where ξ_k is between x_k and $r = x_k + e_k$.
The proof then proceeds by showing that there is some region about r such that
   (a) in this region $|f''(x)|$ is not too large and $|f'(x)|$ is not too small, and
   (b) if x_k is in the region, then x_{k+1} is in the region.

In particular the region is chosen to be so small that if x_k is in the region, then the factor e_k^2 will outweigh the factor $|f''(\xi_k)| / 2 |f'(x_k)|$. You can roughly think of this region as an "attractor" region for the root.

3. The theorem never guarantees that some x_k will fall into the "attractor" region of a root r of the function, as in Example 4.10 and Example 4.11. The theorem that follows gives sufficient conditions for convergence to a particular root.

Theorem 4.13 (Newton's Method Convergence II [**?**])**.** *If $f(x)$ has two continuous derivatives on $[a, b]$, and*

1. *$f(a)f(b) < 0$,*
2. *$f'(x) \neq 0$ on $[a, b]$,*
3. *$f''(x)$ does not change sign on $[a, b]$,*
4. *Both $|f(a)| \leq (b-a)\,|f'(a)|$ and $|f(b)| \leq (b-a)\,|f'(b)|$ hold,*

then Newton's method converges to the unique root of $f(x) = 0$ for any choice of $x_0 \in [a, b]$.

4.3.5 Using Newton's Method

Newton's Method can be used to program more complex functions using only simple functions. Suppose you had a computer which could perform addition, subtraction, multiplication, division, and storing and retrieving numbers, and it was your task to write a subroutine to compute some complex function $g(\cdot)$. One way of solving this problem is to have your subroutine use Newton's Method to solve some equation equivalent to $g(z) - x = 0$, where z is the input to the subroutine. Note that it is assumed the subroutine cannot evaluate $g(z)$ directly, so this equation needs to be modified to satisfy the computer's constraints.

Quite often when dealing with problems of this type, students make the mistake of using Newton's Method to try to solve a linear equation. This should be an indication that a mistake was made, since Newton's Method can solve a linear equation in a single step:

Example 4.14. *Consider Newton's method applied to the function $f(x) = ax + b$. The iteration is given as*

$$x_{k+1} \leftarrow x_k - \frac{ax_k + b}{a}.$$

This can be rewritten simply as $x_{k+1} \leftarrow -b/a$.

The following example problems should illustrate this process of "bootstrapping" via Newton's Method.

Example Problem 4.15. *Devise a subroutine using only subtraction and multiplication that can find the multiplicative inverse of an input number z,* i.e.*, can find $(1/z)$.* Solution: *We are tempted to use the linear function $f(x) = (1/z) - x$. But this is a linear equation for which Newton's Method would reduce to $x_{k+1} \leftarrow (1/z)$. Since the subroutine can only use subtraction and multiplication, this will not work.*

Instead apply Newton's Method to $f(x) = z - (1/x)$. The Newton step is

$$x_{k+1} \leftarrow x_k - \frac{z - (1/x_k)}{(1/x_k^2)} = x_k - zx_k^2 + x_k = x_k\,(2 - zx_k)\,.$$

Note this step uses only multiplication and subtraction. The subroutine is given in Algorithm 2. ⊣

Algorithm 2: Algorithm for finding a multiplicative inverse using simple operations.

Input: a number
Output: its multiplicative inverse
INVS(z)
(1) if $z = 0$ throw an error.
(2) Let $x \leftarrow \text{sign}(z), n \leftarrow 0$.
(3) **while** $n \leq 50$
(4) Let $x \leftarrow x(2 - zx)$.
(5) Let $n \leftarrow n + 1$.

Example Problem 4.16. *Devise a subroutine using only simple operations which computes the square root of an input number z.* Solution: *The temptation is to find a zero for $f(x) = \sqrt{z} - x$. However, this equation is linear in x. Instead let $f(x) = z - x^2$. You can easily see that if x is a positive root of $f(x) = 0$, then $x = \sqrt{z}$. The Newton step becomes*

$$x_{k+1} \leftarrow x_k - \frac{z - x_k^2}{-2x_k}.$$

after some simplification this becomes

$$x_{k+1} \leftarrow \frac{x_k}{2} + \frac{z}{2x_k}.$$

Note this relies only on addition, multiplication and division.
The final subroutine is given in Algorithm 3.

Algorithm 3: Algorithm for finding a square root using simple operations.

Input: a number
Output: its square root
SQRT(z)
(1) if $z < 0$ throw an error.
(2) Let $x \leftarrow 1, n \leftarrow 0$.
(3) **while** $n \leq 50$
(4) Let $x \leftarrow (x + z/x)/2$.
(5) Let $n \leftarrow n + 1$.

⊣

4.4 Secant Method

The secant method for root finding is roughly based on Newton's method; however, it is not assumed that the derivative of the function is known, rather the

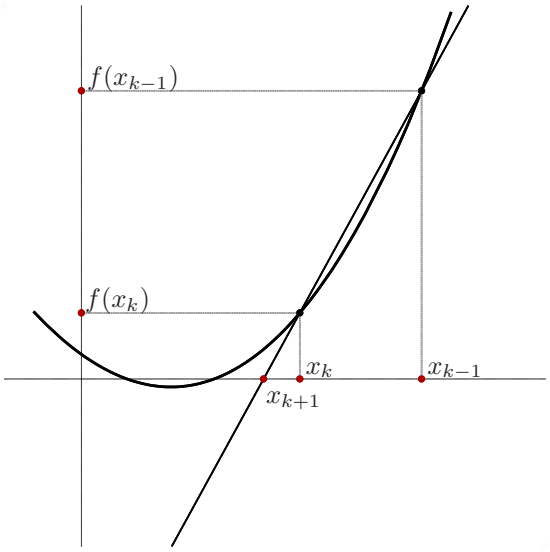


Figure 4.4: One iteration of the Secant method is shown for some quadratic function $f(x)$. The secant line through $(x_{k-1}, f(x_{k-1}))$ and $(x_k, f(x_k))$ is shown. It happens to be the case that $|f(x_{k+1})|$ is smaller than $|f(x_k)|$, *i.e.*, x_{k+1} is a better guess than x_k.

derivative is approximated by the value of the function at some of the iterates, x_k. More specifically, the slope of the *tangent* line at $(x_k, f(x_k))$, which is $f'(x_k)$ is approximated by the slope of the *secant* line passing through $(x_{k-1}, f(x_{k-1}))$ and $(x_k, f(x_k))$, which is

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Thus the iterate x_{k+1} is the root of this secant line. That is, it is a root to the equation

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} (x - x_k) = y - f(x_k).$$

Since the root has a y value of 0, we have

$$\begin{aligned}
\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} (x_{k+1} - x_k) &= -f(x_k), \\
x_{k+1} - x_k &= -\left(\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \right) f(x_k), \\
x_{k+1} &= x_k - \left(\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \right) f(x_k). \qquad (4.2)
\end{aligned}$$

You will note this is the recurrence of Newton's method, but with the slope $f'(x_k)$ substituted by the slope of the secant line. Note also that the secant

method requires two initial guesses, x_0, x_1, but can be used on a black box function. The secant method can suffer from some of the same problems that Newton's method does, as we will see.

An iteration of the secant method is shown in Figure 4.4, along with the secant line.

Example 4.17. *Consider the secant method used on* $x^3 + x^2 - x - 1$, *with* $x_0 = 2, x_1 = \frac{1}{2}$.
Note that this function is continuous and has roots ± 1. *We give the iterates here:*

k	x_k	$f(x_k)$
0	2	9
1	0.5	-1.125
2	0.666666666666667	-0.925925925925926
3	1.44186046511628	2.63467367653163
4	0.868254072087394	-0.459842466254495
5	0.953491494113659	-0.177482458876898
6	1.00706900811804	0.0284762692197613
7	0.999661272951803	-0.0013544492875992
8	0.999997617569723	$-9.52969840528617 \times 10^{-06}$
9	1.0000000008072	$3.22880033820638 \times 10^{-09}$
10	0.999999999999998	$-7.43849426498855 \times 10^{-15}$

4.4.1 Problems

As with Newton's method, convergence is dependent on $f(x)$, and the initial estimates x_0, x_1. We illustrate a few possible problems:

Example 4.18. *Consider the secant method for the function* $f(x) = \frac{\ln x}{x}$, *with* $x_0 = 3, x_1 = 4$.
As with Newton's method, the iterates diverge towards infinity, looking for a

nonexistant root. We give some iterates here:

k	x_k	$f(x_k)$
0	3	0.366204096222703
1	4	0.346573590279973
2	21.6548475770851	0.142011128224341
3	33.9111765137635	0.103911011441661
4	67.3380435135758	0.0625163004418104
5	117.820919458675	0.0404780904944712
6	210.543986613847	0.0254089165873003
7	366.889164762149	0.0160949419231219
8	637.060241341843	0.010135406045582
9	1096.54125113444	0.00638363233543847
10	1878.34688714646	0.00401318169994875
11	3201.94672271613	0.0025208146648422
12	5437.69020766155	0.00158175793894727
13	9203.60222260594	0.000991714984152597
14	15533.1606791089	0.000621298692241343
15	26149.7196085218	0.000388975250950428
16	43924.8466075548	0.000243375589137882
17	73636.673898472	0.000152191807070607

Example 4.19. *Consider Newton's method applied to the function* $f(x) = \frac{1}{1+x^2} - \frac{1}{17}$*, with initial estimates* $x_0 = -1, x_1 = 1$.
You can easily verify that $f(x_0) = f(x_1)$*, and thus the secant line is horizontal. And thus* x_2 *is not defined.*

Algorithm 4: Algorithm for finding root by secant method.

Input: a function, initial guesses, an iteration limit, and a tolerance
Output: a point for which the function has small value.

RUN_SECANT$(f, x0, x1, N, tol)$
(1) Let $x \leftarrow x1, xp \leftarrow x0, fh \leftarrow f(x1), fp \leftarrow f(x0), n \leftarrow 0$.
(2) **while** $n \leq N$
(3) **if** $|fh| < tol$
(4) **return** x.
(5) Let $fpx \leftarrow (fh - fp)/(x - xp)$.
(6) **if** $|fpx| < tol$
(7) Warn "secant slope is too small; giving up."
(8) **return** x.
(9) Let $xp \leftarrow x, fp \leftarrow fh$.
(10) Let $x \leftarrow x - fh/fpx$.
(11) Let $fh \leftarrow f(x)$.
(12) Let $n \leftarrow n + 1$.

4.4.2 Convergence

We consider convergence analysis as in Newton's method. We assume that r is a root of $f(x)$, and let $e_n = r - x_n$. Because the secant method involves two iterates, we assume that we will find some relation between e_{k+1} and the previous two errors, e_k, e_{k-1}.

Indeed, this is the case. Omitting all the nasty details (see Cheney & Kincaid [**?**]), we arrive at the imprecise equation:

$$e_{k+1} \approx -\frac{f''(r)}{2f'(r)} e_k e_{k-1} = C e_k e_{k-1}.$$

Again, the proof relies on finding some "attractor" region and using continuity.

We now postulate that the error terms for the secant method follow some power law of the following type:

$$|e_{k+1}| \sim A\,|e_k|^{\alpha}.$$

Recall that this held true for Newton's method, with $\alpha = 2$. We try to find the α for the secant method. Note that

$$|e_k| = A\,|e_{k-1}|^{\alpha},$$

so

$$|e_{k-1}| = A^{-\frac{1}{\alpha}}\,|e_k|^{\frac{1}{\alpha}}.$$

Then we have

$$A\,|e_k|^{\alpha} = |e_{k+1}| = C\,|e_k|\,|e_{k-1}| = CA^{-\frac{1}{\alpha}}\,|e_k|^{1+\frac{1}{\alpha}},$$

Since this equation is to hold for all e_k, we must have

$$\alpha = 1 + \frac{1}{\alpha}.$$

This is solved by $\alpha = \frac{1}{2}\left(1+\sqrt{5}\right) \approx 1.62$. Thus we say that the secant method enjoys *superlinear* convergence; This is somewhere between the convergence rates for bisection and Newton's method.

EXERCISES

(4.1) Consider the bisection method applied to find the zero of the function $f(x) = x^2 - 5x + 3$, with $a_0 = 0, b_0 = 1$. What are a_1, b_1? What are a_2, b_2 ?

(4.2) Approximate $\sqrt{10}$ by using two steps of Newton's method, with an initial estimate of $x_0 = 3$. (*cf.* Example Problem 4.16) Your answer should be correct to 5 decimal places.

(4.3) Consider bisection for finding the root to $\cos x = 0$. Let the initial interval I_0 be $[0, 2]$. What is the next interval considered, call it I_1? What is I_2? I_6?

(4.4) Use functional iteration to find the unique postive root of $f(x) = \sin(x) - x/2$.

(4.5) In example 4.8 we solved $\sin(x/2) = x\cos(x/2)$ by iterating $f(x) = 2\arctan(x)$. Determine the set of all seeds that converge to the desired solution. How might you prove that this set of seeds works?
Show that $x^2 = 3$ is algebraically equivalent to $x = (x + 3/x)/2$ and use this to approximate $\sqrt{3}$ via functional iteration. What is the rate of convergence?

(4.6) What does the sequence defined by

$$x_0 = 1, \qquad x_{k+1} = \frac{1}{2}x_k + \frac{1}{x_k}$$

converge to?

(4.7) Devise a subroutine using only simple operations that finds, via Newton's Method, the cubed root of some input number z.

(4.8) Use Newton's Method to approximate $\sqrt[3]{9}$. Start with $x_0 = 2$. Find x_2.

(4.9) Use Newton's Method to devise a sequence $x_0, x_1, \ldots$ such that $x_k \to \ln 10$. Is this a reasonable way to write a subroutine that, given z, computes $\ln z$? (*Hint:* such a subroutine would require computation of e^{x_k}. Is this possible for rational x_k without using a logarithm? Is it practical?)

(4.10) Give an example (graphical or analytic) of a function, $f(x)$ for which Newton's Method:
  (a) Does not find a root for some choices of x_0.
  (b) Finds a root for every choice of x_0.
  (c) Falls into a cycle for some choice of x_0.
  (d) Converges slowly to the zero of $f(x)$.

(4.11) How will Newton's Method perform for finding the root to $f(x) = \sqrt{|x|} = 0$?

(4.12) Implement the inverse finder described in Example Problem 4.15. Your m-file should have header line like:

```
function c = invs(z)
```

where `z` is the number to be inverted. You may wish to use the builtin function `sign`. As an extra termination condition, you should have the subroutine return the current iterate if zx is sufficiently close to 1, say within the interval $(0.9999, 0.0001)$. Can you find some z for which the algorithm performs poorly?

(4.13) Implement the bisection method in octave/Matlab. Your m-file should have header line like:
`function c = run_bisection(f, a, b, tol)`
where `f` is the name of the function. Recall that `feval(f,x)` when `f` is a string with the name of some function evaluates that function at `x`. This works for builtin functions and m-files.
   (a) Run your code on the function $f(x) = \cos x$, with $a_0 = 0, b_0 = 2$. In this case you can set `f = "cos"`.
   (b) Run your code on the function $f(x) = x^2 - 5x + 3$, with $a_0 = 0, b_0 = 1$. In this case you will have to set `f` to the name of an m-file (without the ".m") which will evaluate the given $f(x)$.
   (c) Run your code on the function $f(x) = x - \cos x$, with $a_0 = 0, b_0 = 1$.

(4.14) Implement Newton's Method. Your m-file should have header line like:
`function x = run_newton(f, fp, x0, N, tol)`
where `f` is the name of the function, and `fp` is its derivative. Run your code to find zeroes of the following functions:
   (a) $f(x) = \tan x - x$.
   (b) $f(x) = x^2 - (2 + \epsilon)x + 1 + \epsilon$, for ϵ small.
   (c) $f(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$.
   (d) $f(x) = (x - 1)^7$.

(4.15) Implement the secant method Your m-file should have header line like:
`function x = run_secant(f, x0, x1, N, tol)`
where `f` is the name of the function. Run your code to find zeroes of the following functions:
   (a) $f(x) = \tan x - x$.
   (b) $f(x) = x^2 + 1$. (*Note:* This function has no zeroes.)
   (c) $f(x) = 2 + \sin x$. (*Note:* This function also has no zeroes.)

Chapter 5

Interpolation

5.1 Polynomial Interpolation

We consider the problem of finding a polynomial that interpolates a given set of values:

x	x_0	x_1	$\dots$	x_n
y	y_0	y_1	$\dots$	y_n

where the x_i are all distinct. A polynomial $p(x)$ is said to interpolate these data if $p(x_i) = y_i$ for $i = 0, 1, \dots, n$. The x_i values are called "nodes."

Sometimes, we will consider a variant of this problem: we have some black box function, $f(x)$, which we want to approximate with a polynomial $p(x)$. We do this by finding the polynomial interpolant to the data

x	x_0	x_1	$\dots$	x_n
$f(x)$	$f(x_0)$	$f(x_1)$	$\dots$	$f(x_n)$

for some choice of distinct nodes x_i.

5.1.1 Lagranges Method

As you might have guessed, for any such set of data, there is an n-degree polynomial that interpolates it. We present a constructive proof of this fact by use of Lagrange Polynomials.

Definition 5.1 (Lagrange Polynomials)**.** *For a given set of $n+1$ nodes x_i, the Lagrange polynomials are the $n+1$ polynomials ℓ_i defined by*

$$\ell_i(x_j) = \delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

Then we define the interpolating polynomial as

$$p_n(x) = \sum_{i=0}^{n} y_i \ell_i(x).$$

If each Lagrange Polynomial is of degree at most n, then p_n also has this property. The Lagrange Polynomials can be characterized as follows:

$$\boxed{\ell_i(x) = \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}.} \tag{5.1}$$

By evaluating this product for each x_j, we see that this is indeed a characterization of the Lagrange Polynomials. Moreover, each polynomial is clearly the product of n monomials, and thus has degree no greater than n.

This gives the theorem

Theorem 5.2 (Interpolant Existence and Uniqueness)**.** *Let $\{x_i\}_{i=0}^n$ be distinct nodes. Then for any values at the nodes, $\{y_i\}_{i=0}^n$, there is exactly one polynomial, $p(x)$ of degree no greater than n such that $p(x_i) = y_i$ for $i = 0, 1, \ldots, n$.*

Proof. The Lagrange Polynomial construction gives existence of such a polynomial $p(x)$ of degree no greater than n.

Suppose there were two such polynomials, call them $p(x), q(x)$, each of degree no greater than n, both interpolating the data. Let $r(x) = p(x) - q(x)$. Note that $r(x)$ can have degree no greater than n, yet it has roots at $x_0, x_1, \ldots, x_n$. The only polynomial of degree $\leq n$ that has $n + 1$ distinct roots is the zero polynomial, *i.e.*, $0 \equiv r(x) = p(x) - q(x)$. Thus $p(x), q(x)$ are equal everywhere, *i.e.*, they are the same polynomial. □

Example Problem 5.3. *Construct the polynomial interpolating the data*

x	1	$\frac{1}{2}$	3
y	3	-10	2

by using Lagrange Polynomials. Solution: *We construct the Lagrange Polynomials:*

$$\ell_0(x) = \frac{(x - \frac{1}{2})(x - 3)}{(1 - \frac{1}{2})(1 - 3)} = -(x - \frac{1}{2})(x - 3)$$
$$\ell_1(x) = \frac{(x - 1)(x - 3)}{(\frac{1}{2} - 1)(\frac{1}{2} - 3)} = \frac{4}{5}(x - 1)(x - 3)$$
$$\ell_2(x) = \frac{(x - 1)(x - \frac{1}{2})}{(3 - 1)(3 - \frac{1}{2})} = \frac{1}{5}(x - 1)(x - \frac{1}{2})$$

Then the interpolating polynomial, in "Lagrange Form" is

$$p_2(x) = -3(x - \frac{1}{2})(x - 3) - 8(x - 1)(x - 3) + \frac{2}{5}(x - 1)(x - \frac{1}{2})$$

⊣

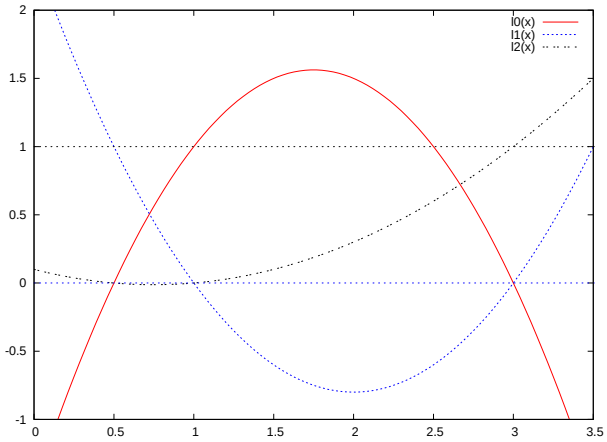


Figure 5.1: The 3 Lagrange polynomials for the example are shown. Verify, graphically, that these polynomials have the property $\ell_i(x_j) = \delta_{ij}$ for the nodes $1, \frac{1}{2}, 3$.

5.1.2 Newton's Method

There is another way to prove existence. This method is also constructive, and leads to a different algorithm for constructing the interpolant. One way to view this construction is to imagine how one would update the Lagrangian form of an interpolant. That is, suppose some data were given, and the interpolating polynomial calculated using Lagrange Polynomials; then a new point was given (x_{n+1}, y_{n+1}), and an interpolant for the augmented set of data is to be found. Each Lagrange Polynomial would have to be updated. This could take a lot of calculation (especially if n is large).

So the alternative method constructs the polynomials iteratively. Thus we create polynomials $p_k(x)$ such that $p_k(x_i) = y_i$ for $0 \leq i \leq k$. This is simple for $k = 0$, we simply let

$$p_0(x) = y_0,$$

the constant polynomial with value y_0. Then assume we have a proper $p_k(x)$ and want to construct $p_{k+1}(x)$. The following construction works:

$$p_{k+1}(x) = p_k(x) + c(x - x_0)(x - x_1) \cdots (x - x_k),$$

for some constant c. Note that the second term will be zero for any x_i for $0 \leq i \leq k$, so $p_{k+1}(x)$ will interpolate the data at $x_0, x_1, \ldots, x_k$. To get the value of the constant we calculate c such that

$$y_{k+1} = p_{k+1}(x_{k+1}) = p_k(x_{k+1}) + c(x_{k+1} - x_0)(x_{k+1} - x_1) \cdots (x_{k+1} - x_k).$$

This construction is known as Newton's Algorithm, and the resultant form is Newton's form of the interpolant

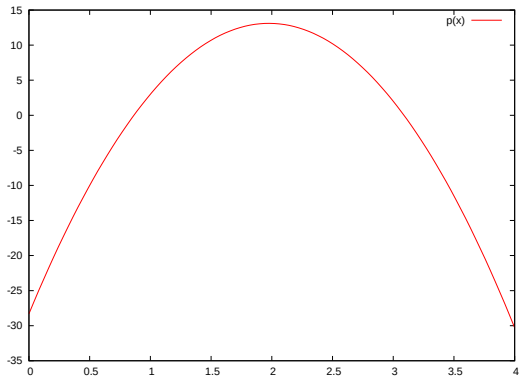


Figure 5.2: The interpolating polynomial for the example is shown.

Example Problem 5.4. *Construct the polynomial interpolating the data*

x	1	$\frac{1}{2}$	3
y	3	-10	2

by using Newton's Algorithm. Solution: *We construct the polynomial iteratively:*

$$p_0(x) = 3$$
$$p_1(x) = 3 + c(x-1)$$

We want $-10 = p_1(\frac{1}{2}) = 3 + c(-\frac{1}{2})$, *and thus* $c = 26$. *Then*

$$p_2(x) = 3 + 26(x-1) + c(x-1)(x-\frac{1}{2})$$

We want $2 = p_2(3) = 3 + 26(2) + c(2)(\frac{5}{2})$, *and thus* $c = \frac{-53}{5}$. *Then we get*

$$p_2(x) = 3 + 26(x-1) + \frac{-53}{5}(x-1)(x-\frac{1}{2}).$$

⊣

Does Newton's Algorithm give a different polynomial? It is easy to show, by induction, that the degree of $p_n(x)$ is no greater than n. Then, by Theorem 5.2, it must be the same polynomial as the Lagrange interpolant.

The two methods give the same interpolant, we may wonder "Which should we use?" Newton's Algorithm seems more flexible–it can deal with adding new data. There also happens to be a way of storing Newton's form of the interpolant that makes the polynomial simple to evaluate (in the sense of number of calculations required).

5.1.3 Newton's Nested Form

Recall the iterative construction of Newton's Form:

$$p_{k+1}(x) = p_k(x) + c_k(x - x_0)(x - x_1)\cdots(x - x_k).$$

The previous iterate $p_k(x)$ was constructed similarly, so we can write:

$$p_{k+1}(x) = [p_{k-1}(x) + c_{k-1}(x - x_0)(x - x_1)\cdots(x - x_{k-1})] + c_k(x-x_0)(x-x_1)\cdots(x-x_k).$$

Continuing in this way we see that we can write

$$\boxed{p_n(x) = \sum_{k=0}^{n} c_k \left[\prod_{0 \le j < k} (x - x_j) \right],} \tag{5.2}$$

where an empty product has value 1 by convention. This can be rewritten in a funny form, where a monomial is factored out of each successive summand:

$$p_n(x) = c_0 + (x - x_0)[c_1 + (x - x_1)[c_2 + (x - x_2)\,[\ldots]]]$$

Supposing for an instant that the constants c_k were known, this provides a better way of calculating $p_n(t)$ at arbitrary t. By "better" we mean requiring few multiplications and additions. This nested calculation is performed iteratively:

$$\begin{aligned}
v_0 &= c_n \\
v_1 &= c_{n-1} + (t - x_{n-1})v_0 \\
v_2 &= c_{n-2} + (t - x_{n-2})v_1 \\
&\vdots \\
v_n &= c_0 + (t - x_0)v_{n-1}
\end{aligned}$$

This requires only n multiplications and $2n$ additions. Compare this with the number required for using the Lagrange form: at least n^2 additions and multiplications.

5.1.4 Divided Differences

It turns out that the coefficients c_k for Newton's nested form can be calculated relatively easily by using *divided differences*. We assume, for the remainder of this section, that we are considering interpolating a function, that is, we have values of $f(x_i)$ at the nodes x_i.

Definition 5.5 (Divided Differences)**.** *For a given collection of nodes $\{x_i\}_{i=0}^n$ and values $\{f(x_i)\}_{x=0}^n$, a k^{th} order divided difference is a function of $k+1$ (not necessarily distinct) nodes, written as*

$$f\,[x_{i_0}, x_{i_1}, \ldots, x_{i_k}]$$

The divided differences are defined recursively as follows:

- *The 0^{th} order divided differences are simply defined:*

$$f[x_i] = f(x_i).$$

- *Higher order divided differences are the ratio of differences:*

$$f[x_{i_0}, x_{i_1}, \ldots, x_{i_k}] = \frac{f[x_{i_1}, x_{i_2}, \ldots, x_{i_k}] - f\left[x_{i_0}, x_{i_1}, \ldots, x_{i_{k-1}}\right]}{x_{i_k} - x_{i_0}}$$

We care about divided differences because coefficients for the Newton nested form are divided differences:

$$\boxed{c_k = f[x_0, x_1, \ldots, x_k].} \tag{5.3}$$

Because we are only interested in the Newton method coefficients, we will only consider divided differences with successive nodes, *i.e.*, those of the form $f[x_j, x_{j+1}, \ldots, x_{j+k}]$. In this case the higher order differences can more simply be written as

$$f[x_j, x_{j+1}, \ldots, x_{j+k}] = \frac{f[x_{j+1}, x_{j+2}, \ldots, x_{j+k}] - f[x_j, x_{j+1}, \ldots, x_{j+k-1}]}{x_{j+k} - x_j}$$

The graphical way to calculate these things is a "pyramid scheme"[1], where you compute the following table columnwise, from left to right:

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$	$f[\,,\,,\,,\,]$
x_0	$f[x_0]$			
		$f[x_0, x_1]$		
x_1	$f[x_1]$		$f[x_0, x_1, x_2]$	
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$
x_2	$f[x_2]$		$f[x_1, x_2, x_3]$	
		$f[x_2, x_2]$		
x_3	$f[x_3]$			

Note that by definition, the first column just echoes the data: $f[x_i] = f(x_i)$.

We drag out our example one last time:

Example Problem 5.6. *Find the divided differences for the following data*

x	1	$\frac{1}{2}$	3
$f(x)$	3	-10	2

Solution: *We start by writing in the data:*

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$
1	3		
$\frac{1}{2}$	-10		
3	2		

[1] That's supposed to be a joke.

Then we calculate:

$$f[x_0, x_1] = \frac{-10-3}{(1/2)-1} = 26, \quad \textit{and} \quad f[x_1, x_2] = \frac{2--10}{3-(1/2)} = 24/5.$$

Adding these to the table, we have

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$
1	3		
		26	
$\frac{1}{2}$	-10		
		$\frac{24}{5}$	
3	2		

Then we calculate

$$f[x_0, x_1, x_2] = \frac{24/5 - 26}{3-1} = \frac{-53}{5}.$$

So we complete the table:

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$
1	3		
		26	
$\frac{1}{2}$	-10		$\frac{-53}{5}$
		$\frac{24}{5}$	
3	2		

You should verify that along the top line of this pyramid you can read off the coefficients for Newton's form, as found in Example Problem 5.4. ⊣

5.2 Errors in Polynomial Interpolation

We now consider two questions:

1. If we want to interpolate some function $f(x)$ at $n+1$ nodes over some closed interval, how should we pick the nodes?
2. How accurate can we make a polynomial interpolant over a closed interval?

You may be surprised to find that the answer to the first question is *not* that we should make the x_i equally spaced over the closed interval, as the following example illustrates.

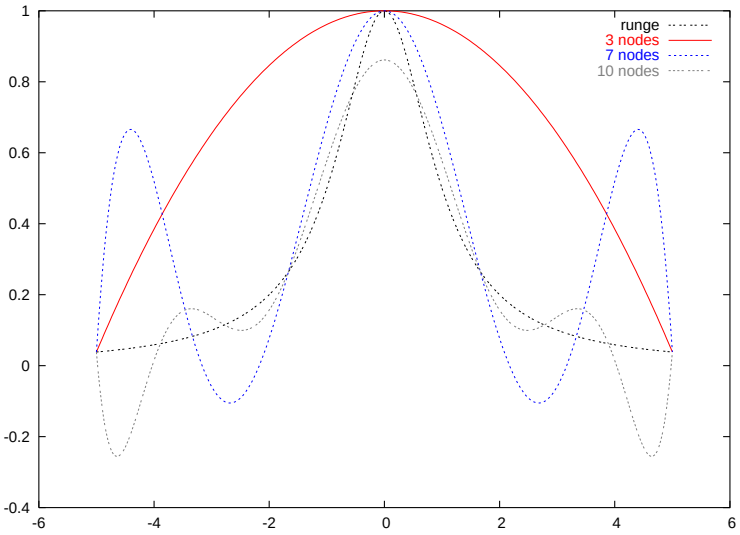
Example 5.7 (Runge Function)**.** *Let*

$$f(x) = \left(1 + x^2\right)^{-1},$$

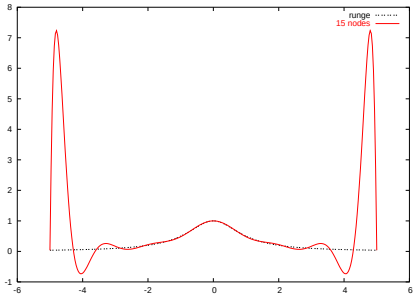
(known as the Runge Function*), and let $p_n(x)$ interpolate f on n equally spaced nodes, including the endpoints, on $[-5, 5]$. Then*

$$\lim_{n\to\infty} \max_{x\in[-5,5]} |p_n(x) - f(x)| = \infty.$$

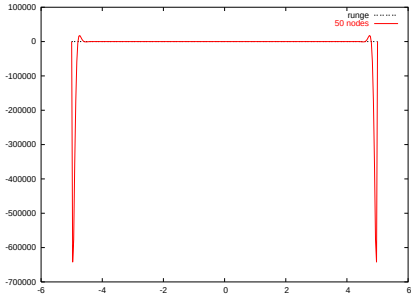
This behaviour is shown in Figure 5.3.



(a) 3,7,10 equally spaced nodes



(b) 15 equally spaced nodes



(c) 50 equally spaced nodes

Figure 5.3: The Runge function is poorly interpolated at n equally spaced nodes, as n gets large. At first, the interpolations appear to improve with larger n, as in (a). In (b) it becomes apparent that the interpolant is good in center of the interval, but bad at the edges. As shown in (c), this gets worse as n gets larger.

It turns out that a much better choice is related to the *Chebyshev Polynomials* ("of the first kind"). If our closed interval is $[-1, 1]$, then we want to define our nodes as

$$x_i = \cos\left[\left(\frac{2i+1}{2n+2}\right)\pi\right], \quad 0 \le i \le n.$$

Literally interpreted, these *Chebyshev Nodes* are the projections of points uniformly spaced on a semi circle; see Figure 5.4. By using the Chebyshev nodes, a good polynomial interpolant of the Runge function of Example 5.7 can be found, as shown in Figure 5.5.

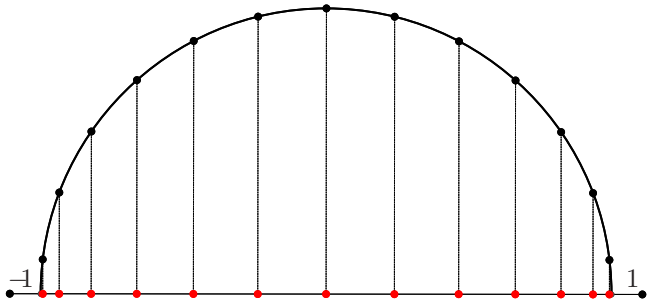
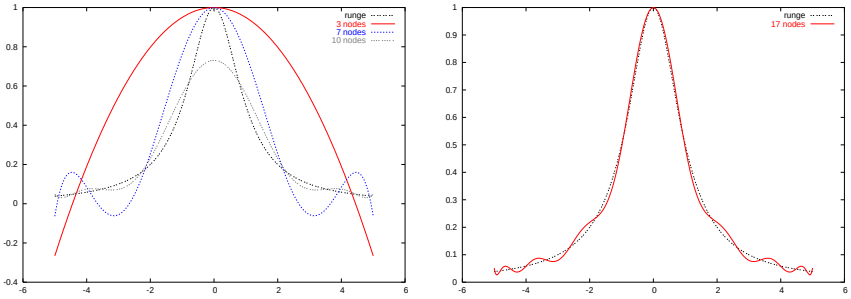


Figure 5.4: The Chebyshev nodes are the projections of nodes equally spaced on a semi circle.



(a) 3,7,10 Chebyshev nodes (b) 17 Chebyshev nodes

Figure 5.5: The Chebyshev nodes yield good polynomial interpolants of the Runge function. Compare these figures to those of Figure 5.3. For more than about 25 nodes, the interpolant is indistinguishable from the Runge function by eye.

5.2.1 Interpolation Error Theorem

We did not just invent the Chebyshev nodes. The fact that they are "good" for interpolation follows from the following theorem:

Theorem 5.8 (Interpolation Error Theorem)**.** *Let p be the polynomial of degree at most n interpolating function f at the $n+1$ distinct nodes $x_0, x_1, \ldots, x_n$ on $[a,b]$. Let $f^{(n+1)}$ be continuous. Then for each $x \in [a,b]$ there is some $\xi \in [a,b]$ such that*

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i) .$$

You should be thinking that the term on the right hand side resembles the error term in Taylor's Theorem.

Proof. First consider when x is one of the nodes x_i; in this case both the LHS and RHS are zero. So assume x is not a node. Make the following definitions

$$\begin{aligned} w(t) &= \prod_{i=0}^{n} (t - x_i) , \\ c &= \frac{f(x) - p(x)}{w(x)}, \\ \phi(t) &= f(t) - p(t) - cw(t). \end{aligned}$$

Since x is not a node, $w(x)$ is nonzero. Now note that $\phi(x_i)$ is zero for each node x_i, and that by definition of c, that $\phi(x) = 0$ for our x. That is $\phi(t)$ has $n+2$ roots.

Some other facts: f, p, w have $n+1$ continuous derivatives, by assumption and definition; thus ϕ has this many continuous derivatives. Apply Rolle's Theorem to $\phi(t)$ to find that $\phi'(t)$ has $n+1$ roots. Then apply Rolle's Theorem again to find $\phi''(t)$ has n roots. In this way we see that $\phi^{(n+1)}(t)$ has a root, call it ξ. That is

$$0 = \phi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - p^{(n+1)}(\xi) - cw^{(n+1)}(\xi).$$

But $p(t)$ is a polynomial of degree $\leq n$, so $p^{(n+1)}$ is identically zero. And $w(t)$ is a polynomial of degree $n+1$ in t, so its $n+1^{\text{th}}$ derivative is easily seen to be $(n+1)!$ Thus

$$\begin{aligned} 0 &= f^{(n+1)}(\xi) - c(n+1)! \\ c(n+1)! &= f^{(n+1)}(\xi) \\ \frac{f(x) - p(x)}{w(x)} &= \frac{1}{(n+1)!} f^{(n+1)}(\xi) \\ f(x) - p(x) &= \frac{1}{(n+1)!} f^{(n+1)}(\xi) w(x), \end{aligned}$$

which is what was to be proven. □

Thus the error in a polynomial interpolation is given as

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i).$$

We have no control over the function $f(x)$ or its derivatives, and once the nodes and f are fixed, p is determined; thus the only way we can make the error $|f(x) - p(x)|$ small is by judicious choice of the nodes x_i.

The Chebyshev nodes on $[-1, 1]$ have the remarkable property that

$$\left| \prod_{i=0}^{n} (t - x_i) \right| \leq 2^{-n}$$

for any $t \in [-1, 1]$. Moreover, it can be shown that for *any* choice of nodes x_i that

$$\max_{t \in [-1,1]} \left| \prod_{i=0}^{n} (t - x_i) \right| \geq 2^{-n}.$$

Thus the Chebyshev nodes are considered the best for polynomial interpolation.

Merging this result with Theorem 5.8, the error for polynomial interpolants defined on Chebyshev nodes can be bounded as

$$|f(x) - p(x)| \leq \frac{1}{2^n (n+1)!} \max_{\xi \in [-1,1]} \left| f^{(n+1)}(\xi) \right|.$$

The Chebyshev nodes can be rescaled and shifted for use on the general interval $[\alpha, \beta]$. In this case they take the form

$$x_i = \frac{\beta - \alpha}{2} \cos\left[\left(\frac{2i+1}{2n+2} \right) \pi \right] + \frac{\alpha + \beta}{2}, \quad 0 \leq i \leq n.$$

In this case, the rescaling of the nodes changes the bound on $\prod(t - x_i)$ so the overall error bound becomes

$$|f(x) - p(x)| \leq \frac{(\beta - \alpha)^{n+1}}{2^{2n+1} (n+1)!} \max_{\xi \in [\alpha,\beta]} \left| f^{(n+1)}(\xi) \right|,$$

for $x \in [\alpha, \beta]$.

Example Problem 5.9. *How many Chebyshev nodes are required to interpolate the function $f(x) = \sin(x) + \cos(x)$ to within 10^{-8} on the interval $[0, \pi]$?*
Solution: *We first find the derivatives of f*

$$\begin{aligned} f'(x) &= \cos(x) - \sin(x) \\ f''(x) &= -\sin(x) - \cos(x) \\ f'''(x) &= -\cos(x) + \sin(x) \\ &\vdots \end{aligned}$$

As a crude approximation we can assert that

$$\left|f^{(k)}(x)\right| \leq |\cos x| + |\sin x| \leq 2.$$

Thus it suffices to take n large enough such that

$$\frac{\pi^{n+1}}{2^{2n+1}\,(n+1)!}2 \leq 10^{-8}$$

By trial and error we see that $n = 10$ *suffices.* ⊣

5.2.2 Interpolation Error for Equally Spaced Nodes

Despite the proven superiority of Chebyshev Nodes, and the problems with the Runge Function, equally spaced nodes are frequently used for interpolation, since they are easy to calculate[2]. We now consider bounding

$$\max_{x\in[a,b]} \prod_{i=0}^{n} |x - x_i|\,,$$

where

$$x_i = a + hi = a + \frac{(b-a)}{n}i, \qquad i = 0, 1, \ldots, n.$$

Start by picking an x. We can assume x is not one of the nodes, otherwise the product in question is zero. Then x is between some x_j, x_{j+1} We can show that

$$|x - x_j|\,|x - x_{j+1}| \leq \frac{h^2}{4}.$$

by simple calculus.

Now we claim that $|x - x_i| \leq (j - i + 1)\,h$ for $i < j$, and $|x - x_i| \leq (i - j)\,h$ for $j + 1 < i$. Then

$$\prod_{i=0}^{n} |x - x_i| \leq \frac{h^2}{4}\left[(j+1)!h^j\right]\left[(n-j)!h^{n-j-1}\right].$$

It can be shown that $(j + 1)!(n - j)! \leq n!$, and so we get an overall bound

$$\prod_{i=0}^{n} |x - x_i| \leq \frac{h^{n+1}n!}{4}.$$

The interpolation theorem then gives us

$$|f(x) - p(x)| \leq \frac{h^{n+1}}{4\,(n+1)} \max_{\xi\in[a,b]} \left|f^{(n+1)}(\xi)\right|,$$

where $h = (b - a)/n$.

The reason this result does not seem to apply to Runge's Function is that $f^{(n)}$ for Runge's Function becomes unbounded as $n \to \infty$.

[2]Never underestimate the power of laziness.

Example Problem 5.10. *How many equally spaced nodes are required to interpolate the function* $f(x) = \sin(x) + \cos(x)$ *to within* 10^{-8} *on the interval* $[0, \pi]$*?*
Solution: *As in Example Problem 5.9, we make the crude approximation*

$$\left| f^{(k)}(x) \right| \leq |\cos x| + |\sin x| \leq 2.$$

Thus we want to make n *sufficiently large such that*

$$\frac{h^{n+1}}{4(n+1)} 2 \leq 10^{-8},$$

where $h = (\pi - 0)/n$*. That is we want to find* n *large enough such that*

$$\frac{\pi^{n+1}}{2n^{n+1}(n+1)} \leq 10^{-8},$$

By simply trying small numbers, we can see this is satisfied if $n = 12$. ⊣

Exercises

(5.1) Find the Lagrange Polynomials for the nodes $\{-1, 1\}$.
(5.2) Find the Lagrange Polynomials for the nodes $\{-1, 1, 5\}$.
(5.3) Find the polynomial of degree no greater than 2 that interpolates

x	-1	1	5
y	3	3	-2

(5.4) Complete the divided differences table:

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$
-1	3		
1	3		
5	-2		

Find the Newton form of the polynomial interpolant.
(5.5) Find the polynomial of degree no greater than 3 that interpolates

x	-1	1	5	-3
y	3	3	-2	4

(*Hint:* reuse the Newton form of the polynomial from the previous question.)
(5.6) Find the nested form of the polynomial interpolant of the data

x	1	3	4	6
y	-3	13	21	1

by completing the following divided differences table:

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$	$f[\,,\,,\,,\,]$
1	-3			
3	13			
4	21			
6	1			

(5.7) Find the polynomial of degree no greater than 3 that interpolates

x	1	0	$3/2$	2
y	3	2	$37/8$	8

(5.8) Complete the divided differences table:

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$	$f[\,,\,,\,,\,]$
-1	6			
0	3			
1	2			
2	3			

(Something a little odd should have happened in the last column.) Find the Newton form of the polynomial interpolant. Of what degree is the polynomial interpolant?

(5.9) Let $p(x)$ interpolate the function $\cos(x)$ at n equally spaced nodes on the interval $[0, 2]$. Bound the error

$$\max_{0\le x\le 2} |p(x) - \cos(x)|$$

as a function of n. How small is the error when $n = 10$? How small would the error be if Chebyshev nodes were used instead? How about when $n = 10$?

(5.10) Let $p(x)$ interpolate the function x^{-2} at n equally spaced nodes on the interval $[0.5, 1]$. Bound the error

$$\max_{0.5\le x\le 1} \left|p(x) - x^{-2}\right|$$

as a function of n. How small is the error when $n = 10$?

(5.11) How many Chebyshev nodes are required to interpolate the function $\frac{1}{x}$ to within 10^{-6} on the interval $[1, 2]$?

(5.12) Write code to calculate the Newton form coefficients, by divided differences, for the nodes x_i and values $f(x_i)$. Your m-file should have header line like:

```
function coefs = newtonCoef(xs,fxs)
```

where `xs` is the vector of $n+1$ nodes, and `fxs` the vector of $n+1$ values. Test your code on the following input:

```
octave:1> xs = [1 -1 2 -2 3 -3 4 -4];
octave:2> fxs = [1 1 2 3 5 8 13 21];
octave:3> newtonCoef(xs,fxs)
ans =

   1.00000  -0.00000   0.33333  -0.08333   0.05000   0.00417  -0.00020  -0.00040
```

(a) What do you get when you try the following?

```
octave:4> xs = [1 4 5 9 14 23 37 60];
octave:5> fxs = [3 1 4 1 5 9 2 6];
octave:6> newtonCoef(xs,ys)
```

(b) Try the following:

```
octave:7> xs = [1 3 4 2 8 -2 0 14 23 15];
octave:8> fxs = xs.*xs + xs .+ 4;
octave:9> newtonCoef(xs,fxs)
```

(5.13) Write code to calculate a polynomial interpolant from its Newton form coefficients and the node values. Your m-file should have header line like:
`function y = calcNewton(t,coefs,xs)`
where `coefs` is a vector of the Newton coefficients, `xs` is a vector of the nodes x_i, and `y` is the value of the interpolating polynomial at `t`. Check your code against the following values:

```
octave:1> xs = [1 3 4];
octave:2> coefs = [6 5 1];
octave:3> calcNewton (5,coefs,xs)
ans = 34
octave:4> calcNewton (-3,coefs,xs)
ans = 10
```

(a) What do you get when you try the following?

```
octave:5> xs = [3 1 4 5 9 2 6 8 7 0];
octave:6> coefs = [1 -1 2 -2 3 -3 4 -4 5 -5];
octave:7> calcNewton(0.5,coefs,xs)
```

(b) Try the following

```
octave:8> xs = [3 1 4 5 9 2 6 8 7 0];
octave:9> coefs = [1 -1 2 -2 3 -3 4 -4 5 -5];
octave:10> calcNewton(1,coefs,xs)
```

Chapter 6

Spline Interpolation

Splines are used to approximate complex functions and shapes. A spline is a function consisting of simple functions glued together. In this way a spline is different from a polynomial interpolation, which consists of a single well defined function that approximates a given shape; splines are normally piecewise polynomial.

6.1 First and Second Degree Splines

Splines make use of partitions, which are a way of cutting an interval into a number of subintervals.

Definition 6.1 (Partition)**.** *A* partition *of the interval* $[a, b]$ *is an ordered sequence* $\{t_i\}_{i=0}^{n}$ *such that*

$$a = t_0 < t_1 < \cdots < t_{n-1} < t_n = b$$

The numbers t_i *are known as* knots*.*

A spline of degree 1, also known as a *linear spline*, is a function which is linear on each subinterval defined by a partition:

Definition 6.2 (Linear Splines)**.** *A function* S *is a spline of degree* 1 *on* $[a, b]$ *if*

1. *The domain of* S *is* $[a, b]$*.*
2. S *is continuous on* $[a, b]$*.*
3. *There is a partition* $\{t_i\}_{i=0}^{n}$ *of* $[a, b]$ *such that on each* $[t_i, t_{i+1}]$*,* S *is a linear polynomial.*

A linear spline is defined entirely by its value at the knots. That is, given

t	t_0	t_1	$\ldots$	t_n
y	y_0	y_1	$\cdots$	y_n

there is only one linear spline with these values at the knots and linear on each given subinterval.

For a spline with this data, the linear polynomial on each subinterval is defined as

$$S_i(x) = y_i + \frac{y_{i+1} - y_i}{t_{i+1} - t_i}(x - t_i).$$

Note that if $x \in [t_i, t_{i+1}]$, then $x - t_i > 0$, but $x - t_{i-1} \leq 0$. Thus if we wish to evaluate $S(x)$, we search for the largest i such that $x - t_i > 0$, then evaluate $S_i(x)$.

Example 6.3. *The linear spline for the following data*

t	0.0	0.1	0.4	0.5	0.75	1.0
y	1.3	4.5	2.0	2.1	5.0	3

is shown in Figure 6.1.

[height=0.65angle=270,clip=]figs/spline1.eps

Figure 6.1: A linear spline. The spline is piecewise linear, and is linear between each knot.

6.1.1 First Degree Spline Accuracy

As with polynomial functions, splines are used to interpolate tabulated data as well as functions. In the latter case, if the spline is being used to interpolate the function f, say, then this is equivalent to interpolating the data

t	t_0	t_1	$\ldots$	t_n
y	$f(t_0)$	$f(t_1)$	$\ldots$	$f(t_n)$

A function and its linear spline interpolant are shown in Figure 6.2. The spline interpolant in that figure is fairly close to the function over some of the interval in question, but it also deviates greatly from the function at other points of the interval. We are interested in finding bounds on the possible error between a function and its spline interpolant.

To find the error bound, we will consider the error on a single interval of the partition, and use a little calculus.[1] Suppose $p(t)$ is the linear polynomial interpolating $f(t)$ at the endpoints of the subinterval $[t_i, t_{i+1}]$, then for $t \in [t_i, t_{i+1}]$,

$$|f(t) - p(t)| \leq \max\{|f(t) - f(t_i)|, |f(t) - f(t_{i+1})|\}.$$

That is, $|f(t) - p(t)|$ is no larger than the "maximum variation" of $f(t)$ on this interval.

[1] Although we could directly claim Theorem 5.8, it is a bit of overkill.

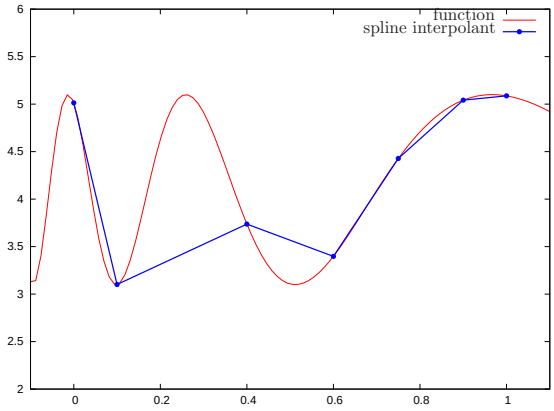


Figure 6.2: The function $f(t) = 4.1 + \sin(1/(0.08t + 0.5))$ is shown, along with the linear spline interpolant, for the knots $0, 0.1, 0.4, 0.6, 0.75, 0.9, 1.0$ For some values of t, the function and its interpolant are very close, while they vary greatly in the middle of the interval.

In particular, if $f'(t)$ exists and is bounded by M_1 on $[t_i, t_{i+1}]$, then

$$|f(t) - p(t)| \leq \frac{M_1}{2}(t_{i+1} - t_i).$$

Similarly, if $f''(t)$ exists and is bounded by M_2 on $[t_i, t_{i+1}]$, then

$$|f(t) - p(t)| \leq \frac{M_2}{8}(t_{i+1} - t_i)^2.$$

Over a given partition, these become, respectively

$$\boxed{\begin{aligned} |f(t) - p(t)| &\leq \frac{M_1}{2} \max_{0 \leq i < n} (t_{i+1} - t_i), \\ |f(t) - p(t)| &\leq \frac{M_2}{8} \max_{0 \leq i < n} (t_{i+1} - t_i)^2. \end{aligned}} \tag{6.1}$$

If equally spaced nodes are used, these bounds guarantee that spline interpolants become better as the number of nodes is increased. This contrasts with polynomial interpolants, which may get worse as the number of nodes is increased, *cf.* Example 5.7.

6.1.2 Second Degree Splines

Piecewise quadratic splines, or splines of degree 2, are defined similarly:

Definition 6.4 (Quadratic Splines)**.** *A function Q is a quadratic spline on $[a, b]$ if*

1. *The domain of Q is $[a, b]$.*
2. *Q is continuous on $[a, b]$.*
3. *Q' is continuous on (a, b).*
4. *There is a partition $\{t_i\}_{i=0}^n$ of $[a, b]$ such that on $[t_i, t_{i+1}]$, Q is a polynomial of degree at most 2.*

Example 6.5. *The following is a quadratic splne:*

$$Q(x) = \begin{cases} -x & x \leq 0, \\ x^2 - x & 0 \leq x \leq 2, \\ -x^2 + 7x - 8 & 2 \leq x. \end{cases}$$

Unlike linear splines, quadratic splines are *not* defined entirely by their values at the knots. We consider why that is. The spline $Q(x)$ is defined by its piecewise polynomials,

$$Q_i(x) = a_i x^2 + b_i x + c_i.$$

Thus there are $3n$ parameters to define $Q(x)$.

For each of the n subintervals, the data

t	t_0	t_1	$\dots$	t_n
y	y_0	y_1	$\dots$	y_n

give two equations regarding $Q_i(x)$, namely that $Q_i(t_i)$ must equal y_i and $Q_i(t_{i+1})$ must equal y_{i+1}. This is $2n$ equations. The condition on continuity of Q' gives a single equation for each of the $n - 1$ internal nodes. This totals $3n - 1$ equations, but $3n$ unknowns. This system is underdetermined.

Thus some additional user-chosen condition is required to determine the quadratic spline. One might choose, for example, $Q'(a) = 0$, or $Q''(a) = 0$, or some other condition.

6.1.3 Computing Second Degree Splines

Suppose the data

t	t_0	t_1	$\dots$	t_n
y	y_0	y_1	$\dots$	y_n

are given. Let $z_i = Q_i'(t_i)$, and suppose that the additional condition to define the quadratic spline is given by specifying z_0. We want to be able to compute the form of $Q_i(x)$.

Because $Q_i(t_i) = y_i, Q_i'(t_i) = z_i, Q_i'(t_{i+1}) = z_{i+1}$, we see that we can define

$$Q_i(x) = \frac{z_{i+1} - z_i}{2\,(t_{i+1} - t_i)}\,(x - t_i)^2 + z_i\,(x - t_i) + y_i.$$

Use this at t_{i+1} :

$$y_{i+1} = Q_i(t_{i+1}) = \frac{z_{i+1} - z_i}{2\,(t_{i+1} - t_i)}\,(t_{i+1} - t_i)^2 + z_i\,(t_{i+1} - t_i) + y_i,$$
$$y_{i+1} - y_i = \frac{z_{i+1} - z_i}{2}\,(t_{i+1} - t_i) + z_i\,(t_{i+1} - t_i)\,,$$
$$y_{i+1} - y_i = \frac{z_{i+1} + z_i}{2}\,(t_{i+1} - t_i)\,.$$

Thus we can determine, from the data alone, z_{i+1} from z_i:

$$z_{i+1} = 2\frac{y_{i+1} - y_i}{t_{i+1} - t_i} - z_i.$$

6.2 (Natural) Cubic Splines

If you recall the definition of the linear and quadratic splines, probably you can guess the definition of the spline of degree k:

Definition 6.6 (Splines of Degree k)**.** *A function S is a spline of degree k on $[a, b]$ if*

1. *The domain of S is $[a, b]$.*
2. *$S, S', S'', \ldots, S^{(k-1)}$ are continuous on (a, b).*
3. *There is a partition $\{t_i\}_{i=0}^n$ of $[a, b]$ such that on $[t_i, t_{i+1}]$, S is a polynomial of degree $\leq k$.*

You would also expect that a spline of degree k has $k-1$ "degrees of freedom," as we show here. If the partition has $n+1$ knots, the spline of degree k is defined by $n(k + 1)$ parameters. The given data

t	t_0	t_1	$\ldots$	t_n
y	y_0	y_1	$\ldots$	y_n

provide $2n$ equations. The continuity of $S', S'', \ldots, S^{(k-1)}$ at the $n - 1$ internal knots gives $(k - 1)(n - 1)$ equations. This is a total of $n(k + 1) - (k - 1)$ equations. Thus we have $k - 1$ more unknowns than equations. Thus, barring some singularity, we can (and must) add $k - 1$ constraints to uniquely define the spline. These are the degrees of freedom.

Often k is chosen as 3. This yields cubic splines. We must add 2 extra constraints to define the spline. The usual choice is to make

$$S''(t_0) = S''(t_n) = 0.$$

This yields the *natural cubic spline.*

6.2.1 Why Natural Cubic Splines?

It turns out that natural cubic splines are a good choice in the sense that they are the "interpolant of minimal H^2 seminorm." The corollary following this theorem states this in more easily understandable terms:

Theorem 6.7. *Suppose f has two continuous derivatives, and S is the natural cubic spline interpolating f at knots $a = t_0 < t_1 < \ldots < t_n = b$. Then*

$$\int_a^b \left[S''(x)\right]^2 \,\mathrm{d}x \leq \int_a^b \left[f''(x)\right]^2 \,\mathrm{d}x$$

Proof. We let $g(x) = f(x) - S(x)$. Then $g(x)$ is zero on the $(n+1)$ knots t_i. Derivatives are linear, meaning that

$$f''(x) = S''(x) + g''(x).$$

Then

$$\int_a^b \left[f''(x)\right]^2 \,\mathrm{d}x = \int_a^b \left[S''(x)\right]^2 \,\mathrm{d}x + \int_a^b \left[g''(x)\right]^2 \,\mathrm{d}x + \int_a^b 2S''(x)g''(x)\,\mathrm{d}x.$$

We show that the last integral is zero. Integrating by parts we get

$$\int_a^b S''(x)g''(x)\,\mathrm{d}x = S''g'\Big|_a^b - \int_a^b S'''g'\,\mathrm{d}x = -\int_a^b S'''g'\,\mathrm{d}x,$$

because $S''(a) = S''(b) = 0$. Then notice that S is a polynomial of degree ≤ 3 on each interval, thus $S'''(x)$ is a piecewise constant function, taking value c_i on each interval $[t_i, t_{i+1}]$. Thus

$$\int_a^b S'''g'\,\mathrm{d}x = \sum_{i=0}^{n-1} \int_a^b c_i g'\,\mathrm{d}x = \sum_{i=0}^{n-1} c_i g\Big|_{t_i}^{t_{i+1}} = 0,$$

with the last equality holding because $g(x)$ is zero at the knots. □

Corollary 6.8. *The natural cubic spline is best twice-continuously differentiable interpolant for a twice-continuously differentiable function, under the measure given by the theorem.*

Proof. Let f be twice-continuously differentiable, and let S be the natural cubic spline interpolating $f(x)$ at some given nodes $\{t_i\}_{i=0}^n$. Let $R(x)$ be some twice-continuously differentiable function which also interpolates $f(x)$ at these nodes. Then $S(x)$ interpolates $R(x)$ at these nodes. Apply the theorem to get

$$\int_a^b \left[S''(x)\right]^2 \,\mathrm{d}x \leq \int_a^b \left[R''(x)\right]^2 \,\mathrm{d}x$$

□

6.2.2 Computing Cubic Splines

First we present an example of computing the natural cubic spline by hand:

Example Problem 6.9. *Construct the natural cubic spline for the following data:*

t	-1	0	2
y	3	-1	3

Solution: *The natural cubic spline is defined by eight parameters:*

$$S(x) = \begin{cases} ax^3 + bx^2 + cx + d & x \in [-1, 0] \\ ex^3 + fx^2 + gx + h & x \in [0, 2] \end{cases}$$

We interpolate to find that $d = h = -1$ *and*

$$-a + b - c - 1 = 3$$
$$8e + 4f + 2g - 1 = 3$$

We take the derivative of S*:*

$$S'(x) = \begin{cases} 3ax^2 + 2bx + c & x \in [-1, 0] \\ 3ex^2 + 2fx + g & x \in [0, 2] \end{cases}$$

Continuity at the middle node gives $c = g$*. Now take the second derivative of* S*:*

$$S''(x) = \begin{cases} 6ax + 2b & x \in [-1, 0] \\ 6ex + 2f & x \in [0, 2] \end{cases}$$

Continuity at the middle node gives $b = f$*. The natural cubic spline condition gives* $-6a + 2b = 0$ *and* $12e + 2f = 0$*. Solving this by "divide and conquer" gives*

$$S(x) = \begin{cases} x^3 + 3x^2 - 2x - 1 & x \in [-1, 0] \\ -\frac{1}{2}x^3 + 3x^2 - 2x - 1 & x \in [0, 2] \end{cases}$$

⊣

Finding the constants for the previous example was fairly tedious. And this is for the case of only three nodes. We would like a method easier than setting up the $4n$ equations and unknowns, something akin to the description in Subsection 6.1.3. The method is rather tedious, so we leave it to the exercises.

6.3 B Splines

The B splines form a *basis* for spline functions, whence the name. We presuppose the existence of an *infinite* number of knots:

$$\ldots < t_2 < t_1 < t_0 < t_1 < t_2 < \ldots,$$

with $\lim_{k \to -\infty} t_k = -\infty$ and $\lim_{k \to \infty} t_k = \infty$.

The B splines of degree 0 are defined as single "blocks":

$$B_i^0(x) = \begin{cases} 1 & t_i \leq x < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

The zero degree B splines are continuous from the right, are nonzero only on one subinterval $[t_i, t_{i+1})$, sum to 1 everywhere.

We justify the description of B splines as basis splines: If S is a spline of degree 0 on the given knots and is continuous from the right then

$$S(x) = \sum_i S(x_i) B_i^0(x).$$

That is, the basis splines work in the same way that Lagrange Polynomials worked for polynomial interpolation.

The B splines of degree k are defined recursively:

$$B_i^k(x) = \left(\frac{x - t_i}{t_{i+k} - t_i} \right) B_i^{k-1}(x) + \left(\frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} \right) B_{i+1}^{k-1}(x).$$

Some B splines are shown in Figure 6.3.

The B splines quickly become unwieldy. We focus on the case $k = 1$. The B spline $B_i^1(x)$ is

- Piecewise linear.
- Continuous.
- Nonzero only on (t_i, t_{i+2}).
- 1 at t_{i+1}.

These B splines are sometimes called *hat functions*. Imagine wearing a hat shaped like this! Whatever.

The nice thing about the hat functions is they allow us to use analogy. Harken back to polynomial interpolation and the Lagrange Functions. The hat functions play a similar role because

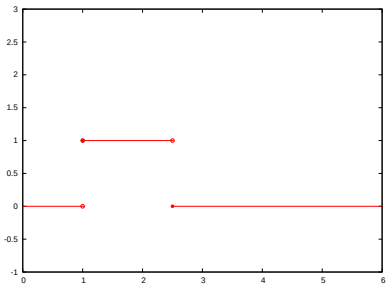
$$B_i^1(t_j) = \delta_{(i+1)j} = \begin{cases} 1 & (i+1) = j \\ 0 & (i+1) \neq j \end{cases}$$

Then if we want to interpolate the following data with splines of degree 1:

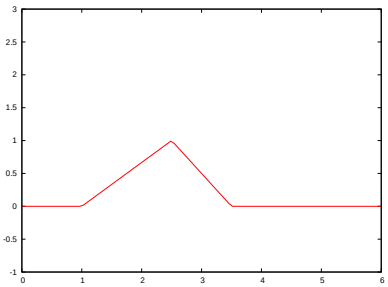
t	t_0	t_1	$\dots$	t_n
y	y_0	y_1	$\dots$	y_n

We can immediately set

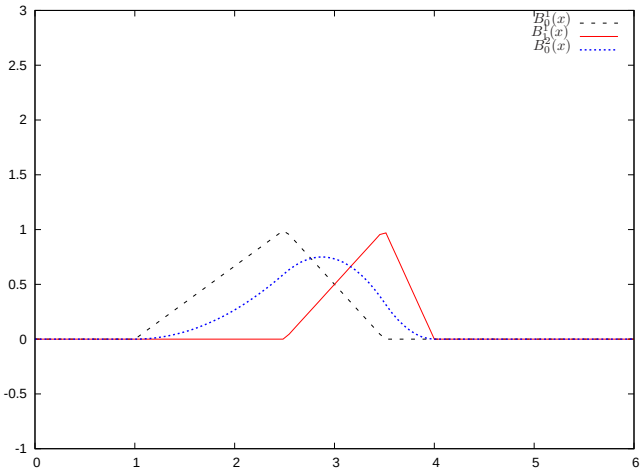
$$S(x) = \sum_{i=0}^{n} y_i B_{i-1}^1(x).$$



(a) Degree 0 B spline



(b) Degree 1 B spline



(c) Degree 2 B spline, with 2 Degree 1 B splines

Figure 6.3: Some B splines for the knots $t_0 = 1, t_1 = 2.5, t_2 = 3.5, t_3 = 4$ are shown. In (a), one of the degree 0 B splines; in (b), a degree 1 B spline; in (c), two of the degree 1 B splines and a degree 2 B spline are shown. The two hat functions "merge" together to give the quadratic B spline.

EXERCISES

(6.1) Is the following function a linear spline on $[0, 4]$? Why or why not?

$$S(x) = \begin{cases} 3x + 2 & : 0 \le x < 1 \\ -2x + 4 & : 1 \le x \le 4 \end{cases}$$

(6.2) Is the following function a linear spline on $[0, 2]$? Why or why not?

$$S(x) = \begin{cases} x + 3 & : 0 \le x < 1 \\ 3 & : 1 \le x \le 2 \end{cases}$$

(6.3) Is the following function a linear spline on $[0, 4]$? Why or why not?

$$S(x) = \begin{cases} x^2 + 3 & : 0 \le x < 3 \\ 5x - 6 & : 3 \le x \le 4 \end{cases}$$

(6.4) Find constants, α, β such that the following is a linear spline on $[0, 5]$.

$$S(x) = \begin{cases} 4x - 2 & : 0 \le x < 1 \\ \alpha x + \beta & : 1 \le x < 3 \\ -2x + 10 & : 3 \le x \le 5 \end{cases}$$

(6.5) Is the following function a quadratic spline on $[0, 4]$? Why or why not?

$$Q(x) = \begin{cases} x^2 + 3 & : 0 \le x < 3 \\ 5x - 6 & : 3 \le x \le 4 \end{cases}$$

(6.6) Is the following function a quadratic spline on $[0, 2]$? Why or why not?

$$Q(x) = \begin{cases} x^2 + 3x + 2 & : 0 \le x < 1 \\ 2x^2 + x + 3 & : 1 \le x \le 2 \end{cases}$$

(6.7) Find constants, α, β, γ such that the following is a quadratic spline on $[0, 5]$.

$$Q(x) = \begin{cases} \frac{1}{2}x^2 + 2x + \frac{3}{2} & : 0 \le x < 1 \\ \alpha x^2 + \beta x + \gamma & : 1 \le x < 3 \\ 3x^2 - 7x + 12 & : 3 \le x \le 5 \end{cases}$$

(6.8) Find the quadratic spline that interpolates the following data:

t	0	1	4
y	1	-2	1

To resolve the single degree of freedom, assume that $Q'(0) = -Q'(4)$. Assume your solution takes the form

$$Q(x) = \begin{cases} \alpha_1 (x-1)^2 + \beta_1 (x-1) - 2 & : 0 \le x < 1 \\ \alpha_2 (x-1)^2 + \beta_2 (x-1) - 2 & : 1 \le x \le 4 \end{cases}$$

Find the constants $\alpha_1, \beta_1, \alpha_2, \beta_2$.

(6.9) Find the natural cubic spline that interpolates the data

x	0	1	3
y	4	2	7

It may help to assume your answer has the form

$$S(x) = \begin{cases} Ax^3 + Bx^2 + Cx + 4 & : 0 \le x < 1 \\ D(x-1)^3 + E(x-1)^2 + F(x-1) + 2 & : 1 \le x \le 3 \end{cases}$$

Chapter 7

Solving Linear Systems

A number of problems in numerical analysis can be reduced to, or approximated by, a system of linear equations.

7.1 Gaussian Elimination with Naïve Pivoting

Our goal is the automatic solution of systems of linear equations:

$$\begin{array}{ccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ a_{31}x_1 & + & a_{32}x_2 & + & a_{33}x_3 & + & \cdots & + & a_{3n}x_n & = & b_3 \\ \vdots & & \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & a_{n3}x_3 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array}$$

In these equations, the a_{ij} and b_i are given real numbers. We also write this as

$$\mathsf{A}\boldsymbol{x} = \boldsymbol{b},$$

where A is a matrix, whose element in the i^{th} row and j^{th} column is a_{ij}, and $\boldsymbol{b}$ is a column vector, whose i^{th} entry is b_i.

This gives the easier way of writing this equation:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix} \tag{7.1}$$

7.1.1 Elementary Row Operations

You may remember that one way to solve linear equations is by applying *elementary row operations* to a given equation of the system. For example, if we

are trying to solve the given system of equations, they should have the same solution as the following system:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \kappa a_{i1} & \kappa a_{i2} & \kappa a_{i3} & \cdots & \kappa a_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \kappa b_i \\ \vdots \\ b_n \end{bmatrix}$$

where κ is some given number which is *not zero*. It suffices to solve this system of linear equations, as it has the same solution(s) as our original system. Multiplying a row of the system by a nonzero constant is one of the elementary row operations.

The second elementary row operation is to replace a row by the sum of that row and a constant times another. Thus, for example, the following system of equations has the same solution as the original system:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{(i-1)1} & a_{(i-1)2} & a_{(i-1)3} & \cdots & a_{(i-1)n} \\ a_{i1} + \beta a_{j1} & a_{i2} + \beta a_{j2} & a_{i3} + \beta a_{j3} & \cdots & a_{in} + \beta a_{jn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{(i-1)} \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{(i-1)} \\ b_i + \beta b_j \\ \vdots \\ b_n \end{bmatrix}$$

We have replaced the i^{th} row by the i^{th} row plus β times the j^{th} row.

The third elementary row operation is to switch rows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_3 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

We have here switched the second and third rows. The purpose of this e.r.o. is mainly to make things look nice.

Note that none of the e.r.o.'s change the structure of the solution vector $\boldsymbol{x}$. For this reason, it is customary to drop the solution vector entirely and to write

the matrix A and the vector $\boldsymbol{b}$ together in *augmented form*:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array}\right)$$

The idea of Gaussian Elimination is to use the elementary row operations to put a system into *upper triangular form* then use *back substitution*. We'll give an example here:

Example Problem 7.1. *Solve the set of linear equations:*

$$\begin{aligned} x_1 + x_2 - x_3 &= 2 \\ -3x_1 - 4x_2 + 4x_3 &= -7 \\ 2x_1 + 1x_2 + 1x_3 &= 7 \end{aligned}$$

Solution: *We start by rewriting in the augmented form:*

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 2 \\ -3 & -4 & 4 & -7 \\ 2 & 1 & 1 & 7 \end{array}\right)$$

We add 3 *times the first row to the second, and* -2 *times the first row to the third to get:*

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 2 \\ 0 & -1 & 1 & -1 \\ 0 & -1 & 3 & 3 \end{array}\right)$$

We now add -1 *times the second row to the third row to get:*

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 2 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & 2 & 4 \end{array}\right)$$

The matrix is now in upper triangular form: there are no nonzero entries below the diagonal. This corresponds to the set of equations:

$$\begin{aligned} x_1 + x_2 - x_3 &= 2 \\ -x_2 + x_3 &= -1 \\ 2x_3 &= 4 \end{aligned}$$

We now solve this by back substitution. Because the matrix is in upper triangular form, we can solve x_3 *by looking only at the last equation; namely* $x_3 = 2$. *However, once* x_3 *is known, the second equation involves only one unknown,* x_2, *and can be solved only by* $x_2 = 3$. *Then the first equation has only one unknown, and is solved by* $x_1 = 1$. ⊣

All sorts of funny things can happen when you attempt Gaussian Elimination: it may turn out that your system has no solution, or has a single solution (as above), or an infinite number of solutions. We should expect that an algorithm for automatic solution of systems of equations should detect these problems.

7.1.2 Algorithm Terminology

The method outlined above is fine for solving small systems. We should like to devise an algorithm for doing the same thing which can be applied to large systems of equations. The algorithm will take the system (in augmented form):

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array}\right)$$

The algorithm then selects the first row as the *pivot equation* or *pivot row*, and the first element of the first row, a_{11} is the *pivot element*. The algorithm then *pivots* on the pivot element to get the system:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & a'_{32} & a'_{33} & \cdots & a'_{3n} & b'_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & a'_{n2} & a'_{n3} & \cdots & a'_{nn} & b'_n \end{array}\right)$$

Where

$$\left.\begin{array}{rl} a'_{ij} & = a_{ij} - \left(\frac{a_{i1}}{a_{11}}\right) a_{1j} \\ b'_i & = b_i - \left(\frac{a_{i1}}{a_{11}}\right) b_1 \end{array}\right\} \qquad (2 \le i \le n,\ 1 \le j \le n)$$

Effectively we are carrying out the e.r.o. of replacing the i^{th} row by the i^{th} row minus $\left(\frac{a_{i1}}{a_{11}}\right)$ times the first row. The quantity $\left(\frac{a_{i1}}{a_{11}}\right)$ is the *multiplier* for the i^{th} row.

Hereafter the algorithm will not alter the first row or first column of the system. Thus, the algorithm could be written recursively. By pivoting on the second row, the algorithm then generates the system:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & 0 & a''_{33} & \cdots & a''_{3n} & b''_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & a''_{n3} & \cdots & a''_{nn} & b''_n \end{array}\right)$$

In this case

$$\left.\begin{array}{rcl} a''_{ij} & = & a'_{ij} - \left(\frac{a'_{i2}}{a'_{22}}\right) a'_{2j} \\ b''_{i} & = & b'_{i} - \left(\frac{a'_{i2}}{a'_{22}}\right) b'_{2} \end{array}\right\} \qquad (3 \le i \le n,\, 1 \le j \le n)$$

7.1.3 Algorithm Problems

The pivoting strategy we examined in this section is called 'naïve' because a real algorithm is a bit more complicated. The algorithm we have outlined is far too rigid–it always chooses to pivot on the k^{th} row during the k^{th} step. This would be bad if the pivot element were zero; in this case all the multipliers $\frac{a_{ik}}{a_{kk}}$ are not defined.

Bad things can happen if a_{kk} is merely small instead of zero. Consider the following example:

Example 7.2. *Solve the system of equations given by the augmented form:*

$$\left(\begin{array}{rr|r} -0.0590 & 0.2372 & -0.3528 \\ 0.1080 & -0.4348 & 0.6452 \end{array}\right)$$

Note that the exact solution of this system is $x_1 = 10$, $x_2 = 1$. Suppose, however, that the algorithm uses only 4 significant figures for its calculations. The algorithm, naïvely, pivots on the first equation. The multiplier for the second row is

$$\frac{0.1080}{-0.0590} \approx -1.830508...,$$

which will be rounded to -1.831 by the algorithm.

The second entry in the matrix is replaced by

$$-0.4348 - (-1.831)(0.2372) = -0.4348 + 0.4343 = -0.0005,$$

where the arithmetic is rounded to four significant figures each time. There is some serious subtractive cancellation going on here. We have lost three figures with this subtraction. The errors get worse from here. Similarly, the second vector entry becomes:

$$0.6452 - (-1.831)(-0.3528) = 0.6452 - 0.6460 = -0.0008,$$

where, again, intermediate steps are rounded to four significant figures, and again there is subtractive cancelling. This puts the system in the form

$$\left(\begin{array}{rr|r} -0.0590 & 0.2372 & -0.3528 \\ 0 & -0.0005 & -0.0008 \end{array}\right)$$

When the algorithm attempts back substitution, it gets the value

$$x_2 = \frac{-0.0008}{-0.0005} = 1.6.$$

This is a bit off from the actual value of 1. The algorithm now finds

$$x_1 = (-0.3528 - 0.2372 \cdot 1.6)\,/{-0.059} = (-0.3528 - 0.3795)\,/{-0.059} = (-0.7323)\,/{-0.059} = 12.41,$$

where each step has rounding to four significant figures. This is also a bit off.

7.2 Pivoting Strategies for Gaussian Elimination

Gaussian Elimination can fail when performed in the wrong order. If the algorithm selects a zero pivot, the multipliers are undefined, which is no good. We also saw that a pivot small in magnitude can cause failure. As here:

$$\begin{aligned} \epsilon x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}$$

The naïve algorithm solves this as

$$\begin{aligned} x_2 &= \frac{2 - \frac{1}{\epsilon}}{1 - \frac{1}{\epsilon}} = 1 - \frac{\epsilon}{1 - \epsilon} \\ x_1 &= \frac{1 - x_2}{\epsilon} = \frac{1}{1 - \epsilon} \end{aligned}$$

If ϵ is very small, then $\frac{1}{\epsilon}$ is enormous compared to both 1 and 2. With poor rounding, the algorithm solves x_2 as 1. Then it solves $x_1 = 0$. This is nearly correct for x_2, but is an awful approximation for x_1. Note that this choice of x_1, x_2 satisfies the first equation, but not the second.

Now suppose the algorithm changed the order of the equations, then solved:

$$\begin{aligned} x_1 + x_2 &= 2 \\ \epsilon x_1 + x_2 &= 1 \end{aligned}$$

The algorithm solves this as

$$\begin{aligned} x_2 &= \frac{1 - 2\epsilon}{1 - \epsilon} \\ x_1 &= 2 - x_2 \end{aligned}$$

There's no problem with rounding here.

The problem is not the small entry *per se*: Suppose we use an e.r.o. to scale the first equation, then use naïve G.E.:

$$\begin{aligned} x_1 + \frac{1}{\epsilon} x_2 &= \frac{1}{\epsilon} \\ x_1 + x_2 &= 2 \end{aligned}$$

This is still solved as

$$\begin{aligned} x_2 &= \frac{2 - \frac{1}{\epsilon}}{1 - \frac{1}{\epsilon}} \\ x_1 &= \frac{1 - x_2}{\epsilon}, \end{aligned}$$

and rounding is still a problem.

7.2.1 Scaled Partial Pivoting

The naïve G.E. algorithm uses the rows 1, 2, . . . , n-1 in order as pivot equations. As shown above, this can cause errors. Better is to pivot first on row ℓ_1, then row ℓ_2, etc, until finally pivoting on row ℓ_{n-1}, for some permutation $\{\ell_i\}_{i=1}^n$ of the integers 1, 2, . . . , n. The strategy of *scaled partial pivoting* is to compute this permutation so that G.E. works well.

In light of our example, we want to pivot on an element which is not small compared to other elements in its row. So our algorithm first determines "smallness" by calculating a scale, row-wise:

$$s_i = \max_{1 \le j \le n} |a_{ij}| .$$

The scales are only computed once.

Then the first pivot, ℓ_1, is chosen to be the i such that

$$\frac{|a_{i,1}|}{s_i}$$

is maximized. The algorithm pivots on row ℓ_1, producing a bunch of zeros in the first column. Note that the algorithm should *not* rearrange the matrix–this takes too much work.

The second pivot, ℓ_2, is chosen to be the i such that

$$\frac{|a_{i,2}|}{s_i}$$

is maximized, but without choosing $\ell_2 = \ell_1$. The algorithm pivots on row ℓ_2, producing a bunch of zeros in the second column.

In the k^{th} step ℓ_k is chosen to be the i not among $\ell_1, \ell_2, \ldots, \ell_{k-1}$ such that

$$\frac{|a_{i,k}|}{s_i}$$

is maximized. The algorithm pivots on row ℓ_k, producing a bunch of zeros in the k^{th} column.

The slick way to implement this is to first set $\ell_i = i$ for $i = 1, 2, \ldots, n$. Then rearrange this vector in a kind of "bubble sort": when you find the index that should be ℓ_1, swap them, *i.e.*, find the j such that ℓ_j should be the first pivot and switch the values of ℓ_1, ℓ_j.

Then at the k^{th} step, search only those indices in the tail of this vector: *i.e.*, only among ℓ_j for $k \le j \le n$, and perform a swap.

7.2.2 An Example

We present an example of using scaled partial pivoting with G.E. It's hard to come up with an example where the numbers do not come out as ugly fractions. We'll look at a homework question.

$$\left(\begin{array}{cccc|c} 2 & -1 & 3 & 7 & 15 \\ 4 & 4 & 0 & 7 & 11 \\ 2 & 1 & 1 & 3 & 7 \\ 6 & 5 & 4 & 17 & 31 \end{array}\right)$$

The scales are as follows: $s_1 = 7, s_2 = 7, s_3 = 3, s_4 = 17$.

We pick ℓ_1. It should be the index which maximizes $|a_{i1}|/s_i$. These values are:

$$\frac{2}{7}, \frac{4}{7}, \frac{2}{3}, \frac{6}{17}.$$

We pick $\ell_1 = 3$, and pivot:

$$\left(\begin{array}{cccc|c} 0 & -2 & 2 & 4 & 8 \\ 0 & 2 & -2 & 1 & -3 \\ 2 & 1 & 1 & 3 & 7 \\ 0 & 2 & 1 & 8 & 10 \end{array}\right)$$

We pick ℓ_2. It should *not* be 3, and should be the index which maximizes $|a_{i2}|/s_i$. These values are:

$$\frac{2}{7}, \frac{2}{7}, \frac{2}{17}.$$

We have a tie. In this case we pick the second row, *i.e.*, $\ell_2 = 2$. We pivot:

$$\left(\begin{array}{cccc|c} 0 & 0 & 0 & 5 & 5 \\ 0 & 2 & -2 & 1 & -3 \\ 2 & 1 & 1 & 3 & 7 \\ 0 & 0 & 3 & 7 & 13 \end{array}\right)$$

The matrix is in permuted upper triangular form. We could proceed, but would get a zero multiplier, and no changes would occur.

If we did proceed we would have $\ell_3 = 4$. Then $\ell_4 = 1$. Our row permutation is $3, 2, 4, 1$. When we do back substitution, we work in this order *reversed* on the rows, solving x_4, then x_3, x_2, x_1.

We get $x_4 = 1$, so

$$\begin{aligned} x_3 &= \frac{1}{3}(13 - 7 * 1) = 2 \\ x_2 &= \frac{1}{2}(-3 - 1 * 1 + 2 * 2) = 0 \\ x_1 &= \frac{1}{2}(7 - 3 * 1 - 1 * 2 - 1 * 0) = 1 \end{aligned}$$

7.2.3 Another Example and A Real Algorithm

Sometimes we want to solve

$$\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$$

for a number of different vectors $\boldsymbol{b}$. It turns out we can run G.E. on the matrix A alone and come up with all the multipliers, which can then be used multiple times on different vectors $\boldsymbol{b}$. We illustrate with an example:

$$\mathsf{M}_0 = \begin{pmatrix} 1 & 2 & 4 & 1 \\ 4 & 2 & 1 & 2 \\ 2 & 1 & 2 & 3 \\ 1 & 3 & 2 & 1 \end{pmatrix}, \quad \boldsymbol{\ell} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}.$$

The scale vector is $\boldsymbol{s} = \begin{bmatrix} 4 & 4 & 3 & 3 \end{bmatrix}^\top$.

Our scale choices are $\frac{1}{4}, \frac{4}{4}, \frac{2}{3}, \frac{1}{3}$. We choose $\ell_1 = 2$, and swap ℓ_1, ℓ_2. In the places where there would be zeros in the real matrix, we will put the multipliers. We will illustrate them here boxed:

$$\mathsf{M}_1 = \begin{pmatrix} \boxed{\frac{1}{4}} & \frac{3}{2} & \frac{15}{4} & \frac{1}{2} \\ 4 & 2 & 1 & 2 \\ \boxed{\frac{1}{2}} & 0 & \frac{3}{2} & 2 \\ \boxed{\frac{1}{4}} & \frac{5}{2} & \frac{7}{4} & \frac{1}{2} \end{pmatrix}, \quad \boldsymbol{\ell} = \begin{bmatrix} 2 \\ 1 \\ 3 \\ 4 \end{bmatrix}.$$

Our scale choices are $\frac{3}{8}, \frac{0}{3}, \frac{5}{6}$. We choose $\ell_2 = 4$, and so swap ℓ_2, ℓ_4:

$$\mathsf{M}_2 = \begin{pmatrix} \boxed{\frac{1}{4}} & \boxed{\frac{3}{5}} & \frac{27}{10} & \frac{1}{5} \\ 4 & 2 & 1 & 2 \\ \boxed{\frac{1}{2}} & \boxed{0} & \frac{3}{2} & 2 \\ \boxed{\frac{1}{4}} & \frac{5}{2} & \frac{7}{4} & \frac{1}{2} \end{pmatrix}, \quad \boldsymbol{\ell} = \begin{bmatrix} 2 \\ 4 \\ 3 \\ 1 \end{bmatrix}.$$

Our scale choices are $\frac{27}{40}, \frac{1}{2}$. We choose $\ell_3 = 1$, and so swap ℓ_3, ℓ_4:

$$\mathsf{M}_3 = \begin{pmatrix} \boxed{\frac{1}{4}} & \boxed{\frac{3}{5}} & \frac{27}{10} & \frac{1}{5} \\ 4 & 2 & 1 & 2 \\ \boxed{\frac{1}{2}} & \boxed{0} & \boxed{\frac{5}{9}} & \frac{17}{9} \\ \boxed{\frac{1}{4}} & \frac{5}{2} & \frac{7}{4} & \frac{1}{2} \end{pmatrix}, \quad \boldsymbol{\ell} = \begin{bmatrix} 2 \\ 4 \\ 1 \\ 3 \end{bmatrix}.$$

Now suppose we had to solve the linear system for $\boldsymbol{b} = \begin{bmatrix} -1 & 8 & 2 & 1 \end{bmatrix}^\top$.

We scale $\boldsymbol{b}$ by the multipliers in order: $\ell_1 = 2$, so, we sweep through the first column of M_3, picking off the boxed numbers (your computer doesn't really

have boxed variables), and scaling $\boldsymbol{b}$ appropriately:

$$\begin{bmatrix} -1 \\ 8 \\ 2 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} -3 \\ 8 \\ -2 \\ -1 \end{bmatrix}$$

This continues:

$$\begin{bmatrix} -3 \\ 8 \\ -2 \\ -1 \end{bmatrix} \Rightarrow \begin{bmatrix} -\frac{12}{5} \\ 8 \\ -2 \\ -1 \end{bmatrix} \Rightarrow \begin{bmatrix} -\frac{12}{5} \\ 8 \\ -\frac{2}{3} \\ -1 \end{bmatrix}$$

We then perform a permuted backwards substitution on the augmented system

$$\left(\begin{array}{cccc|c} 0 & 0 & \frac{27}{10} & \frac{1}{5} & -\frac{12}{5} \\ 4 & 2 & 1 & 2 & 8 \\ 0 & 0 & 0 & \frac{17}{9} & -\frac{2}{3} \\ 0 & \frac{5}{2} & \frac{7}{4} & \frac{1}{2} & -1 \end{array}\right)$$

This proceeds as

$$\begin{aligned} x_4 &= \frac{-2}{3}\frac{9}{17} = \frac{-6}{17} \\ x_3 &= \frac{10}{27}\left(-\frac{12}{5} - \frac{1}{5}\frac{-6}{17}\right) = \dots \\ x_2 &= \frac{2}{5}\left(-1 - \frac{1}{2}\frac{-6}{17} - \frac{7}{4}x_3\right) = \dots \\ x_1 &= \frac{1}{4}\left(8 - 2\frac{-6}{17} - x_3 - 2x_2\right) = \dots \end{aligned}$$

Fill in your own values here.

7.3 LU Factorization

We examined G.E. to solve the system

$$\mathsf{A}\boldsymbol{x} = \boldsymbol{b},$$

where A is a matrix:

$$\mathsf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}.$$

We want to show that G.E. actually factors A into lower and upper triangular parts, that is $\mathsf{A} = \mathsf{LU}$, where

$$\mathsf{L} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix}, \qquad \mathsf{U} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}.$$

We call this a *LU Factorization* of A.

7.3.1 An Example

We consider solution of the following augmented form:

$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & 3 & 7 \\ 4 & 4 & 0 & 7 & 11 \\ 6 & 5 & 4 & 17 & 31 \\ 2 & -1 & 0 & 7 & 15 \end{array}\right) \tag{7.2}$$

The naïve G.E. reduces this to

$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & 3 & 7 \\ 0 & 2 & -2 & 1 & -3 \\ 0 & 0 & 3 & 7 & 13 \\ 0 & 0 & 0 & 12 & 18 \end{array}\right)$$

We are going to run the naïve G.E., and see how it is a LU Factorization. Since this is the naïve version, we first pivot on the first row. Our multipliers are $2, 3, 1$. We pivot to get

$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & 3 & 7 \\ 0 & 2 & -2 & 1 & -3 \\ 0 & 2 & 1 & 8 & 10 \\ 0 & -2 & -1 & 4 & 8 \end{array}\right)$$

Careful inspection shows that we've merely multiplied A and $\boldsymbol{b}$ by a lower triangular matrix M_1:

$$\mathsf{M}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -3 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

The entries in the first column are the negative e.r.o. multipliers for each row. Thus after the first pivot, it is like we are solving the system

$$\mathsf{M}_1\mathsf{A}\boldsymbol{x} = \mathsf{M}_1\boldsymbol{b}.$$

We pivot on the second row to get:

$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & 3 & 7 \\ 0 & 2 & -2 & 1 & -3 \\ 0 & 0 & 3 & 7 & 13 \\ 0 & 0 & -3 & 5 & 5 \end{array}\right)$$

The multipliers are $1, -1$. We can view this pivot as a multiplication by M_2, with

$$\mathsf{M}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

We are now solving

$$\mathsf{M}_2\mathsf{M}_1\mathsf{A}\boldsymbol{x} = \mathsf{M}_2\mathsf{M}_1\boldsymbol{b}.$$

We pivot on the third row, with a multiplier of -1. Thus we get

$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & 3 & 7 \\ 0 & 2 & -2 & 1 & -3 \\ 0 & 0 & 3 & 7 & 13 \\ 0 & 0 & 0 & 12 & 18 \end{array}\right)$$

We have multiplied by M_3 :

$$\mathsf{M}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

We are now solving

$$\mathsf{M}_3\mathsf{M}_2\mathsf{M}_1\mathsf{A}\boldsymbol{x} = \mathsf{M}_3\mathsf{M}_2\mathsf{M}_1\boldsymbol{b}.$$

But we have an upper triangular form, that is, if we let

$$\mathsf{U} = \begin{bmatrix} 2 & 1 & 1 & 3 \\ 0 & 2 & -2 & 1 \\ 0 & 0 & 3 & 7 \\ 0 & 0 & 0 & 12 \end{bmatrix}$$

Then we have

$$\begin{aligned} \mathsf{M}_3\mathsf{M}_2\mathsf{M}_1\mathsf{A} &= \mathsf{U}, \\ \mathsf{A} &= (\mathsf{M}_3\mathsf{M}_2\mathsf{M}_1)^{-1}\,\mathsf{U}, \\ \mathsf{A} &= {\mathsf{M}_1}^{-1}{\mathsf{M}_2}^{-1}{\mathsf{M}_3}^{-1}\mathsf{U}, \\ \mathsf{A} &= \mathsf{L}\mathsf{U}. \end{aligned}$$

We are hoping that L is indeed lower triangular, and has ones on the diagonal. It turns out that the inverse of each M_i matrix has a nice form (See Exercise (7.6)). We write them here:

$$\begin{aligned}\mathsf{L} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix}\end{aligned}$$

This is really crazy: the matrix L looks to be composed of ones on the diagonal and multipliers under the diagonal.

Now we check to see if we made any mistakes:

$$\begin{aligned}\mathsf{LU} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 3 \\ 0 & 2 & -2 & 1 \\ 0 & 0 & 3 & 7 \\ 0 & 0 & 0 & 12 \end{bmatrix} \\ &= \begin{bmatrix} 2 & 1 & 1 & 3 \\ 4 & 4 & 0 & 7 \\ 6 & 5 & 4 & 17 \\ 2 & -1 & 0 & 7 \end{bmatrix} = \mathsf{A}.\end{aligned}$$

7.3.2 Using LU Factorizations

We see that the G.E. algorithm can be used to actually calculate the LU factorization. We will look at this in more detail in another example. We now examine how we can use the LU factorization to solve the equation

$$\mathsf{A}\boldsymbol{x} = \boldsymbol{b},$$

Since we have $\mathsf{A} = \mathsf{LU}$, we first solve

$$\mathsf{L}\boldsymbol{z} = \boldsymbol{b},$$

then solve

$$\mathsf{U}\boldsymbol{x} = \boldsymbol{z}.$$

Since L is lower triangular, we can solve for $\boldsymbol{z}$ with a *forward* substitution. Similarly, since U is upper triangular, we can solve for $\boldsymbol{x}$ with a back substitution. We drag out the previous example (which we never got around to solving):

$$\left(\begin{array}{cccc|c} 2 & 1 & 1 & 3 & 7 \\ 4 & 4 & 0 & 7 & 11 \\ 6 & 5 & 4 & 17 & 31 \\ 2 & -1 & 0 & 7 & 15 \end{array}\right)$$

We had found the LU factorization of A as

$$\mathsf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 3 \\ 0 & 2 & -2 & 1 \\ 0 & 0 & 3 & 7 \\ 0 & 0 & 0 & 12 \end{bmatrix}$$

So we solve

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix} \boldsymbol{z} = \begin{bmatrix} 7 \\ 11 \\ 31 \\ 15 \end{bmatrix}$$

We get

$$\boldsymbol{z} = \begin{bmatrix} 7 \\ -3 \\ 13 \\ 18 \end{bmatrix}$$

Now we solve

$$\begin{bmatrix} 2 & 1 & 1 & 3 \\ 0 & 2 & -2 & 1 \\ 0 & 0 & 3 & 7 \\ 0 & 0 & 0 & 12 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 7 \\ -3 \\ 13 \\ 18 \end{bmatrix}$$

We get the ugly solution

$$\boldsymbol{z} = \begin{bmatrix} \frac{37}{24} \\ \frac{-17}{12} \\ \frac{5}{6} \\ \frac{3}{2} \end{bmatrix}$$

7.3.3 Some Theory

We aren't doing much proving here. The following theorem has an ugly proof in the Cheney & Kincaid [**?**].

Theorem 7.3. *If* A *is an* $n \times n$ *matrix, and naïve Gaussian Elimination does not encounter a zero pivot, then the algorithm generates a LU factorization of* A*, where* L *is the lower triangular part of the output matrix, and* U *is the upper triangular part.*

This theorem relies on us using the fancy version of G.E., which saves the multipliers in the spots where there should be zeros. If correctly implemented, then, L is the lower triangular part but with ones put on the diagonal.

This theorem is proved in Cheney & Kincaid [**?**]. This appears to me to be a case of something which can be better illustrated with an example or two and some informal investigation. The proof is an unillustrating index-chase–read it at your own risk.

7.3.4 Computing Inverses

We consider finding the inverse of A. Since

$$\mathsf{A}\mathsf{A}^{-1} = \mathsf{I},$$

then the j^{th} column of the inverse A^{-1} solves the equation

$$\mathsf{A}\boldsymbol{x} = \boldsymbol{e_j},$$

where $\boldsymbol{e_j}$ is the column matrix of all zeros, but with a one in the j^{th} position.

Thus we can find the inverse of A by running n linear solves. Obviously we are only going to run G.E. once, to put the matrix in LU form, then run n solves using forward and backward substitutions.

7.4 Iterative Solutions

Recall we are trying to solve

$$\mathsf{A}\boldsymbol{x} = \boldsymbol{b}.$$

We examine the computational cost of Gaussian Elimination to motivate the search for an alternative algorithm.

7.4.1 An Operation Count for Gaussian Elimination

We consider the number of floating point operations ("flops") required to solve the system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$. Gaussian Elimnation first uses row operations to transform the problem into an equivalent problem of the form $\mathsf{U}\boldsymbol{x} = \boldsymbol{b'}$, where U is upper triangular. Then back substitution is used to solve for $\boldsymbol{x}$.

First we look at how many floating point operations are required to reduce

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array}\right)$$

to

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & a'_{32} & a'_{33} & \cdots & a'_{3n} & b'_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & a'_{n2} & a'_{n3} & \cdots & a'_{nn} & b'_n \end{array}\right)$$

First a multiplier is computed for each row. Then in each row the algorithm performs n multiplies and n adds. This gives a total of $(n-1) + (n-1)n$ multiplies (counting in the computing of the multiplier in each of the $(n-1)$

rows) and $(n-1)n$ adds. In total this is $2n^2 - n - 1$ floating point operations to do a single pivot on the n by n system.

Then this has to be done recursively on the lower right subsystem, which is an $(n-1)$ by $(n-1)$ system. This requires $2(n-1)^2 - (n-1) - 1$ operations. Then this has to be done on the next subsystem, requiring $2(n-2)^2 - (n-2) - 1$ operations, and so on.

In total, then, we use I_n total floating point operations, with

$$I_n = 2\sum_{j=1}^{n} j^2 - \sum_{j=1}^{n} j - \sum_{j=1}^{n} 1.$$

Recalling that

$$\sum_{j=1}^{n} j^2 = \frac{1}{6}(n)(n+1)(2n+1), \quad \text{and} \quad \sum_{j=1}^{n} j = \frac{1}{2}(n)(n+1),$$

We find that

$$I_n = \frac{1}{6}(4n-1)n(n+1) - n \approx \frac{2}{3}n^3.$$

Now consider the costs of back substitution. To solve

$$\left(\begin{array}{ccccc|c} a_{11} & \cdots & a_{1,n-2} & a_{1,n-1} & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & a_{n-2,n-2} & a_{n-2,n-1} & a_{n-2,n} & b_{n-2} \\ 0 & \cdots & 0 & a_{n-1,n-1} & a_{n-1,n} & b_{n-1} \\ 0 & \cdots & 0 & 0 & a_{nn} & b_n \end{array}\right)$$

for x_n requires only a single division. Then to solve for x_{n-1} we compute

$$x_{n-1} = \frac{1}{a_{n-1,n-1}}\left[b_{n-1} - a_{n-1,n}x_n\right],$$

and requires 3 flops. Similarly, solving for x_{n-2} requires 5 flops. Thus in total back substitution requires B_n total floating point operations with

$$B_n = \sum_{j=1}^{n} 2j - 1 = n(n-1) - n = n(n-2) \approx n^2$$

7.4.2 Dividing by Multiplying

We saw that Gaussian Elimination requires around $\frac{2}{3}n^3$ operations just to find the LU factorization, then about n^2 operations to solve the system, when A is $n \times n$. When n is large, this may take too long to be practical. Additionally, if A is sparse (has few nonzero elements per row), we would like the complexity of our computations to scale with the sparsity of A. Thus we look for an alternative algorithm.

First we consider the simplest case, $n = 1$. Suppose we are to solve the equation

$$Ax = b.$$

for scalars A, b. We solve this by

$$x = \frac{1}{A}b = \frac{1}{\omega A}\omega b = \frac{1}{1-(1-\omega A)}\omega b = \frac{1}{1-r}\omega b,$$

where $\omega \neq 0$ is some real number chosen to "weight" the problem appropriately, and $r = 1 - \omega A$. Now suppose that ω is chosen such that $|r| < 1$. This can be done so long as $A \neq 0$, which would have been a problem anyway. Now use the geometric expansion:

$$\frac{1}{1-r} = 1 + r + r^2 + r^3 + \ldots$$

Because of the assumption $|r| < 1$, the terms r^n converge to zero as $n \to \infty$. This gives the approximate solution to our one dimensional problem as

$$\begin{aligned} x &\approx \left[1 + r + r^2 + r^3 + \ldots + r^k\right] \omega b \\ &= \omega b + \left[r + r^2 + r^3 + \ldots + r^k\right] \omega b \\ &= \omega b + r\left[1 + r + r^2 + \ldots + r^{k-1}\right] \omega b \end{aligned}$$

This suggests an *iterative* approach to solving $Ax = b$. First let $x^{(0)} = \omega b$, then let

$$x^{(k)} = \omega b + r x^{(k-1)}.$$

The iterates $x^{(k)}$ will converge to the solution of $Ax = b$ if $|r| < 1$.

You should now convince yourself that because $r^n \to 0$, that the choice of the initial iterate $x^{(0)}$ was immaterial, *i.e.*, that under *any* choice of initial iterate convergence is guaranteed.

We now translate this scalar result into the vector case. The algorithm proceeds as follows: first fix some initial estimate of the solution, $\boldsymbol{x}^{(0)}$. A good choice might be $\omega\boldsymbol{b}$, but this is not necessary. Then calculate successive approximations to the actual solution by updates of the form

$$\boldsymbol{x}^{(k)} = \omega\boldsymbol{b} + (\mathsf{I} - \omega\mathsf{A})\,\boldsymbol{x}^{(k-1)}.$$

It turns out that we can consider a slightly more general form of the algorithm, one in which successive iterates are defined *implicitly*. That is we consider iterates of the form

$$\boxed{\mathsf{Q}\boldsymbol{x}^{(k+1)} = (\mathsf{Q} - \omega\mathsf{A})\,\boldsymbol{x}^{(k)} + \omega\boldsymbol{b},} \tag{7.3}$$

for some matrix Q, and some scaling factor ω. Note that this update relies on vector additions and possibly by premultiplication of a vector by A or Q. In the case where these two matrices are sparse, such an update can be relatively cheap.

Now suppose that as $k \to \infty$, $\boldsymbol{x}^{(k)}$ converges to some vector $\boldsymbol{x}^*$, which is a fixed point of the iteration. Then

$$\begin{aligned} \mathsf{Q}\boldsymbol{x}^* &= (\mathsf{Q} - \omega\mathsf{A})\,\boldsymbol{x}^* + \omega\boldsymbol{b}, \\ \mathsf{Q}\boldsymbol{x}^* &= \mathsf{Q}\boldsymbol{x}^* - \omega\mathsf{A}\boldsymbol{x}^* + \omega\boldsymbol{b}, \\ \omega\mathsf{A}\boldsymbol{x}^* &= \omega\boldsymbol{b}, \\ \mathsf{A}\boldsymbol{x}^* &= \boldsymbol{b}. \end{aligned}$$

We have some freedom in choosing Q, but there are two considerations we should keep in mind:

1. Choice of Q affects convergence and speed of convergence of the method. In particular, we want Q to be similar to A.
2. Choice of Q affects ease of computing the update. That is, given

   $$\boldsymbol{z} = (\mathsf{Q} - \mathsf{A})\,\boldsymbol{x}^{(k)} + \boldsymbol{b},$$

   we should pick Q such that the equation

   $$\mathsf{Q}\boldsymbol{x}^{(k+1)} = \boldsymbol{z}$$

   is easy to solve exactly.

These two goals conflict with each other. At one end of the spectrum is the so-called "impossible iteration," at the other is the Richardsons.

7.4.3 Impossible Iteration

I made up the term "impossible iteration." But consider the method which takes Q to be A. This seems to be the best choice for satisfying the first goal. Letting $\omega = 1$, our method becomes

$$\mathsf{A}\boldsymbol{x}^{(k+1)} = (\mathsf{A} - \mathsf{A})\,\boldsymbol{x}^{(k)} + \boldsymbol{b} = \boldsymbol{b}.$$

This method should clearly converge in one step. However, the second goal is totally ignored. Indeed, we are considering iterative methods because we cannot easily solve this linear equation in the first place.

7.4.4 Richardson Iteration

At the other end of the spectrum is the Richardson Iteration, which chooses Q to be the identity matrix. Solving the system

$$\mathsf{Q}\boldsymbol{x}^{(k+1)} = \boldsymbol{z}$$

is trivial: we just have $\boldsymbol{x}^{(k+1)} = \boldsymbol{z}$.

Example Problem 7.4. *Use Richardson Iteration with $\omega = 1$ on the system*

$$\mathsf{A} = \begin{bmatrix} 6 & 1 & 1 \\ 2 & 4 & 0 \\ 1 & 2 & 6 \end{bmatrix}, \qquad \mathsf{b} = \begin{bmatrix} 12 \\ 0 \\ 6 \end{bmatrix}.$$

Solution: *We let*

$$\mathsf{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad (\mathsf{Q}-\mathsf{A}) = \begin{bmatrix} -5 & -1 & -1 \\ -2 & -3 & 0 \\ -1 & -2 & -5 \end{bmatrix}.$$

We start with an arbitrary $\boldsymbol{x}^{(0)}$*, say* $\boldsymbol{x}^{(0)} = [2\ 2\ 2]^\top$*. We get* $\boldsymbol{x}^{(1)} = [-2\ -10\ -10]^\top$*, and* $\boldsymbol{x}^{(2)} = [42\ 34\ 78]^\top$*.*

Note the real solution is $\boldsymbol{x} = [2\ -1\ 1]^\top$*. The Richardson Iteration does not appear to converge for this example, unfortunately.* ⊣

Example Problem 7.5. *Apply Richardson Iteration with* $\omega = 1/6$ *on the previous system.* Solution: *Our iteration becomes*

$$\boldsymbol{x}^{(k+1)} = \begin{bmatrix} 0 & -1/6 & -1/6 \\ -1/3 & 1/3 & 0 \\ -1/6 & -1/3 & 0 \end{bmatrix} \boldsymbol{x}^{(k)} + \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}.$$

We start with the same $\boldsymbol{x}^{(0)}$ *as previously,* $\boldsymbol{x}^{(0)} = [2\ 2\ 2]^\top$*. We get* $\boldsymbol{x}^{(1)} = [4/3\ 0\ 0]^\top$*,* $\boldsymbol{x}^{(2)} = [2\ -4/9\ 7/9]^\top$*, and finally* $\boldsymbol{x}^{(12)} = [2\ -0.99998\ 0.99998]^\top$*.*

Thus, the choice of ω *has some affect on convergence.* ⊣

We can rethink the Richardson Iteration as

$$\boldsymbol{x}^{(k+1)} = (\mathsf{I} - \omega\mathsf{A})\,\boldsymbol{x}^{(k)} + \omega\boldsymbol{b} = \boldsymbol{x}^{(k)} + \omega\left(\boldsymbol{b} - \mathsf{A}\boldsymbol{x}^{(k)}\right).$$

Thus at each step we are adding some scaled version of the *residual*, defined as $\boldsymbol{b} - \mathsf{A}\boldsymbol{x}^{(k)}$, to the iterate.

7.4.5 Jacobi Iteration

The Jacobi Iteration chooses Q to be the matrix consisting of the diagonal of A. This is more similar to A than the identity matrix, but nearly as simple to invert.

Example Problem 7.6. *Use Jacobi Iteration, with* $\omega = 1$*, to solve the system*

$$\mathsf{A} = \begin{bmatrix} 6 & 1 & 1 \\ 2 & 4 & 0 \\ 1 & 2 & 6 \end{bmatrix}, \qquad \mathsf{b} = \begin{bmatrix} 12 \\ 0 \\ 6 \end{bmatrix}.$$

Solution: *We let*

$$\mathsf{Q} = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}, \qquad (\mathsf{Q}-\mathsf{A}) = \begin{bmatrix} 0 & -1 & -1 \\ -2 & 0 & 0 \\ -1 & -2 & 0 \end{bmatrix}, \qquad \mathsf{Q}^{-1} = \begin{bmatrix} \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{6} \end{bmatrix}.$$

We start with an arbitrary $\boldsymbol{x}^{(0)}$*, say* $\boldsymbol{x}^{(0)} = [2\ 2\ 2]^\top$*. We get* $\boldsymbol{x}^{(1)} = \left[\frac{4}{3}\ -1\ 0\right]^\top$*. Then* $\boldsymbol{x}^{(2)} = \left[\frac{13}{6}\ -\frac{2}{3}\ \frac{10}{9}\right]^\top$*. Continuing, we find that* $\boldsymbol{x}^{(5)} \approx [1.987\ -1.019\ 0.981]^\top$*.*

Note the real solution is $\boldsymbol{x} = [2\ -1\ 1]^\top$*.* ⊣

There is an alternative way to describe the Jacobi Iteration for $\omega = 1$. By considering the update elementwise, we see that the operation can be described by

$$\boldsymbol{x}_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{i=1, i \neq j}^{n} a_{ji} \boldsymbol{x}_i^{(k)} \right).$$

Thus an update takes less than $2n^2$ operations. In fact, if A is sparse, with less than k nonzero entries per row, the update should take less than $2nk$ operations.

7.4.6 Gauss Seidel Iteration

The Gauss Seidel Iteration chooses Q to be lower triangular part of A, including the diagonal. In this case solving the system

$$\mathsf{Q}\boldsymbol{x}^{(k+1)} = \boldsymbol{z}$$

is performed by forward substitution. Here the Q is more like A than for Jacobi Iteration, but involves more work for inverting.

Example Problem 7.7. *Use Gauss Seidel Iteration to again solve for*

$$\mathsf{A} = \begin{bmatrix} 6 & 1 & 1 \\ 2 & 4 & 0 \\ 1 & 2 & 6 \end{bmatrix}, \qquad \mathsf{b} = \begin{bmatrix} 12 \\ 0 \\ 6 \end{bmatrix}.$$

Solution: *We let*

$$\mathsf{Q} = \begin{bmatrix} 6 & 0 & 0 \\ 2 & 4 & 0 \\ 1 & 2 & 6 \end{bmatrix}, \qquad (\mathsf{Q} - \mathsf{A}) = \begin{bmatrix} 0 & -1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

We start with an arbitrary $\boldsymbol{x}^{(0)}$*, say* $\boldsymbol{x}^{(0)} = [2\,2\,2]^\top$*. We get* $\boldsymbol{x}^{(1)} = \left[\frac{4}{3}\ -\frac{2}{3}\ 1\right]^\top$*. Then* $\boldsymbol{x}^{(2)} = \left[\frac{35}{18}\ -\frac{35}{36}\ 1\right]^\top$*.*

Already this is fairly close to the actual solution $\boldsymbol{x} = [2\ -1\ 1]^\top$*.* ⊣

Just as with Jacobi Iteration, there is an easier way to describe the Gauss Seidel Iteration. In this case we will keep a single vector $\boldsymbol{x}$ and overwrite it, element by element. Thus for $j = 1, 2, \ldots, n$, we set

$$x_j \leftarrow \frac{1}{a_{jj}} \left(b_j - \sum_{i=1, i \neq j}^{n} a_{ji} x_i \right).$$

This looks exactly like the Jacobi update. However, in the sum on the right there are some "old" values of x_i and some "new" values; the new values are those x_i for which $i < j$.

Again this takes less than $2n^2$ operations. Or less than $2nk$ if A is sufficiently sparse.

An alteration of the Gauss Seidel Iteration is to make successive "sweeps" of this redefinition, one for $j = 1, 2, \ldots, n$, the next for $j = n, n-1, \ldots, 2, 1$. This amounts to running Gauss Seidel once with Q the lower triangular part of A, then running it with Q the upper triangular part. This iterative method is known as "red-black Gauss Seidel."

7.4.7 Error Analysis

Suppose that $\boldsymbol{x}$ is the solution to equation 7.4. Define the error vector:

$$\boldsymbol{e}^{(k)} = \boldsymbol{x}^{(k)} - \boldsymbol{x}.$$

Now notice that

$$\begin{aligned}
\boldsymbol{x}^{(k+1)} &= \mathsf{Q}^{-1}\left(\mathsf{Q} - \omega\mathsf{A}\right)\boldsymbol{x}^{(k)} + \mathsf{Q}^{-1}\omega\boldsymbol{b}, \\
\boldsymbol{x}^{(k+1)} &= \mathsf{Q}^{-1}\mathsf{Q}\boldsymbol{x}^{(k)} - \omega\mathsf{Q}^{-1}\mathsf{A}\boldsymbol{x}^{(k)} + \omega\mathsf{Q}^{-1}\mathsf{A}\boldsymbol{x}, \\
\boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} - \omega\mathsf{Q}^{-1}\mathsf{A}\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}\right), \\
\boldsymbol{x}^{(k+1)} - \boldsymbol{x} &= \boldsymbol{x}^{(k)} - \boldsymbol{x} - \omega\mathsf{Q}^{-1}\mathsf{A}\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}\right), \\
\boldsymbol{e}^{(k+1)} &= \boldsymbol{e}^{(k)} - \omega\mathsf{Q}^{-1}\mathsf{A}\boldsymbol{e}^{(k)}, \\
\boldsymbol{e}^{(k+1)} &= \left(\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right)\boldsymbol{e}^{(k)}.
\end{aligned}$$

Reusing this relation we find that

$$\begin{aligned}
\boldsymbol{e}^{(k)} &= \left(\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right)\boldsymbol{e}^{(k-1)}, \\
&= \left(\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right)^2\boldsymbol{e}^{(k-2)}, \\
&= \left(\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right)^k\boldsymbol{e}^{(0)}.
\end{aligned}$$

We want to ensure that $\boldsymbol{e}^{(k+1)}$ is "smaller" than $\boldsymbol{e}^{(k)}$. To do this we recall matrix and vector norms from Subsection **??**.

$$\left\|\boldsymbol{e}^{(k)}\right\|_2 = \left\|\left(\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right)^k\boldsymbol{e}^{(0)}\right\|_2 \leq \left\|\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right\|_2^k\left\|\boldsymbol{e}^{(0)}\right\|_2.$$

(See Example Problem **??**.)

Thus our iteration converges ($\boldsymbol{e}^{(k)}$ goes to the zero vector, *i.e.*, $\boldsymbol{x}^{(k)} \to \boldsymbol{x}$) if

$$\left\|\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right\|_2 < 1.$$

This gives the theorem:

Theorem 7.8. *An iterative solution scheme converges for any starting* $\boldsymbol{x}^{(0)}$ *if and only if all eigenvalues of* $\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}$ *are less than* 1 *in absolute value,* i.e.*, if and only if*

$$\left\|\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right\|_2 < 1$$

Another way of saying this is "the spectral radius of $\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}$ is less than 1."

In fact, the *speed* of convergence is decided by the spectral radius of the matrix–convergence is faster for smaller values. Recall our introduction to iterative methods in the scalar case, where the result relied on ω being chosen such that $|1 - \omega A| < 1$. You should now think about how eigenvalues generalize the absolute value of a scalar, and how this relates to the norm of matrices.

Let $\boldsymbol{y}$ be an eigenvector for $\mathsf{Q}^{-1}\mathsf{A}$, with corresponding eigenvalue λ. Then

$$\left(\mathsf{I} - \omega\mathsf{Q}^{-1}\mathsf{A}\right)\boldsymbol{y} = \boldsymbol{y} - \omega\mathsf{Q}^{-1}\mathsf{A}\boldsymbol{y} = \boldsymbol{y} - \omega\lambda\boldsymbol{y} = (1 - \omega\lambda)\,\boldsymbol{y}.$$

This relation may allow us to pick the optimal ω for given A,Q. It can also show us that sometimes no choice of ω will give convergence of the method. There are a number of different related results that show when various methods will work for certain choices of ω. We leave these to the exercises.

Example Problem 7.9. *Find conditions on ω which guarantee convergence of Richardson's Iteration for finding approximate iterative solutions to the system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$, where*

$$\mathsf{A} = \begin{bmatrix} 6 & 1 & 1 \\ 2 & 4 & 0 \\ 1 & 2 & 6 \end{bmatrix}, \qquad \mathsf{b} = \begin{bmatrix} 12 \\ 0 \\ 6 \end{bmatrix}.$$

Solution: *By Theorem 7.8, with Q the identity matrix, we have convergence if and only if*

$$\|\mathsf{I} - \omega\mathsf{A}\|_2 < 1$$

We now use the fact that "eigenvalues commute with polynomials;" that is if $f(x)$ is a polynomial and λ is an eigenvalue of a matrix A, then $f(\lambda)$ is an eigenvalue of the matrix $f(\mathsf{A})$. In this case the polynomial we consider is $f(x) = x^0 - \omega x^1$. The eigenvalues of A are approximately $7.7321, 4.2679$, and 4. Thus the eigenvalues of $\mathsf{I} - \omega\mathsf{A}$ are approximately

$$1 - 7.7321\omega,\ 1 - 4.2679\omega,\ 1 - 4\omega.$$

With some work it can be shown that all three of these values will be less than one in absolute value if and only if

$$0 < \omega < \frac{3}{7.7321} \approx 0.388$$

(See also Exercise (7.10).)

Compare this to the results of Example Problem 7.4, where for this system, $\omega = 1$ apparently did not lead to convergence, while for Example Problem 7.5, with $\omega = 1/6$, convergence was observed.

⊣

7.4.8 A Free Lunch?

The analysis leading to Theorem 7.8 leads to an interesting possible variant of the iterative scheme. For simplicity we will only consider an alteration of Richardson's Iteration. In the altered algorithm we presuppose the existence, via some oracle, of a sequence of weightings, ω_i, which we use in each iterative update. Thus our algorithm becomes:

1. Select some initial iterate $\boldsymbol{x}^{(0)}$.
2. Given iterate $\boldsymbol{x}^{(k-1)}$, define

$$\boldsymbol{x}^{(k)} = (\mathsf{I} - \omega_k \mathsf{A})\, \boldsymbol{x}^{(k-1)} + \omega_k \boldsymbol{b}.$$

Following the analysis for Theorem 7.8, it can be shown that

$$\boldsymbol{e}^{(k)} = (\mathsf{I} - \omega_k \mathsf{A})\, \boldsymbol{e}^{(k-1)}$$

where, again, $\boldsymbol{e}^{(k)} = \boldsymbol{x}^{(k)} - x$, with x the actual solution to the linear system. Expanding $\boldsymbol{e}^{(k-1)}$ similarly gives

Remember that we want $\boldsymbol{e}^{(k)}$ to be small in magnitude, or, better yet, to be zero. One way to guarantee that $\boldsymbol{e}^{(k)}$ is zero, regardless of the choice of $\boldsymbol{e}^{(0)}$ it to somehow ensure that the matrix

$$\mathsf{B} = \prod_{i=1}^{k} \mathsf{I} - \omega_i \mathsf{A}$$

has all zero eigenvalues.

We again make use of the fact that eigenvalues "commute" with polynomials to claim that if λ_j is an eigenvalue of A, then

$$\prod_{i=1}^{k} 1 - \omega_i \lambda_j$$

is an eigenvalue of B. This eigenvalue is zero if one of the ω_i for $1 \leq i \leq k$ is $1/\lambda_j$. This suggests how we are to pick the weightings: let them be the inverses of the eigenvalues of A.

In fact, if A has a small number of distinct eigenvalues, say m eigenvalues, then convergence to the exact solution could be guaranteed after only m iterations, regardless of the *size* of the matrix.

As you may have guessed from the title of this subsection, this is not exactly a practical algorithm. The problem is that it is not simple to find, for a given arbitrary matrix A, one, some, or all its eigenvalues. This problem is of sufficient complexity to outweigh any savings to be gotten from our "free lunch" algorithm.

However, in some limited situations this algorithm might be practical if the eigenvalues of A are known *a priori*.

Exercises

(7.1) Find the LU decomposition of the following matrices, using naïve Gaussian Elimination (a) $\begin{bmatrix} 3 & -1 & -2 \\ 9 & -1 & -4 \\ -6 & 10 & 13 \end{bmatrix}$ (b) $\begin{bmatrix} 8 & 24 & 16 \\ 1 & 12 & 11 \\ 4 & 13 & 19 \end{bmatrix}$ (c) $\begin{bmatrix} -3 & 6 & 0 \\ 1 & -2 & 0 \\ -4 & 5 & -8 \end{bmatrix}$

(7.2) Perform back substitution to solve the equation

$$\begin{bmatrix} 1 & 3 & 5 & 3 \\ 0 & 2 & 4 & 3 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 2 \end{bmatrix}$$

(7.3) Perform Naïve Gaussian Elimination to prove Cramer's rule for the 2D case. That is, prove that the solution to

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}$$

is given by

$$y = \frac{\det \begin{bmatrix} a & f \\ c & g \end{bmatrix}}{\det \begin{bmatrix} a & b \\ c & d \end{bmatrix}} \quad \text{and} \quad x = \frac{\det \begin{bmatrix} f & b \\ g & d \end{bmatrix}}{\det \begin{bmatrix} a & b \\ c & d \end{bmatrix}}$$

(7.4) Implement Cramer's rule to solve a pair of linear equations in 2 variables. Your m-file should have header line like:

```
function x = cramer2(A,b)
```

where `A` is a 2×2 matrix, and `b` and `x` are 2×1 vectors. Your code should find the `x` such that `Ax` = `b`. (See Exercise (7.3))

Test your code on the following (augmented) systems:

(a) $\left(\begin{array}{cc|c} 3 & -2 & 1 \\ 4 & -3 & -1 \end{array}\right)$ (b) $\left(\begin{array}{cc|c} 1.24 & -3.48 & 1 \\ -0.744 & 2.088 & 2 \end{array}\right)$ (c) $\left(\begin{array}{cc|c} 1.24 & -3.48 & 1 \\ -0.744 & 2.088 & -0.6 \end{array}\right)$

(d) $\left(\begin{array}{cc|c} -0.0590 & 0.2372 & -0.3528 \\ 0.1080 & -0.4348 & 0.6452 \end{array}\right)$

(7.5) Given two lines parametrized by $\boldsymbol{f}(t) = \boldsymbol{a}t + \boldsymbol{b}$, and $\boldsymbol{g}(s) = \boldsymbol{c}s + \boldsymbol{d}$, set up a linear 2×2 system of equations to find the t, s at the point of intersection of the two lines. If you were going to write a program to detect the intersection of two lines, how would you detect whether they are parallel? How is this related to the form of the solution to a system of two linear equations? (See Exercise (7.3))

(7.6) Prove that the inverse of the matrix

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ a_2 & 1 & 0 & \cdots & 0 \\ a_3 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_n & 0 & 0 & \cdots & 1 \end{bmatrix}$$

is the matrix

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -a_2 & 1 & 0 & \cdots & 0 \\ -a_3 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_n & 0 & 0 & \cdots & 1 \end{bmatrix}$$

(*Hint:* Multiply them together.)

(7.7) Under the strategy of scaled partial pivoting, which row of the following matrix will be the first pivot row?

$$\begin{bmatrix} 10 & 17 & -10 & 0.1 & 0.9 \\ -3 & 3 & -3 & 0.3 & -4 \\ 0.3 & 0.1 & 0.01 & -1 & 0.5 \\ 2 & 3 & 4 & -3 & 5 \\ 10 & 100 & 1 & 0.1 & 0 \end{bmatrix}$$

(7.8) Let A be a symmetric positive definite $n \times n$ matrix with n distinct eigenvalues. Letting $\boldsymbol{y}^{(0)} = \boldsymbol{b}/\left\|\boldsymbol{b}\right\|_2$, consider the iteration

$$\boldsymbol{y}^{(k+1)} = \frac{\mathsf{A}\boldsymbol{y}^{(k)}}{\left\|\mathsf{A}\boldsymbol{y}^{(k)}\right\|_2}.$$

(a) What is $\left\|\boldsymbol{y}^{(k)}\right\|_2$?

(b) Show that $\boldsymbol{y}^{(k)} = \mathsf{A}^k\boldsymbol{b}/\left\|\mathsf{A}^k\boldsymbol{b}\right\|_2$.

(c) Show that as $k \to \infty$, $\boldsymbol{y}^{(k)}$ converges to the (normalized) eigenvector associated with the largest eigenvalue of A.

(7.9) Consider the equation

$$\begin{bmatrix} 1 & 3 & 5 \\ -2 & 2 & 4 \\ 4 & -3 & -4 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} -5 \\ -6 \\ 10 \end{bmatrix}$$

Letting $\boldsymbol{x}^{(0)} = [1\ 1\ 0]^\top$, find the iterate $\boldsymbol{x}^{(1)}$ by one step of Richardson's Method. And by one step of Jacobi Iteration. And by Gauss Seidel.

(7.10) Let A be a symmetric $n \times n$ matrix with eigenvalues in the interval $[\alpha, \beta]$, with $0 < \alpha \leq \beta$, and $\alpha + \beta \neq 0$. Consider Richardson's Iteration

$$\boldsymbol{x}^{(k+1)} = (\mathsf{I} - \omega\mathsf{A})\,\boldsymbol{x}^{(k)} + \omega\boldsymbol{b}.$$

Recall that $\boldsymbol{e}^{(k+1)} = (\mathsf{I} - \omega\mathsf{A})\,\boldsymbol{e}^{(k)}$.

(a) Show that the eigenvalues of $\mathsf{I} - \omega\mathsf{A}$ are in the interval $[1 - \omega\beta, 1 - \omega\alpha]$.

(b) Prove that

$$\max\{|\lambda| : 1 - \omega\beta \leq \lambda \leq 1 - \omega\alpha\}$$

is minimized when we choose ω such that $1 - \omega\beta = -(1 - \omega\alpha)$. (*Hint:* It may help to look at the graph of something versus ω.)

(c) Show that this relationship is satisfied by $\omega = 2/(\alpha + \beta)$.
(d) For this choice of ω show that the spectral radius of $\mathsf{I} - \omega\mathsf{A}$ is

$$\frac{|\alpha - \beta|}{|\alpha + \beta|}.$$

(e) Show that when $0 < \alpha$, this quantity is always smaller than 1.
(f) Prove that if A is positive definite, then there is an ω such that Richardson's Iteration with this ω will converge for any choice of $\boldsymbol{x}^{(0)}$.
(g) For which matrix do you expect faster convergence of Richardson's Iteration: $\mathsf{A_1}$ with eigenvalues in $[10, 20]$ or $\mathsf{A_2}$ with eigenvalues in $[1010, 1020]$? Why?

(7.11) Implement Richardson's Iteration to solve the system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$. Your m-file should have header line like:
`function xk = richardsons(A,b,x0,w,k)`
Your code should return $\boldsymbol{x}^{(k)}$ based on the iteration

$$\boldsymbol{x}^{(j+1)} = \boldsymbol{x}^{(j)} - \omega\left(\mathsf{A}\boldsymbol{x}^{(j)} - \boldsymbol{b}\right).$$

Let `w` take the place of ω, and let `x0` be the initial iterate $\boldsymbol{x}^{(0)}$. Test your code for $\mathsf{A}, \boldsymbol{b}$ for which you know the actual solution to the problem. (*Hint:* Start with A and the solution $\boldsymbol{x}$ and *generate* $\boldsymbol{b}$.) Test your code on the following matrices:

- Let A be the *Hilbert Matrix*.
  Try different values of ω, including $\omega = 1$.
- Let A be a *Toeplitz* matrix of the form:

$$\mathsf{A} = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -2 \end{bmatrix}$$

  Try different values of ω, including $\omega = -1/2$.

(7.12) Let A be a nonsingular $n \times n$ matrix. We wish to solve $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$. Let $\boldsymbol{x}^{(0)}$ be some starting vector, let $\mathcal{D}_k$ be span $\left\{\boldsymbol{r}^{(0)}, \mathsf{A}\boldsymbol{r}^{(0)}, \ldots, \mathsf{A}^k\boldsymbol{r}^{(0)}\right\}$, and let $\mathcal{P}_k$ be the set of polynomials, $p(x)$ of degree k with $p(0) = 1$.
Consider the following iterative method: Let $\boldsymbol{x}^{(k+1)}$ be the $\boldsymbol{x}$ that solves

$$\min_{\boldsymbol{x} \in \boldsymbol{x}^{(0)} + \mathcal{D}_k} \|\boldsymbol{b} - \mathsf{A}\boldsymbol{x}\|_2 .$$

Let $\boldsymbol{r}^{(k)} = \boldsymbol{b} - \mathsf{A}\boldsymbol{x}^{(k)}$.
(a) Show that if $\boldsymbol{x} \in \boldsymbol{x}^{(0)} + \mathcal{D}_k$, then $\boldsymbol{b} - \mathsf{A}\boldsymbol{x} = p(\mathsf{A})\boldsymbol{r}^{(0)}$ for some $p \in \mathcal{P}_k$.
(b) Prove that, conversely, for any $p \in \mathcal{P}_k$ there is some $\boldsymbol{x} \in \boldsymbol{x}^{(0)} + \mathcal{D}_k$, such that $\boldsymbol{b} - \mathsf{A}\boldsymbol{x} = p(\mathsf{A})\boldsymbol{r}^{(0)}$.

(c) Argue that

$$\left\|\boldsymbol{r}^{(k+1)}\right\|_2 = \min_{p\in\mathcal{P}_k} \left\|p(\mathsf{A})\boldsymbol{r}^{(0)}\right\|_2 .$$

(d) Prove that this iteration converges in at most n steps. (*Hint:* Argue for the existence of a polynomial in $\mathcal{P}_n$ that vanishes at all the eigenvalues of A. Use this polynomial to show that $\left\|\boldsymbol{r}^{(n)}\right\|_2 \leq 0$.)

Chapter 8

Least Squares

8.1 Least Squares

Least squares is a general class of methods for fitting observed data to a theoretical model function. In the general setting we are given a set of data

x	x_0	x_1	$\dots$	x_n
y	y_0	y_1	$\dots$	y_n

and some class of functions, $\mathcal{F}$. The goal then is to find the "best" $f \in \mathcal{F}$ to fit the data to $y = f(x)$. Usually the class of functions $\mathcal{F}$ will be determined by some small number of parameters; the number of parameters will be smaller (usually much smaller) than the number of data points. The theory here will be concerned with defining "best," and examining methods for finding the "best" function to fit the data.

8.1.1 The Definition of Ordinary Least Squares

First we consider the data to be two vectors of length $n+1$. That is we let

$$\boldsymbol{x} = [x_0\ x_1\ \dots\ x_n]^\top, \quad \text{and} \quad \boldsymbol{y} = [y_0\ y_1\ \dots\ y_n]^\top.$$

The *error*, or *residual*, of a given function with respect to this data is the vector $\boldsymbol{r} = \boldsymbol{y} - f(\boldsymbol{x})$. That is

$$\boldsymbol{r} = [r_0\ r_1\ \dots\ r_n]^\top, \quad \text{where } r_i = y_i - f(x_i).$$

Our goal is to find $f \in \mathcal{F}$ such that $\boldsymbol{r}$ is reasonably small. We measure the size of a vector by the use of norms, which were explored in Subsection **??**. The most useful norms are the ℓ^p norms. For a given p with $0 < p < \infty$, the ℓ^p norm of $\boldsymbol{r}$ is defined as

$$\|\boldsymbol{r}\|_p = \left(\sum_{i=0}^{n} r_i^p \right)^{1/p}$$

For the purposes of approximation, the easiest norm to use is the ℓ^2 norm:

Definition 8.1.1 ((Ordinary) Least Squares Best Approximant)**.** The *least-squares best approximant* to a set of data, $\boldsymbol{x}$, $\boldsymbol{y}$ from a class of functions, $\mathcal{F}$, is the function $f^* \in \mathcal{F}$ that minimizes the ℓ^2 norm of the error. That is, if f^* is the least squares best approximant, then

$$\|\boldsymbol{y} - f^*(\boldsymbol{x})\|_2 = \min_{f \in \mathcal{F}} \|\boldsymbol{y} - f(\boldsymbol{x})\|_2$$

We will generally assume uniqueness of the minimum. This method is sometimes called the *ordinary* least squares method. It assumes there is no error in the measurement of the data $\boldsymbol{x}$, and usually admits a relatively straightforward solution.

8.1.2 Linear Least Squares

We illustrate this definition using the class of linear functions as an example, as this case is reasonably simple. We are assuming

$$\mathcal{F} = \{ f(x) = ax + b \mid a, b \in \mathbb{R} \}.$$

We can now think of our job as drawing the "best" line through the data.

By Definition 8.1.1, we want to find the function $f(x) = ax+b$ that minimizes

$$\left(\sum_{i=0}^{n} [y_i - f(x_i)]^2 \right)^{1/2}.$$

This is a minimization problem over two variables, x and y. As you may recall from calculus class, it suffices to minimize the sum rather than its square root. That is, it suffices to minimize

$$\sum_{i=0}^{n} [ax_i + b - y_i]^2.$$

We can illustrate this approach in Python easily enough.

```
In[]:
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import minimize
from numpy.random import seed, randn

seed(1)
xs = np.array(list(range(5)))
ys = [2*x+1 + randn()/2 for x in xs]
def squares(ab):
    a = ab[0]
```

```
    b = ab[1]
    return sum([((a*xs[i]+b)-ys[i])**2 for i in range(len(xs))])
result = minimize(squares,[2,1])
result
```

```
Out[]:
  message: 'Optimization terminated successfully.'
     njev: 6
      fun: 1.196223130993053
      nit: 4
     nfev: 24
  success: True
      jac: array([  2.98023224e-08,   1.49011612e-08])
   status: 0
        x: array([ 1.9010456 ,  1.22559441])
 hess_inv: array([[ 0.05, -0.1 ],
       [-0.1 ,  0.3 ]])
```

The result is essentially a dictionary with some extra formatting. The `fun` key tells us the minimum value and the `x` key tells us how to choose the parameter to get the minimum. The fit is illustrated in figure 8.1.

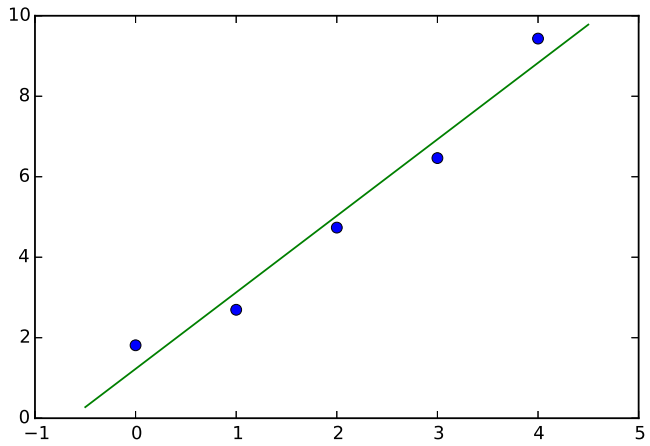


Figure 8.1: A simple least squares approximation

We minimize this function with calculus. We set

$$\begin{aligned} 0 = \frac{\partial \phi}{a} &= \sum_{k=0}^{n} 2x_k \left(ax_k + b - y_k \right) \\ 0 = \frac{\partial \phi}{b} &= \sum_{k=0}^{n} 2 \left(ax_k + b - y_k \right) \end{aligned}$$

These are called the *normal equations*. Believe it or not these are two equations in two unknowns. They can be reduced to

$$\begin{aligned} \sum x_k^2 a + \sum x_k b &= \sum x_k y_k \\ \sum x_k a + (n+1) b &= \sum y_k \end{aligned}$$

The solution is found by naïve Gaussian Elimination, and is ugly. Let

$$\begin{aligned} d_{11} &= \sum x_k^2 \\ d_{12} = d_{21} &= \sum x_k \\ d_{22} &= n+1 \\ e_1 &= \sum x_k y_k \\ e_2 &= \sum y_k \end{aligned}$$

We want to solve

$$\begin{aligned} d_{11} a + d_{12} b &= e_1 \\ d_{21} a + d_{22} b &= e_2 \end{aligned}$$

Gaussian Elimination produces

$$\begin{aligned} a &= \frac{d_{22} e_1 - d_{12} e_2}{d_{22} d_{11} - d_{12} d_{21}} \\ b &= \frac{d_{11} e_2 - d_{21} e_1}{d_{22} d_{11} - d_{12} d_{21}} \end{aligned}$$

The answer is not so enlightening as the means of finding the solution.

We should, for a moment, consider whether this is indeed the solution. Our calculations have only shown an extrema at this choice of (a, b); could it not be a maxima?

8.1.3 Least Squares from Basis Functions

In many, but not all cases, the class of functions, $\mathcal{F}$, is the span of a small set of functions. This case is simpler to explore and we consider it here. In this case $\mathcal{F}$ can be viewed as a vector space over the real numbers. That is, for $f, g \in \mathcal{F}$,

and $\alpha, \beta \in \mathbb{R}$, then $\alpha f + \beta g \in \mathcal{F}$, where the function αf is that function such that $(\alpha f)(x) = \alpha f(x)$.

Now let $\{g_j(x)\}_{j=0}^m$ be a set of $m+1$ linearly independent functions, *i.e.*,

$$c_0 g_0(x) + c_1 g_1(x) + \ldots + c_m g_m(x) = 0 \quad \forall x \quad \Rightarrow \quad c_0 = c_1 = \ldots = c_m = 0$$

Then we say that $\mathcal{F}$ is *spanned* by the functions $\{g_j(x)\}_{j=0}^m$ if

$$\mathcal{F} = \left\{ f(x) = \sum_j c_j g_j(x) \,|\, c_j \in \mathbb{R},\, j = 0, 1, \ldots, m \right\}.$$

In this case the functions g_j are *basis functions* for $\mathcal{F}$. Note the basis functions need not be unique: a given class of functions will usually have more than one choice of basis functions.

Example 8.1. *The class $\mathcal{F} = \{ f(x) = ax + b \,|\, a, b \in \mathbb{R} \}$ is spanned by the two functions $g_0(x) = 1$, and $g_1(x) = x$. However, it is also spanned by the two functions $\tilde{g}_0(x) = 2x + 1$, and $\tilde{g}_1(x) = x - 1$.*

To find the least squares best approximant of $\mathcal{F}$ for a given set of data, we minimize the square of the ℓ^2 norm of the error; that is we minimize the function

$$\phi\,(c_0, c_1, \ldots, c_m) = \sum_{k=0}^{n} \left[\left(\sum_j c_j g_j(x_k) \right) - y_k \right]^2 \tag{8.1}$$

Again we set partials to zero and solve

$$0 = \frac{\partial \phi}{c_i} = \sum_{k=0}^{n} 2 \left[\left(\sum_j c_j g_j(x_k) \right) - y_k \right] g_i(x_k)$$

This can be rearranged to get

$$\sum_{j=0}^{m} \left[\sum_k g_j(x_k) g_i(x_k) \right] c_j \quad = \quad \sum_{k=0}^{n} y_k g_i(x_k)$$

If we now let

$$d_{ij} = \sum_{k=0}^{n} g_j(x_k) g_i(x_k), \quad e_i = \sum_{k=0}^{n} y_k g_i(x_k),$$

Then we have reduced the problem to the linear system (again, called the normal equations):

$$\begin{bmatrix} d_{00} & d_{01} & d_{02} & \cdots & d_{0m} \\ d_{10} & d_{11} & d_{12} & \cdots & d_{1m} \\ d_{20} & d_{21} & d_{22} & \cdots & d_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{m0} & d_{m1} & d_{m2} & \cdots & d_{mm} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix} \tag{8.2}$$

The choice of the basis functions can affect how easy it is to solve this system. We explore this in Section 8.2. Note that we are talking about the basis $\{g_j\}_{j=0}^m$, and not exactly about the class of functions $\mathcal{F}$.

For example, consider what would happen if the system of normal equations were diagonal. In this case, solving the system would be rather trivial.

Example Problem 8.2. *Consider the case where $m = 0$, and $g_0(x) = \ln x$. Find the least squares approximation of the data*

x	0.50	0.75	1.0	1.50	2.0	2.25	2.75	3.0
y	−1.187098	−0.452472	−0.068077	0.713938	1.165234	1.436975	1.725919	1.841422

Solution: *Essentially we are trying to find the c such that $c \ln x$ best approximates the data. The system of equation 8.2 reduces to the 1-D equation:*

$$\left[\Sigma_k \ln^2 x_k\right] c = \Sigma_k y_k \ln x_k$$

For our data, this reduces to:

$$4.0960c = 6.9844$$

so we find $c = 1.7052$. The data and the least squares interpolant are shown in Figure 8.2. ⊣

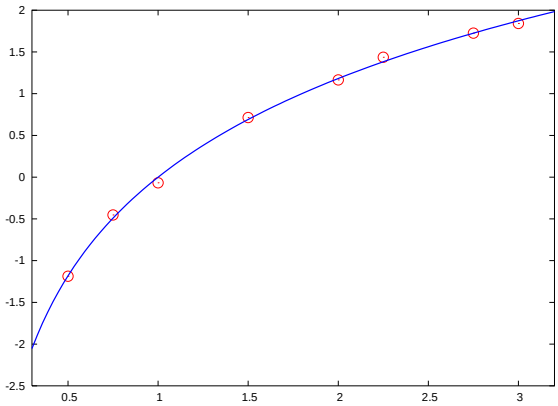


Figure 8.2: The data of Example Problem 8.2 and the least squares interpolant are shown.

Example 8.3. *Consider the awfully chosen basis functions:*

$$\begin{aligned} g_0(x) &= \left(\frac{\epsilon}{2} - 1\right) x^2 - \frac{\epsilon}{2} x + 1 \\ g_1(x) &= x^3 + \left(\frac{\epsilon}{2} - 1\right)\left(x^2 + x\right) + 1 \end{aligned}$$

where ϵ is small, around machine precision.

Suppose the data are given at the nodes $x_0 = 0, x_1 = 1, x_2 = -1$. We want to set up the normal equations, so we compute some of the d_{ij}. First we have to evaluate the basis functions at the nodes x_i. But this example was rigged to give:

$$g_0(x_0) = 1,\ g_0(x_1) = 0,\ g_0(x_2) = \epsilon$$
$$g_1(x_0) = 1,\ g_1(x_1) = \epsilon,\ g_1(x_2) = 0$$

After much work we find we want to solve

$$\begin{bmatrix} 1+\epsilon^2 & 1 \\ 1 & 1+\epsilon^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} y_0 + \epsilon y_2 \\ y_0 + \epsilon y_1 \end{bmatrix}$$

However, the computer would only find this if it had infinite precision. Since it does not, and since ϵ is rather small, the computer thinks $\epsilon^2 = 0$, and so tries to solve the system

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} y_0 + \epsilon y_2 \\ y_0 + \epsilon y_1 \end{bmatrix}$$

When $y_1 \neq y_2$, this has no solution. Bummer.

This kind of thing is common in the method of least squares: the coefficients of the normal equations include terms like

$$g_i(x_k) g_j(x_k).$$

When the g_i are small at the nodes x_k, these coefficients can get really small, since we are squaring.

Now we draw a rough sketch of the basis functions. We find they do a pretty poor job of discriminating around all the nodes.

8.2 Orthonormal Bases

In the previous section we saw that poor choice of basis vectors can lead to numerical problems. Roughly speaking, if $g_i(x_k)$ is small for some i's and k's, then some d_{ij} can have a loss of precision when two small quantities are multiplied together and rounded to zero.

Poor choice of basis vectors can also lead to numerical problems in solution of the normal equations, which will be done by Gaussian Elimination.

Consider the case where $\mathcal{F}$ is the class of polynomials of degree no greater than m. For simplicity we will assume that all x_i are in the interval $[0, 1]$. The most obvious choice of basis functions is $g_j(x) = x^j$. This certainly gives a basis for $\mathcal{F}$, but actually a rather poor one. To see why, look at the graph of the basis functions in Figure 8.4. The basis functions look too much alike on the given interval.

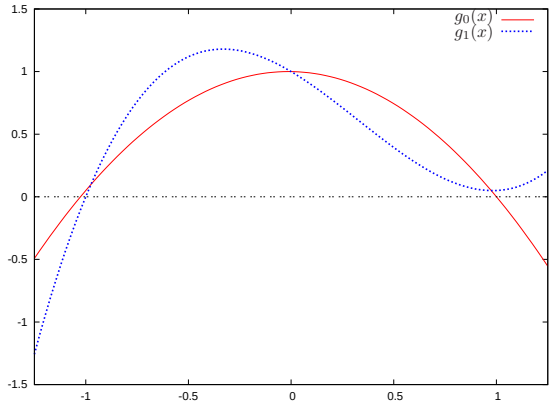


Figure 8.3: The basis functions $g_0(x) = \left(\frac{\epsilon}{2} - 1\right) x^2 - \frac{\epsilon}{2} x + 1$, and $g_1(x) = x^3 + \left(\frac{\epsilon}{2} - 1\right)\left(x^2 + x\right) + 1$ are shown for $\epsilon = 0.05$. Note that around the three nodes $0, 1, -1$, these two functions take nearly identical values. This can lead to a system of normal equations with no solution.

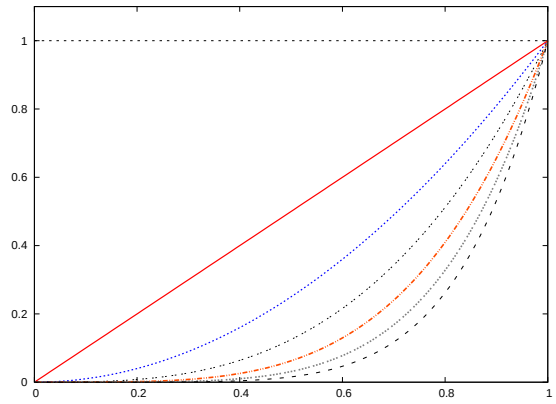


Figure 8.4: The polynomials x^i for $i = 0, 1, \ldots, 6$ are shown on $[0, 1]$. These polynomials make a bad basis because they look so much alike, essentially.

A better basis for this problem is the set of Chebyshev Polynomials of the first kind, *i.e.*, $g_j(x) = T_j(x)$, where

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x).$$

These polynomials are illustrated in Figure 8.5.

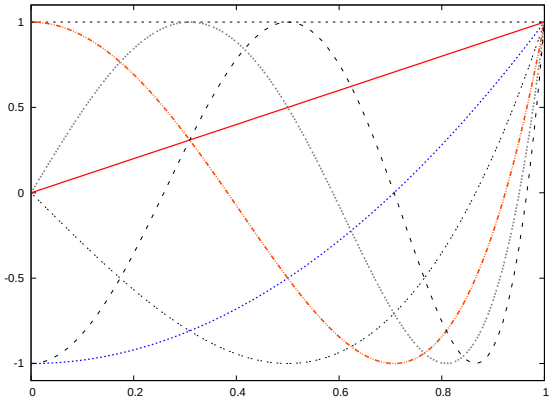


Figure 8.5: The Chebyshev polynomials $T_j(x)$ for $j = 0, 1, \ldots, 6$ are shown on $[0, 1]$. These polynomials make a better basis for least squares because they are orthogonal under some inner product. Basically, they do not look like each other.

8.2.1 Alternatives to Normal Equations

It turns out that the Normal Equations method isn't really so great. We consider other methods. First, we define A as the $n \times m$ matrix defined by the entries:

$$a_{ij} = g_j(x_i), \quad i = 0, 1, \ldots, n, \; j = 0, 1, \ldots, m.$$

That is

$$\mathsf{A} = \begin{bmatrix} g_0(x_0) & g_1(x_0) & g_2(x_0) & \cdots & g_m(x_0) \\ g_0(x_1) & g_1(x_1) & g_2(x_1) & \cdots & g_m(x_1) \\ g_0(x_2) & g_1(x_2) & g_2(x_2) & \cdots & g_m(x_2) \\ g_0(x_3) & g_1(x_3) & g_2(x_3) & \cdots & g_m(x_3) \\ g_0(x_4) & g_1(x_4) & g_2(x_4) & \cdots & g_m(x_4) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_0(x_n) & g_1(x_n) & g_2(x_n) & \cdots & g_m(x_n) \end{bmatrix}$$

We write it in this way since we are thinking of the case where $n \gg m$, so A is "tall."

After some inspection, we find that the Normal Equations can be written as:

$$\boxed{\mathsf{A}^\top \mathsf{A}\, \boldsymbol{c} = \mathsf{A}^\top \boldsymbol{y}.} \tag{8.3}$$

Now let the vector $\boldsymbol{c}$ be identified, in the natural way, with a function in $\mathcal{F}$. That is $\boldsymbol{c}$ is identified with $f(x) = \sum_{j=0}^{m} c_j g_j(x)$. You should now convince yourself that

$$\mathsf{A}\boldsymbol{c} = f(\boldsymbol{x}).$$

And thus the residual, or error, of this function is $\boldsymbol{r} = \boldsymbol{y} - \mathsf{A}\boldsymbol{c}$.

In our least squares theory we attempted to find that $\boldsymbol{c}$ that minimized

$$\|\boldsymbol{y} - \mathsf{A}\boldsymbol{c}\|_2^2 = (\boldsymbol{y} - \mathsf{A}\,\boldsymbol{c})^\top (\boldsymbol{y} - \mathsf{A}\,\boldsymbol{c})$$

We can see this as minimizing the Euclidian distance from $\boldsymbol{y}$ to $\mathsf{A}\,\boldsymbol{c}$. For this reason, we will have that the residual is orthogonal to the column space of A, that is we want

$$\mathsf{A}^\top \boldsymbol{r} = \mathsf{A}^\top (\boldsymbol{y} - \mathsf{A}\,\boldsymbol{c}) = \boldsymbol{0}.$$

This is just the normal equations. We could rewrite this, however, in the following form: find $\boldsymbol{c}$, $\boldsymbol{r}$ such that

$$\begin{bmatrix} \mathsf{I} & \mathsf{A} \\ \mathsf{A}^\top & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{c} \end{bmatrix} = \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{bmatrix}$$

This is now a system of $n + m$ variables and unknowns, which can be solved by specialized means. This is known as the *augmented form*. We briefly mention that naïve Gaussian Elimination is *not* appropriate to solve the augmented form, as it turns out to be equivalent to using the normal equations method.

8.3 Orthogonal Least Squares

The method described in Section 8.1, sometimes referred to as "ordinary least squares," assumes that measurement error is found entirely in the dependent variable, and that there is no error in the independent variables.

For example, consider the case of two variables, x and y, which are thought to be related by an equation of the form $y = mx+b$. A number of measurements are made, giving the two sequences $\{x_i\}_{i=1}^n$ and $\{y_i\}_{i=1}^n$. Ordinary least squares assumes, that

$$y_i = mx_i + b + \epsilon_i,$$

where ϵ_i, the error of the i^{th} measurement, is a random variable. It is usually assumed that $\mathrm{E}[\epsilon_i] = 0$, *i.e.*, that the measurements are "unbiased." Under the further assumption that the errors are independent and have the same variance, the ordinary least squares solution is a very good one.[1]

However, what if it were the case that

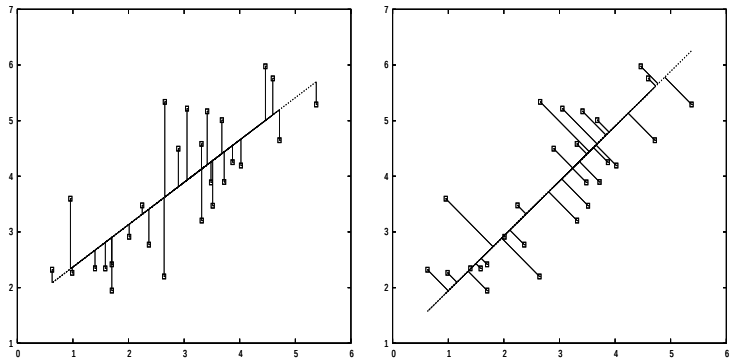
$$y_i = m(x_i + \xi_i) + b + \epsilon_i,$$

i.e., that there is actually error in the measurement of the x_i? In this case, the orthogonal least squares method is appropriate.

The difference between the two methods is illustrated in Figure 8.6. In Figure 8.6a, the ordinary least squares method is shown; it minimizes the sum of the squared lengths of vertical lines from observations to a line, reflecting

[1] This means that the ordinary least squares solution gives an unbiased estimator of m and b, and, moreover, gives the estimators with the lowest variance among all linear, *a priori* estimators. For more details on this, locate the Gauss-Markov Theorem in a good statistics textbook.

the assumption of no error in x_i. The orthogonal least squares method will minimize the sum of the squared distances from observations to a line, as shown in Figure 8.6b.



(a) Ordinary Least Squares (b) Orthogonal Least Squares

Figure 8.6: The ordinary least squares method, as shown in (a), minimizes the sum of the squared lengths of the vertical lines from the observed points to the regression line. The orthogonal least squares, shown in (b), minimizes the sum of the squared distances from the points to the line. The offsets from points to the regression line in (b) may not look orthogonal due to improper aspect ratio of the figure.

We construct the solution geometrically. Let $\left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{m}$ be the set of observations, expressed as vectors in $\mathbb{R}^n$. The problem is to find the vector $\boldsymbol{n}$ and number d such that the hyperplane $\boldsymbol{n}^\top \boldsymbol{x} = d$ is the plane that minimizes the sum of the squared distances from the points to the plane. We can solve this problem using vector calculus.

The function

$$\frac{1}{\|\boldsymbol{n}\|_2^2} \sum_{i=1}^{m} \left\| \boldsymbol{n}^\top \boldsymbol{x}^{(i)} - d \right\|_2^2.$$

is the one to be minimized with respect to $\boldsymbol{n}$ and d. It gives the sum of the squared distances from the points to the plane described by $\boldsymbol{n}$ and d. To simplify its computation, we will minimize it subject to the constraint that $\|\boldsymbol{n}\|_2^2 = 1$. Thus our problem is to find $\boldsymbol{n}$ and d that solve

$$\min_{\boldsymbol{n}^\top \boldsymbol{n}=1} \sum_{i=1}^{m} \left\| \boldsymbol{n}^\top \boldsymbol{x}^{(i)} - d \right\|_2^2.$$

We can express this as $\min f(\boldsymbol{n}, d)$ subject to $g(\boldsymbol{n}, d) = 1$. The solution of this problem uses the Lagrange multipliers technique.

Let $\mathcal{L}(\boldsymbol{n}, d, \lambda) = f(\boldsymbol{n}, d) - \lambda g(\boldsymbol{n}, d)$. The Lagrange multipliers theory tells us that a necessary condition for $\boldsymbol{n}, d$ to be the solution is that there exists λ such

that the following hold:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial d}(\boldsymbol{n}, d, \lambda) = 0 \\ \nabla_{\boldsymbol{n}} \mathcal{L}(\boldsymbol{n}, d, \lambda) = \mathbf{0} \\ g(\boldsymbol{n}, d) = 1 \end{cases}$$

Let X be the $m \times n$ matrix whose i^{th} row is the vector ${\boldsymbol{x}^{(i)}}^\top$. We can rewrite the Lagrange function as

$$\begin{aligned} \mathcal{L}(\boldsymbol{n}, d, \lambda) &= (\mathsf{X}\boldsymbol{n} - d\mathbf{1}_{\boldsymbol{m}})^\top (\mathsf{X}\boldsymbol{n} - d\mathbf{1}_{\boldsymbol{m}}) - \lambda \boldsymbol{n}^\top \boldsymbol{n} \\ &= \boldsymbol{n}^\top \mathsf{X}^\top \mathsf{X}\boldsymbol{n} - 2d{\mathbf{1}_{\boldsymbol{m}}}^\top \mathsf{X}\boldsymbol{n} + d^2 {\mathbf{1}_{\boldsymbol{m}}}^\top \mathbf{1}_{\boldsymbol{m}} - \lambda \boldsymbol{n}^\top \boldsymbol{n} \end{aligned}$$

We solve the necessary condition on $\frac{\partial \mathcal{L}}{\partial d}$.

$$0 = \frac{\partial \mathcal{L}}{\partial d}(\boldsymbol{n}, d, \lambda) = 2d{\mathbf{1}_{\boldsymbol{m}}}^\top \mathbf{1}_{\boldsymbol{m}} - 2{\mathbf{1}_{\boldsymbol{m}}}^\top \mathsf{X}\boldsymbol{n} \quad \Rightarrow \quad d = {\mathbf{1}_{\boldsymbol{m}}}^\top \mathsf{X}\boldsymbol{n} / {\mathbf{1}_{\boldsymbol{m}}}^\top \mathbf{1}_{\boldsymbol{m}}$$

This essentially tells us that we will zero the first moment of the data around the line. Note that this determination of d requires knowledge of $\boldsymbol{n}$. However, since this is a necessary condition, we can plug it into the gradient equation:

$$\mathbf{0} = \nabla_{\boldsymbol{n}} \mathcal{L}(\boldsymbol{n}, d, \lambda) = 2\mathsf{X}^\top \mathsf{X}\boldsymbol{n} - 2d\left(\mathbf{1}_{\boldsymbol{m}}{}^\top \mathsf{X}\right)^\top - 2\lambda \boldsymbol{n} = 2\left[\mathsf{X}^\top \mathsf{X}\boldsymbol{n} - \frac{{\mathbf{1}_{\boldsymbol{m}}}^\top \mathsf{X}\boldsymbol{n}}{{\mathbf{1}_{\boldsymbol{m}}}^\top \mathbf{1}_{\boldsymbol{m}}}\left(\mathbf{1}_{\boldsymbol{m}}{}^\top \mathsf{X}\right)^\top - \lambda \boldsymbol{n}\right]$$

The middle term is a scalar times a vector, so the multiplication can be commuted, this gives

$$\mathbf{0} = \left[\mathsf{X}^\top \mathsf{X}\boldsymbol{n} - \frac{\mathsf{X}^\top \mathbf{1}_{\boldsymbol{m}} {\mathbf{1}_{\boldsymbol{m}}}^\top \mathsf{X}}{{\mathbf{1}_{\boldsymbol{m}}}^\top \mathbf{1}_{\boldsymbol{m}}} \boldsymbol{n} - \lambda \boldsymbol{n}\right] = \left[\mathsf{X}^\top \mathsf{X} - \frac{\mathsf{X}^\top \mathbf{1}_{\boldsymbol{m}} {\mathbf{1}_{\boldsymbol{m}}}^\top \mathsf{X}}{{\mathbf{1}_{\boldsymbol{m}}}^\top \mathbf{1}_{\boldsymbol{m}}} - \lambda \mathsf{I}\right] \boldsymbol{n}$$

Thus $\boldsymbol{n}$ is an eigenvector of the matrix

$$\mathsf{M} = \mathsf{X}^\top \mathsf{X} - \frac{\mathsf{X}^\top \mathbf{1}_{\boldsymbol{m}} {\mathbf{1}_{\boldsymbol{m}}}^\top \mathsf{X}}{{\mathbf{1}_{\boldsymbol{m}}}^\top \mathbf{1}_{\boldsymbol{m}}}$$

The final condition of the three Lagrange conditions is that $g(\boldsymbol{n}, d) = \boldsymbol{n}^\top \boldsymbol{n} = 1$. Thus if $\boldsymbol{n}$ is a minimizer, then it is a unit eigenvector of M.

It is not clear which eigenvector is the right one, so we might have to check *all* the eigenvectors. However, further work tells us which eigenvector it is. First we rewrite the function to be minimized, when the optimal d is used:

$\hat{f}(\boldsymbol{n}) = f(\boldsymbol{n}, \mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n}/\mathbf{1}_m{}^\top \mathbf{1}_m)$. We have

$$\begin{aligned}
\hat{f}(\boldsymbol{n}) &= \boldsymbol{n}^\top \mathsf{X}^\top \mathsf{X}\boldsymbol{n} - 2\frac{\mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n}}{\mathbf{1}_m{}^\top \mathbf{1}_m}\mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n} + \left(\frac{\mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n}}{\mathbf{1}_m{}^\top \mathbf{1}_m}\right)^2 \mathbf{1}_m{}^\top \mathbf{1}_m \\
&= \boldsymbol{n}^\top \mathsf{X}^\top \mathsf{X}\boldsymbol{n} - 2\frac{\boldsymbol{n}^\top \mathsf{X}^\top \mathbf{1}_m \mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n}}{\mathbf{1}_m{}^\top \mathbf{1}_m} + \frac{\boldsymbol{n}^\top \mathsf{X}^\top \mathbf{1}_m \mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n} \mathbf{1}_m{}^\top \mathbf{1}_m}{\mathbf{1}_m{}^\top \mathbf{1}_m \mathbf{1}_m{}^\top \mathbf{1}_m} \\
&= \boldsymbol{n}^\top \mathsf{X}^\top \mathsf{X}\boldsymbol{n} - 2\frac{\boldsymbol{n}^\top \mathsf{X}^\top \mathbf{1}_m \mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n}}{\mathbf{1}_m{}^\top \mathbf{1}_m} + \frac{\boldsymbol{n}^\top \mathsf{X}^\top \mathbf{1}_m \mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n}}{\mathbf{1}_m{}^\top \mathbf{1}_m} \\
&= \boldsymbol{n}^\top \mathsf{X}^\top \mathsf{X}\boldsymbol{n} - \frac{\boldsymbol{n}^\top \mathsf{X}^\top \mathbf{1}_m \mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n}}{\mathbf{1}_m{}^\top \mathbf{1}_m} \\
&= \boldsymbol{n}^\top \left[\mathsf{X}^\top \mathsf{X} - \frac{\mathsf{X}^\top \mathbf{1}_m \mathbf{1}_m{}^\top \mathsf{X}}{\mathbf{1}_m{}^\top \mathbf{1}_m}\right] \boldsymbol{n} \\
&= \boldsymbol{n}^\top \mathsf{M}\boldsymbol{n}
\end{aligned}$$

This sequence of equations only required that the d be the optimal d for the given $\boldsymbol{n}$, and used the fact that a scalar is it's own transpose, thus $\mathbf{1}_m{}^\top \mathsf{X}\boldsymbol{n} = \boldsymbol{n}^\top \mathsf{X}^\top \mathbf{1}_m$.

Now if $\boldsymbol{n}$ is a unit eigenvector of M, with corresponding eigenvalue λ, then $\boldsymbol{n}^\top \mathsf{M}\boldsymbol{n} = \lambda$. Note that because $f(\boldsymbol{n}, d)$ is defined as $\boldsymbol{w}^\top \boldsymbol{w}$, for some vector $\boldsymbol{w}$, then the matrix M must be positive definite, *i.e.*, λ must be positive. However, we want to minimize λ. Thus we take $\boldsymbol{n}$ to be the unit eigenvector associated with the smallest eigenvalue.

Note that, by roundoff, you may compute that M has some negative eigenvalues, though they are small in magnitude. Thus one should select, as $\boldsymbol{n}$, the unit eigenvector associated with the smallest eigenvalue in absolute value.

Example Problem 8.4. *Find the equation of the line which best approximates the 2D data* $\{(x_i, y_i)\}_{i=1}^m$*, by the orthogonal least squares method.* Solution: *In this case the matrix* X *is*

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \vdots & \vdots \\ x_m & y_m \end{bmatrix}$$

Thus we have that

$$\begin{aligned}
\mathsf{M} &= \mathsf{X}^\top\mathsf{X} - \frac{\mathsf{X}^\top \mathbf{1_m} \mathbf{1_m}^\top \mathsf{X}}{\mathbf{1_m}^\top \mathbf{1_m}}, \\
&= \begin{bmatrix} \sum x_i^2 & \sum x_i y_i \\ \sum x_i y_i & \sum y_i^2 \end{bmatrix} - \frac{1}{m} \begin{bmatrix} \left(\sum x_i\right)^2 & \sum x_i \sum y_i \\ \sum x_i \sum y_i & \left(\sum y_i\right)^2 \end{bmatrix}, \\
&= \begin{bmatrix} \sum x_i^2 & \sum x_i y_i \\ \sum x_i y_i & \sum y_i^2 \end{bmatrix} - m \begin{bmatrix} \bar{x}^2 & \bar{x}\bar{y} \\ \bar{x}\bar{y} & \bar{y}^2 \end{bmatrix}, \\
&= \begin{bmatrix} \sum x_i^2 - m\bar{x}^2 & \sum x_i y_i - m\bar{x}\bar{y} \\ \sum x_i y_i - m\bar{x}\bar{y} & \sum y_i^2 - m\bar{y}^2 \end{bmatrix} \\
&= \begin{bmatrix} 2\alpha & \beta \\ \beta & 2\gamma \end{bmatrix}. \qquad (8.4)
\end{aligned}$$

where we use $\bar{x}, \bar{y}$, to mean, respectively, the mean of the x- and y-values. The characteristic polynomial of the matrix, whose roots are the eigenvalues of the matrix is

$$p(\lambda) = \det \begin{bmatrix} 2\alpha - \lambda & \beta \\ \beta & 2\gamma - \lambda \end{bmatrix} = 4\alpha\gamma - \beta^2 - 2(\alpha+\gamma)\lambda + \lambda^2.$$

The roots of this polynomial are given by the quadratic equation

$$\lambda_\pm = \frac{2(\alpha+\gamma) \pm \sqrt{4\left(\alpha+\gamma\right)^2 - 4\left(4\alpha\gamma - \beta^2\right)}}{2} = (\alpha+\gamma) \pm \sqrt{\left(\alpha-\gamma\right)^2 + \beta^2}$$

We want the smaller eigenvalue, $\lambda_- = (\alpha+\gamma) - \sqrt{\left(\alpha-\gamma\right)^2 + \beta^2}$. The associated eigenvector is in the null space of the matrix

$$\mathbf{0} = \begin{bmatrix} 2\alpha - \lambda_- & \beta \\ \beta & 2\gamma - \lambda_- \end{bmatrix} \boldsymbol{v} = \begin{bmatrix} 2\alpha - \lambda_- & \beta \\ \beta & 2\gamma - \lambda_- \end{bmatrix} \begin{bmatrix} -k \\ 1 \end{bmatrix} = \begin{bmatrix} \beta - k\left(2\alpha - \lambda_-\right) \\ -k\beta + 2\gamma - \lambda_- \end{bmatrix}$$

This is solved by

$$\begin{aligned}
k &= \frac{2\gamma - \lambda_-}{\beta} = \frac{(\gamma-\alpha) + \sqrt{\left(\gamma-\alpha\right)^2 + \beta^2}}{\beta} \\
&= \frac{\sum(y_i^2 - x_i^2) - m\left(\bar{y}^2 - \bar{x}^2\right) + \sqrt{\left(\sum(y_i^2 - x_i^2) - m\left(\bar{y}^2 - \bar{x}^2\right)\right)^2 + 4\left(\sum x_i y_i - m\bar{x}\bar{y}\right)^2}}{2\sum x_i y_i - m\bar{x}\bar{y}}. \qquad (8.5)
\end{aligned}$$

Thus the best plane through the data is of the form

$$-kx + 1y = d \quad \textit{or} \quad y = kx + d$$

Note that the eigenvector, $\boldsymbol{v}$, that we used is not *a unit vector. However it is parallel to the unit vector we want, and can still use the regular formula to*

compute the optimal d.

$$d = \mathbf{1_m}^\top \mathsf{X}\boldsymbol{n}/\mathbf{1_m}^\top\mathbf{1_m} = \frac{1}{m}\mathbf{1_m}^\top\mathsf{X}\begin{bmatrix} -k \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} \bar{x} & \bar{y} \end{bmatrix}\begin{bmatrix} -k \\ 1 \end{bmatrix} = \bar{y} - k\bar{x}$$

Thus the optimal line through the data is

$$y = kx + d = k\,(x - \bar{x}) + \bar{y} \quad or \quad y - \bar{y} = k\,(x - \bar{x})\,,$$

with k defined in equation 8.5.

⊣

8.3.1 Computing the Orthogonal Least Squares Approximant

Just as the Normal Equations equation 8.3 define the solution to the ordinary least squares approximant, but are not normally used to solve the problem, so too is the above described means of computing the orthogonal least squares not a good idea.

First we define

$$\mathsf{W} = \mathsf{X} - \frac{\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\mathbf{1_m}^\top\mathbf{1_m}}.$$

Now note that

$$\mathsf{W}^\top\mathsf{W} = \left(\mathsf{X} - \frac{\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\mathbf{1_m}^\top\mathbf{1_m}}\right)^\top\left(\mathsf{X} - \frac{\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\mathbf{1_m}^\top\mathbf{1_m}}\right) = \mathsf{X}^\top\mathsf{X} - 2\frac{\mathsf{X}^\top\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\mathbf{1_m}^\top\mathbf{1_m}} + \frac{\left(\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}\right)^\top\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\left(\mathbf{1_m}^\top\mathbf{1_m}\right)^2}$$
$$= \mathsf{X}^\top\mathsf{X} - 2\frac{\mathsf{X}^\top\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\mathbf{1_m}^\top\mathbf{1_m}} + \frac{\mathsf{X}^\top\mathbf{1_m}(\mathbf{1_m}^\top\mathbf{1_m})\mathbf{1_m}^\top\mathsf{X}}{\left(\mathbf{1_m}^\top\mathbf{1_m}\right)^2} = \mathsf{X}^\top\mathsf{X} - 2\frac{\mathsf{X}^\top\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\mathbf{1_m}^\top\mathbf{1_m}} + \frac{\mathsf{X}^\top\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\mathbf{1_m}^\top\mathbf{1_m}}$$
$$= \mathsf{X}^\top\mathsf{X} - \frac{\mathsf{X}^\top\mathbf{1_m}\mathbf{1_m}^\top\mathsf{X}}{\mathbf{1_m}^\top\mathbf{1_m}} = \mathsf{M}.$$

We now need some linear algebra magic:

Definition 8.3.1. A square matrix U is called *unitary* if its inverse is its transpose, *i.e.*,

$$\mathsf{U}^\top\mathsf{U} = \mathsf{I} = \mathsf{U}\mathsf{U}^\top.$$

From the first equation, we see that the columns of U form a collection of orthonormal vectors. The second equation tells that the rows of U are also such a collection.

Definition 8.3.2. Every $m \times n$ matrix B can be decomposed as

$$\mathsf{B} = \mathsf{U}\Sigma\mathsf{V}^\top,$$

where U and V are unitary matrices, U is $m \times m$, V is $n \times n$, and Σ is $m \times n$, and has nonzero elements only on its diagonal. The values of the diagonal of Σ are the *singular values* of B. The column vectors of V span the row space of B, and the column vectors of U contain the column space of B.

The singular value decomposition generalizes the notion of eigenvalues and eigenvalues to the case of nonsquare matrices. Let $\boldsymbol{u}^{(i)}$ be the i^{th} column vector of U, $\boldsymbol{v}^{(i)}$ the i^{th} column of V, and let σ_i be the i^{th} element of the diagonal of Σ. Then if $\boldsymbol{x} = \sum_i \alpha_i \boldsymbol{v}^{(i)}$, we have

$$\mathsf{B}\boldsymbol{x} = \mathsf{U}\Sigma\mathsf{V}^\top \sum_i \alpha_i \boldsymbol{v}^{(i)} = \mathsf{U}\Sigma \left[\alpha_1 \alpha_2 \dots \alpha_n\right]^\top = \mathsf{U} \begin{bmatrix} \sigma_1\alpha_1 & 0 & \dots & 0 \\ 0 & \sigma_2\alpha_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n\alpha_n \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} = \sum_i \sigma_i \alpha_i \boldsymbol{u}^{(i)}.$$

Thus $\boldsymbol{v}^{(i)}$ and $\boldsymbol{u}^{(i)}$ act as an "eigenvector pair" with "eigenvalue" σ_i.

The singular value decomposition is computed by the octave/Matlab command `[U,S,V] = svd(B)`. The decomposition is computed such that the diagonal elements of `S`, the singular values, are positive, and decreasing with increasing index.

Now we can use the singular value decomposition on W:

$$\mathsf{W} = \mathsf{U}\Sigma\mathsf{V}^\top.$$

Thus

$$\mathsf{M} = \mathsf{W}^\top\mathsf{W} = \mathsf{V}\Sigma^\top\mathsf{U}^\top\mathsf{U}\Sigma\mathsf{V}^\top = \mathsf{V}\Sigma^\top\mathsf{I}\Sigma\mathsf{V}^\top = \mathsf{V}\mathsf{S}^2\mathsf{V}^\top,$$

where S^2 is the $n \times n$ diagonal matrix whose i^{th} diagonal is σ_i^2. Thus we see that the solution to the orthogonal least squares problem, the eigenvector associated with the smallest eigenvalue of M, is the column vector of V associated with the smallest, in absolute value, singular value of W. In octave/Matlab, this is `V(:,n)`.

This may not appear to be a computational savings, but if M is computed up front, and its eigenvectors are computed, there is loss of precision in its smallest eigenvalues which might lead us to choose the wrong normal direction (especially if there is more than one!). On the other hand, M is a $n \times n$ matrix, whereas W is $m \times n$. If storage is at a premium, and the method is being reapplied with new observations, it may be preferrable to keep M, rather than keeping W. That is, it may be necessary to trade conditioning for space.

8.3.2 Principal Component Analysis

The above analysis leads us to the topic of Principal Component Analysis. Suppose the $m \times n$ matrix X represents m noisy observations of some process,

where each observation is a vector in $\mathbb{R}^n$. Suppose the observations nearly lie in some k-dimensional subspace of $\mathbb{R}^n$, for $k < n$. How can you find an approximate value of k, and how do you find the k-dimensional subspace?

As in the previous subsection, let

$$\mathsf{W} = \mathsf{X} - \frac{\mathbf{1}_m \mathbf{1}_m{}^\top \mathsf{X}}{\mathbf{1}_m{}^\top \mathbf{1}_m} = \mathsf{X} - \mathbf{1}_m \bar{\boldsymbol{X}}, \quad \text{where } \bar{\boldsymbol{X}} = \mathbf{1}_m{}^\top \mathsf{X} / \mathbf{1}_m{}^\top \mathbf{1}_m.$$

Now note that $\bar{\boldsymbol{X}}$ is the mean of the m observations, as a row vector in $\mathbb{R}^n$. Under the assumption that the noise is approximately the same for each observation, we should think that $\bar{\boldsymbol{X}}$ is likely to be in or very close to the k-dimensional subspace. Then each row of W should be like a vector which is contained in the subspace. If we take some vector $\boldsymbol{v}$ and multiply $\mathsf{W}\boldsymbol{v}$, we expect the resultant vector to have small norm if $\boldsymbol{v}$ is not in the k-dimensional subspace, and to have a large norm if it is in the subspace.

You should now convince yourself that this is related to the singular value decomposition of W. Decomposing $\boldsymbol{v} = \sum_i \alpha_i \boldsymbol{v}^{(i)}$, we have $\mathsf{W}\boldsymbol{v} = \sum_i \sigma_i \alpha_i \boldsymbol{u}^{(i)}$, and thus product vector has small norm if the α_i associated with large σ_i are small, and large norm otherwise. That is the principal directions of the "best" k-dimensional subspace associated with X are the k columns of V associated with the largest singular values of W.

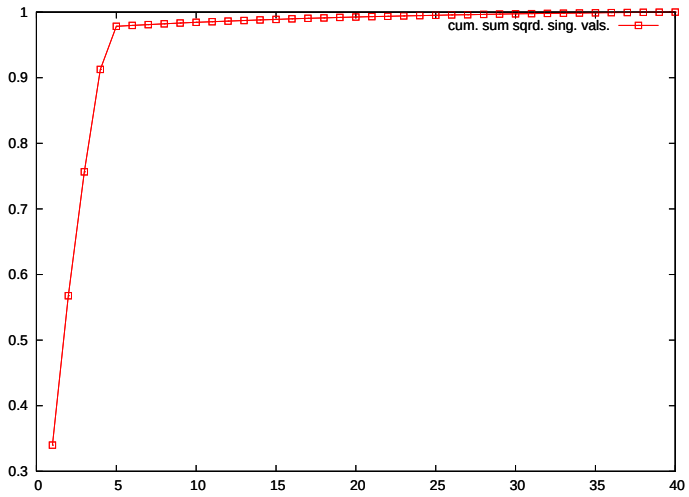


Figure 8.7: Noisy 5-dimensional data lurking in $\mathbb{R}^{40}$ are treated to Principal Component Analysis. This figure shows the normalized cumulative sum of squared singular values of W. There is a sharp "elbow" at $k = 5$, which indicates the data is actually 5-dimensional.

If k is unknown *a priori*, a good value can be obtained by eyeing a graph of the cumulative sum of squared singular values of W, and selecting the k that makes this appropriately large.

EXERCISES

(8.1) How do you know that the choice of constants c_i in our least squares analysis actually find a minimum of equation 8.1, and not, say, a maximum?

(8.2) Our "linear" least squares might be better called the "affine" least squares. In this exercise you will find the best linear function which approximates a set of data. That is, find the function $f(x) = cx$ which is the ordinary least squares best approximant to the given data

x	x_0	x_1	$\dots$	x_n
y	y_0	y_1	$\dots$	y_n

(8.3) Find the constant that best approximates, in the ordinary least squares sense, the given data $\boldsymbol{x}$, $\boldsymbol{y}$. (*Hint:* you can use equation 8.2 or equation 8.3 using a single basis function $g_0(x) = 1$.) Do the x_i values affect your answer?

(8.4) Find the function $f(x) = c$ which best approximates, in the ordinary least squares sense, the data

x	1	-2	5
y	1	-2	4

(8.5) Find the function $ax + b$ that best approximates the data

x	0	-1	2
y	0	1	-1

Use the ordinary least squares method.

(8.6) Find the function $ax + b$ that best approximates the data of the previous problem using the orthogonal least squares method.

(8.7) Find the function $ax^2 + b$ that best approximates the data

x	0	1	2
y	0.3	0.1	0.5

(8.8) Prove that the least squares solution is the solution of the normal equations, equation 8.3, and thus takes the form

$$\boldsymbol{c} = \left(\mathsf{A}^\top \mathsf{A}\right)^{-1} \mathsf{A}^\top \boldsymbol{y}.$$

(8.9) For ordinary least squares regression to the line $y = mx + b$, express the approximate m in terms of the α, β, and γ parameters from equation 8.4. Compare this to that found in equation 8.5.

(8.10) Any symmetric positive definite matrix, G can be used to define a norm in the following way:

$$\|\boldsymbol{v}\|_{\mathsf{G}} =_{\text{df}} \sqrt{\boldsymbol{v}^\top \mathsf{G} \boldsymbol{v}}$$

The *weighted* least squares method for G and data A and $\boldsymbol{y}$, finds the $\hat{\boldsymbol{c}}$ that solves

$$\min_{\boldsymbol{c}} \|\mathsf{A}\boldsymbol{c} - \boldsymbol{y}\|_{\mathsf{G}}^2$$

Prove that the normal equations form of the solution of this problem is

$$\mathsf{A}^\top \mathsf{G}\mathsf{A}\,\hat{\boldsymbol{c}} = \mathsf{A}^\top \mathsf{G}\boldsymbol{y}.$$

(8.11) (Continuation) Let the symmetric positive definite matrix G have Cholesky Factorization $\mathsf{G} = \mathsf{L}\mathsf{L}^\top$, where L is a lower triangular matrix. Show that the solution to the weighted least squares method is the $\hat{\boldsymbol{c}}$ that is the ordinary (unweighted) least squares best approximate solution to the problem

$$\mathsf{L}^\top \mathsf{A}\boldsymbol{c} = \mathsf{L}^\top \boldsymbol{y}.$$

(8.12) The r^2 statistic is often used to describe the "goodness-of-fit" of the ordinary least squares approximation to data. It is defined as

$$r^2 = \frac{\|\mathsf{A}\hat{\boldsymbol{c}}\|_2^2}{\|\boldsymbol{y}\|_2^2},$$

where $\hat{\boldsymbol{c}}$ is the approximant $\mathsf{A}^\top \mathsf{A}^{-1} \mathsf{A}^\top \boldsymbol{y}$.

(a) Prove that we also have

$$r^2 = 1 - \frac{\|\mathsf{A}\hat{\boldsymbol{c}} - \boldsymbol{y}\|_2^2}{\|\boldsymbol{y}\|_2^2}.$$

(b) Prove that the r^2 parameter is "scale-invariant," that is, if we change the units of our data, the parameter is unchanged (although the value of $\hat{\boldsymbol{c}}$ may change.)

(c) Devise a similar statistic for the orthogonal least squares approximation which is scale-invariant and rotation-invariant.

(8.13) As proved in Example Problem 8.4, the orthogonal least squares best approximate line for data $\{(x_i, y_i)\}_{i=1}^m$ goes through the point $(\bar{x}, \bar{y})$. Does the same thing hold for the ordinary least squares best approximate line?

(8.14) Find the constant c such that $f(x) = \ln(cx)$ best approximates, in the least squares sense, the given data

x	x_0	x_1	$\ldots$	x_n
y	y_0	y_1	$\ldots$	y_n

(*Hint:* You cannot use basis functions and equation 8.2 to solve this. You must use the Definition 8.1.1.) The *geometric mean* of the numbers $a_1, a_2, \ldots, a_n$ is defined as $\left(\prod a_i\right)^{1/n}$. How does your answer relate to the geometric mean?

Chapter 9

Approximating Derivatives

9.1 Finite Differences

Suppose we have some blackbox function $f(x)$ and we wish to calculate $f'(x)$ at some given x. Not surprisingly, we start with Taylor's theorem:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)h^2}{2}.$$

Rearranging we get

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(\xi)h}{2}.$$

Remember that ξ is between x and $x+h$, but its exact value is not known. If we wish to calculate $f'(x)$, we cannot evaluate $f''(\xi)$, so we approximate the derivative by dropping the last term. That is, we calculate $[f(x+h) - f(x)]/h$ as an approximation[1] to $f'(x)$. In so doing, we have dropped the last term. If there is a finite bound on $f''(z)$ on the interval in question then the dropped term is bounded by a constant times h. That is,

$$\boxed{f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)} \tag{9.1}$$

The error that we incur when we approximate $f'(x)$ by calculating $[f(x+h) - f(x)]/h$ is called *truncation error*. It has nothing to do with the kind of error that you get when you do calculations with a computer with limited precision; even if you worked in infinite precision, you would still have truncation error.

The truncation error can be made small by making h small. However, as h gets smaller, precision will be lost in equation 9.1 due to subtractive cancellation. The error in calculation for small h is called *roundoff error*. Generally the roundoff error will increase as h decreases. Thus there is a nonzero h for which

[1] This approximation for $f'(x)$ should remind you of the definition of $f'(x)$ as a limit.

the sum of these two errors is minimized. See Example Problem 9.5 for an example of this.

The truncation error for this approximation is $\mathcal{O}(h)$. We may want a more precise approximation. By now, you should know that any calculation starts with Taylor's Theorem:

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(\xi_1)}{3!}h^3 \\ f(x-h) &= f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(\xi_2)}{3!}h^3 \end{aligned}$$

By subtracting these two lines, we get

$$f(x+h) - f(x-h) = 2f'(x)h + \frac{f'''(\xi_1) + f'''(\xi_2)}{3!}h^3.$$

Thus

$$\begin{aligned} 2f'(x)h &= f(x+h) - f(x-h) - \frac{f'''(\xi_1) + f'''(\xi_2)}{3!}h^3 \\ f'(x) &= \frac{f(x+h) - f(x-h)}{2h} - \left[\frac{f'''(\xi_1) + f'''(\xi_2)}{2}\right]\frac{h^2}{6} \end{aligned}$$

If $f'''(x)$ is continuous, then there is some ξ between ξ_1, ξ_2 such that $f'''(\xi) = \frac{f'''(\xi_1)+f'''(\xi_2)}{2}$. (This is the MVT at work.) Assuming some uniform bound on $f'''(\cdot)$, we get

$$\boxed{f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}\left(h^2\right)} \tag{9.2}$$

In some situations it may be necessary to use evaluations of a function at "odd" places to approximate a derivative. These are usually straightforward to derive, involving the use of Taylor's Theorem. The following examples illustrate:

Example Problem 9.1. *Use evaluations of f at $x+h$ and $x+2h$ to approximate $f'(x)$, assuming $f(x)$ is an* analytic *function,* i.e.*, one with infintely many derivatives.* Solution: *First use Taylor's Theorem to expand $f(x+h)$ and $f(x+2h)$, then subtract to get some factor of $f'(x)$:*

$$\begin{aligned} f(x+2h) &= f(x) + 2hf'(x) + \tfrac{4h^2}{2!}f''(x) + \tfrac{8h^3}{3!}f'''(x) + \tfrac{16h^4}{4!}f^{(4)}(x) + \dots \\ f(x+h) &= f(x) + hf'(x) + \tfrac{h^2}{2!}f''(x) + \tfrac{h^3}{3!}f'''(x) + \tfrac{h^4}{4!}f^{(4)}(x) + \dots \\ \hline f(x+2h) - f(x+h) &= hf'(x) + \tfrac{3h^2}{2!}f''(x) + \tfrac{7h^3}{3!}f'''(x) + \tfrac{15h^4}{4!}f^{(4)}(x) + \dots \\ (f(x+2h) - f(x+h))/h &= f'(x) + \tfrac{3h}{2!}f''(x) + \tfrac{7h^2}{3!}f'''(x) + \tfrac{15h^3}{4!}f^{(4)}(x) + \dots \end{aligned}$$

Thus $(f(x+2h) - f(x+h))/h = f'(x) + \mathcal{O}(h)$ ⊣

Example Problem 9.2. *Show that*

$$\frac{4f(x+h) - f(x+2h) - 3f(x)}{2h} = f'(x) + \mathcal{O}\left(h^2\right),$$

for f with sufficient number of derivatives Solution: *In this case we do not have to find the approximation scheme, it is given to us. We only have to expand the appropriate terms with Taylor's Theorem. As before:*

$$\begin{aligned}
f(x+h) &= f(x) + hf'(x) + \tfrac{h^2}{2!}f''(x) + \tfrac{h^3}{3!}f'''(x) + \dots \\
4f(x+h) &= 4f(x) + 4hf'(x) + \tfrac{4h^2}{2!}f''(x) + \tfrac{4h^3}{3!}f'''(x) + \dots \\
f(x+2h) &= f(x) + 2hf'(x) + \tfrac{4h^2}{2!}f''(x) + \tfrac{8h^3}{3!}f'''(x) + \dots \\
\hline
4f(x+h) - f(x+2h) &= 3f(x) + 2hf'(x) + 0f''(x) + \tfrac{-4h^3}{3!}f'''(x) + \dots \\
4f(x+h) - f(x+2h) - 3f(x) &= 2hf'(x) + \tfrac{-4h^3}{3!}f'''(x) + \dots \\
\left(4f(x+h) - f(x+2h) - 3f(x)\right)/2h &= f'(x) + \tfrac{-2h^2}{3!}f'''(x) + \dots
\end{aligned}$$

⊣

9.1.1 Approximating the Second Derivative

Suppose we want to approximate the second derivative of some blackbox function $f(x)$. Again, start with Taylor's Theorem:

$$\begin{aligned}
f(x+h) &= f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{3!}h^3 + \frac{f^{(4)}(x)}{4!}h^4 + \dots \\
f(x-h) &= f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{3!}h^3 + \frac{f^{(4)}(x)}{4!}h^4 - \dots
\end{aligned}$$

Now *add* the two series to get

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + 2\frac{f^{(4)}(x)}{4!}h^4 + 2\frac{f^{(6)}(x)}{6!}h^6 + \dots$$

Then let

$$\begin{aligned}
\psi(h) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} &= f''(x) + 2\frac{f^{(4)}(x)}{4!}h^2 + 2\frac{f^{(6)}(x)}{6!}h^4 + \dots, \\
&= f''(x) + \sum_{k=1}^{\infty} b_{2k}h^{2k}.
\end{aligned}$$

Thus we can use Richardson Extrapolation on $\psi(h)$ to get higher order approximations.

This derivation also gives us the *centered difference* approximation to the second derivative:

$$\boxed{f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \mathcal{O}\left(h^2\right).} \tag{9.3}$$

9.2 Richardson Extrapolation

The centered difference approximation gives a truncation error of $\mathcal{O}\left(h^2\right)$, which is better than $\mathcal{O}\left(h\right)$. Can we do better? Let's define

$$\phi(h) = \frac{1}{2h}\left[f(x+h) - f(x-h)\right].$$

Had we expanded the Taylor's Series for $f(x+h), f(x-h)$ to more terms we would have seen that

$$\phi(h) \quad = \quad f'(x) + a_2h^2 + a_4h^4 + a_6h^6 + a_8h^8 + \ldots$$

The constants a_i are a function of $f^{(i+1)}(x)$ only. (In fact, they should take the value of $\frac{f^{(i+1)}(x)}{(i+1)!}$.) What happens if we now calculate $\phi(h/2)$?

$$\phi(h/2) \quad = \quad f'(x) + \frac{1}{4}a_2h^2 + \frac{1}{16}a_4h^4 + \frac{1}{64}a_6h^6 + \frac{1}{256}a_8h^8 + \ldots$$

But we can combine this with $\phi(h)$ to get better accuracy. We have to be a little tricky, but we can get the $\mathcal{O}\left(h^2\right)$ terms to cancel by taking the right multiples of these two approximations:

$$\begin{aligned}
\phi(h) - 4\phi(h/2) &= -3f'(x) + \frac{3}{4}4h^4 + \frac{15}{16}a_6h^6 + \frac{63}{64}a_8h^8 + \ldots \\
\frac{4\phi(h/2) - \phi(h)}{3} &= f'(x) - \frac{1}{4}4h^4 - \frac{5}{16}a_6h^6 - \frac{21}{64}a_8h^8 + \ldots
\end{aligned}$$

This approximation has a truncation error of $\mathcal{O}\left(h^4\right)$.

This technique of getting better approximations is known as the *Richardson Extrapolation*, and can be repeatedly applied. We will also use this technique later to get better quadrature rules–that is, ways of approximating the definite integral of a function.

9.2.1 Abstracting Richardson's Method

We now discuss Richardson's Method in a more abstract framework. Suppose you want to calculate some quantity L, and have found, through theory, some approximation:

$$\phi(h) = L + \sum_{k=1}^{\infty} a_{2k}h^{2k}.$$

Let

$$D(n,0) = \phi\left(\frac{h}{2^n}\right).$$

Now define

$$\boxed{D(n,m) = \frac{4^m D(n,m-1) - D(n-1,m-1)}{4^m - 1}.} \tag{9.4}$$

We will be interested in calculating $D(n,n)$ for some n. We claim that

$$D(n,n) = L + \mathcal{O}\left(h^{2(n+1)}\right).$$

First we examine the recurrence for $D(n,m)$. As in divided differences, we use a pyramid table:

$$\begin{array}{lllll} D(0,0) & & & & \\ D(1,0) & D(1,1) & & & \\ D(2,0) & D(2,1) & D(2,2) & & \\ \vdots & \vdots & \vdots & \ddots & \\ D(n,0) & D(n,1) & D(n,2) & \cdots & D(n,n) \end{array}$$

By definition we know how to calculate the first column of this table; every other entry in the table depends on two other entries, one directly to the left, and the other to the left and up one space. Thus to calculate $D(n,n)$ we have to compute this whole lower triangular array.

We want to show that $D(n,n) = L + \mathcal{O}\left(h^{2(n+1)}\right)$, that is $D(n,n)$ is a $\mathcal{O}\left(h^{2(n+1)}\right)$ approximation to L. The following theorem gives this result:

Theorem 9.3 (Richardson Extrapolation)**.** *There are constants $a_{k,m}$ such that*

$$D(n,m) = L + \sum_{k=m+1}^{\infty} a_{k,m}\left(\frac{h}{2^n}\right)^{2k} \qquad (0 \le m \le n).$$

The proof is by an easy, but tedious, induction. We skip the proof.

9.2.2 Using Richardson Extrapolation

We now try out the technique on an example or two.

Example Problem 9.4. *Approximate the derivative of $f(x) = \log x$ at $x = 1$.*
Solution: *The real answer is $f'(1) = 1/1 = 1$, but our computer doesn't know that. Define*

$$\phi(h) = \frac{1}{2h}\left[f(1+h) - f(1-h)\right] = \frac{\log\frac{1+h}{1-h}}{2h}.$$

Let's use $h = 0.1$. We now try to find $D(2,2)$, which is supposed to be a $\mathcal{O}\left(h^6\right)$ approximation to $f'(1) = 1$:

$n\backslash m$	0	1	2
0	$\frac{\log\frac{1.1}{0.9}}{0.2} \approx 1.003353477$		
1	$\frac{\log\frac{1.05}{0.95}}{0.1} \approx 1.000834586$	≈ 0.999994954	
2	$\frac{\log\frac{1.025}{0.975}}{0.05} \approx 1.000208411$	≈ 0.999999686	≈ 1.000000002

This shows that the Richardson method is pretty good. However, notice that for this simple example, we have, already, that $\phi(0.00001) \approx 0.999999999$. ⊣

Example Problem 9.5. *Consider the ugly function:*

$$f(x) = \arctan(x).$$

Attempt to find $f'(\sqrt{2})$. *Recall that* $f'(x) = \frac{1}{1+x^2}$, *so the value that we are seeking is* $\frac{1}{3}$. Solution: *Let's use* $h = 0.01$. *We now try to find* $D(2,2)$, *which is supposed to be a* $\mathcal{O}\left(h^6\right)$ *approximation to* $\frac{1}{3}$*:*

$n\backslash m$	0	1	2
0	0.333339506181068		
1	0.333334876543723	0.333333333331274	
2	0.33333371913582	0.333333333333186	0.333333333333313

Note that we have some motivation to use Richardson's method in this case: If we let

$$\phi(h) = \frac{1}{2h}\left[f(\sqrt{2}+h) - f(\sqrt{2}-h)\right],$$

then making h *small gives a good approximation to* $f'\left(\sqrt{2}\right)$ until subtractive cancelling takes over. *The following table illustrates this:*

h	$\phi(h)$
1.0	0.39269908169872408
0.1	0.33395069677431943
0.01	0.33333950618106845
0.001	0.33333339506169679
0.0001	0.33333333395058062
1×10^{-5}	0.33333333334106813
1×10^{-6}	0.33333333332441484
1×10^{-7}	0.33333333315788138
1×10^{-8}	0.33333332760676626
1×10^{-9}	0.33333336091345694
1×10^{-10}	0.333333360913457
1×10^{-11}	0.333333360913457
1×10^{-12}	0.33339997429493451
1×10^{-13}	0.33306690738754696
1×10^{-14}	0.33306690738754696
1×10^{-15}	0.33306690738754691
1×10^{-16}	0

The data are illustrated in Figure 9.1. Notice that $\phi(h)$ *gives at most 10 decimal places of accuracy, then begins to deteriorate; Note however, we get 13 decimal places from* $D(2,2)$. ⊣

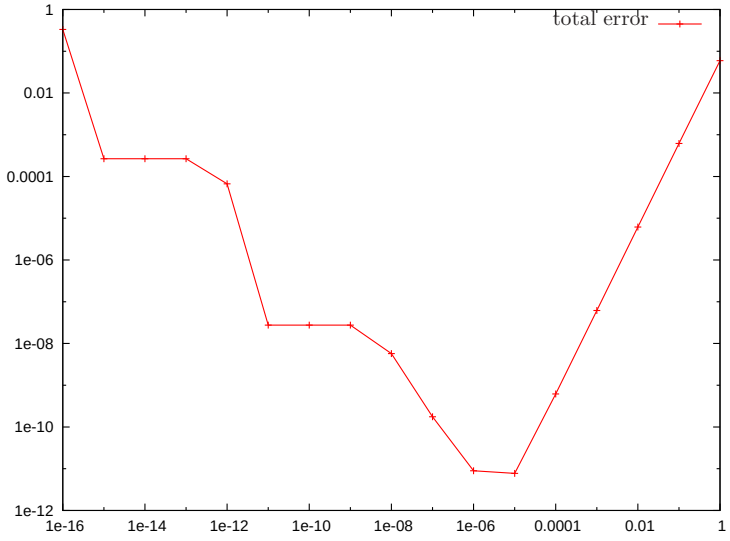


Figure 9.1: The total error for the centered difference approximation to $f'(\sqrt{2})$ is shown versus h. The total error is the sum of a truncation term which decreases as h decreases, and a roundoff term which increases. The optimal h value is around 1×10^{-5}. Note that Richardson's $D(2,2)$ approximation with $h = 0.01$ gives much better results than this optimal h.

Exercises

(9.1) Derive the approximation

$$f'(x) \approx \frac{4f(x+h) - 3f(x) - f(x-2h)}{6h}$$

using Taylor's Theorem.

(a) Assuming that $f(x)$ has bounded derivatives, give the accuracy of the above approximation. Your answer should be something like $\mathcal{O}\left(h^{?}\right)$.

(b) Let $f(x) = x^3$. Approximate $f'(0)$ with this approximation, using $h = \frac{1}{4}$.

(9.2) Let $f(x)$ be an *analytic* function, *i.e.*, one which is infinitely differentiable. Let $\psi(h)$ be the centered difference approximation to the first derivative:

$$\psi(h) = \frac{f(x+h) - f(x-h)}{2h}$$

(a) Show that $\psi(h) = f'(x) + \frac{h^2}{3!}f'''(x) + \frac{h^4}{5!}f^{(5)}(x) + \frac{h^6}{7!}f^{(7)}(x) + \ldots$

(b) Show that

$$\frac{8\left(\psi(h) - \psi(h/2)\right)}{h^2} = f'''(x) + \mathcal{O}\left(h^2\right).$$

(9.3) Derive the approximation

$$f'(x) \approx \frac{4f(x+3h) + 5f(x) - 9f(x-2h)}{30h}$$

using Taylor's Theorem.

(a) What order approximation is this? (Assume $f(x)$ has bounded derivatives of arbitrary order.)

(b) Use this formula to approximate $f'(0)$, where $f(x) = x^4$, and $h = 0.1$

(9.4) Suppose you want to know quantity Q, and can approximate it with some formula, say $\phi(h)$, which depends on parameter h, and such that $\phi(h) = Q + a_1h + a_2h^2 + a_3h^3 + a_4h^4 + \ldots$ Find some linear combination of $\phi(h)$ and $\phi(-h)$ which is a $\mathcal{O}\left(h^2\right)$ approximation to Q.

(9.5) Assuming that $\phi(h) = Q + a_2h^2 + a_4h^4 + a_6h^6 \ldots$, find some combination of $\phi(h), \phi(h/3)$ which is a $\mathcal{O}\left(h^4\right)$ approximation to Q.

(9.6) Let λ be some number in $(0, 1)$. Assuming that $\phi(h) = Q + a_2h^2 + a_4h^4 + a_6h^6 \ldots$, find some combination of $\phi(h), \phi(\lambda h)$ which is a $\mathcal{O}\left(h^4\right)$ approximation to Q. To make the constant associated with the h^4 term small in magnitude, what should you do with λ? Is this practical? Note that the method of Richardson Extrapolation that we considered used the value $\lambda = 1/2$.

(9.7) Assuming that $\phi(h) = Q + a_2h^2 + a_4h^4 + a_6h^6 \ldots$, find some combination of $\phi(h), \phi(h/4)$ which is a $\mathcal{O}\left(h^4\right)$ approximation to Q.

(9.8) Suppose you have some great computational approximation to the quantity Q such that $\psi(h) = Q + a_3h^3 + a_6h^6 + a_9h^9 \ldots$ Can you find some combination of $\psi(h), \psi(h/2)$ which is a $\mathcal{O}\left(h^6\right)$ approximation to Q?

(9.9) Complete the following Richardson's Extrapolation Table, assuming the first column consists of values $D(n,0)$ for $n = 0, 1, 2$:

$n\backslash m$	0	1	2
0	2		
1	1.5	?	
2	1.25	?	?

(See equation 9.4 if you've forgotten the definitions.)

(9.10) Write code to complete a Richardson's Method table, given the first column.

Your m-file should have header line like:

```
function Dnn = richardsons(col0)
```

where `Dnn` is the value at the lower left corner of the table, $D(n,n)$ while `col0` is the column of $n+1$ values $D(i,0)$, for $i = 0, 1, \ldots, n$. Test your code on the following input:

```
octave:1> col0 = [1 0.5 0.25 0.125 0.0625 0.03125];
octave:2> richardsons(col0)
ans = 0.019042
```

(a) What do you get when you try the following?

```
octave:5> col0 = [1.5 0.5 1.5 0.5 1.5 0.5 1.5];
octave:6> richardsons(col0)
```

(b) What do you get when you try the following?

```
octave:7> col0 = [0.9 0.99 0.999 0.9999 0.99999];
octave:8> richardsons(col0)
```

Chapter 10

Integrals and Quadrature

10.1 The Definite Integral

Often enough the numerical analyst is presented with the challenge of finding the definite integral of some function:

$$\int_a^b f(x)\,\mathrm{d}x.$$

In your golden years of Calculus, you learned the Fundamental Theorem of Calculus, which claims that if $f(x)$ is continuous, and $F(x)$ is an antiderivative of $f(x)$, then

$$\int_a^b f(x)\,\mathrm{d}x = F(b) - F(a).$$

What you might not have been told in Calculus is there are some functions for which a closed form antiderivative does not exist or at least is not known to humankind. Nevertheless, you may find yourself in a situation where you have to evaluate an integral for just such an integrand. An approximation will have to do.

10.1.1 Upper and Lower Sums

We will review the definition of the Riemann integral of a function. A *partition* of an interval $[a, b]$ is a finite, ordered collection of nodes x_i:

$$a = x_0 < x_1 < x_2 < \cdots < x_n = b.$$

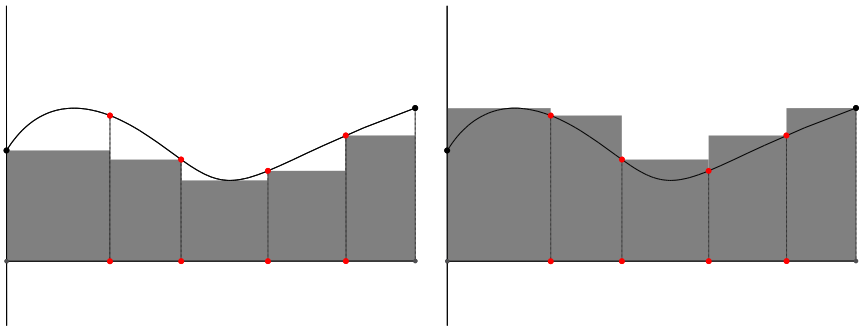
Given such a partition, P, define the upper and lower bounds on each subinterval $[x_j, x_{j+1}]$ as follows:

$$m_i = \inf\{f(x) \mid x_i \le x \le x_{i+1}\}$$
$$M_i = \sup\{f(x) \mid x_i \le x \le x_{i+1}\}$$

Then for this function f and partition P, define the upper and lower sums:

$$L(f,P) = \sum_{i=0}^{n-1} m_i \left(x_{i+1} - x_i\right)$$
$$U(f,P) = \sum_{i=0}^{n-1} M_i \left(x_{i+1} - x_i\right)$$

We can interpret the upper and lower sums graphically as the sums of areas of rectangles defined by the function f and the partition P, as in Figure 10.1.



(a) The Lower Sum (b) The Upper Sum

Figure 10.1: The (a) lower, and (b) upper sums of a function on a given interval are shown. These approximations to the integral are the sums of areas of rectangles. Note that the lower sums are an underestimate, and the upper sums an overestimate of the integral.

Notice a few things about the upper, lower sums:

(i) $L(f,P) \leq U(f,P)$.

(ii) If we switch to a "better" partition (*i.e.*, a finer one), we expect that $L(f,\cdot)$ increases and $U(f,\cdot)$ decreases.

The notion of integrability familiar from Calculus class (that is Riemann Integrability) is defined in terms of the upper and lower sums.

Definition 10.1. *A function f is Riemann Integrable over interval $[a,b]$ if*

$$\sup_P L(f,P) = \inf_P U(f,P),$$

where the supremum and infimum are over all partitions of the interval $[a,b]$. Moreover, in case $f(x)$ is integrable, we define the integral

$$\int_a^b f(x)\,\mathrm{d}x = \inf_P U(f,P),$$

You may recall the following

Theorem 10.2. *Every continuous function on a closed bounded interval of the real line is Riemann Integrable (on that interval).*

Continuity is sufficient, but not necessary.

Example 10.3. *Consider the* Heaviside function*:*

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & 0 \leq x \end{cases}$$

This function is not continuous on any interval containing 0*, but is Riemann Integrable on every closed bounded interval.*

Example 10.4. *Consider the* Dirichlet function*:*

$$f(x) = \begin{cases} 0 & x \text{ rational} \\ 1 & x \text{ irrational} \end{cases}$$

For any partition P of any interval $[a, b]$, we have $L(f, P) = 0$, while $U(f, P) =$ 1*, so*

$$\sup_P L(f, P) = 0 \neq 1 = \inf_P U(f, P),$$

so this function is not *Riemann Integrable.*

10.1.2 Approximating the Integral

The definition of the integral gives a simple method of approximating an integral $\int_a^b f(x)\,\mathrm{d}x$. The method cuts the interval $[a, b]$ into a partition of n equal subintervals $x_i = a + \frac{b-a}{n}$, for $i = 0, 1, \ldots, n$. The algorithm then has to somehow find the supremum and infimum of $f(x)$ on each interval $[x_i, x_{i+1}]$. The integral is then approximated by the mean of the lower and upper sums:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{1}{2}\left(L(f, P) + U(f, P)\right).$$

Because the value of the integral is between $L(f, P)$ and $U(f, P)$, this approximation has error at most

$$\frac{1}{2}\left(U(f, P) - L(f, P)\right).$$

Note that in general, or for a black box function, it is usually not feasible to find the suprema and infima of $f(x)$ on the subintervals, and thus the lower and upper sums cannot be calculated. However, if some information is known about the function, it becomes easier:

Example 10.5. *Consider for example, using this method on some function $f(x)$ which is* monotone increasing*, that is $x \leq y$ implies $f(x) \leq f(y)$. In this case, the infimum of $f(x)$ on each interval occurs at the leftmost endpoint, while*

the supremum occurs at the right hand endpoint. Thus for this partition, P, we have

$$L(f,P) = \sum_{k=0}^{n-1} m_i \left|x_{k+1} - x_k\right| = \frac{|b-a|}{n} \sum_{k=0}^{n-1} f(x_k)$$

$$U(f,P) = \sum_{k=0}^{n-1} M_i \left|x_{k+1} - x_k\right| = \frac{|b-a|}{n} \sum_{k=0}^{n-1} f(x_{k+1}) = \frac{|b-a|}{n} \sum_{k=1}^{n} f(x_k)$$

Then the error of the approximation is

$$\frac{1}{2}\left(U(f,P) - L(f,P)\right) = \frac{1}{2}\frac{|b-a|}{n}\left[f(x_n) - f(x_0)\right] = \frac{|b-a|\left[f(b) - f(a)\right]}{2n}.$$

10.1.3 Simple and Composite Rules

For the remainder of this chapter we will study "simple" quadrature rules, *i.e.*, rules which approximate the integral of a function, $f(x)$ over an interval $[a,b]$ by means of a number of evaluations of f at points in this interval. The error of a simple quadrature rule usually depends on the function f, and the width of the interval $[a,b]$ to some power which is determined by the rule. That is we usually think of a simple rule as being applied to a small interval.

To use a simple rule on a larger interval, we usually cast it into a "composite" rule. Thus the trapezoidal rule, which we will study next becomes the composite trapezoidal rule. The means of extending a simple rule to a composite rule is straightforward: Partition the given interval into subintervals, apply the simple rule to each subinterval, and sum the results. Thus, for example if the interval in question is $[\alpha,\beta]$, and the partition is $\alpha = x_0 < x_1 < x_2 < \ldots < x_n = \beta$, we have

$$\text{composite rule on } [\alpha,\beta] = \sum_{i=0}^{n-1} \text{simple rule applied to } [x_i, x_{i+1}].$$

10.2 Trapezoidal Rule

Suppose we are trying to approximate the integral

$$\int_a^b f(x)\,\mathrm{d}x,$$

for some unpleasant or black box function $f(x)$.

The trapezoidal rule approximates the integral

$$\int_a^b f(x)\,\mathrm{d}x$$

by the (signed) area of the trapezoid through the points $(a, f(a))$, $(b, f(b))$, and with one side the segment from a to b. See Figure 10.2.

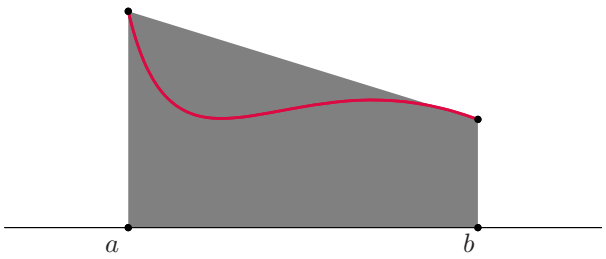


Figure 10.2: The trapezoidal rule for approximating the integral of a function over $[a, b]$ is shown.

By old school math, we can find this signed area easily. This gives the (simple) trapezoidal rule:

$$\int_a^b f(x)\,\mathrm{d}x \approx (b-a)\,\frac{f(a)+f(b)}{2}.$$

The composite trapezoidal rule can be written in a simplified form, one which you saw in your calculus class, if the interval in question is partitioned into equal width subintervals. That is if you let $[\alpha, \beta]$ be partitioned by

$$\alpha = x_0 < x_1 < x_2 < x_3 < \ldots < x_n = \beta,$$

with $x_i = \alpha + ih$, where $h = (\beta - \alpha)/n$, then the composite trapezoidal rule is

$$\int_\alpha^\beta f(x)\,\mathrm{d}x = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x \approx \frac{1}{2}\sum_{i=0}^{n-1} (x_{i+1}-x_i)\left[f(x_i)+f(x_{i+1})\right]. \quad (10.1)$$

Since each subinterval has equal width, $x_{i+1} - x_i = h$, and we have

$$\boxed{\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{2}\sum_{i=0}^{n-1}\left[f(x_i)+f(x_{i+1})\right].} \quad (10.2)$$

In your calculus class, you saw this in the less comprehensible form:

$$\int_a^b f(x)\,\mathrm{d}x \approx h\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right].$$

Note that the composite trapezoidal rule for equal subintervals is the same as the approximation we found for increasing functions in Example 10.5.

Example Problem 10.6. *Approximate the integral*

$$\int_0^2 \frac{1}{1+x^2}\,\mathrm{d}x$$

by the composite trapezoidal rule with a partition of equally spaced points, for $n = 2$. Solution: *We have* $h = \frac{2-0}{2} = 1$, *and* $f(x_0) = 1, f(x_1) = \frac{1}{2}, f(x_2) = \frac{1}{5}$. *Then the composite trapezoidal rule gives the value*

$$\frac{1}{2}\left[f(x_0) + f(x_1) + f(x_1) + f(x_2)\right] = \frac{1}{2}\left[1 + 1 + \frac{1}{5}\right] = \frac{11}{10}.$$

The actual value is $\arctan 2 \approx 1.107149$, *and our approximation is correct to two decimal places.* ⊣

10.2.1 How Good is the Composite Trapezoidal Rule?

We consider the composite trapezoidal rule for partitions of equal subintervals. Let $p_i(x)$ be the polynomial of degree ≤ 1 that interpolates $f(x)$ at x_i, x_{i+1}. Let

$$I_i = \int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x, \quad T_i = \int_{x_i}^{x_{i+1}} p_i(x)\,\mathrm{d}x = (x_{i+1} - x_i)\,\frac{p_i(x_i) + p_i(x_{i+1})}{2} = \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right).$$

That's right: the composite trapezoidal rule approximates the integral of $f(x)$ over $[x_i, x_{i+1}]$ by the integral of $p_i(x)$ over the same interval.

Now recall our theorem on polynomial interpolation error. For $x \in [x_i, x_{i+1}]$, we have

$$f(x) - p_i(x) = \frac{1}{(2)!} f^{(2)}(\xi_x)\,(x - x_i)\,(x - x_{i+1}),$$

for some $\xi_x \in [x_i, x_{i+1}]$. Recall that ξ_x depends on x. To make things simpler, call it $\xi(x)$.

Now integrate:

$$I_i - T_i = \int_{x_i}^{x_{i+1}} f(x) - p_i(x)\,\mathrm{d}x = \frac{1}{2}\int_{x_i}^{x_{i+1}} f''(\xi(x))\,(x - x_i)\,(x - x_{i+1})\,\mathrm{d}x.$$

We will now attack the integral on the right hand side. Recall the following theorem:

Theorem 10.7 (Mean Value Theorem for Integrals)**.** *Suppose* f *is continuous,* g *is Riemann Integrable and does not change sign on* $[\alpha, \beta]$. *Then there is some* $\zeta \in [\alpha, \beta]$ *such that*

$$\int_\alpha^\beta f(x)g(x)\,\mathrm{d}x = f(\zeta)\int_\alpha^\beta g(x)\,\mathrm{d}x.$$

We use this theorem on our integral. Note that $(x - x_i)\,(x - x_{i+1})$ is non-positive on the interval of question, $[x_i, x_{i+1}]$. We assume continuity of $f''(x)$, and wave our hands to get continuity of $f''(\xi(x))$. Then we have

$$I_i - T_i = \frac{1}{2} f''(\xi)\int_{x_i}^{x_{i+1}} (x - x_i)\,(x - x_{i+1})\,\mathrm{d}x,$$

for some $\xi_i \in [x_i, x_{i+1}]$. By boring calculus and algebra, we find that

$$\int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1})\, \mathrm{d}x = -\frac{h^3}{6}.$$

This gives

$$I_i - T_i = -\frac{h^3}{12} f''(\xi_i),$$

for some $\xi_i \in [x_i, x_{i+1}]$.

We now sum over all subintervals to find the total error of the composite trapezoidal rule

$$E = \sum_{i=0}^{n-1} I_i - T_i = -\frac{h^3}{12} \sum_{i=0}^{n-1} f''(\xi_i) = -\frac{(b-a)\,h^2}{12} \left[\frac{1}{n} \sum_{i=0}^{n-1} f''(\xi_i) \right].$$

On the far right we have an average value, $\frac{1}{n}\sum_{i=0}^{n-1} f''(\xi_i)$, which lies between the least and greatest values of f'' on the inteval $[a, b]$, and thus by the IVT, there is some ξ which takes this value. So

$$E = -\frac{(b-a)\,h^2}{12} f''(\xi)$$

This gives us the theorem:

Theorem 10.8 (Error of the Composite Trapezoidal Rule)**.** *Let $f''(x)$ be continuous on $[a, b]$. Let T be the value of the trapezoidal rule applied to $f(x)$ on this interval with a partition of uniform spacing, h, and let $I = \int_a^b f(x)\,\mathrm{d}x$. Then there is some $\xi \in [a, b]$ such that*

$$I - T = -\frac{(b-a)\,h^2}{12} f''(\xi).$$

Note that this theorem tells us not only the magnitude of the error, but the sign as well. Thus if, for example, $f(x)$ is concave up and thus f'' is positive, then $I - T$ will be negative, *i.e.*, the trapezoidal rule gives an *overestimate* of the integral I. See Figure 10.3.

10.2.2 Using the Error Bound

Example Problem 10.9. *How many intervals are required to approximate the integral*

$$\ln 2 = I = \int_0^1 \frac{1}{1+x}\,\mathrm{d}x$$

to within 1×10^{-10}? Solution: *We have $f(x) = \frac{1}{1+x}$, thus $f'(x) = -\frac{1}{(1+x)^2}$. And $f''(x) = \frac{2}{(1+x)^3}$. Thus $f''(\xi)$ is continuous and bounded by 2 on $[0, 1]$. If we use n equal subintervals then Theorem 10.8 tells us the error will be*

$$-\frac{1-0}{12}\left(\frac{1-0}{n}\right)^2 f''(\xi) = -\frac{f''(\xi)}{12n^2}.$$

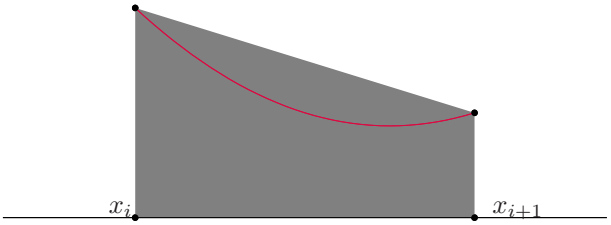


Figure 10.3: The trapezoidal rule is an overestimate for a function which is concave up, *i.e.*, has positive second derivative.

To make this smaller than 1×10^{-10}*, in absolute value, we need only take*

$$\frac{1}{6n^2} \leq 1 \times 10^{-10},$$

and so $n \geq \sqrt{\frac{1}{6}} \times 10^5$ *suffices. Because* $f''(x)$ *is positive on this interval, the trapezoidal rule will be an overestimate.* ⊣

Example Problem 10.10. *How many intervals are required to approximate the integral*

$$\int_0^2 x^3 - 1 \, \mathrm{d}x$$

to within 1×10^{-6}*?* Solution: *We have* $f(x) = x^3 - 1$*, thus* $f'(x) = 3x^2$*, and* $f''(x) = 6x$*. Thus* $f''(\xi)$ *is continuous and bounded by* 12 *on* $[0, 2]$*. If we use* n *equal subintervals then by Theorem 10.8 the error will be*

$$-\frac{2-0}{12}\left(\frac{2-0}{n}\right)^2 f''(\xi) = -\frac{2f''(\xi)}{3n^2}.$$

To make this smaller than 1×10^{-6}*, in absolute value, it suffices to take*

$$\frac{24}{3n^2} \leq 1 \times 10^{-6},$$

and so $n \geq \sqrt{8} \times 10^3$ *suffices. Because* $f''(x)$ *is positive on this interval, the trapezoidal rule will be an overestimate.* ⊣

10.3 Romberg Algorithm

Theorem 10.8 tells us, approximately, that the error of the composite trapezoidal rule approximation is $\mathcal{O}\left(h^2\right)$. If we halve h, the error is quartered. Sometimes

we want to do better than this. We'll use the same trick that we did from Richardson extrapolation. In fact, the forms are exactly the same.

Towards this end, suppose that f, a, b are given. For a given n, we are going to use the trapezoidal rule on a partition of 2^n equal subintervals of $[a, b]$. That is $h = \frac{b-a}{2^n}$. Then define

$$\begin{aligned}\phi(n) &= \frac{1}{2}\frac{b-a}{2^n}\sum_{i=0}^{2^n-1} f(x_i) + f(x_{i+1}) \\ &= \frac{b-a}{2^n}\left[\frac{f(a)}{2} + \frac{f(b)}{2} + \sum_{i=1}^{2^n-1} f\left(a + i\frac{b-a}{2^n}\right)\right].\end{aligned}$$

The intervals used to calculate $\phi(n+1)$ are half the size of those for $\phi(n)$. As mentioned above, this means the error is one quarter.

It turns out that if we had proved the error theorem differently, we would have proved the relation:

$$\phi(n) = \int_a^b f(x)\,\mathrm{d}x + a_2 h_n^2 + a_4 h_n^4 + a_6 h_n^6 + a_8 h_n^8 + \ldots,$$

where $h_n = \frac{b-a}{2^n}$. The constants a_i are a function of $f^{(i)}(x)$ only. This should look just like something from Chapter 9. What happens if we now calculate $\phi(n+1)$? We have

$$\begin{aligned}\phi(n+1) &= \int_a^b f(x)\,\mathrm{d}x + a_2 h_{n+1}^2 + a_4 h_{n+1}^4 + a_6 h_{n+1}^6 + a_8 h_{n+1}^8 + \ldots, \\ &= \int_a^b f(x)\,\mathrm{d}x + \frac{1}{4}a_2 h_n^2 + \frac{1}{16}a_4 h_n^4 + \frac{1}{64}a_6 h_n^6 + \frac{1}{256}a_8 h_n^8 + \ldots.\end{aligned}$$

This happens because $h_{n+1} = \frac{b-a}{2^{n+1}} = \frac{1}{2}\frac{b-a}{2^n} = \frac{h_n}{2}$. As with Richardon's method for approximating derivatives, we now combine the right multiples of these:

$$\begin{aligned}\phi(n) - 4\phi(n+1) = &\quad -3\int_a^b f(x)\,\mathrm{d}x + \frac{3}{4}4h_n^4 + \frac{15}{16}a_6 h_n^6 + \frac{63}{64}a_8 h_n^8 + \ldots \\ \frac{4\phi(n) - \phi(n+1)}{3} = &\quad \int_a^b f(x)\,\mathrm{d}x - \frac{1}{4}4h_n^4 - \frac{5}{16}a_6 h_n^6 - \frac{21}{64}a_8 h_n^8 + \ldots\end{aligned}$$

This approximation has a truncation error of $\mathcal{O}\left(h_n^4\right)$.

Like in Richardson's method, we can use this to get better and better approximations to the integral. We do this by constructing a triangular array of approximations, each entry depending on two others. Towards this end, we let

$$R(n, 0) = \phi(n),$$

then define, for $m > 0$

$$\boxed{R(n,m) = \frac{4^m R(n, m-1) - R(n-1, m-1)}{4^m - 1}.} \tag{10.3}$$

The familiar pyramid table then is:

$$\begin{array}{lllll} R(0,0) & & & & \\ R(1,0) & R(1,1) & & & \\ R(2,0) & R(2,1) & R(2,2) & & \\ \vdots & \vdots & \vdots & \ddots & \\ R(n,0) & R(n,1) & R(n,2) & \cdots & R(n,n) \end{array}$$

Even though this is *exactly* the same as Richardon's method, it has another name: this is called the Romberg Algorithm.

Example Problem 10.11. *Approximating the integral*

$$\int_0^2 \frac{1}{1+x^2}\,\mathrm{d}x$$

by Romberg's Algorithm; find $R(1,1)$. Solution: *The first column is calculated by the trapezoidal rule. Successive columns are found by combining members of previous columns. So we first calculate* $R(0,0)$ *and* $R(1,0)$. *These are fairly simple, the first is the trapezoidal rule on a single subinterval, the second is the trapezoidal rule on two subintervals. Then*

$$\begin{aligned} R(0,0) &= \frac{2-0}{1}\frac{1}{2}\left[f(0)+f(2)\right] = \frac{6}{5}, \\ R(1,0) &= \frac{2-0}{2}\frac{1}{2}\left[f(0)+f(1)+f(1)+f(2)\right] = \frac{11}{10}. \end{aligned}$$

Then, using Romberg's Algorithm we have

$$R(1,1) = \frac{4R(1,0)-R(0,0)}{4-1} = \frac{\frac{44}{10}-\frac{12}{10}}{3} = \frac{32}{30} = 1.0\bar{6}.$$

⊣

At this point we are tempted to use Richardson's analysis. This would claim that $R(n,n)$ is a $\mathcal{O}\left(h_0^{2(n+1)}\right)$ approximation to the integral. However, $h_0 = b-a$, and need not be smaller than 1. This is a bit different from Richardson's method, where the original h is independently set before starting the triangular array; for Romberg's algorithm, h_0 is determined by a and b.

We can easily deal with this problem by picking some k such that $\frac{b-a}{2^k}$ is small enough, say smaller than 1. Then calculating the following array:

$$\begin{array}{lllll} R(k,0) & & & & \\ R(k+1,0) & R(k+1,1) & & & \\ R(k+2,0) & R(k+2,1) & R(k+2,2) & & \\ \vdots & \vdots & \vdots & \ddots & \\ R(k+n,0) & R(k+n,1) & R(k+n,2) & \cdots & R(k+n,n) \end{array}$$

Quite often Romberg's Algorithm is used to compute columns of this array. Subtractive cancelling or unbounded higher derivatives of $f(x)$ can make successive approximations *less* accurate. For this reason, entries in ever rightward columns are usually not calculated, rather lower entries in a single column are calculated instead. That is, the user calculates the array:

$$\begin{array}{ccccc}
R(k,0) & & & & \\
R(k+1,0) & R(k+1,1) & & & \\
R(k+2,0) & R(k+2,1) & R(k+2,2) & & \\
\vdots & \vdots & \vdots & \ddots & \\
R(k+n,0) & R(k+n,1) & R(k+n,2) & \cdots & R(k+n,n) \\
R(k+n+1,0) & R(k+n+1,1) & R(k+n+1,2) & \cdots & R(k+n+1,n) \\
R(k+n+2,0) & R(k+n+2,1) & R(k+n+2,2) & \cdots & R(k+n+2,n) \\
R(k+n+3,0) & R(k+n+3,1) & R(k+n+3,2) & \cdots & R(k+n+3,n) \\
\vdots & \vdots & \vdots & & \vdots
\end{array}$$

Then $R(k+n+l,n)$ makes a fine approximation to the integral as $l \to \infty$. Usually n is small, like 2 or 3.

10.3.1 Recursive Trapezoidal Rule

It turns out there is an efficient way of calculating $R(n+1,0)$ given $R(n,0)$; first notice from the above example that

$$\begin{aligned}
R(0,0) &= \frac{b-a}{1}\frac{1}{2}\left[f(a)+f(b)\right], \\
R(1,0) &= \frac{b-a}{2}\frac{1}{2}\left[f(a)+f(\frac{a+b}{2})+f(\frac{a+b}{2})+f(b)\right].
\end{aligned}$$

It would be best to calculate $R(1,0)$ without recalculating $f(a)$ and $f(b)$. It turns out this is possible. Let $h_n = \frac{b-a}{2^n}$, and recall that

$$R(n,0) = \phi(n) = h_n\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{2^n-1} f\left(a+ih_n\right)\right].$$

Thus

$$\begin{aligned}
R(n+1,0) = \phi(n+1) &= h_{n+1}\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{2^{n+1}-1} f\left(a+ih_{n+1}\right)\right], \\
&= \frac{1}{2}h_n\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{2^n-1} f\left(a+(2i-1)h_{n+1}\right) + f\left(a+(2i)\frac{1}{2}h_n\right)\right], \\
&= \frac{1}{2}h_n\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{2^n-1} f\left(a+ih_n\right) + \sum_{i=1}^{2^n-1} f\left(a+(2i-1)h_{n+1}\right)\right], \\
&= \frac{1}{2}R(n,0) + h_{n+1}\sum_{i=1}^{2^n-1} f\left(a+(2i-1)h_{n+1}\right).
\end{aligned}$$

Then calculating $R(n+1,0)$ requires only $2^n - 1$ additional evaluations of $f(x)$, instead of the $2^{n+1}+1$ usually required.

10.4 Gaussian Quadrature

The word *quadrature* refers to a method of approximating the integral of a function as the linear combination of the function at certain points, *i.e.*,

$$\boxed{\int_a^b f(x)\,\mathrm{d}x \approx A_0 f(x_0) + A_1 f(x_1) + \ldots A_n f(x_n),} \tag{10.4}$$

for some collection of nodes $\{x_i\}_{i=0}^n$, and weights $\{A_i\}_{i=0}^n$. Normally one finds the nodes and weights in a table somewhere; we expect a quadrature rule with more nodes to be more accurate in some sense–the tradeoff is in the number of evaluations of $f(\cdot)$. We will examine how these rules are created.

10.4.1 Determining Weights (Lagrange Polynomial Method)

Suppose that the nodes $\{x_i\}_{i=0}^n$ are given. An easy way to find "good" weights $\{A_i\}_{i=0}^n$ for these nodes is to rig them so the quadrature rule gives the integral of $p(x)$, the polynomial of degree $\leq n$ which interpolates $f(x)$ on these nodes. Recall

$$p(x) = \sum_{i=0}^n f(x_i)\ell_i(x),$$

where $\ell_i(x)$ is the i^{th} Lagrange polynomial. Thus our rigged approximation is the one that gives

$$\int_a^b f(x)\,\mathrm{d}x \approx \int_a^b p(x)\,\mathrm{d}x = \sum_{i=0}^n f(x_i)\int_a^b \ell_i(x)\,\mathrm{d}x.$$

If we let

$$A_i = \int_a^b \ell_i(x)\,\mathrm{d}x,$$

then we have a quadrature rule.

If $f(x)$ is a polynomial of degree $\leq n$ then $f(x) = p(x)$, and the quadrature rule is exact.

Example Problem 10.12. *Construct a quadrature rule on the interval* $[0, 4]$ *using nodes* $0, 1, 2$. Solution: *The nodes are given, we determine the weights by constructing the Lagrange Polynomials, and integrating them.*

$$\begin{aligned}
\ell_0(x) &= \frac{(x-1)(x-2)}{(0-1)(0-2)} = \frac{1}{2}(x-1)(x-2),\\
\ell_1(x) &= \frac{(x-0)(x-2)}{(1-0)(1-2)} = -(x)(x-2),\\
\ell_2(x) &= \frac{(x-0)(x-1)}{(2-0)(2-1)} = \frac{1}{2}(x)(x-1).
\end{aligned}$$

Then the weights are

$$\begin{aligned}
A_0 &= \int_0^4 \ell_0(x)\,\mathrm{d}x = \int_0^4 \frac{1}{2}(x-1)(x-2)\,\mathrm{d}x = \frac{8}{3},\\
A_1 &= \int_0^4 \ell_1(x)\,\mathrm{d}x = \int_0^4 -(x)(x-2)\,\mathrm{d}x = -\frac{16}{3},\\
A_2 &= \int_0^4 \ell_2(x)\,\mathrm{d}x = \int_0^4 \frac{1}{2}(x)(x-1)\,\mathrm{d}x = \frac{20}{3}.
\end{aligned}$$

Thus our quadrature rule is

$$\boxed{\int_0^4 f(x)\,\mathrm{d}x \approx \frac{8}{3}f(0) - \frac{16}{3}f(1) + \frac{20}{3}f(2).}$$

We expect this rule to be exact for a quadratic function $f(x)$. *To illustrate this, let* $f(x) = x^2 + 1$. *By calculus we have*

$$\int_0^4 x^2 + 1\ \mathrm{d}x = \frac{1}{3}x^3 + x\Big|_0^4 = \frac{64}{3} + 4 = \frac{76}{3}.$$

The approximation is

$$\int_0^4 x^2 + 1\ \mathrm{d}x \approx \frac{8}{3}\left[0+1\right] - \frac{16}{3}\left[1+1\right] + \frac{20}{3}\left[4+1\right] = \frac{76}{3}.$$

⊣

10.4.2 Determining Weights (Method of Undetermined Coefficients)

Using the Lagrange Polynomial Method to find the weights A_i is fine for a computer, but can be tedious (and error-prone) if done by hand (say, on an exam). The method of undetermined coefficients is a good alternative for finding the weights by hand, and for small n.

The idea behind the method is to find $n+1$ equations involving the $n+1$ weights. The equations are derived by letting the quadrature rule be exact for $f(x) = x^j$ for $j = 0, 1, \ldots, n$. That is, setting

$$\int_a^b x^j \, \mathrm{d}x = \sum_{i=0}^{n} A_i (x_i)^j.$$

For example, we reconsider Example Problem 10.12.

Example Problem 10.13. *Construct a quadrature rule on the interval* $[0, 4]$ *using nodes* $0, 1, 2$. Solution: *The method of undetermined coefficients gives the equations:*

$$\begin{aligned} \int_0^4 1 \, \mathrm{d}x &= 4 = A_0 + A_1 + A_2 \\ \int_0^4 x \, \mathrm{d}x &= 8 = A_1 + 2A_2 \\ \int_0^4 x^2 \, \mathrm{d}x &= 64/3 = A_1 + 4A_2. \end{aligned}$$

We perform Naïve Gaussian Elimination on the system:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 1 & 2 & 8 \\ 0 & 1 & 4 & 64/3 \end{array}\right)$$

We get the same weights as in Example Problem 10.12: $A_2 = \frac{20}{3}, A_1 = -\frac{16}{3}, A_0 = \frac{8}{3}$. ⊣

Notice the difference compared to the Lagrange Polynomial Method: undetermined coefficients requires solution of a linear system, while the former method calculates the weights "directly." Since we will not consider n to be very large, solving the linear system may not be too burdensome.

Moreover, the method of undetermined coefficients is useful in more general settings, as illustrated by the next example:

Example Problem 10.14. *Determine a "quadrature" rule of the form*

$$\int_0^1 f(x) \, \mathrm{d}x \approx Af(1) + Bf'(1) + Cf''(1)$$

that is exact for polynomials of highest possible degree. What is the highest degree polynomial for which this rule is exact? Solution: *Since there are three unknown coefficients to be determined, we look for three equations. We get these equations by plugging in successive polynomials. That is, we plug in* $f(x) = 1$, x, x^2 *and assuming the coefficients give equality:*

$$\begin{aligned}\int_0^1 1\,\mathrm{d}x &= 1 = A\,1 + B\,0 + C\,0 = A\\ \int_0^1 x\,\mathrm{d}x &= 1/2 = A\,1 + B\,1 + C\,0 = A + B\\ \int_0^1 x^2\,\mathrm{d}x &= 1/3 = A\,1 + B\,2 + C\,2 = A + 2B + 2C\end{aligned}$$

This is solved by $A = 1$, $B = -1/2$, $C = 1/6$. *This rule should be exact for polynomials of degree no greater than* 2, *but it might be better. We should check:*

$$\int_0^1 x^3\,\mathrm{d}x = 1/4 \neq 1/2 = 1 - 3/2 + 1 = A\,1 + B\,3 + C\,6,$$

and thus the rule is not *exact for cubic polynomials, or those of higher degree.* ⊣

10.4.3 Gaussian Nodes

It would seem this is the best we can do: using $n+1$ nodes we can devise a quadrature rule that is exact for polynomials of degree $\leq n$ by choosing the weights correctly. It turns out that by choosing the *nodes* in the right way, we can do far better. Gauss discovered that the right nodes to choose are the $n+1$ roots of the (nontrivial) polynomial, $q(x)$, of degree $n+1$ which has the property

$$\int_a^b x^k q(x)\,\mathrm{d}x = 0 \quad (0 \leq k \leq n)\,.$$

(If you view the integral as an inner product, you could say that $q(x)$ is orthogonal to the polynomials x^k in the resultant inner product space, but that's just fancy talk.)

Suppose that we have such a $q(x)$–we will not prove existence or uniqueness. Let $f(x)$ be a polynomial of degree $\leq 2n+1$. We write

$$f(x) = p(x)q(x) + r(x).$$

Both $p(x), r(x)$ are of degree $\leq n$. Because of how we picked $q(x)$ we have

$$\int_a^b p(x)q(x)\,\mathrm{d}x = 0.$$

Thus

$$\int_a^b f(x)\,\mathrm{d}x = \int_a^b p(x)q(x)\,\mathrm{d}x + \int_a^b r(x)dx = \int_a^b r(x)dx.$$

Now suppose that the A_i are chosen by Lagrange Polynomials so the quadrature rule on the nodes x_i is exact for polynomials of degree $\leq n$. Then

$$\sum_{i=0}^{n} A_i f(x_i) = \sum_{i=0}^{n} A_i \left[p(x_i)q(x_i) + r(x_i)\right] = \sum_{i=0}^{n} A_i r(x_i).$$

The last equality holds because the x_i are the roots of $q(x)$. Because of how the A_i are chosen we then have

$$\sum_{i=0}^{n} A_i f(x_i) = \sum_{i=0}^{n} A_i r(x_i) = \int_a^b r(x)\,\mathrm{d}x = \int_a^b f(x)\,\mathrm{d}x.$$

Thus this rule is exact for $f(x)$. We have (or rather, Gauss has) made quadrature twice as good.

Theorem 10.15 (Gaussian Quadrature Theorem)**.** *Let x_i be the $n+1$ roots of a (nontrivial) polynomial, $q(x)$, of degree $n+1$ which has the property*

$$\int_a^b x^k q(x)\,\mathrm{d}x = 0 \quad (0 \leq k \leq n)\,.$$

Let A_i be the coefficients for these nodes chosen by integrating the Lagrange Polynomials. Then the quadrature rule for this choice of nodes and coefficients is exact for polynomials of degree $\leq 2n+1$.

10.4.4 Determining Gaussian Nodes

We can determine the Gaussian nodes in the same way we determine coefficients. The example is illustrative

Example Problem 10.16. *Find the two Gaussian nodes for a quadrature rule on the interval $[0,2]$.* Solution: *We will find the function $q(x)$ of degree 2, which is "orthogonal" to $1, x$ under the inner product of integration over $[0,2]$. Thus we let $q(x) = c_0 + c_1 x + c_2 x^2$. The orthogonality condition becomes*

$$\int_0^2 1q(x)\,\mathrm{d}x = \int_0^2 xq(x)\,\mathrm{d}x = 0 \qquad \textit{that is,}$$
$$\int_0^2 c_0 + c_1 x + c_2 x^2\,\mathrm{d}x = \int_0^2 c_0 x + c_1 x^2 + c_2 x^3\,\mathrm{d}x = 0$$

Evaluating these integrals gives the following system of linear equations:

$$2c_0 + 2c_1 + \frac{8}{3}c_2 = 0,$$
$$2c_0 + \frac{8}{3}c_1 + 4c_2 = 0.$$

This system is "underdetermined," that is, there are two equations, but three unknowns. Notice, however, that if $q(x)$ satisfies the orthogonality conditions,

then so does $\hat{q}(x) = \alpha q(x)$, *for any real number* α. *That is, we can pick the scaling of* $q(x)$ *as we wish.*

With great foresight, we "guess" that we want $c_2 = 3$. *This reduces the equations to*

$$2c_0 + 2c_1 = -8,$$
$$2c_0 + \frac{8}{3}c_1 = -12.$$

*Simple Gaussian Elimination (*cf. *Chapter 7) yields the answer* $c_0 = 2, c_1 = -6, c_2 = 3$.

Then our nodes are the roots of $q(x) = 2 - 6x + 3x^2$. *That is the roots*

$$\frac{6 \pm \sqrt{36 - 24}}{6} = 1 \pm \frac{\sqrt{3}}{3}.$$

These nodes are a bit ugly. Rather than construct the Lagrange Polynomials, we will use the method of undetermined coefficients. Remember, we want to construct A_0, A_1 *such that*

$$\int_0^2 f(x)\,\mathrm{d}x \approx A_0 f(1 - \frac{\sqrt{3}}{3}) + A_1 f(1 + \frac{\sqrt{3}}{3})$$

is exact for polynomial $f(x)$ *of degree* ≤ 1. *It suffices to make this approximation exact for the "building blocks" of such polynomials, that is, for the functions* 1 *and* x. *That is, it suffices to find* A_0, A_1 *such that*

$$\int_0^2 1\,\mathrm{d}x = A_0 + A_1$$
$$\int_0^2 x\,\mathrm{d}x = A_0(1 - \frac{\sqrt{3}}{3}) + A_1(1 + \frac{\sqrt{3}}{3})$$

This gives the equations

$$2 = A_0 + A_1$$
$$2 = A_0(1 - \frac{\sqrt{3}}{3}) + A_1(1 + \frac{\sqrt{3}}{3})$$

This is solved by $A_0 = A_1 = 1$.

Thus our quadrature rule is

$$\boxed{\int_0^2 f(x)\,\mathrm{d}x \approx f(1 - \frac{\sqrt{3}}{3}) + f(1 + \frac{\sqrt{3}}{3})}.$$

We expect this rule to be exact for cubic polynomials. ⊣

Example Problem 10.17. *Verify the results of the previous example problem for* $f(x) = x^3$. Solution: *We have*

$$\int_0^2 f(x)\,\mathrm{d}x = (1/4)\,x^4\Big|_0^2 = 4.$$

The quadrature rule gives

$$\begin{aligned} f(1-\frac{\sqrt{3}}{3}) + f(1+\frac{\sqrt{3}}{3}) &= \left(1-\frac{\sqrt{3}}{3}\right)^3 + \left(1+\frac{\sqrt{3}}{3}\right)^3 \\ &= \left(1-3\frac{\sqrt{3}}{3}+3\frac{3}{9}-\frac{3\sqrt{3}}{27}\right) + \left(1+3\frac{\sqrt{3}}{3}+3\frac{3}{9}+\frac{3\sqrt{3}}{27}\right) \\ &= \left(2-\sqrt{3}-\sqrt{3}/9\right) + \left(2+\sqrt{3}+\sqrt{3}/9\right) = 4 \end{aligned}$$

Thus the quadrature rule is exact for $f(x)$. ⊣

10.4.5 Reinventing the Wheel

While it is good to know the theory, it doesn't make sense in practice to recompute these things all the time. There are books full of quadrature rules; any good textbook will list a few. The simplest ones are given in Table 10.1.
See also `http://mathworld.wolfram.com/Legendre-GaussQuadrature.html`

n	x_i	A_i
0	$x_0 = 0$	$A_0 = 2$
1	$x_0 = -\sqrt{1/3}$	$A_0 = 1$
	$x_1 = \sqrt{1/3}$	$A_1 = 1$
2	$x_0 = -\sqrt{3/5}$	$A_0 = 5/9$
	$x_1 = 0$	$A_1 = 8/9$
	$x_2 = \sqrt{3/5}$	$A_1 = 5/9$

Table 10.1: Gaussian Quadrature rules for the interval $[-1, 1]$. Thus $\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \sum_{i=0}^{n} A_i f(x_i)$, with this relation holding exactly for all polynomials of degree no greater than $2n + 1$.

Quadrature rules are normally given for the interval $[-1, 1]$. On first consideration, it would seem you need a different rule for each interval $[a, b]$. This is not the case, as the following example problem illustrates:

Example Problem 10.18. *Given a quadrature rule which is good on the interval* $[-1, 1]$*, derive a version of the rule to apply to the interval* $[a, b]$. Solution: *Consider the substitution:*

$$x = \frac{b-a}{2}t + \frac{b+a}{2}, \quad \textit{so} \quad \mathrm{d}x = \frac{b-a}{2}\,\mathrm{d}t.$$

Then

$$\int_a^b f(x)\,\mathrm{d}x = \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right)\frac{b-a}{2}\,\mathrm{d}t.$$

Letting

$$g(t) = \frac{b-a}{2} f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right),$$

if $f(x)$ is a polynomial, $g(t)$ is a polynomial of the same degree. Thus we can use the quadrature rule for $[-1,1]$ on $g(t)$ to evaluate the integral of $f(x)$ over $[a,b]$. ⊣

Example Problem 10.19. *Derive the quadrature rules of Example Problem 10.16 by using the technique of Example Problem 10.18 and the quadrature rules of Table 10.1.* Solution: *We have $a = 0, b = 2$. Thus*

$$g(t) = \frac{b-a}{2} f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right) = f(t+1).$$

To integrate $f(x)$ over $[0,2]$, we integrate $g(t)$ over $[-1,1]$. The standard Gaussian Quadrature rule approximates this as

$$\int_{-1}^1 g(t)\,\mathrm{d}t \approx g(-\sqrt{1/3}) + g(\sqrt{1/3}) = f(1-\sqrt{1/3}) + f(1+\sqrt{1/3}).$$

This is the same rule that was derived (with much more work) in Example Problem 10.16. ⊣

Exercises

(10.1) Use the composite trapezoidal rule, by hand, to approximate

$$\int_0^3 x^2 \, \mathrm{d}x \; (= 9)$$

Use the partition $\{x_i\}_{i=0}^2 = \{0, 1, 3\}$. Why is your approximation an overestimate?

(10.2) Use the composite trapezoidal rule, by hand, to approximate

$$\int_0^1 \frac{1}{x+1} \, \mathrm{d}x \; (= \ln 2 \approx 0.693)$$

Use the partition $\{x_i\}_{i=0}^3 = \left\{0, \frac{1}{4}, \frac{1}{2}, 1\right\}$. Why is your approximation an overestimate? (*Check:* I think the answer is 0.7)

(10.3) Use the composite trapezoidal rule, by hand to approximate

$$\int_0^1 \frac{4}{1+x^2} \, \mathrm{d}x.$$

Use $n = 4$ subintervals. How good is your answer?

(10.4) Use Theorem 10.8 to bound the error of the composite trapezoidal rule approximation of $\int_0^2 x^3 \, \mathrm{d}x$ with $n = 10$ intervals. You should find that the approximation is an overestimate.

(10.5) How many equal subintervals of $[0, 1]$ are required to approximate $\int_0^1 \cos x \; \mathrm{d}x$ with error smaller than 1×10^{-6} by the composite trapezoidal rule? (Use Theorem 10.8.)

(10.6) How many equal subintervals would be required to approximate

$$\int_0^1 \frac{4}{1+x^2} \, \mathrm{d}x.$$

to within 0.0001 by the composite trapezoidal rule? (*Hint:* Use the fact that $|f''(x)| \leq 8$ on $[0, 1]$ for $f(x) = 4/(1 + x^2)$)

(10.7) How many equal subintervals of $[2, 3]$ are required to approximate $\int_2^3 e^x \; \mathrm{d}x$ with error smaller than 1×10^{-3} by the composite trapezoidal rule?

(10.8) *Simpson's Rule* for quadrature is given as

$$\int_a^b f(x) \, \mathrm{d}x \approx \frac{\Delta x}{3} \left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n) \right],$$

where $\Delta x = (b-a)/n$, and n is assumed to be even. Show that Simpson's Rule for $n = 2$ is actually given by Romberg's Algorithm as $R(1, 1)$. As such we expect Simpson's Rule to be a $\mathcal{O}\left(h^4\right)$ approximation to the integral.

(10.9) Find a quadrature rule of the form

$$\int_0^1 f(x)\,\mathrm{d}x \approx Af(0) + Bf(1/2) + Cf(1)$$

that is exact for polynomials of highest possible degree. What is the highest degree polynomial for which this rule is exact?

(10.10) Determine a "quadrature" rule of the form

$$\int_{-1}^1 f(x)\,\mathrm{d}x \approx Af(0) + Bf'(-1) + Cf'(1)$$

that is exact for polynomials of highest possible degree. What is the highest degree polynomial for which this rule is exact? (Since this rule uses derivatives of f, it does not exactly fit our definition of a quadrature rule, but it may be applicable in some situations.)

(10.11) Determine a "quadrature" rule of the form

$$\int_0^1 f(x)\,\mathrm{d}x \approx Af(0) + Bf'(0) + Cf(1)$$

that is exact for polynomials of highest possible degree. What is the highest degree polynomial for which this rule is exact?

(10.12) Consider the so-called order n *Chebyshev Quadrature* rule:

$$\int_{-1}^1 f(x)\,\mathrm{d}x \approx c_n \sum_{i=0}^{n} f(x_i)$$

Find the weighting c_n and nodes x_i for the case $n = 2$ and the case $n = 3$. For what order polynomials are these rules exact?

(10.13) Find the Gaussian Quadrature rule with 2 nodes for the interval $[1, 5]$, *i.e.*, find a rule

$$\int_1^5 f(x)\,\mathrm{d}x \approx Af(x_0) + Bf(x_1)$$

Before you solve the problem, consider the following questions: do you expect the nodes to be the endpoints 1 and 5? do you expect the nodes to be arranged symmetrically around the midpoint of the interval?

(10.14) Find the Gaussian Quadrature rule with 3 nodes for the interval $[-1, 1]$, *i.e.*, find a rule

$$\int_{-1}^1 f(x)\,\mathrm{d}x \approx Af(x_0) + Bf(x_1) + Cf(x_2)$$

To find the nodes x_0, x_1, x_2 you will have to find the zeroes of a cubic equation, which could be difficult. However, you may use the simplifying assumption that the nodes are symmetrically placed in the interval $[-1, 1]$.

(10.15) Write code to approximate the integral of a f on $[a, b]$ by the composite trapezoidal rule on n equal subintervals. Your m-file should have header line like:

```
function iappx = trapezoidal(f,a,b,n)
```

You may wish to use the code:

```
x = a .+ (b-a) .* (0:n) ./ n;
```

If `f` is defined to work on vectors element-wise, you can probably speed up your computation by using

```
bigsum = 0.5 * ( f(x(1)) + f(x(n+1)) ) + sum( f(x(2:(n))) );
```

(10.16) Write code to implement the Gaussian Quadrature rule for $n = 2$ to integrate f on the interval $[a, b]$. Your m-file should have header line like:

```
function iappx = gauss2(f,a,b)
```

(10.17) Write code to implement composite Gaussian Quadrature based on code from the previous problem. Something like the following probably works:

```
function iappx = gaussComp(f,a,b,n)
% code to approximate integral of f over n equal subintervals of [a,b]
x = a .+ (b-a) .* (0:n) ./ n;
iappx = 0;
for i=1:n
  iappx += gauss2(f,x(i),x(i+1));
end
```

Use your code to approximate the *error function*:

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} \, \mathrm{d}t.$$

Compare your results with SciPy's builtin function `erf`. (See `http://mathworld.wolfram.com/Er`
The error function is used in probability. In particular, the probability that a normal random variable is within z standard deviations from its mean is

$$\operatorname{erf}(z/\sqrt{2})$$

Thus $\operatorname{erf}(1/\sqrt{2}) \approx 0.683$, and $\operatorname{erf}(2/\sqrt{2}) \approx 0.955$. These numbers should look familiar to you.

Chapter 11

Ordinary Differential Equations

Ordinary Differential Equations, or ODEs, are used in approximating some physical systems. Some classical uses include simulation of the growth of populations and trajectory of a particle. They are usually easier to solve or approximate than the more general Partial Differential Equations, PDEs, which can contain partial derivatives.

Much of the background material of this chapter should be familiar to the student of calculus. We will focus more on the approximation of solutions, than on analytical solutions. For more background on ODEs, see any of the standard texts, *e.g.*, [**?**].

11.1 Elementary Methods

A one-dimensional ODE is posed as follows: find some function $x(t)$ such that

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t)\right), \\ x(a) = c. \end{cases} \tag{11.1}$$

In calculus classes, you probably studied the case where $f\left(t, x(t)\right)$ is independent of its second variable. For example the ODE given by

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = t^2 - t, \\ x(a) = c. \end{cases}$$

has solution $x(t) = \frac{1}{3}t^3 - \frac{1}{2}t^2 + K$, where K is chosen such that $x(a) = c$.

A more general case is when $f(t, x)$ is separable, that is $f(t, x) = g(t)h(x)$ for some functions g, h. You may recall that the solution to such a separable ODE usually involved the following equation:

$$\int \frac{\mathrm{d}x}{h(x)} = \int g(t)\,\mathrm{d}t$$

Finding an analytic solution can be considerably more difficult in the general case, thus we turn to approximation schemes.

11.1.1 Integration and 'Stepping'

We attempt to solve the ODE by integrating both sides. That is

$$\begin{aligned} \frac{\mathrm{d}x(t)}{\mathrm{d}t} &= f\left(t, x(t)\right), \quad \text{yields} \\ \int_t^{t+h} \mathrm{d}x &= \int_t^{t+h} f\left(r, x(r)\right)\, \mathrm{d}r, \quad \text{thus} \\ x(t+h) &= x(t) + \int_t^{t+h} f\left(r, x(r)\right)\, \mathrm{d}r. \end{aligned} \tag{11.2}$$

If we can approximate the integral then we have a way of 'stepping' from t to $t+h$, *i.e.*, if we have a good approximate of $x(t)$ we can approximate $x(t+h)$. Note that this means that all the work we have put into approximating integrals can be used to approximate the solution of ODEs.

Using stepping schemes we can approximate $x(t_{final})$ given $x(t_{initial})$ by taking a number of steps. For simplicity we usually use the same step size, h, for each step, though nothing precludes us from varying the step size.

If we apply the left-hand rectangle rule for approximating integrals to equation 11.2, we get

$$\boxed{x(t+h) \approx x(t) + hf(t, x(t)).} \tag{11.3}$$

This is called *Euler's Method.* Note this is essentially the same as the "forward difference" approximation we made to the derivative, equation 9.1.

Trapezoid rule gives

$$x(t+h) \approx x(t) + \frac{h}{2}\left[f(t, x(t)) + f(t+h, x(t+h))\right].$$

But we cannot evaluate this exactly, since $x(t+h)$ appears on both sides of the equation. And it is embedded on the right hand side. Bummer. But if we could approximate it, say by using Euler's Method, maybe the formula would work. This is the idea behind the Runge-Kutta Method (see Section 11.2).

11.1.2 Taylor's Series Methods

We see that our more accurate integral approximations will be useless since they require information we do not know, *i.e.*, evaluations of $f(t, x)$ for yet unknown x values. Thus we fall back on Taylor's Theorem (Theorem 2.6). We can also view this as using integral approximations where all information comes from the left-hand endpoint.

By Taylor's theorem, if x has at least $m+1$ continuous derivatives on the interval in question, we can write

$$x(t+h) = x(t)+hx'(t)+\frac{1}{2}h^2x''(t)+\ldots+\frac{1}{m!}h^m x^{(m)}(t)+\frac{1}{(m+1)!}h^{m+1}x^{(m+1)}(\tau),$$

where τ is between t and $t+h$. Since τ is essentially unknown, our best calculable approximation to this expression is

$$x(t+h) \approx x(t) + hx'(t) + \frac{1}{2}h^2x''(t) + \ldots + \frac{1}{m!}h^m x^{(m)}(t).$$

This approximate solution to the ODE is called a *Taylor's series method of order* m. We also say that this approximation is a *truncation* of the Taylor's series. The difference between the actual value of $x(t+h)$, and this approximation is called the *truncation error.* The truncation error exists independently from any error in computing the approximation, called roundoff error. We discuss this more in Subsection **??**.

11.1.3 Euler's Method

When $m = 1$ we recover Euler's Method:

$$x(t+h) = x(t) + hx'(t) = x(t) + hf(t, x(t)).$$

Example 11.1. *Consider the ODE*

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = x, \\ x(0) = 1. \end{cases}$$

The actual solution is $x(t) = e^t$. Euler's Method will underestimate $x(t)$ because the curvature of the actual $x(t)$ is positive, and thus the function is always above its linearization. In Figure 11.1, we see that eventually the Euler's Method approximation is very poor.

11.1.4 Higher Order Methods

Higher order methods are derived by taking more terms of the Taylor's series expansion. Note however, that they will require information about the higher derivatives of $x(t)$, and thus may not be appropriate for the case where $f(t, x)$ is given as a "black box" function.

Example Problem 11.2. *Derive the Taylor's series method for $m = 3$ for the ODE*

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = x + e^x, \\ x(0) = 1. \end{cases}$$

Solution: *By simple calculus:*

$$\begin{aligned} x'(t) &= x + e^x \\ x''(t) &= x' + e^x x' \\ x'''(t) &= x''\left(1 + e^x\right) + x' e^x x' \end{aligned}$$

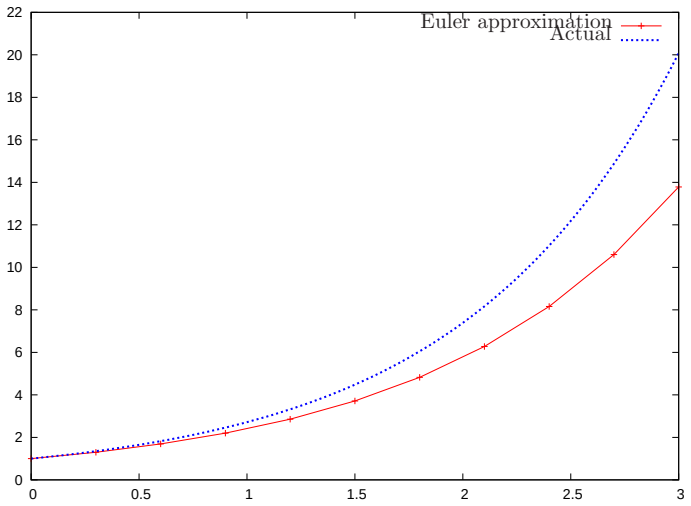


Figure 11.1: Euler's Method applied to approximate $x' = x$, $x(0) = 1$. Each step proceeds on the linearization of the curve, and is thus an underestimate, as the actual solution, $x(t) = e^t$ has positive curvature. Here step size $h = 0.3$ is used.

We can then write our step like a program:

$$\begin{aligned} x'(t) &\leftarrow x(t) + e^{x(t)} \\ x''(t) &\leftarrow x'(t)\left(1 + e^{x(t)}\right) \\ x'''(t) &\leftarrow x''(t)\left(1 + e^{x(t)}\right) + \left(x'(t)\right)^2 e^{x(t)} \\ x(t+h) &\approx x(t) + hx'(t) + \frac{1}{2}h^2x''(t) + \frac{1}{6}h^3x'''(t) \end{aligned}$$

⊣

11.1.5 A basic error estimate

Let's try to get a grip on how much error is introduced in each step of the Euler method. That is, suppose we already know x_{i-1} and we compute x_i by $x_i = x_{i-1} + f\left(t_{i-1}, x_{i-1}\right) h$.

Recall that x_i is our approximation of $x\left(t_i\right)$. Expanding x out in a Taylor series about t_{i-1} we get

$$x\left(t_i\right) = x\left(t_{i-1}\right) + x'\left(t_{i-1}\right) h + \frac{1}{2}x''(\xi)h^2$$

for some $\xi \in [t_{i-1}, t_i]$.

Thus, the error in one step is proportional to h^2. Now, if we apply Euler's method over an interval $[a, b]$, then we take approximately $(b-a)/h$ steps. Thus

the error is proportional to h. In particular, the error approaches 0 with the step size.

11.1.6 Error theorems

Since Taylor's theorem was applied to an unknown function x in the prior discussion, it can be a bit tricky to obtain an apriori bound on the error generated by Euler's method. Using the fact that $x' = f(t,x)$, however, we get that $x'' = f_t + f_x x' = f_t + f_x f$ from the multi-variable chain rule. Thus, it can be done. Here are a couple of theorems to that effect.

Theorem 11.1.1. *Suppose that f has continuous partial derivatives on the rectangle $R = [0,T] \times [A,B]$ in the tx-plane. Suppose also that*

$$M_1 = \sup_R \left| \frac{\partial f}{\partial x} \right| < \infty$$

and

$$M_2 = \sup_R \left| \frac{\partial f}{\partial t} + f \frac{\partial f}{\partial x} \right| < \infty.$$

Finally (and notably), suppose that the solution x to problem (1) satisfies $A \leq x(t) \leq B$ for all $t \in [0,T]$. Then the global error associated with Euler's method satisfies

$$|y(t_i) - y_i| \leq \left(\frac{e^{TM_1} - 1}{2M_1} \right) M_2 h,$$

for each $i = 1, 2, \ldots, n$.

The obvious problem with theorem 11.1.1 is that it still requires knowledge of the function x to be solved for. It does not require much information, however, so it can be useful. A strong conclusion under stronger assumptions is offered via the following theorem.

Theorem 11.1.2. *Suppose that f has continuous partial derivatives on the strip $S = [0,T] \times [-\infty, \infty]$ in the tx-plane. Suppose that*

$$M_1 = \sup_S \left| \frac{\partial f}{\partial x} \right| < \infty$$

and

$$M_2 = \sup_S \left| \frac{\partial f}{\partial t} + f \frac{\partial f}{\partial x} \right| < \infty.$$

Then the global error associated with Euler's method satisfies

$$|y(t_i) - y_i| \leq \left(\frac{e^{TM_1} - 1}{2M_1} \right) M_2 h,$$

for each $i = 1, 2, \ldots, n$.

11.1.7 Examples

Example 1

Consider the IVP $x' = x$, $x(0) = 1$. Of course, the solution is $x(t) = e^t$ so we have something to compare to. Suppose we want to approximate the solution over $[0, 2]$ with Euler's method and we want to ensure that we are within 0.01 of the actual value. How large should n be?

First note that $f(t, x) = x$, so $f_t = 0$ and $f_x = 1$. Thus, $M_1 = 1$. Unfortunately, to obtain a bound on M_2, we need to know how large x can be. This is essentially what we're trying to find in the first place! Nonetheless, let us assume that x never grows larger than 10. (In fact, we know that x never grows larger than $e^2 \approx 7.389$ on this interval.) Then $M_2 = 10$ so we need

$$\frac{e^2 - 1}{2} 10h \leq 10^{-2}$$

or $h < 0.000313$. Thus $n = \lceil 2/h \rceil < 6400$ will do.

In practice the bound for M_2 can frequently be obtained via an initial crude run of Euler's method.

Example 2

We now consider the IVP $x' = -\text{sint}(t) \sin(x) e^{-x^2}$, $x(0) = 1$. We'd like to approximate the solution over the interval $[0, 4]$ and obtain a solution that is good to within 0.01 of the actual value. How many pieces do we need and how do we do it?

Perhaps this is not the prettiest ODE you've ever seen, but keep in mind that we'll determine much relevant information (such as M_1 and M_2) numerically. The key point we're trying to understand is application of the error bound inequalities. From this perspective, the e^{-x^2} term is very nice since it converges to zero *very* rapidly as $u \to \pm\infty$, forcing the partial derivatives of $f(t, x)$ to be globally bounded.

In order to determine M_1 and M_2, we simply graphf f_x and $f_t + f f_x$ as shown in figure 11.2. It appears that the maximum of $|f_x|$ is about 1 and that the maximum of $|f_t + f f_x|$ is about 0.4.

Thus, we can take $M_1 = 1$ and $M_2 = 0.4$ to get

$$\frac{e^2 - 1}{2} 0.4h \leq 10^{-2}$$

or $h < 0.00782$. Thus, we can take $n = \lceil 4/0.00782 \rceil < 520$.

11.1.8 Stability

Suppose you had an exact method of solving an ODE, but had the wrong data. That is, you were trying to solve

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t)\right), \\ x(a) = c, \end{cases}$$

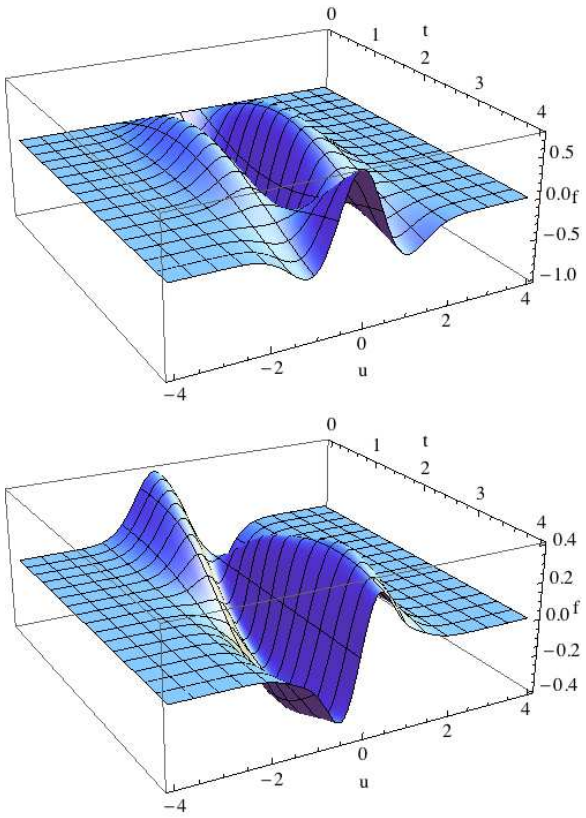


Figure 11.2: The graphs of f_x (top) and $f_t + f f_x$ (bottom)

but instead fed the wrong data into the computer, which then solved exactly the ODE

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t)\right), \\ x(a) = c + \epsilon. \end{cases}$$

This solution can diverge from the correct one because of the different starting conditions. We illustrate this with the usual example.

Example 11.3. *Consider the ODE*

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = x.$$

This has solution $x(t) = x(0)e^t$. The curves for different starting conditions diverge, as in Figure 11.3a.

However, if we instead consider the ODE

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = -x,$$

which has solution $x(t) = x(0)e^{-t}$, we see that differences in the initial conditions become immaterial, as in Figure 11.3b.

Thus the latter ODE exhibits stability: roundoff and truncation errors accrued at a given step will become irrelevant as more steps are taken. The former ODE exhibits the opposite behavior–accrued errors will be amplified.

It turns out there is a simple test for stability. If $f_x > \delta$ for some positive δ, then the ODE is instable; if $f_x < -\delta$ for a positive δ, the equation is stable. Some ODEs fall through this test, however.

We can prove this without too much pain. Define $x(t, s)$ to be the solution to

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t)\right), \\ x(a) = s, \end{cases}$$

for $t \geq a$.

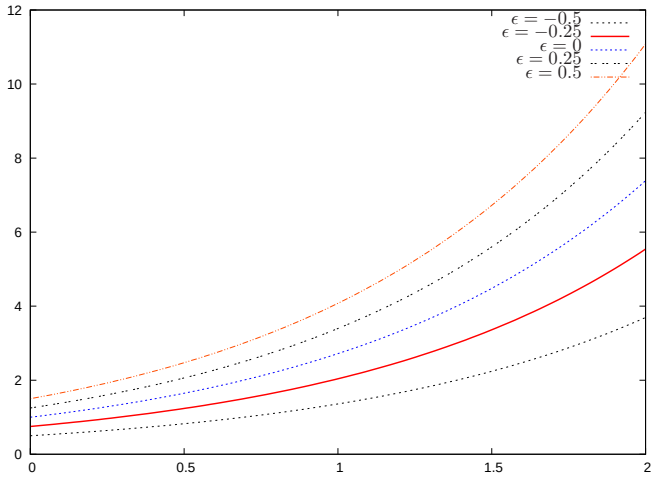
Then instability means that

$$\lim_{t\to\infty} \left| \frac{\partial}{\partial s} x(t, s) \right| = \infty.$$

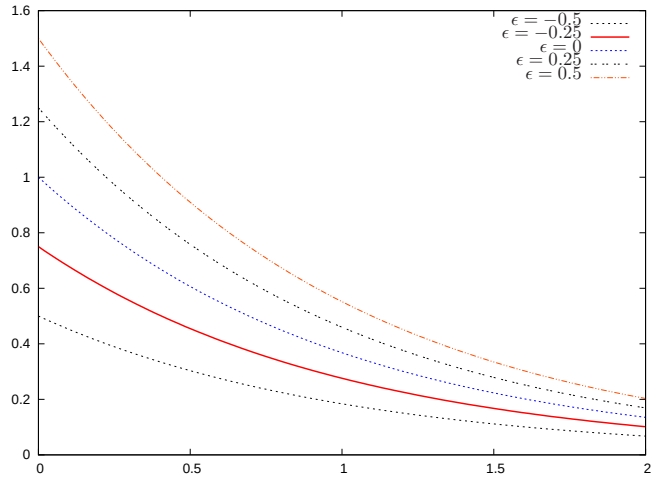
Think about this in terms of the curves for our simple example $x' = x$.

If we take the derivative of the ODE we get

$$\begin{aligned} \frac{\partial}{\partial t} x(t, s) &= f(t, x(t, s)), \\ \frac{\partial}{\partial s}\frac{\partial}{\partial t} x(t, s) &= \frac{\partial}{\partial s} f(t, x(t, s)), \quad \textit{i.e.}, \\ \frac{\partial}{\partial s}\frac{\partial}{\partial t} x(t, s) &= f_t(t, x(t))\frac{\partial t}{\partial s} + f_x(t, x(t, s))\frac{\partial x(t, s)}{\partial s}. \end{aligned}$$



(a) $(1+\epsilon)e^{t}$



(b) $(1+\epsilon)e^{-t}$

Figure 11.3: Exponential functions with different starting conditions: in (a), the functions $(1+\epsilon)e^{t}$ are shown, while in (b), $(1+\epsilon)e^{-t}$ are shown. The latter exhibits stability, while the former exhibits instability.

By the independence of t, s the first part of the RHS is zero, so

$$\frac{\partial}{\partial s}\frac{\partial}{\partial t}x(t,s) = f_x(t,x(t,s))\frac{\partial x(t,s)}{\partial s}.$$

We use continuity of x to switch the orders of differentiation:

$$\frac{\partial}{\partial t}\frac{\partial}{\partial s}x(t,s) = f_x(t,x(t,s))\frac{\partial}{\partial s}x(t,s).$$

Defining $u(t) = \frac{\partial}{\partial s}x(t,s)$, and $q(t) = f_x(t,x(t,s))$, we have

$$\frac{\partial}{\partial t}u(t) = q(t)u(t).$$

The solution is $u(t) = ce^{Q(t)}$, where

$$Q(t) = \int_a^t q(r)\,\mathrm{d}r.$$

If $q(r) = f_x(r,x(r,s)) \geq \delta > 0$, then $\lim Q(t) = \infty$, and so $\lim u(t) = \infty$. But note that $u(t) = \frac{\partial}{\partial s}x(t,s)$, and thus we have instability.

Similarly if $q(r) \leq -\delta < 0$, then $\lim Q(t) = -\infty$, and so $\lim u(t) = 0$, giving stability.

11.1.9 Backwards Euler's Method

Euler's Method is the most basic numerical technique for solving ODEs. However, it may suffer from instability, which may make the method inappropriate or impractical. However, it can be recast into a method which may have superior stability at the cost of limited applicability.

This method, the so-called Backwards Euler's Method, is a stepping method: given a reasonable approximation to $x(t)$, it calculates an approximation to $x(t+h)$. As usual, Taylor's Theorem is the starting point. Letting $t_f = t + h$, Taylor's Theorem states that

$$x(t_f - h) = x(t_f) + (-h)x'(t_f) + \frac{(-h)^2}{2}x''(t_f) + \ldots,$$

for an analytic function. Assuming that x has two continuous derivatives on the interval in question, the second order term is truncated to give

$$\begin{aligned} x(t) = x(t+h-h) = x(t_f - h) &\approx x(t_f) + (-h)x'(t_f) \\ \text{and so,} \qquad x(t+h) &\approx x(t) + hx'(t+h) \end{aligned}$$

The complication with applying this method is that $x'(t+h)$ may not be computable if $x(t+h)$ is not known, thus cannot be used to calculate an approximation for $x(t+h)$. Since this approximation may contain $x(t+h)$ on both

sides, we call this method an *implicit method*. In contrast, Euler's Method, and the Runge-Kutta methods in the following section are *explicit methods*: they can be used to directly compute an approximate step. We make this clear by examples.

Example 11.4. *Consider the ODE*

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = \cos x, \\ x(0) = 2\pi. \end{cases}$$

In this case, if we wish to approximate $x(0+h)$ using Backwards Euler's Method, we have to find $x(h)$ such that $x(h) = x(0) + \cos x(h)$. This is equivalent to finding a root of the equation

$$g(y) = 2\pi + \cos y - y = 0$$

The function $g(y)$ is nonincreasing ($g'(y)$ is nonpositive), and has a zero between 6 and 8, as seen in Figure 11.4. The techniques of Chapter 4 are needed to find the zero of this function.

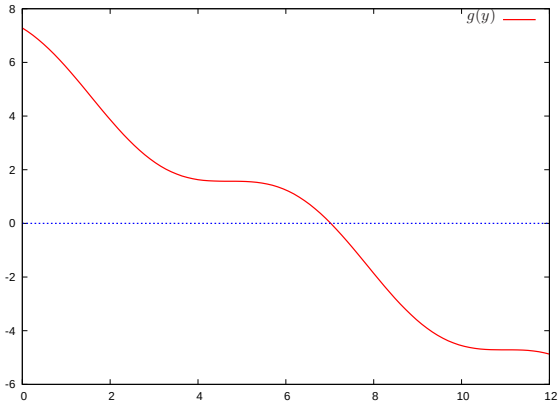


Figure 11.4: The nonincreasing function $g(y) = 2\pi + \cos y - y$ is shown.

Example 11.5. *Consider the ODE from Example 11.1:*

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = x, \\ x(0) = 1. \end{cases}$$

The actual solution is $x(t) = e^t$. Euler's Method underestimates $x(t)$, as shown in Figure 11.1. Using Backwards Euler's Method, gives the approximation:

$$x(t+h) = x(t) + hx(t+h),$$
$$x(t+h) = \frac{x(t)}{1-h}.$$

Using Backwards Euler's Method gives an over*estimate, as shown in Figure 11.5. This is because each step proceeds on the tangent line to a curve ke^t to a point on that curve. Generally Backwards Euler's Method is preferred over vanilla Euler's Method because it gives equivalent stability for larger stepsize h. This effect is not evident in this case.*

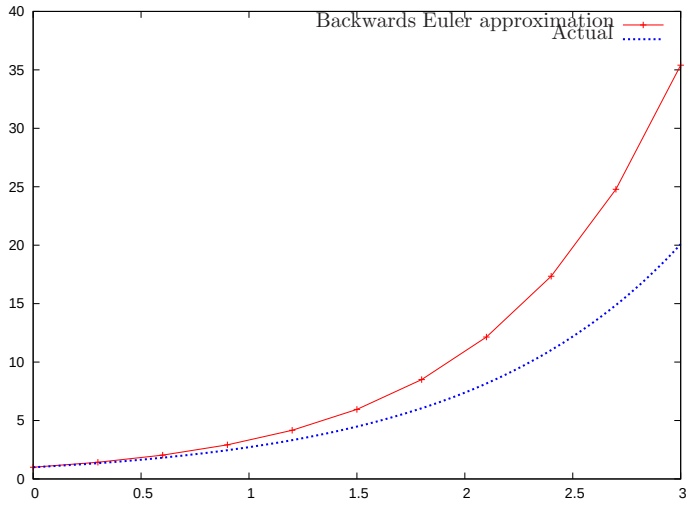


Figure 11.5: Backwards Euler's Method applied to approximate $x' = x$, $x(0) = 1$. The approximation is an overestimate, as each step is to a point on a curve ke^t, through the tangent at that point. Compare this figure to Figure 11.1, which shows the same problem approximated by Euler's Method, with the same stepsize, $h = 0.3$.

11.2 Runge-Kutta Methods

Recall the ODE problem: find some $x(t)$ such that

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t)\right), \\ x(a) = c, \end{cases}$$

where f, a, c are given.

Recall that the Taylor's series method has problems: either you are using the first order method (Euler's Method), which suffers inaccuracy, or you are using a higher order method which requires evaluations of higher order derivatives of $x(t)$, *i.e.*, $x''(t), x'''(t)$, etc. This makes these methods less useful for the general setting of f being a blackbox function. The Runge-Kutta Methods (don't ask me how it's pronounced) seek to resolve this.

11.2.1 Taylor's Series Redux

We fall back on Taylor's series, in this case the 2-dimensional version:

$$f(x+h,y+k)=\sum_{i=0}^{\infty}\frac{1}{i!}\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{i}f(x,y).$$

The thing in the middle is an operator on $f(x,y)$. The partial derivative operators are interpreted exactly as if they were algebraic terms, that is:

$$\begin{aligned}
\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{0}f(x,y)&=f(x,y),\\
\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{1}f(x,y)&=h\frac{\partial f(x,y)}{\partial x}+k\frac{\partial f(x,y)}{\partial y},\\
\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{2}f(x,y)&=h^2\frac{\partial^2 f(x,y)}{\partial x^2}+2hk\frac{\partial^2 f(x,y)}{\partial x\partial y}+k^2\frac{\partial^2 f(x,y)}{\partial y^2},\\
&\vdots
\end{aligned}$$

There is a truncated version of Taylors series:

$$f(x+h,y+k)=\sum_{i=0}^{n}\frac{1}{i!}\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{i}f(x,y)+\frac{1}{(n+1)!}\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{n+1}f(\bar{x},\bar{y}).$$

where $(\bar{x},\bar{y})$ is a point on the line segment with endpoints (x,y) and $(x+h,y+k)$.

For practice, we will write this for $n=1$:

$$f(x+h,y+k)=f(x,y)+hf_x(x,y)+kf_y(x,y)+\frac{1}{2}\left(h^2f_{xx}(\bar{x},\bar{y})+2hkf_{xy}(\bar{x},\bar{y})+k^2f_{yy}(\bar{x},\bar{y})\right)$$

11.2.2 Deriving the Runge-Kutta Methods

We now introduce the Runge-Kutta Method for stepping from $x(t)$ to $x(t+h)$. We suppose that α,β,w_1,w_2 are fixed constants. We then compute:

$$\begin{aligned}
K_1&\leftarrow hf(t,x)\\
K_2&\leftarrow hf(t+\alpha h,x+\beta K_1).
\end{aligned}$$

Then we approximate:

$$x(t+h)\leftarrow x(t)+w_1K_1+w_2K_2.$$

We now examine the "proper" choices of the constants. Note that $w_1=1,w_2=0$ corresponds to Euler's Method, and does not require computation of K_2. We will pick another choice.

Also notice that the definition of K_2 should be related to Taylor's theorem in two dimensions. Let's look at it:

$$\begin{aligned} K_2/h &= f(t+\alpha h, x+\beta K_1) \\ &= f(t+\alpha h, x+\beta h f(t,x)) \\ &= f + \alpha h f_t + \beta h f f_x + \frac{1}{2}\left(\alpha^2 h^2 f_{tt}(\bar{t},\bar{x}) + \alpha\beta f h^2 f_{tx}(\bar{t},\bar{x}) + \beta^2 h^2 f^2 f_x x(\bar{t},\bar{x})\right). \end{aligned}$$

Now reconsider our step:

$$\begin{aligned} x(t+h) &= x(t) + w_1 K_1 + w_2 K_2 \\ &= x(t) + w_1 h f + w_2 h f + w_2 \alpha h^2 f_t + w_2 \beta h^2 f f_x + \mathcal{O}\left(h^3\right). \\ &= x(t) + (w_1+w_2) h x'(t) + w_2 h^2 (\alpha f_t + \beta f f_x) + \mathcal{O}\left(h^3\right). \\ &= x(t) + (w_1+w_2) h x'(t) + w_2 h^2 (\alpha f_t + \beta f f_x) + \mathcal{O}\left(h^3\right). \end{aligned}$$

If we happen to choose our constants such that

$$w_1 + w_2 = 1, \quad \alpha w_2 = \frac{1}{2} = \beta w_2,$$

then we get

$$\begin{aligned} x(t+h) &= x(t) + h x'(t) + \frac{1}{2} h^2 (f_t + f f_x) + \mathcal{O}\left(h^3\right) \\ &= x(t) + h x'(t) + \frac{1}{2} h^2 x''(t) + \mathcal{O}\left(h^3\right), \end{aligned}$$

i.e., our choice of the constants makes the approximate $x(t+h)$ good up to a $\mathcal{O}\left(h^3\right)$ term, because we end up with the Taylor's series expansion up to that term. cool.

The usual choice of constants is $\alpha = \beta = 1, w_1 = w_2 = \frac{1}{2}$. This gives the *second order Runge-Kutta Method*:

$$x(t+h) \leftarrow x(t) + \frac{h}{2} f(t,x) + \frac{h}{2} f(t+h, x+hf(t,x)).$$

This can be written (and evaluated) as

$$\boxed{\begin{aligned} K_1 &\leftarrow h f(t,x) \\ K_2 &\leftarrow h f(t+h, x+K_1) \\ x(t+h) &\leftarrow x(t) + \frac{1}{2}(K_1 + K_2). \end{aligned}} \tag{11.4}$$

Another choice is $\alpha = \beta = 2/3, w_1 = 1/4, w_2 = 3/4$. This gives

$$x(t+h) \leftarrow x(t) + \frac{h}{4} f(t,x) + \frac{3h}{4} f\left(t + \frac{2h}{3}, x + \frac{2h}{3} f(t,x)\right).$$

The Runge-Kutta Method of order two has error term $\mathcal{O}\left(h^3\right)$. Sometimes this is not enough and higher-order Runge-Kutta Methods are used. The next Runge-Kutta Method is the order four method:

$$\boxed{\begin{aligned} K_1 &\leftarrow hf(t,x) \\ K_2 &\leftarrow hf(t+h/2, x+K_1/2) \\ K_3 &\leftarrow hf(t+h/2, x+K_2/2) \\ K_4 &\leftarrow hf(t+h, x+K_3) \\ x(t+h) &\leftarrow x(t) + \frac{1}{6}\left(K_1+2K_2+2K_3+K_4\right). \end{aligned}} \tag{11.5}$$

This method has order $\mathcal{O}\left(h^5\right)$. (See `http://mathworld.wolfram.com/Runge-KuttaMethod.html`)

The Runge-Kutta Method can be extrapolated to even higher orders. However, the number of function evaluations grows faster than the accuracy of the method. Thus the methods of order higher than four are normally not used.

We already saw that $w_1 = 1, w_2 = 0$ corresponds to Euler's Method. We consider now the case $w_1 = 0, w_2 = 1$. The method becomes

$$x(t+h) \leftarrow x(t) + hf\left(t + \frac{h}{2}, x + \frac{h}{2}f(t,x)\right).$$

This is called the *modified Euler's Method*. Note this gives a *different* value than if Euler's Method was applied twice with step size $h/2$.

11.2.3 Examples

Example 11.6. *Consider the ODE:*

$$\begin{cases} x' = (tx)^3 - \left(\frac{x}{t}\right)^2 \\ x(1) = 1 \end{cases}$$

Use $h = 0.1$ to compute $x(1.1)$ using both Taylor's Series Methods and Runge-Kutta methods of order 2.

11.3 Systems of ODEs

Recall the regular ODE problem: find some $x(t)$ such that

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t)\right), \\ x(a) = c, \end{cases}$$

where f, a, c are given.

Sometimes the physical systems we are considering are more complex. For example, we might be interested in the system of ODEs:

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t), y(t)\right), \\ \frac{\mathrm{d}y(t)}{\mathrm{d}t} = g\left(t, x(t), y(t)\right), \\ x(a) = c, \\ y(a) = d. \end{cases}$$

Example 11.7. *Consider the following system of ODEs:*

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = t^2 - x \\ \frac{\mathrm{d}y(t)}{\mathrm{d}t} = y^2 + y - t, \\ x(0) = 1, \\ y(0) = 0. \end{cases}$$

You should immediately notice that this is not a system at all, but rather a collection of two ODEs:

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = t^2 - x \\ x(0) = 1, \end{cases}$$

and

$$\begin{cases} \frac{\mathrm{d}y(t)}{\mathrm{d}t} = y^2 + y - t, \\ y(0) = 0. \end{cases}$$

These two ODEs can be solved separately. We call such a system uncoupled. *In an uncoupled system the function $f(t, x, y)$ is independent of y, and $g(t, x, y)$ is independent of x. A system which is not uncoupled, is, of course,* coupled. *We will not consider uncoupled systems.*

11.3.1 Larger Systems

There is no need to stop at two functions. We may imagine we have to solve the following problem: find $x_1(t), x_2(t), \ldots, x_n(t)$ such that

$$\begin{cases} \frac{\mathrm{d}x_1(t)}{\mathrm{d}t} = f_1\left(t, x_1(t), x_2(t), \ldots, x_n(t)\right), \\ \frac{\mathrm{d}x_2(t)}{\mathrm{d}t} = f_2\left(t, x_1(t), x_2(t), \ldots, x_n(t)\right), \\ \vdots \\ \frac{\mathrm{d}x_n(t)}{\mathrm{d}t} = f_n\left(t, x_1(t), x_2(t), \ldots, x_n(t)\right), \\ x_1(a) = c_1,\ x_2(a) = c_2,\ \ldots, x_n(a) = c_n. \end{cases}$$

The idea is to not be afraid of the notation, write everything as vectors, then *do exactly the same thing as for the one dimensional case!* That's right, there is nothing new but notation:

Let

$$\boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}, \quad \boldsymbol{X'} = \begin{bmatrix} x_1' \\ x_2' \\ \cdots \\ x_n' \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} c_1 \\ c_2 \\ \cdots \\ c_n \end{bmatrix}.$$

Then we want to find $\boldsymbol{X}$ such that

$$\begin{cases} \boldsymbol{X'}(t) = \boldsymbol{F}\left(t, \boldsymbol{X}(t)\right) \\ \boldsymbol{X}\left(a\right) = \boldsymbol{C}. \end{cases} \tag{11.6}$$

Compare this to equation 11.1.

11.3.2 Recasting Single ODE Methods

We will consider stepping methods for solving higher order ODEs. That is, we know $\boldsymbol{X}(t)$, and want to find $\boldsymbol{X}(t+h)$. Most of the methods we looked at for solving one dimensional ODEs can be rewritten to solve higher dimensional problems.

For example, Euler's Method, equation 11.3 can be written as

$$\boxed{\boldsymbol{X}(t+h) \leftarrow \boldsymbol{X}(t) + h\boldsymbol{F}\left(t, \boldsymbol{X}(t)\right).}$$

As in 1D, this method is derived from approximating $\boldsymbol{X}$ by its linearization.

In fact, our general strategy of using Taylor's Theorem also carries through without change. That is we can use the k^{th} order method to step as follows:

$$\boldsymbol{X}(t+h) \leftarrow \boldsymbol{X}(t) + h\boldsymbol{X'}(t) + \frac{h^2}{2}\boldsymbol{X''}(t) + \ldots + \frac{h^k}{k!}\boldsymbol{X^{(k)}}(t).$$

Additionally we can write the Runge-Kutta Methods as follows:

Order two:

$$\boxed{\begin{aligned} \boldsymbol{K_1} &\leftarrow h\boldsymbol{F}\left(t, \boldsymbol{X}\right) \\ \boldsymbol{K_2} &\leftarrow h\boldsymbol{F}\left(t+h, \boldsymbol{X}+\boldsymbol{K_1}\right) \\ \boldsymbol{X}(t+h) &\leftarrow \boldsymbol{X}(t) + \frac{1}{2}\left(\boldsymbol{K_1}+\boldsymbol{K_2}\right). \end{aligned}}$$

Order four:

$$\boxed{\begin{aligned} \boldsymbol{K_1} &\leftarrow h\boldsymbol{F}\left(t, \boldsymbol{X}\right) \\ \boldsymbol{K_2} &\leftarrow h\boldsymbol{F}\left(t+\frac{1}{2}h, \boldsymbol{X}+\frac{1}{2}\boldsymbol{K_1}\right) \\ \boldsymbol{K_3} &\leftarrow h\boldsymbol{F}\left(t+\frac{1}{2}h, \boldsymbol{X}+\frac{1}{2}\boldsymbol{K_2}\right) \\ \boldsymbol{K_4} &\leftarrow h\boldsymbol{F}\left(t+h, \boldsymbol{X}+\boldsymbol{K_3}\right) \\ \boldsymbol{X}(t+h) &\leftarrow x(t) + \frac{1}{6}\left(\boldsymbol{K_1}+2\boldsymbol{K_2}+2\boldsymbol{K_3}+\boldsymbol{K_4}\right). \end{aligned}}$$

Example Problem 11.8. *Consider the system of ODEs:*

$$\begin{cases} x_1'(t) = x_2 - x_3^2 \\ x_2'(t) = t + x_1 + x_3 \\ x_3'(t) = x_2 - x_1^2 \\ x_1(0) = 1, \\ x_2(0) = 0, \\ x_3(0) = 1. \end{cases}$$

Approximate $\boldsymbol{X}(0.1)$ by taking a single step of size 0.1, for Euler's Method, and the Runge-Kutta Method of order 2. Solution: *We write*

$$\boldsymbol{F}(t, \boldsymbol{X}(t)) = \begin{bmatrix} x_2(t) - x_3^2(t) \\ t + x_1(t) + x_3(t) \\ x_2(t) - x_1^2(t) \end{bmatrix} \quad \boldsymbol{X}(0) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Thus we have

$$\boldsymbol{F}(0, \boldsymbol{X}(0)) = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}$$

Euler's Method then makes the approximation

$$\boldsymbol{X}(0.1) \leftarrow \boldsymbol{X}(0) + 0.1\boldsymbol{F}(0, \boldsymbol{X}(0)) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.1 \\ 0.2 \\ -0.1 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.2 \\ 0.9 \end{bmatrix}$$

The Runge-Kutta Method computes:

$$\boldsymbol{K_1} \leftarrow 0.1\boldsymbol{F}(0, \boldsymbol{X}(0)) = \begin{bmatrix} -0.1 \\ 0.2 \\ -0.1 \end{bmatrix} \quad \textit{and} \quad \boldsymbol{K_2} \leftarrow 0.1\boldsymbol{F}(0.1, \boldsymbol{X}(0)+\boldsymbol{K_1}) = \begin{bmatrix} -0.061 \\ 0.19 \\ -0.061 \end{bmatrix}$$

Then

$$\boldsymbol{X}(0.1) \leftarrow \boldsymbol{X}(0) + \frac{1}{2}(\boldsymbol{K_1} + \boldsymbol{K_2}) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.0805 \\ 0.195 \\ -0.0805 \end{bmatrix} = \begin{bmatrix} 0.9195 \\ 0.195 \\ 0.9195 \end{bmatrix}$$

⊣

11.3.3 It's Only Systems

The fact we can simply solve systems of ODEs using old methods is good news. It allows us to solve higher order ODEs. For example, consider the following problem: given $f, a, c_0, c_1, \ldots, c_n$, find $x(t)$ such that

$$\begin{cases} x^{(n)}(t) = f\left(t, x(t), x'(t), \ldots, x^{(n-1)}(t)\right), \\ x(a) = c_0,\ x'(a) = c_1,\ x''(a) = c_2,\ \ldots,\ x^{(n-1)}(a) = c_{n-1}. \end{cases}$$

We can put this in terms of a system by letting

$$x_0 = x,\ x_1 = x',\ x_2 = x'',\ \ldots,\ x_{n-1} = x^{(n-1)}.$$

Then the ODE plus these $n-1$ equations can be written as

$$\begin{cases} x'_{n-1}(t) = f\left(t, x_0(t), x_1(t), x_2(t), \ldots, x_{n-1}(t)\right) \\ x'_0(t) = x_1(t) \\ x'_1(t) = x_2(t) \\ \vdots \\ x'_{n-2}(t) = x_{n-1}(t) \\ x_0(a) = c_0,\ x_1(a) = c_1,\ x_2(a) = c_2,\ \ldots,\ x_{n-1}(a) = c_{n-1}. \end{cases}$$

This is just a system of ODEs that we can solve like any other.

Note that this trick also works on systems of higher order ODEs. That is, we can transform any system of higher order ODEs into a (larger) system of first order ODEs.

Example Problem 11.9. *Rewrite the following system as a system of first order ODEs:*

$$\begin{cases} x''(t) = (t/y(t)) + x'(t) - 1 \\ y'(t) = \frac{1}{(x'(t)+y(t))} \\ x(0) = 1,\, x'(0) = 0, y(0) = -1 \end{cases}$$

Solution: *We let* $x_0 = x$, $x_1 = x'$, $x_2 = y$, *then we can rewrite as*

$$\begin{cases} x_1'(t) = (t/x_2(t)) + x_1(t) - 1 \\ x_0'(t) = x_1(t) \\ x_2'(t) = \frac{1}{(x_1(t)+x_2(t))} \\ x_0(0) = 1,\, x_1(0) = 0, x_2(0) = -1 \end{cases}$$

⊣

11.3.4 It's Only Autonomous Systems

In fact, we can use a similar trick to get rid of the time-dependence of an ODE. Consider the first order system

$$\begin{cases} x_1' = f_1(t, x_1, x_2, \ldots, x_n), \\ x_2' = f_2(t, x_1, x_2, \ldots, x_n), \\ x_3' = f_3(t, x_1, x_2, \ldots, x_n), \\ \quad \vdots \\ x_n' = f_n(t, x_1, x_2, \ldots, x_n), \\ x_1(a) = c_1,\, x_2(a) = c_2,\, \ldots, x_n(a) = c_n. \end{cases}$$

We can get rid of the time dependence and thus make the system *autonomous* by making t another variable function to be found. We do this by letting $x_0 = t$, then adding the equations $x_0' = 1$, and $x_0(a) = a$, to get the system:

$$\begin{cases} x_0' = 1, \\ x_1' = f_1(x_0, x_1, x_2, \ldots, x_n), \\ x_2' = f_2(x_0, x_1, x_2, \ldots, x_n), \\ x_3' = f_3(x_0, x_1, x_2, \ldots, x_n), \\ \quad \vdots \\ x_n' = f_n(x_0, x_1, x_2, \ldots, x_n), \\ x_0(a) = a,\, x_1(a) = c_1,\, x_2(a) = c_2,\, \ldots, x_n(a) = c_n. \end{cases}$$

An autonomous system can be written in the more elegant form:

$$\begin{cases} \boldsymbol{X'} = \boldsymbol{F}(\boldsymbol{X}) \\ \boldsymbol{X}(a) = \boldsymbol{C}. \end{cases}$$

Moreover, we can consider the *phase curve* of an autonomous system.

One can consider $\boldsymbol{X}$ to be the position of a particle which moves through $\mathbb{R}^n$ over time. For an autonomous system of ODEs, the movement of the particle

is dependent only on its current position and not on time.[1] A phase curve is a flow line in the vector field $\boldsymbol{F}$. You can think of the vector field $\boldsymbol{F}$ as being the velocity, at a given point, of a river, the flow of which is time independent. Under this analogy, the phase curve is the path of a vessel placed in the river. In our deterministic world, phase curves never cross. This is because at the point of crossing, there would have to be two different values of the tangent of the curve, *i.e.*, the vector field would have to have two different values at that point.

The following example illustrates the idea of phase curves for autonomous ODE.

Example 11.10. *Consider the autonomous system of ODEs, without an initial value:*

$$\begin{cases} x_1'(t) = -2(x_2 - 2.4) \\ x_2'(t) = 3(x_1 - 1.2) \end{cases}$$

We can rewrite this ODE as

$$\boldsymbol{X}'(t) = \boldsymbol{F}\left(\boldsymbol{X}(t)\right).$$

This is an autonomous ODE. The associated vector field can be expressed as

$$\boldsymbol{F}\left(\boldsymbol{X}\right) = \begin{bmatrix} 0 & -2 \\ 3 & 0 \end{bmatrix} \boldsymbol{X} + \begin{bmatrix} 4.8 \\ -3.6 \end{bmatrix}.$$

This vector field is plotted in Figure 11.6.

You should verify that this ODE is solved by

$$\boldsymbol{X}\left(t\right) = r \begin{bmatrix} \sqrt{2}\cos(\sqrt{6}t + t_0) \\ \sqrt{3}\sin(\sqrt{6}t + t_0) \end{bmatrix} + \begin{bmatrix} 1.2 \\ 2.4 \end{bmatrix},$$

for any $r \geq 0$, and $t_0 \in (0, 2\pi]$. Given an initial value, the parameters r and t_0 are uniquely determined. Thus the trajectory of $\boldsymbol{X}$ over time is that of an ellipse in the plane.[2] *Thus the family of such ellipses, taking all $r \geq 0$, form the phase curves of this ODE.* We would expect the approximation of an ODE to never cross a phase curve. *This is because the phase curve represents the*

Euler's Method and the second order Runge-Kutta Method were used to approximate the ODE with initial value

$$\boldsymbol{X}\left(0\right) = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}.$$

Euler's Method was used for step size $h = 0.005$ for 3000 steps. The Runge-Kutta Method was employed with step size $h = 0.015$ for 3000 steps. The approximations are shown in Figure 11.7.

[1] Note that if you have converted a system of n equations with a time dependence to an autonomous system, then the autonomous system should be considered a particle moving through $\mathbb{R}^{n+1}$. In this case time becomes one of the dimensions, and thus we speak of position, instead of time.

[2] In the case $r = 0$, the ellipse is the "trivial" ellipse which consists of a point.

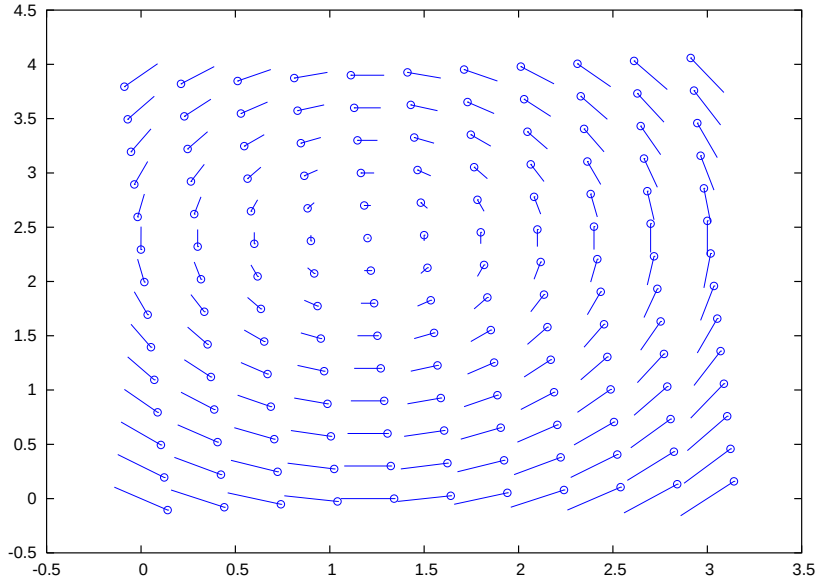
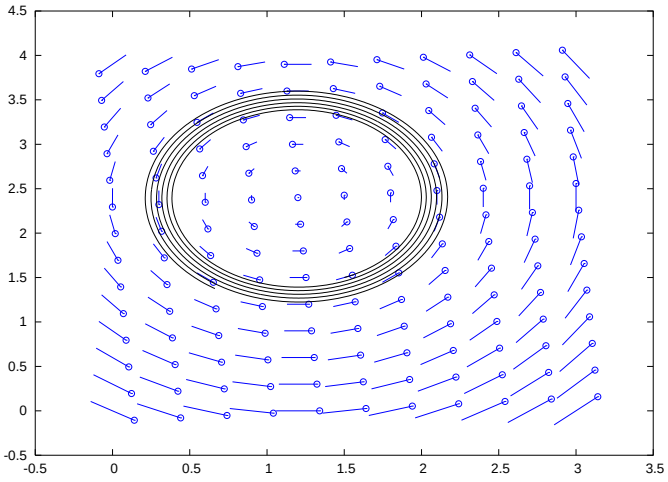


Figure 11.6: The vector field $\boldsymbol{F}(\boldsymbol{X})$ is shown in $\mathbb{R}^2$. The tips of the vectors are shown as circles. (Due to budget restrictions, arrowheads were not available for this figure.)

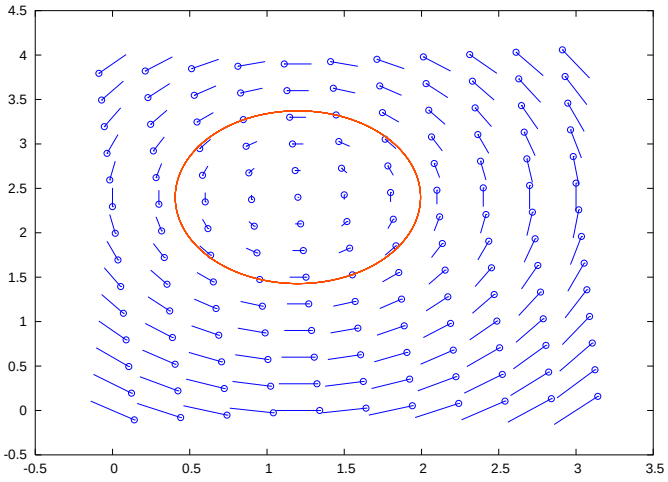
Given that the actual solution of this initial value problem is an ellipse, we see that Euler's Method performed rather poorly, spiralling out from the ellipse. This must be the case for any *step size, since Euler's Method steps in the direction tangent to the actual solution; thus every step of Euler's Method puts the approximation on an ellipse of larger radius. Smaller stepsize minimizes this effect, but at greater computational cost.*

The Runge-Kutta Method performs much better, and for larger step size. At the given resolution, no spiralling is observable. Thus the Runge-Kutta Method outperforms Euler's Method, and at lesser computational cost.[3]

[3]Because the Runge-Kutta Method of order two requires two evaluations of $\boldsymbol{F}$, whereas Euler's Method requires only one, per step, it is expected that they would be 'evenly matched' when Runge-Kutta Method uses a step size twice that of Euler's Method.



(a) Euler's Method



(b) Runge-Kutta Method

Figure 11.7: The simulated phase curves of the system of Example 11.10 are shown for (a) Euler's Method and (b) the Runge-Kutta Method of order 2. The Runge-Kutta Method used larger step size, but was more accurate. The actual solution to the ODE is an ellipse, thus the "spiralling" observed for Euler's Method is spurious.

EXERCISES

(11.1) Given the ODE:

$$\begin{cases} x'(t) = t^2 - x^2 \\ x(1) = 1 \end{cases}$$

Approximate $x(1.1)$ by using one step of Euler's Method. Approximate $x(1.1)$ using one step of the second order Runge-Kutta Method.

(11.2) Given the ODE:

$$\begin{cases} x_1'(t) = x_1 x_2 + t \\ x_2'(t) = 2x_2 - x_1^2 \\ x_1(1) = 0 \\ x_2(1) = 3 \end{cases}$$

Approximate $\boldsymbol{X}(1.1)$ by using one step of Euler's Method. Approximate $\boldsymbol{X}(1.1)$ using one step of the second order Runge-Kutta Method.

(11.3) Consider the separable ODE

$$x'(t) = x(t)g(t),$$

for some function g. Use backward's Euler (or, alternatively, the right endpoint rule for approximating integrals) to derive the approximation scheme

$$x(t+h) \leftarrow \frac{x(t)}{1 - hg(t+h)}.$$

What could go wrong with using such a scheme?

(11.4) Consider the ODE

$$x'(t) = x(t) + t^2.$$

Using Taylor's Theorem, derive a higher order approximation scheme to find $x(t+h)$ based on $x(t)$, t, and h.

(11.5) Which of the following ODEs can you show are stable? Which are unstable? Which seem ambiguous?
(a) $x'(t) = -x - \arctan x$ (b) $x'(t) = 2x + \tan x$ (c) $x'(t) = -4x - e^x$
(d) $x'(t) = x + x^3$ (e) $x'(t) = t + \cos x$

(11.6) Rewrite the following higher order ODE as a system of first order ODEs:

$$\begin{cases} x''(t) = x(t) - \sin x'(t), \\ x(0) = 0, \\ x'(0) = \pi. \end{cases}$$

(11.7) Rewrite the system of ODEs as a system of first order ODEs

$$\begin{cases} x''(t) = y(t) + t - x(t), \\ y'(t) = x'(t) + x(t) - 4, \\ x'(0) = 1, \\ x(0) = 2, \\ y(0) = 0. \end{cases}$$

(11.8) Rewrite the following higher order system of ODEs as a system of first order ODEs:

$$\begin{cases} x'''(t) = y''(t) - x'(t), \\ y'''(t) = x''(t) + y'(t), \\ x(0) = x''(0) = y'(0) = -1, \\ x'(0) = y(0) = y''(0) = 1. \end{cases}$$

(11.9) Implement Euler's Method for approximating the solution to

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t)\right), \\ x(a) = c. \end{cases}$$

Your m-file should have header line like:
`function xfin = euler(f,a,c,h,steps)`
where `xfin` should be the approximate solution to the ODE at time `a` + `steps` ⋆ `h`.

(a) Run your code with `f(t,x)` = −`x`, using `a` = 0 ≠ `c`, and using `h` ⋆ `steps` = 1.0. In this case the actual solution is

$$\texttt{xfin} = \texttt{c}e^{-1}.$$

This ODE is stable, so your approximation should be good (*cf.* Example 11.3). Experiment with different values of `h`.

(b) Run your code with `f(t,x)` = `x`, using `a` = 0 ≠ `c`, and using `h` ⋆ `steps` = 1.0. In this case the actual solution is

$$\texttt{xfin} = \texttt{c}e.$$

Your actual solution may be a rather poor approximation. Prepare a plot of error versus `h` for varying values of `h`. (*cf.* Figure 9.1 and Figure **??**)

(c) Run your code with `f(t,x)` = 1/(1 − `x`), using `a` = 0, `c` = 0.1, and using `h` ⋆ `steps` = 2.0. Plot your approximations for varying values of `steps`. For example, try `steps` values of 35, 36, and 90, 91. Also try large values of `steps`, like 100, 500, 1000. Can you guess what the actual solution is supposed to look like? Can you explain the poor performance of Euler's Method for this ODE?

(11.10) Implement the Runge-Kutta Method of order two for approximating the solution to

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left(t, x(t)\right), \\ x(a) = c. \end{cases}$$

Your m-file should have header line like:
`function xfin = rkm(f,a,c,h,steps)`
where `xfin` should be the approximate solution to the ODE at time `a` + `steps` ⋆ `h`.
Test your code with the ODEs from the previous problem. Does the Runge-Kutta Method perform any better than Euler's Method for the last ODE?

Appendix A

GNU Free Documentation License

Version 1.2, November 2002
Copyright ©2000,2001,2002 Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or

XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example,

statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or

any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.